

Eastern
Economy
Edition

ARTIFICIAL NEURAL NETWORKS

B. YEGNANARAYANA



Artificial Neural Networks

B. YEGNANARAYANA

Professor

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Chennai

Prentice-Hall of India Private Limited

New Delhi - 110 001

2005

Rs. 275.00

ARTIFICIAL NEURAL NETWORKS
by **B. Yegnanarayana**

© 1999 by Prentice-Hall of **India** Private Limited, New Delhi. All rights reserved.
No part of this book may be reproduced in any form, by mimeograph or any
other means, without permission in writing from the publisher.

ISBN-81-203-1253-8

The export rights of this book are vested solely with the publisher.

Eleventh Printing June, 2005

Published by **Asoke K. Ghosh**, Prentice-Hall of **India** Private Limited, M-97,
Connaught Circus, New Delhi-110001 and Printed by **Rajkamal** Electric Press,
B-35/9, G.T. Karnal Road Industrial Area, Delhi-110033.

To My Parents
B. Ramamurthy
and
B. Savitri

Contents

<i>Preface</i>	<i>ix</i>
<i>Acknowledgements</i>	<i>xiii</i>
INTRODUCTION	1-14
Trends in Computing	2
Pattern and Data	4
Pattern Recognition Tasks	6
Methods for Pattern Recognition Tasks	8
Organization of the Topics	10
REVIEW QUESTIONS	13
1. BASICS OF ARTIFICIAL NEURAL NETWORKS	15-39
1.1 Characteristics of Neural Networks	15
1.2 Historical Development of Neural Network Principles	21
1.3 Artificial Neural Networks: Terminology	24
1.4 Models of Neuron	26
1.5 Topology	29
1.6 Basic Learning Laws	31
1.7 Summary	36
REVIEW QUESTIONS	37
PROBLEMS	38
2. ACTIVATION AND SYNAPTIC DYNAMICS	40-75
2.1 Introduction	40
2.2 Activation Dynamics Models	42
2.3 Synaptic Dynamics Models	52
2.4 Learning Methods	57
2.5 Stability and Convergence	68
2.6 Recall in Neural Networks	72
2.7 Summary	73
REVIEW QUESTIONS	73
PROBLEMS	74
3. FUNCTIONAL UNITS OF ANN FOR PATTERN RECOGNITION TASKS	76-07
3.1 Pattern Recognition Problem	77

3.2	Basic Functional Units	78
3.3	Pattern Recognition Tasks by the Functional Units	79
	REVIEW QUESTIONS	87
4.	FEEDFORWARD NEURAL NETWORKS	88-141
4.1	Introduction	88
4.2	Analysis of Pattern Association Networks	90
4.3	Analysis of Pattern Classification Networks	99
4.4	Analysis of Pattern Mapping Networks	113
4.5	Summary and Discussion	135
	REVIEW QUESTIONS	136
	PROBLEMS	138
5.	FEEDBACK NEURAL NETWORKS	142-200
5.1	Introduction	142
5.2	Analysis of Linear Autoassociative FF Networks	144
5.3	Analysis of Pattern Storage Networks	146
5.4	Stochastic Networks and Simulated Annealing	165
5.5	Boltzmann Machine	183
5.6	Summary	196
	REVIEW QUESTIONS	197
	PROBLEMS	199
6.	COMPETITIVE LEARNING NEURAL NETWORKS	201-232
6.1	Introduction	201
6.2	Components of a Competitive Learning Network	203
6.3	Analysis of Feedback Layer for Different Output Functions	211
6.4	Analysis of Pattern Clustering Networks	218
6.5	Analysis of Feature Mapping Network	223
6.6	Summary	228
	REVIEW QUESTIONS	229
	PROBLEMS	230
7.	ARCHITECTURES FOR COMPLEX PATTERN RECOGNITION TASKS	233-277
7.1	Introduction	233
7.2	Associative Memory	235
7.3	Pattern Mapping	240
7.4	Stability-Plasticity Dilemma: ART'	258
7.5	Temporal Patterns	265
7.6	Pattern Variability: Neocognitron	271
7.7	Summary	273
	REVIEW QUESTIONS	273
	PROBLEMS	276

8. APPLICATIONS OF ANN	278–339
8.1 Introduction	278
8.2 Direct Applications	280
8.3 Application Areas	306
8.4 Summary	334
REVIEW QUESTIONS	336
PROBLEMS	338
 <i>Appendices</i>	 341–397
A — Features of Biological Neural Networks through Parallel and Distributed Processing Models	341
B — Mathematical Preliminaries	351
C — Basics of Gradient Descent Methods	364
D — Generalization in Neural Networks: An Overview	372
E — Principal Component Neural Networks: An Overview	379
F — Current Trends in Neural Networks	391
 <i>Bibliography</i>	 399–431
<i>Author Index</i>	433–440
<i>Subject Index</i>	441–461

Preface

Over the past **fifteen** years, a view has emerged that computing based on models inspired by our understanding of the structure and function of the biological neural networks may hold the key to the success of solving intelligent tasks by machines. The new field is called ***Artificial Neural Networks***, although it is more apt to describe it as parallel and distributed processing. This introductory book is aimed at giving the basic principles of computing with models of artificial neural networks, without giving any judgment on their capabilities in solving intelligent tasks.

This text is an outgrowth of the author's teaching and research experience for the past 25 years in the areas of speech and image processing, artificial intelligence and neural networks. The principles of neural networks are closely related to such areas as pattern recognition, signal processing and artificial intelligence. Over the past 10 years many excellent books have been published in the area of artificial neural networks and many more are being published. Thus one more book like this may seem redundant. However, there seems to be still a need for a book that could be used as a text book at an introductory level. This text is designed to meet such a demand. It must be pointed out that most of the ideas presented here have been taken from the available references and mainly from the recently published books in this area. The distinguishing feature of this book is the manner in which the various concepts are linked to provide a unified view of the subject.

The book is a self-contained, covering the fundamental principles of artificial neural networks. It can be adopted as a text book for a graduate level course. Students with basic engineering or physics or mathematics background can easily follow the topics discussed. No advanced concepts from any field are assumed. It can also be used by scientists and engineers who have an aptitude to explore new ideas in computing.

The book starts **with** tracing the developments in computing in general, and the trends in artificial intelligence, in particular. The prevailing notions of intelligence and intelligent tasks are discussed in the context of handling these tasks by machines. The primary reasons for the performance gaps in the current systems can be traced to the differences in the perceptions of a given input by machine

and by human beings. The introductory chapter discusses the distinction between data and pattern, and between recognition and understanding, to highlight the differences in machine and human perceptions of input to a system. The chapter also deals with several pattern recognition tasks which human beings are able to perform naturally and effortlessly, whereas there are no good algorithms to implement these tasks on a machine. A brief discussion on existing models and methods of solving pattern recognition tasks is given, followed by an analysis of the need for new models of computing to deal with such tasks.

The basics of artificial neural networks are introduced in Chapter 1. The terminology is introduced with reference to a single computing element (or artificial neuron) and some simple connection topologies of the computing elements. Basic learning laws are also discussed in this chapter.

In an artificial neural network the changes of activation values of units and the connection weights (synapses) between units are governed by the equations describing the activation and synaptic dynamics, respectively. Models for activation and synaptic dynamics are introduced in Chapter 2. Stability and convergence issues of these models are discussed, as these will determine the ability of an artificial neural network to accomplish a given pattern recognition task.

Chapter 3 introduces some basic structures of artificial neural networks and the pattern recognition tasks that these structures can perform. The **structures** are organized into feedforward, feedback and competitive layer networks. The corresponding broad pattern recognition tasks are pattern association, pattern storage and pattern clustering, respectively. Chapters 4–6, the kernel of the book, provide a detailed analysis of the three basic structures of artificial neural networks and discuss the different pattern recognition tasks that these structures address. Chapter 4 deals with feedforward networks, where the pattern association, classification and mapping tasks are analyzed. Perceptron learning and its limitations for adjusting the weights of a multilayer feedforward network are covered. The generalized delta rule or the backpropagation learning is presented for training a multilayer feedforward neural network. In Chapter 5 feedback networks and the associated pattern storage and pattern environment storage tasks are analyzed. Here, the **Hopfield** energy analysis of feedback networks is presented in detail, and the need for stochastic neural networks is clearly brought out, besides introducing the Boltzmann machine to accomplish the task of pattern environment storage. Finally, the chapter concludes with a detailed discussion on the Boltzmann learning law for stochastic neural networks. Competitive learning networks are analyzed in Chapter 6 which presents the details on how pattern clustering and feature

mapping are accomplished through the competitive learning networks. The chapter also discusses the principles of self-organization and the self-organization learning for feature map.

Chapter 7 deals with artificial neural network architectures for complex pattern recognition tasks such as associative memory, pattern mapping, stability-plasticity dilemma, temporal patterns and pattern variability. In each case, a brief description of the task and a suitable architecture for the task is given. Applications of artificial neural network models are covered in Chapter 8. Some direct applications considered are: pattern classification, associative memories, optimization, vector quantization and control. Some of the application areas discussed are: speech and image processing and decision making. In each case a simplified version of the problem to suit an existing neural network architecture is considered for illustration. The chapter also analyzes issues in the development of neural network models for practical problems. It concludes with a discussion on several unresolved issues **that** severely limit the application of models based on artificial neural networks.

The book provides examples and illustrations at appropriate places. It also gives algorithms are given for important learning laws to enable the reader to implement them. Finally, review questions and problems are given at the end of each chapter. A solution manual for all the problems is available and can be obtained either from the publisher or at the **website** [http:// speech.iitm.ernet.in/Main/faculty/yegna/Biodata/solutionmanual.tar.gz](http://speech.iitm.ernet.in/Main/faculty/yegna/Biodata/solutionmanual.tar.gz).

B. YEGNANARAYANA

Acknowledgements

I am grateful to the Director of the Indian Institute of Technology Madras for providing an excellent environment for work with ample facilities and academic freedom. I wish to thank several of my faculty colleagues for providing feedback on **my** lectures as well as my notes. In particular, I would like to thank Dr. C. Chandra Sekhar, Dr. S. Das, Dr. Deepak Khemani and Dr. R. Sundar for many useful interactions.

I have been fortunate to have an excellent group of students and colleagues in the Speech and Vision Laboratory of the Department of Computer Science and Engineering. In particular, the following students have contributed significantly in the preparation of many diagrams, tables and examples: **Anitha** Mary Jacob, **Hemant**, Ikbal, Kedar, Jaganadha Reddy, Mathews, **Pavan** Kumar, Poongodi, Raghu, Rajasekhar, Rajendran, Ravindra Babu, Seetha, Shaji, and Vijay.

The following students have really helped me to improve my understanding of the concepts through discussions and constant criticism: Madhukumar, Manish, Murthy and **Raghu**.

I would like to thank the following students who have contributed significantly in the preparation of the manuscript: A. Ravichandran, C. Chandra Sekhar, P.P. Raghu, N. Sudha, A. Neeharika and Manish **Sarkar**.

One individual who has been associated with this effort almost from the beginning is A. Tamilendran, who patiently typed several versions of this book and also helped me in organizing the material for the book. I am indeed grateful for his untiring and dedicated effort.

Finally, I would like to thank my wife Suseela and my daughters Madhuri and Radhika for their patience and cooperation during the nearly five years period of writing this book. I am indeed very fortunate to have understanding and accommodative members in my family, without whose help I would not have been able to write this book.

B. YEGNANARAYANA

Introduction

The **current** search for new models of computing based on artificial neural networks is motivated by our quest to solve natural (intelligent) tasks by exploiting the developments in computer technology [Marcus and Dam, 1991; **IEEE** Computer, Oct. 1996]. The developments in Artificial Intelligence (**AI**) appear promising, but when applied to real world intelligent tasks such as in speech, vision and natural language processing, the AI techniques show their inadequacies and 'brittleness' in the sense that they become highly task specific [Dreyfus, 1992; Rich and Knight, 1991; Holland, 1986; Pearl, 1984]. Like in the algorithmic methods for problem solving, even the **AI** techniques need clear specification and mapping of the given problem **into** a form suitable for the techniques to be applicable. For example, in order to apply heuristic search methods, one needs to map the problem as a search problem. Likewise, to solve a problem using a rule-based approach, it is necessary to explicitly state the rules. It is here scientists are hoping that computing models inspired by biological neural networks may provide new directions to solve problems arising in natural tasks. In particular, it is hoped that neural networks would extract the relevant features from the input data and perform a pattern recognition task by learning **from** examples without explicitly stating the rules for performing the task. The purpose of this book is to discuss the issues in pattern recognition tasks and some of the current approaches used to address these issues based on Artificial Neural Networks (ANN). We discuss the notion of intelligence and intelligent tasks, and then we briefly trace the developments in AI, in particular, and computer technology, in general. We show that computing in intelligent tasks requires a distinction between pattern and data. To give a better appreciation of the nature of intelligent tasks, we elaborate the distinction between pattern and data with illustrations. We also discuss the nature of patterns and various types of pattern recognition tasks which we encounter in our daily life. We present briefly different methods available for dealing with various pattern recognition tasks, and make a case for new models of computing based on ANNs to address these tasks. To appreciate the ANN models, some basics of ANN, including the terminology, are introduced. We provide a detailed discussion on the operation of ANNs through models of activation and synaptic dynamics of ANNs. Some ANNs are identified as basic **functional**

units which form building blocks for more complex networks. The pattern recognition tasks that these functional units can handle are discussed, and a detailed analysis of the **performance** of these units is given. Specific architectures are needed to address complex pattern recognition tasks. Each of these architectures is typically tailored to deal with some critical issues of the pattern recognition tasks. Principles and architectures of ANNs are currently limited to trivial applications, where the problems are modified to suit the architectures. Some of these direct applications are discussed in detail. The more challenging task is to solve real world problems. Limitations of the existing ANNs and issues that need to be addressed to deal with the real world problems are discussed in the final sections of this book. In the end we notice that our current understanding of problems and the existing models of ANNs still fall too short of our needs and expectations.

Trends in Computing

In this section we trace the background for the development of neural network models from the viewpoint of computing. First we shall consider the prevailing notion of intelligence and intelligent tasks. Then we shall trace the developments in computing and computer technology that led to the belief that intelligent tasks can be realized by a machine. In particular, we shall discuss the trends in AI and the gaps in performance of the current AI systems. The primary reason for the performance gaps can be traced to the differences in the perception of an input by machines and by human beings. These differences will be discussed in the following sections.

The current usage of the terms like AI systems, intelligent systems, knowledge-based systems, expert systems, etc., are intended to convey that it is possible to build machines that can demonstrate intelligences similar to human beings in performing some simple tasks. In these tasks we look for the final result of the performance of the machine for comparison with the performance of a human being. We attribute intelligence to the machine if the performance of the machine and human being are the same. But the ways the tasks are performed by the machine and a human being are basically different. The machine performs a task in a step-by-step sequential manner dictated by an algorithm, which may be modified by some known heuristics. Therefore the **algorithm, and** the heuristics have to be derived for a given task. Once derived, they remain fixed. Typically, implementation of a task requires large number of operations (arithmetic and logical) and a large amount of memory. The trends in computing along several dimensions clearly point out the ability of a machine to handle large number of operations.

Table I shows the developments in device technology, software,

Table I Trends in Computing

1. Technology
Mechanical devices
Vacuum tubes
Transistors
Medium scale integration
Large scale integration
Very large scale integration
Optical devices
2. Software
Machine language
Assembly language
High level languages
LISP, PROLOG
4th generation languages
Object-oriented languages
Distributed languages—JAVA
Natural language
3. Architecture
Uniprocessors
Array processors
Special purpose chips
Supercomputers
Parallel computers
VLSI array processors
Parallel distributed processing models
Optical computers
4. AI Concepts
Numerical processing
Symbolic processing
General problem solving
Logic
Heuristic search
Computational linguistics
Natural language processing
Knowledge representation
Expert systems
Hidden markov models
Artificial neural networks

architecture and artificial intelligence concepts [IEEE Computer, Oct. 1996]. In device technology, the trend is to make each device more and more powerful **and/or** to pack more and more devices in a single chip, thus increasing the packing density. The trend in software is to bring the computer more and more closer to the user by providing multimodal communication and natural input and output facilities. In particular, the goal is to achieve communication through the natural language of the user, instead of using artificial computer languages.

Significant developments are taking place in the evolution of

computer architectures. It is realized that single or multiple processors using Von Neumann model have severe limitations of speed. Parallel computers and optical computers are also limited in scope due to their mismatch with the problems [Partridge, 1997]. The trend is to explore architectures based on Parallel and Distributed Processing (PDP) models motivated by our understanding of the structure and function of the biological neural network [Rumelhart and McClelland, 1986; McClelland and Rumelhart, 1986]. The driving force for these developments is the desire to make machines more and more *intelligent*. Finally, it is in the applications and in the simulation of applications, the real test of the technology developments can be seen.

The notion of intelligence and intelligent systems is changing continuously as can be seen from the evolution of AI concepts (Table I). Originally computing was meant only for numerical calculations. When it was realized that symbols like text also can be manipulated using step-by-step algorithmic approach, it was felt that logical inference could be implemented on a computer. Through logic it was possible to incorporate heuristics in reducing the search in the solution of an AI problem. Therefore it was felt that all AI problems including problems of speech and image understanding could be solved by using clever search methods [Reddy, 1988; Reddy, 1996]. But it was soon realized that mapping a problem onto a problem of heuristic search was possible only for tasks where representation was obvious as in some games and puzzles. To overcome the representational problems, rule-based approaches were suggested, where the rules for the solution of a problem could be explicitly obtained from a human expert. The set of rules constituting the knowledge, together with an inferencing mechanism for the application of the rules, resulted in proliferation of expert systems or knowledge-based systems for various tasks [Feigenbaum et al, 1988]. It did not take much time to realize that explicit statement of rules by a human expert does not constitute all the knowledge he uses for a given problem [Holland, 1986; Dreyfus, 1989]. Moreover, the issues of common sense knowledge and learning, which are so natural to any human being, could not easily be captured in a machine [Dreyfus, 1972; Dreyfus, 1992].

Pattern and Data

Speech, vision and natural language processing problems dominated the attention of designers of intelligent systems [Reddy, 1996]. The most difficult issues in these cases are to derive the description of the pattern in data in terms of symbols and to derive a set of rules representing the knowledge of the problem domain. Even if we can implement highly complex and compute intensive algorithms with the

current technology, it is now realized that we cannot derive the pattern description and knowledge completely for a given problem. Thus the mere ability of a machine to perform large amount of symbolic processing and logical inferencing (as is being done in AI) does not result in an intelligent behaviour.

The main difference between human and machine intelligence comes from the fact that humans perceive everything as a pattern, whereas for a machine everything is data [Greenberger, 1962]. Even in routine data consisting of integer numbers (like telephone numbers, bank account numbers, car numbers), humans tend to perceive a pattern. Recalling the data is also normally **from** a stored pattern. If there is no pattern, then it is very difficult for a human being to remember and reproduce the data later. Thus storage and recall operations in human beings and machines are performed by different mechanisms. The pattern nature in storage and recall automatically gives robustness and fault tolerance for the human system. Moreover, typically far fewer patterns than the estimated capacity of human memory system are stored. Functionally also human beings and machines differ in the sense that human beings understand patterns, whereas machines can be said to *recognise* patterns in data. In other words, human beings can get the whole object in the data even though there is no clear identification of subpatterns in the data. For example, consider the name of a person written in a handwritten cursive script. Even though the individual patterns for each letter may not be evident, the name is understood due to the visual hints provided in the written script. Likewise, speech is understood even though the patterns corresponding to the individual sounds may be distorted, sometimes to unrecognizable extents [Cooper, 1980]. Another major characteristic of a human being is the ability to continuously learn from examples, which is not understood well enough to implement it in an algorithmic fashion in a machine. Human beings are capable of making mental patterns in their biological neural network **from** an input data given in the form of numbers, text, picture, sounds, etc., using their sensory mechanisms of vision, sound, touch, smell and taste. These mental patterns are formed even when the data is noisy or deformed due to variations such as translation, rotation and scaling. The patterns are also formed from a temporal sequence of data as in the case of speech and pictures. Human beings have the ability to recall the stored patterns even when the input information is noisy or partial (incomplete) or mixed with information pertaining to other patterns. Although patterns and data are different, these terms are **used** interchangeably in the literature. While we also use these terms interchangeably throughout the book, the distinction between these must be kept in mind to appreciate the limitations of a machine over human beings for performing pattern recognition tasks.

Pattern Recognition Tasks

The inherent differences in information handling by human beings and machines in the form of patterns and data, respectively, and in their functions in the form of understanding and recognition, respectively, have led us to identify and discuss several pattern recognition tasks which human beings are able to **perform** very naturally and effortlessly, whereas no simple algorithms exist to implement these tasks on a machine. The identification of these tasks below was somewhat influenced by the organization of the artificial neural network models described in this book [Yegnanarayana, 1994; Duda and Hart, 1973].

Pattern Association

Pattern association problem involves storing a set of patterns or a set of input-output pattern pairs in such a way that when a test pattern is presented, the stored pattern or pattern pair corresponding to the test pattern is recalled. This is purely a memory function to be performed for patterns or pattern pairs. Typically, it is desirable to recall the correct pattern even though the test pattern is noisy or incomplete. The problem of storage and recall of patterns is called the autoassociation task. Since this is a content addressable memory function, the system should display *accretive* behaviour, i.e., should recall the stored pattern closest to the given input. The problem of storage and recall of pattern pairs is called a heteroassociation task. It is also necessary to store as many patterns or pattern pairs as possible in a given system. Printed characters or any set of fixed symbols could be considered as examples of patterns for these tasks. Note that in this case the test patterns are the same as the training patterns, but with some noise added or some portions missing. In other words, the test patterns are generated from the same source in an identical manner as the training patterns.

Pattern Classification

In pattern classification, given a set of patterns and the corresponding class label, the objective is to capture the implicit relation among the **patterns** of the same class, so that when a test pattern is given, the corresponding output class label is retrieved. Note that the individual patterns of each class are not memorized or stored. Typically, in this case the test patterns belonging to a class are not the same as the patterns used in the training, although they may originate from the same source. Speech spectra of steady vowels generated by a person, or hand-printed characters, could be considered as examples of patterns for pattern classification problems. Pattern classification problems display *accretive* behaviour. Pattern classification problems are said to belong to the category of *supervised* learning.

Pattern Mapping

In pattern mapping, given a set of input patterns and the corresponding output patterns, the objective is to capture the implicit relationship between the input and output patterns, so that when a test input pattern is given, the pattern corresponding to the output of the generating system is retrieved. Note that the system should perform some kind of *generalization* as opposed to *memorizing* the information. Typically, in this case the test patterns are not the same as the training patterns, although they may originate **from** the same source. Speech spectra of vowels in continuous speech could be considered as examples of patterns for pattern mapping **problems**. Pattern mapping generally displays *interpolative* behaviour.

Pattern Grouping

In this case, given a set of patterns, the problem is to identify the subset of patterns possessing similar distinctive features, and group them together. Since the number of groups and the features of each group are not explicitly stated, this problem belongs to the category of *unsupervised* learning. Note that, this is possible only when the features are unambiguous as in the case of hand-printed characters or steady vowels. In the pattern classification problem the **patterns** for each group are given separately. In pattern **grouping**, on the other hand, patterns belonging to several groups are given, and the system has to resolve them into different groups. Pattern grouping is also called *pattern clustering* task.

Examples of the patterns for this task could be printed **characters** or hand-printed characters. In the former case the grouping can be performed based on the data itself. Moreover, in that case, the test data is also generated from an identical source as for the training data. For hand-printed characters or steady vowel patterns, features of the patterns in the data are used for grouping. In this case the test data is generated from a similar source as for the training data, such that only the pattern features are preserved. The actual training data values are not necessarily reproduced in the test **patterns**.

Feature Mapping

In several patterns the features are not unambiguous. In fact the features vary over a continuum, and hence it is difficult to form groups of patterns having some distinct features. In such cases, it is desirable to display directly the feature variations in the patterns. This again belongs to the unsupervised learning category. In this case what is learnt is the *feature map* of a pattern and not the group or the class to which the pattern belongs. This occurs, for example, in the speech spectra for vowels in continuous speech. Due to changes in

the vocal tract shape for the same vowel occurring in different contexts, the features (formants or resonances of the vocal tract in this case) vary over overlapping regions for different vowels.

Pattern Variability

There are many situations when the features in a pattern undergo unspecified distortions each time the pattern is generated by the system. This can be easily seen in the characters in normal handwritten cursive script. Human beings are able to recognise them due to some implicit interrelations among the features, which are not known precisely. Classification of such patterns falls into the category of pattern variability task.

Temporal Patterns

All the tasks discussed so far refer to the features present in a given static pattern. Human beings are able to capture effortlessly the dynamic features present in a sequence of patterns. This is true, for example, in speech where the changes in the resonance characteristics of the vocal tract system (e.g., formant contours) capture the significant information about the speech message. This is also true in any dynamic scene situation as in a movie on a television. All such situations require handling multiple static patterns simultaneously, looking for changes in the features in the subpatterns in adjacent pattern pairs.

Stability-plasticity Dilemma

In any pattern recognition task the input patterns keep changing. Therefore it is **difficult** to freeze the categorization task based on a set of patterns used in the training set. If it is frozen, then the **system** cannot learn the category that a new pattern may suggest. In other words, the system lacks its plasticity. On the other hand, if the system is allowed to change its categorization continuously, based on new input patterns, it cannot be used for any application such as pattern classification or grouping, as it is not stable: This is called stability-plasticity dilemma in pattern recognition.

Methods for Pattern Recognition Tasks

Methods for solving pattern recognition tasks generally assume a sequential model for the pattern recognition process, consisting of pattern environment, sensors to collect data from the environment, feature extraction from the data and association/storage/classification/grouping using the features [Kanal, 1974; Mantas, 1987].

The simplest solution to a pattern recognition problem is to use a template matching, where the data of the test pattern is matched point by point with the corresponding data in the reference pattern. Obviously, this can work only for very simple and highly restricted pattern recognition tasks. At the next level of complexity, one can assume a deterministic model for the pattern generation process, and derive the parameters of the model from a given pattern in order to represent the information in the **pattern**. Matching the test and reference patterns is done at the **parametric** level. This works well when the model of the generation process is known with reasonable accuracy. One could also assume a stochastic model for the pattern generation process, and derive the parameters of the model from a large set of training patterns. Matching the test and reference patterns can be performed by several statistical methods such as likelihood ratio, variance weighted distance, Bayesian classification, etc. Other approaches for pattern recognition tasks depend on extracting features from parameters or data. These features may be specific for the task. A pattern is described in terms of features, and pattern matching is done using descriptions in terms of the features. Another method based on descriptions is called syntactic pattern recognition in which a pattern is expressed in terms of primitives suitable for the classes of patterns under study. Pattern matching is performed by matching the descriptions of the patterns in terms of the primitives. More recently, methods based on the knowledge of the sources generating the patterns are being explored for pattern recognition tasks. These knowledge-based systems express the knowledge in the form of rules for generating and perceiving patterns.

The main difficulty in each of the pattern recognition techniques alluded to above is that of choosing an appropriate model for the pattern generating process and estimating the parameters of the model in the case of a model-based approach, *or* extraction of features from the **data/parameters** in the case of feature-based methods, *or* selecting appropriate primitives in the case of syntactic pattern recognition, *or* deriving rules in the case of a knowledge-based approach. It is all the more difficult when the test patterns are noisy or distorted versions of the patterns used in the training process. The ultimate goal is to impart to a machine the pattern recognition capabilities comparable to those of human beings. This goal is difficult to achieve using many of the conventional methods, because, as mentioned earlier, these methods assume a sequential model for the pattern recognition process [Devijver and Kittler, 1982; Schalkoff, 1992; Bezdek, 1996]. On the other hand, the human pattern recognition process is an integrated process involving the use of biological neural processing even from the stage of sensing the environment. Thus the neural processing takes place directly on the data for feature extraction and pattern matching. Moreover, the large

size (in **terms** of number of neurons and interconnections) of the biological neural network and the inherently different mechanism of processing may be contributing to our abilities of pattern recognition in spite of variability and noise, and also to our abilities to deal with the temporal patterns **as** well as with the so called stability-plasticity dilemma. It is for these reasons attempts are being made to explore new models of computing. Such models for computing are based on artificial neural networks, the basics of which are introduced in the next chapter.

Organization of the Topics

It is possible to view the topics of interest in artificial neural networks at various levels as follows (Table II):

Table II Organization of Topics in Artificial Neural Networks at Different Levels

(i) Problem level

Issues:

- Understanding human problem solving **as** a pattern recognition process
- Understanding biological neural networks

Topics discussed: (Introduction)

- Distinction between pattern and data
- Distinction between information processing by human beings and by machines
- Pattern recognition tasks
- Methods and models for pattern recognition tasks

(ii) Basics level

Issues:

- Models of neurons
- Models of interconnections
- Models of activation dynamics
- Models of synaptic dynamics
- Global pattern formation by models
- Stability and convergence

Topics discussed: (Chapters 1 and 2)

- Basic principles of biological neural networks
- Three basic models of artificial neurons: MP neuron, **Perceptron** and **Adaline**
- Topology: Basic structures of ANN
- Basic learning laws
- Activation dynamics models: Additive and shunting
- Synaptic dynamics models: Requirements for learning and categories of models of learning process
- Stability theorems
- Neural network recall

Table II (Cont.)

(iii) Functional level*Issues:*

- Identification of basic functional units which can be **analysed**
- Analysis of pattern recognition **tasks** by the functional units
- Functional units as basic building **blocks** for complex architectures of ANN

Topics discussed: (Chapters 3 to 6)

- Three **types** of functional units namely, FF, FB and CL networks
- Analysis of FF networks: Pattern association, perceptron, multilayer perceptron, gradient descent methods, backpropagation algorithm
- Analysis of FB networks: Pattern storage, **Hopfield** network, **Boltzmann** machine, simulated annealing
- Analysis of CL networks: Pattern clustering, self-organization, feature mapping

(iv) Architectural level*Issues:*

- Identification of pattern recognition **tasks** and issues
- Development of architectures for complex pattern recognition **tasks**
- Architectures specific to problems

Topics discussed: (Chapter 7)

- Associative memory: **BAM**
- Pattern mapping: RBF and CPN
- Stability-plasticity dilemma: Adaptive **Resonance** Theory (ART)
- **Temporal** patterns: Avalanche, Time Delay NN (TDNN)
- Pattern variability: Neocognitron

(v) Applications level*Issues:*

- Potential for direct application
- Mapping of the given application onto a neural network model

Topics discussed: (Chapter 8)

- Direct applications: Associative memory, data compression, optimization, vector quantization and control
 - Application areas: Problems in speech, image and decision making
-

(i) Problem level: This involves mapping the real world problems as pattern processors. This may require good understanding of human information processing both from the psychological and biological angles. This topic is not within the scope of this book, although in the introductory chapter we have discussed briefly the issues in problem solving by humans and by machines.

(ii) Basics level: Advances in technology and understanding of human information processing system enable us to evolve better models of neurons as processing units, their interconnections and dynamics (**activation** and synaptic), learning laws and recall procedures. These models in turn will enable us to build sophisticated

artificial neural networks for solving complex pattern recognition tasks. Chapters 1 and 2 deal with some issues at the basics level. In particular, in Chapter 1, we present basic models of artificial neurons and some basic structures obtained by interconnecting these neuron models. The chapter also includes some basic learning laws commonly used in artificial neural networks. In Chapter 2, models of activation and synaptic dynamics are described. The chapter also deals with issues of stability and convergence, pertaining to activation and synaptic dynamics models, respectively.

(iii) Functional level: It is necessary to understand the pattern recognition tasks that can be performed by some of the basic structures of artificial neural networks. These basic structures then will form building blocks for development of new architectures for complex pattern recognition tasks. We have identified three categories of functional units, namely, feedforward, feedbackward and competitive learning networks. Chapters 3 to 6 deal with detailed analysis of the pattern recognition tasks that these functional units can perform. Chapter 3 deals with a description of the functional units and the corresponding pattern recognition tasks. Chapter 4 gives a detailed analysis of feedforward networks, illustrating at each stage the limitations of the networks to solve a given pattern recognition task. The chapter concludes with a discussion on the capabilities and limitations of multilayer feedforward neural networks and the associated backpropagation learning law. Chapter 5 is devoted to analysis of feedback networks. The significance of the nonlinear output function of a processing unit in feedback networks for pattern storage task is discussed. **Hopfield** energy analysis of a feedback network is used to demonstrate the capability and also limitations of such networks. Stochastic network models are introduced to overcome some of the limitations of the **Hopfield** model due to local minima problems. Finally, the Boltzmann machine is presented to address the issue of pattern environment storage. The chapter concludes with a discussion on Boltzmann learning law and its implementation using simulated annealing. Chapter 6 deals with a detailed analysis of competitive learning networks. In particular, simple networks for pattern clustering are considered. Self-organizing neural networks are presented as an extension of the idea of competitive learning. The feature mapping capability of the self-organizing networks is illustrated with examples.

(iv) Architectural level: For complex pattern recognition tasks new architectures need to be evolved from the known principles, components and structures at the basics and functional levels. Chapter 7 discusses development of architectures which address some complex issues in pattern recognition tasks. We present an extended

discussion on some specific architectures for associative memory and pattern mapping tasks. In addition, we discuss Counter Propagation Networks (**CPN**) for data compression, Adaptive Resonance Theory (ART) architecture for stability-plasticity dilemma, **neocognitron** for pattern variability, and avalanche architecture and time delay neural networks for temporal **patterns**. These architectures **are** meant for specific tasks and hence are severely limited in their use. However, understanding the development process of these architectures helps us to evolve new architectures tailored to specific issues.

(v) Applications level: **Currently** most of the neural network models are severely limited in their abilities to solve real world problems. At the application level, one can consider two different categories. In one case it may be possible to map the given application onto a neural network model or architecture. We call such situations **as** direct applications. Simple associative memories, data compression, optimization, vector quantization and pattern mapping fall into the category of direct application. But in case of problems such as in speech recognition, image processing, natural language processing and decision making, it is not normally possible to see a direct mapping of the given problem onto a neural network model. These are natural tasks which human beings are good at, but we still do not understand how we do them. Hence it is a challenging task to **find** suitable neural network models to address these problems [Barnden, 1995; Cowan and Sharp, 1981].

Review Questions

1. Give examples for which heuristic search methods of artificial intelligence **are** applicable.
2. Discuss the developments in artificial intelligence that led to the interest in exploring new models for computing.
3. What is a rule-based expert system? Why do we say such systems are 'brittle'? Discuss your answer with an illustration.
4. What are the differences in the manner of solving problems by human beings and by machines? Illustrate with examples.
5. Explain the distinction between pattern and data.
6. What are the features of pattern processing by human beings?
7. Explain, with examples, differences between the following pattern recognition tasks:
 - (a) Association vs classification
 - (b) Classification vs mapping
 - (c) Classification vs clustering

8. Explain the following pattern recognition issues with illustrations:
 - (a) Pattern variability
 - (b) Temporal patterns
 - (c) Stability-plasticity dilemma
9. What are different methods for solving pattern recognition tasks?
10. What is the difficulty with the existing methods for solving natural pattern recognition problems?
11. **Identify** some difficult pattern recognition problems in the following areas:
 - (a) Speech
 - (b) Vision
 - (c) Natural language processing
12. What are the issues at the architectural level of artificial neural networks?
13. What are the situations for direct applications of artificial neural networks?
14. What is the difficulty in solving a real world problem like speech recognition even by an artificial neural network model?

Chapter 1

Basics of Artificial Neural Networks

New models of computing to perform pattern recognition tasks **are** inspired by the structure and performance of our biological neural network. But these models **are** not expected to reach anywhere near the performance of the biological network for several reasons. Firstly, we do not fully understand the operation of a biological neuron and the neural interconnections. Moreover, it is nearly impossible to simulate: (i) the number of neurons and their interconnections as it exists in a biological network, and (ii) their operations in the natural asynchronous mode.

However, a network consisting of basic computing units can display some of the features of the biological network. In this chapter, the features of neural networks that motivate the study of neural computing are discussed. A simplified description of the biological neural network is given in Section 1.1. The differences in processing by the brain and a computer are then presented. In Section 1.2 a brief history of neurocomputing is presented, indicating some of the significant developments that have led to the current interest in the field. In Section 1.3 the terminology of artificial neural networks is introduced by considering the structure and operation of a basic computing unit, *i.e.*, the artificial neuron. Three classical models of artificial neurons are described in Section 1.4. It is necessary to arrange the units in a suitable manner to handle pattern recognition tasks. In Section 1.5 we discuss a few basic structures which form the building blocks for more complex architectures. The basic training or learning laws for determining the connection weights of a network to represent a given problem are then discussed in Section 1.6. The concluding section gives a summary of the issues discussed in this chapter.

1.1 Characteristics of Neural Networks

1.1.1 Features of Biological Neural Networks

Some attractive features of the biological neural network that make

it superior to even the most sophisticated **AI** computer system for pattern recognition tasks are the following:

(a) **Robustness and fault tolerance:** The decay of nerve cells does not seem to **affect** the performance significantly.

(b) **Flexibility:** The network automatically adjusts to a new environment without using any **preprogrammed** instructions.

(c) **Ability to deal with a variety of data situations:** The network can deal with information that is fuzzy, probabilistic, noisy and inconsistent.

(d) **Collective computation:** The network performs routinely many operations in parallel and also a given task in a distributed manner.

1.1.2 Biological Neural Networks

The features of the biological neural network are attributed to **its** structure and function. The description of the biological neural network in this section is adapted from [Muller and Reinhardt, 1991]. The fundamental unit of the network is called a neuron or a nerve cell. **Figure 1.1** shows a schematic of the structure of a neuron. It

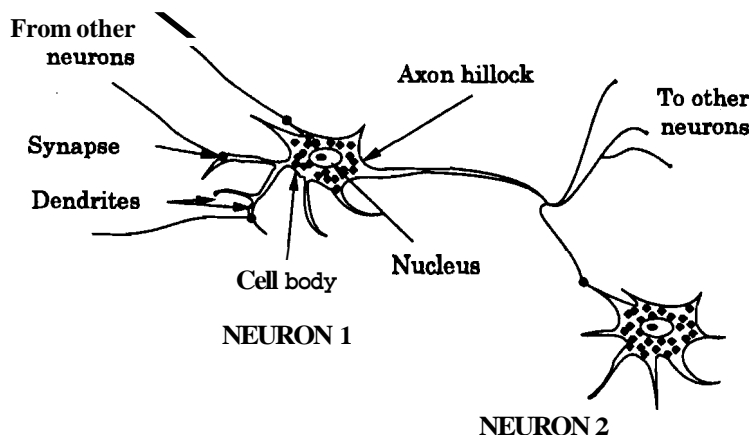


Figure 1.1 Schematic diagram of a typical neuron or nerve cell.

consists of a cell body or soma where the cell nucleus is located. Tree-like nerve fibres called dendrites are associated with the cell body. These dendrites receive signals from other neurons. Extending from the cell body is a single long fibre called the axon, which eventually branches into strands and substrands connecting to many other neurons at the synaptic junctions, or synapses. The receiving ends of these junctions on other cells can be found both on the dendrites and on the cell bodies themselves. The axon of a typical neuron leads to a few thousand synapses associated with other neurons.

The transmission of a signal from one cell to another at a synapse is a complex chemical process in which specific transmitter substances are released from the sending side of the junction. The effect is to raise or lower the electrical potential inside the body of the receiving cell. If this potential reaches a threshold, an electrical activity in the form of short pulses is generated. When this happens, the cell is said to have fired. These electrical signals of fixed strength and duration are sent down the axon. Generally the electrical activity is confined to the interior of a neuron, whereas the chemical mechanism operates at the synapses.

The dendrites serve as receptors for signals from other neurons, whereas the purpose of an axon is transmission of the generated neural activity to other nerve cells (inter-neuron) or to muscle fibres (motor neuron). A third type of neuron, which receives information from muscles or sensory organs, such as the eye or ear, is called a receptor neuron.

The size of the cell body of a typical neuron is approximately in the range **10–80 micrometers (μm)** and the dendrites and **axons** have diameters of the order of a few μm . The gap at the synaptic junction is about **200 nanometers (nm)** wide. The total length of a neuron varies from **0.01 mm** for internal neurons in the human brain up to **1 m** for neurons in the limbs.

In the state of inactivity the interior of the neuron, the protoplasm, is negatively charged against the surrounding neural liquid containing positive Sodium (**Na^+**) ions.. The resulting resting potential of about **-70 mV** is supported by the action of the cell membrane, which is impenetrable for the positive Sodium ions. This causes a deficiency of positive ions in the protoplasm. Signals arriving from the synaptic connections may result in a temporary depolarization of the resting potential. When the potential is increased to a level above **-60 mV** , the membrane suddenly loses its impermeability against Na^+ ions, which enter into the protoplasm and reduce the potential difference. This sudden change in the membrane potential causes the neuron to discharge. Then the neuron is said to have fired. The membrane then gradually recovers its original properties and regenerates the resting potential over a period of several milliseconds. During this recovery period, the neuron remains incapable of further excitation. The discharge, which initially occurs in the cell body, propagates as a signal along the axon to the synapses. The intensity of the signal is encoded in the frequency of the sequence of pulses of activity, which can range **from** about 1 to **100** per second.

The speed of propagation of the discharge signal in the cells of the human brain is about **$0.5\text{--}2 \text{ m/s}$** . The discharge signal travelling along the axon stops at the synapses, because there exists no conducting link to the next neuron. Transmission of the signal across the

synaptic gap is mostly effected by chemical activity. When the signal arrives at the presynaptic nerve terminal, special substances called neurotransmitters are produced in tiny amounts. The neurotransmitter molecules travel across the synaptic junction reaching the postsynaptic neuron within about 0.5 ms. These substances modify the conductance of the postsynaptic membrane for certain ions, causing a polarization or depolarization of the postsynaptic potential. If the induced polarization potential is positive, the synapse is termed excitatory, because the influence of the synapse tends to activate the postsynaptic neuron. If the polarization potential is negative, the synapse is called inhibitory, since it counteracts excitation of the neuron. All the synaptic endings of an axon are either of an excitatory or an inhibitory nature.

The cell body of a neuron acts as a kind of summing device due to the net depolarizing effect of its input signals. This net effect decays with a time constant of 5–10 ms. But if several signals arrive within such a period, their excitatory effects accumulate. When the total magnitude of the depolarization potential in the cell body exceeds the critical threshold (about 10 mV), the neuron fires.

The activity of a given synapse depends on the rate of the arriving signals. An active synapse, which repeatedly triggers the activation of its postsynaptic neuron, will grow in strength, while others will gradually weaken. Thus the strength of a synaptic connection gets modified continuously. This mechanism of synaptic plasticity in the structure of neural connectivity, known as Hebb's rule, appears to play a dominant role in the complex process of learning.

Although all neurons operate on the same basic principles as described above, there exist several different types of neurons, distinguished by the size and degree of branching of their dendritic trees, the length of their axons, and other structural details. The complexity of the human central nervous system is due to the vast number of the neurons and their mutual connections. Connectivity is characterised by the complementary properties of convergence and divergence. In the human cortex every neuron is estimated to receive a converging input on an average from about 10^4 synapses. On the other hand, each cell feeds its output into many hundreds of other neurons. The total number of neurons in the human cortex is estimated to be in the vicinity of 10^{11} , which are distributed in layers over a full depth of the cortical tissue at a constant density of about 15×10^4 neurons per mm^2 . Combined with the average number of synapses per neuron, this yields a total of about 10^{15} synaptic connections in the human brain, the majority of which develop during the first few months after birth. The study of the properties of complex systems built of simple, identical units may lead to an understanding of the mode of operation of the brain in its various functions, although we are still very far from such an understanding.

1.1.3 Performance Comparison of Computer and Biological Neural Networks

A set of processing **units** when assembled in a closely interconnected network, offers a surprisingly rich structure exhibiting some features of the biological neural network. Such a structure is called an artificial neural network (ANN). Since **ANNs** are implemented on computers, it is worth comparing the processing capabilities of a computer with those of the brain [Simpson, 1990].

Speed: Neural **networks are** slow in processing information. For the most advanced computers the cycle time corresponding to execution of one step of a program in the central processing unit is in the range of few nanoseconds. The cycle time corresponding to a neural event prompted by an external stimulus occurs in milliseconds range. Thus the computer processes information nearly a million times faster.

Processing: Neural networks can perform massively parallel operations. Most programs have large number of instructions, and they operate in a sequential mode one instruction **after** another on a conventional computer. On the other hand, the brain operates with massively parallel operations, each of them having comparatively fewer steps. This explains the superior performance of human information processing for certain tasks, despite being several orders of magnitude slower compared to computer processing of information.

Size and complexity: Neural networks have large number of computing elements, and the computing is not restricted to within neurons. The number of neurons in a brain is estimated to be about 10^{11} and the total number of interconnections to be around 10^{15} . It is this size and complexity of connections that may be giving the brain the power of performing complex pattern recognition tasks which we are unable to realize on a computer. The complexity of brain is further compounded by the fact that computing takes place not only inside the cell body, or soma, but also outside in the dendrites and synapses.

Storage: Neural networks store information in the strengths of the interconnections. In a computer, information is stored in the memory which is addressed by its location. Any new information in the same location destroys the old information. In contrast, in a neural network new information is added by adjusting the interconnection strengths, without destroying the old information. Thus information in the brain is adaptable, whereas in the computer it is strictly replaceable.

Fault tolerance: Neural networks exhibit fault tolerance since the information is distributed in the connections throughout the network.

Basics of Artificial Neural Networks

Even if a few connections are snapped or a few neurons are not functioning, the information is still preserved due to the distributed nature of the encoded information. In contrast, computers are inherently not fault tolerant, in the sense that information corrupted in the memory cannot be retrieved.

Control mechanism: There is no central control for processing information in the brain. In a computer there is a control unit which monitors all the activities of computing. In a neural network each neuron **acts** based on the information locally available, and transmits its output to the neurons connected to it. Thus there is no specific control mechanism external to the computing task.

While the superiority of human information processing system over the conventional computer for pattern recognition tasks is evident from the basic structure and operation of the biological neural network, it is possible to realize some of its features using an artificial network consisting of basic computing units. It is possible to show that such a network exhibits parallel and distributed processing capability. In addition, information can be stored in a distributed manner in the connection weights so as to achieve some fault tolerance. These features are illustrated through several parallel and distributed processing models for cognitive tasks in [Rumelhart and McClelland, 1986; McClelland and Rumelhart, 1986; McClelland and Rumelhart, 1988]. Two of these models are described **briefly** in Appendix A.

The motivation to explore new computing models based on ANNs is to solve pattern recognition tasks that may sometimes involve complex optical and acoustical patterns also. It is impossible to derive logical rules for such problems for applying the well known **AI** methods. It is also difficult to divide a pattern recognition task into subtasks, so that each of them could be handled on a separate processor. Thus the inadequacies of the logic-based artificial intelligence and the limitations of the sequential computing have led to the concept of parallel and distributed processing through ANN. It may be possible to realize a large number of simple computing units on a single chip or on a few chips, and assemble them into a neural computer with the present day technology. However, it is difficult to implement the large number of synaptic connections, and it is even more difficult to determine the strategies for synaptic strength adjustment (learning).

Even with these limitations, ANNs can be developed for several pattern recognition tasks for which it is difficult to derive the logical rules explicitly. The network connection weights can be adjusted to learn from example patterns. The architecture of the network can be evolved to deal with the problem of generalization in pattern classification tasks. ANNs can also be designed to implement selective attention feature required for some pattern recognition tasks. While

the adjustment of weights may take a long time, the execution of pattern classification or pattern recall will be much faster, provided the computing units work in parallel as in a dedicated hardware.

Since information is stored in the connections and it is distributed throughout, the network can function as a memory. This memory is content addressable, in the sense that the information may be recalled by providing partial or even erroneous input pattern. The information is stored by association with other stored data like in the brain. Thus ANNs can perform the task of associative memory. This memory can work even in the presence of certain level of internal noise, or with a certain degree of forgetfulness. Thus the short-term memory function of the brain can be realized to some extent. Since information is stored throughout in an associative manner, ANNs are somewhat fault tolerant in the sense that the information is not lost even if some connections are snapped or some units are not **functioning**. Because of the inherent redundancy in information storage, the networks can also recover the complete information from partial or noisy input pattern. Another way of looking at it is that an ANN is a reliable system built **from** intrinsically unreliable units. Any degradation in performance is 'graceful' rather than abrupt as in the conventional computers. A remarkable feature of **ANNs** is that it can deal with data that are not only noisy, but also fuzzy, inconsistent and probabilistic, just as human beings do. All this is due to the associative and **distributed** nature of the stored information and the redundancy in the information storage due to large size of the network. Typically, the stored information is much less than the capacity of the network.

1.2 Historical Development of Neural Network Principles

The key developments in neural network principles are outlined in this section. Table 1.1 gives a list of some significant contributions in this field that have put the field on a strong theoretical and conceptual foundation, as it exists today.

In **1943** Warren McCulloch and Walter Pitts proposed a model of computing element, called **McCulloch-Pitts** neuron, which performs a weighted sum of the inputs to the element followed by a threshold logic operation [McCulloch and Pitts, **1943**]. Combinations of these computing elements were used to realize several logical computations. The main drawback of this model of computation is that the weights are fixed and hence the model could not learn **from** examples.

In **1949** Donald Hebb proposed a learning scheme for adjusting a connection weight based on pre- and post-synaptic values of the variables [Hebb, **1949**]. Hebb's law became a fundamental learning rule in neural networks literature.

In **1954** a learning machine was developed by Marvin Minsky, in which the connection strengths could be adapted automatically

Table 1.1 Historical Development of Neural Network Principles

Key developments	Other significant contributions
McCulloch and Pitts (1943) <ul style="list-style-type: none"> • Model of neuron • Logic operations • Lack of learning 	von Neumann (1946)— General purpose electronic computer Norbert Weiner (1948)— Cybernetics Shannon (1948)— Information theory Ashby (1952)— Design for a Brain Gabor (1954)—Nonlinear adaptive filter Uttley (1956)—Theoretical machine Caianiello (1961)— Statistical theory and learning
Hebb (1949) <ul style="list-style-type: none"> • Synaptic modifications • Hebb's learning law 	Minsky (1961)—Artificial intelligence Steinbuch (1961)— Learnmatrix Minsky and Selfridge (1961)— Credit assignment problem
Minsky (1954) <ul style="list-style-type: none"> • Learning machines 	Nilsson (1965)— Learning machine Amari (1967)—Mathematical solution to credit assignment Kohonen (1971)—Associative memories Willshaw (1971)— Self-organization and generalization
Rosenblatt (1958) <ul style="list-style-type: none"> • Perceptmn learning and convergence • Pattern classification • Linear separability constraint 	Malsburg (1973)— Self-organization Tikhonov (1973)—Regularization theory Little (1974)—Ising model and neural network Grossberg (1976)— Adaptive resonance theory
Widrow and Hoff (1960) <ul style="list-style-type: none"> • Adaline—LMS learning • Adaptive signal processing 	Anderson (1977)—Brain state-in-box model Little and Shaw (1978)— Stochastic law for NN, spin glasses Fukushima (1980)— Neocognitron Kohonen (1982)—Feature mapping Barto, Sutton and Anderson (1983)—Reinforcement learning
Minsky and Papert (1969) <ul style="list-style-type: none"> • Perceptron — Multilayer perceptron (MLP) • Hard problems • No learning for MLP 	Kirkpatrick (1983)— Simulated annealing Peretto (1984)— Stochastic units Mead (1985)—Analog VLSI Amit (1985)— Statistical machines and stochastic networks
Werbos (1974) <ul style="list-style-type: none"> • Error backpropagation 	Klopf (1986)—Drive-reinforcement learning Hecht-Nielsen (1987)— Counterpropagation Linsker (1988)— Self-organization based on information preservation
Hopfield (1982) <ul style="list-style-type: none"> • Energy analysis 	Kosko (1988)— BAM , Fuzzy logic in ANN Broomhead (1988)—Radial basis functions (RBF) Poggio and Girosi (1990)— RBF and regularization theory
Ackley, Hinton and Sejnowski (1985) <ul style="list-style-type: none"> • Boltzmann machine 	
Rumelhart, Hinton and Williams (1986) <ul style="list-style-type: none"> • Generalised delta rule 	

[Minsky, 1954]. But it was in 1958 that Rosenblatt proposed the perceptron model, which has weights adjustable by the perceptron learning law [Rosenblatt, 1958]. The learning law was shown to converge for pattern classification problems, which are linearly separable in the feature space. While a single layer of perceptrons could handle only linearly separable classes, it was shown that a multilayer perceptron could be used to perform any pattern classification task. But there was no systematic learning algorithm to adjust the weights to realize the classification task. In 1969 Minsky and **Papert** demonstrated the limitations of the perceptron model through several illustrative examples [Minsky and **Papert**, 1969]. Lack of suitable learning law for a multilayer perceptron network had put brakes on the development of neural network models for pattern recognition tasks for nearly 15 years till 1984.

In 1960s **Widrow** and his group proposed an Adaline model for a computing element and an LMS learning algorithm to adjust the weights of an Adaline model [**Widrow** and Hoff, 1960]. The convergence of the LMS algorithm was proved. The algorithm was successfully used for adaptive signal processing situations.

The resurgence of interest in artificial neural networks is due to two key developments in early 1980s. The first one is the energy analysis of feedback neural networks by John Hopfield, published in 1982 and 1984 [Hopfield, 1982; Hopfield, 1984]. The analysis has shown the existence of stable equilibrium states in a feedback network, provided that the network has symmetric weights, and that the state update is made asynchronously. Also, in 1986, **Rumelhart** et al have shown that it is possible to adjust the weights of a multilayer feedforward neural network in a systematic way to learn the implicit mapping in a set of input-output pattern pairs [**Rumelhart** et al, 1986a]. The learning law is called generalized delta rule or error backpropagation learning law.

About the same time Ackley, **Hinton** and Sejnowski proposed the Boltzmann machine which is a feedback neural network with stochastic neuron units [Ackley et al, 1985]. A stochastic neuron has an output function **which** is implemented using a probabilistic update rule instead of a deterministic update rule as in the **Hopfield** model. Moreover, the Boltzmann machine has several additional neuron units, called hidden units, which are used to make a given pattern storage problem representable in a feedback network.

Besides these key developments, there are many other significant contributions made in this field during the past thirty years. Notable among them are the concepts of competitive learning, self-organization and simulated annealing. Self-organization led to the realization of feature mapping. Simulated annealing has been very useful in implementing the learning law for the Boltzmann machine. Several new learning laws were also developed, the prominent among

them being the reinforcement learning or learning with critic. Several architectures were developed to address specific issues in pattern recognition. Some of these architectures are: adaptive resonance theory (ART), neocognitron and counterpropagation networks. Currently, fuzzy logic concepts are being used to enhance the capability of the neural networks to deal with real world problems such as in speech, image processing, natural language processing and decision making [Lin and Lee, 1996].

1.3 Artificial Neural Networks: Terminology

Processing unit: We can consider an artificial neural network (ANN) as a highly simplified model of the **structure** of the biological neural network. An ANN consists of interconnected **processing units**. The general model of a processing unit consists of a summing part followed by an output part. The summing part receives N input values, weights each value, and computes a weighted sum. The weighted sum is called the **activation value**. The output part produces a signal from the activation value. The sign of the weight for each input **determines** whether the input is **excitatory** (positive weight) or **inhibitory** (negative weight). The inputs could be discrete or continuous data values, and likewise the outputs also could be discrete or continuous. The input and output could also be deterministic or stochastic or fuzzy.

Interconnections: In an artificial neural network several processing units are interconnected according to some topology to accomplish a pattern recognition task. Therefore the inputs to a processing unit may come from the outputs of other processing units, and/or from external sources. The output of each unit may be given to several units including itself. The amount of the output of one unit received by another unit depends on the strength of the connection between the units, and it is reflected in the **weight** value associated with the **connecting link**. If there are N units in a given ANN, then at any instant of time each unit will have a unique activation value and a unique output value. The set of the N activation values of the network defines the **activation state** of the network at that instant. Likewise, the set of the N output values of the network defines the **output state** of the network at that instant. Depending on the discrete or continuous nature of the activation and output values, the state of the network can be described by a discrete or continuous point in an N -dimensional space.

Operations: In operation, each unit of an ANN receives inputs from other connected units and/or from an external source. A weighted

Artificial Neural Networks: Terminology

sum of the inputs is computed at a given instant of time. The activation value determines the actual output from the ***output function*** unit, i.e., the output state of the unit. The output values and other external inputs in turn determine the activation and output states of the other units. ***Activation dynamics*** determines the activation values of all the units, i.e., the activation state of the network as a function of time. The activation dynamics also determines the dynamics of the output state of the network. The set of all activation states defines the activation ***state space*** of the network. The set of all output states defines the ***output*** state space of the network. Activation dynamics determines the trajectory of the path of the states in the state space of the network. For a given network, defined by the units and their interconnections with appropriate weights, the activation states determine the ***short term memory*** function of the network.

Generally, given an external input, the activation dynamics is followed to ***recall*** a pattern stored in a network. In order to store a pattern in a network, it is necessary to adjust the weights of the connections in the network. The set of all weights on all connections in a network form a weight vector. The set of all possible weight vectors define the ***weight space***. When the weights are changing, then the synaptic dynamics of the network determines the weight vector as a function of time. Synaptic dynamics is followed to adjust the weights in order to store the given patterns in the network. The process of adjusting the weights is referred to as ***learning***. Once the learning process is completed, the final set of weight values corresponds to the ***long term*** memory function of the network. The procedure to incrementally update each of the weights is called a ***learning law*** or ***learning algorithm***.

Update: In implementation, there are several options available for both activation and synaptic dynamics. In **particular**, the updating of the output states of all the units could be performed ***synchronously***. In this case, the activation values of all the units are computed at the same time, assuming a given output state throughout. From the activation values, the new output state of the network is derived. In an ***asynchronous*** update, on the other hand, each unit is updated sequentially, taking the current output state of the network into account each time. For each unit, the output state can be determined from the activation value either ***deterministically*** or ***stochastically***.

In practice, the activation dynamics, including the update, is much more complex in a biological neural network than the simple models mentioned above. The ANN models along with the equations governing the activation and synaptic dynamics are designed according to the pattern recognition task to be handled.

1.4 Models of Neuron

In this section we will consider three classical models for an artificial neuron or processing unit.

1.4.1 McCulloch-Pitts Model

In **McCulloch-Pitts (MP)** model (Figure 1.2) the activation (x) is given by a weighted sum of its M input values (a_i) and a bias term (θ). The

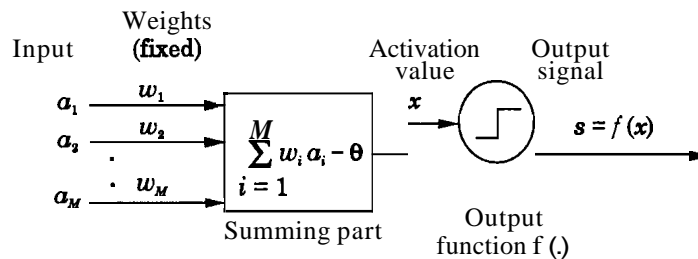


Figure 1.2 McCulloch-Pitts model of a neuron.

output signal (s) is typically a nonlinear function $f(x)$ of the activation value x . The following equations describe the operation of an MP model:

$$\text{Activation:} \quad x = \sum_{i=1}^M w_i a_i - \theta$$

$$\text{Output signal:} \quad s = f(x)$$

Three **commonly** used nonlinear functions (binary, ramp and sigmoid) are shown in Figure 1.3, although only the binary function

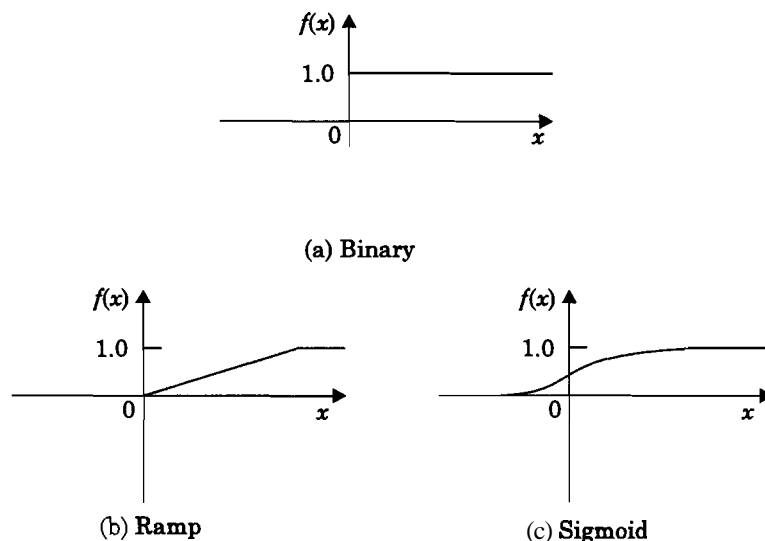
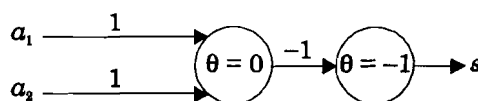


Figure 1.3 Some nonlinear functions.

was used in the original MP model. Networks consisting of MP neurons with binary (**on-off**) output signals can be configured to perform several logical functions [McCulloch and Pitts, 1943; Zurada, 1992]. Figure 1.4 shows some examples of logic circuits realized using

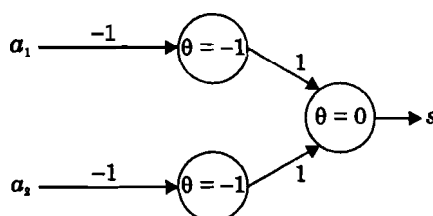
(a) NOR gate

a_1	a_2	s
0	0	1
0	1	0
1	0	0
1	1	0



(b) NAND gate

a_1	a_2	s
0	0	1
0	1	1
1	0	1
1	1	0



(c) Memory cell assuming unit delay for neuron. An initial excitatory input 1 sustains the output 1 and an initial inhibitory input +1 sustains the output 0

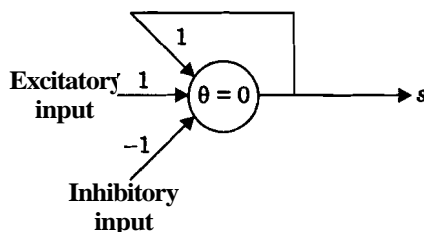


Figure 1.4 Illustration of some elementary logic networks using MP neurons.

the MP model. In this model a binary output function is used with the following logic:

$$f(x) = 1, x > 0 \\ = 0, x \leq 0$$

A single input and a single output MP neuron with proper weight and threshold gives an output a unit time later. This unit delay property of the MP neuron **can** be used to build sequential digital circuits. With feedback, it is also possible to have a memory cell (Figure 1.4c) which can retain the output indefinitely in the absence of any input.

In the MP model the weights are **fixed**. Hence a network using this model does not have the capability of learning. Moreover, the original model **allows** only binary output states, operating at discrete time steps

1.4.2 Perceptron

The **Rosenblatt's** perceptron model (Figure 1.5) for an artificial neuron consists of outputs **from** sensory units to a fixed set of association units, the outputs of which are fed to an MP neuron [Rosenblatt,

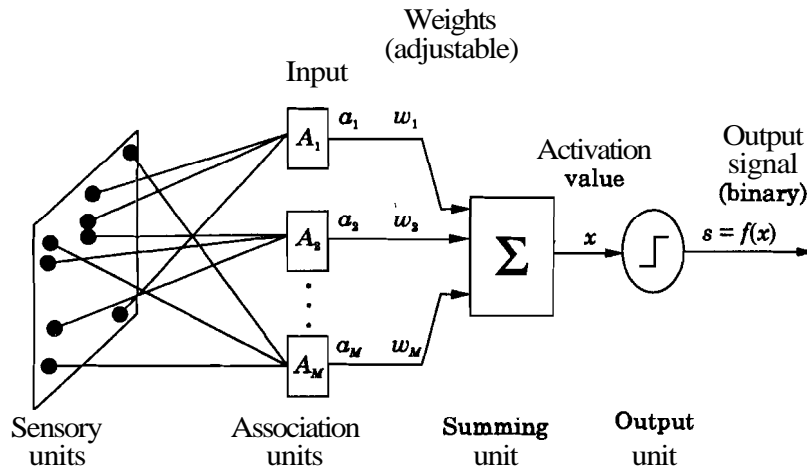


Figure 15 Rosenblatt's perceptron model of a neuron.

19581. The association units perform predetermined manipulations on their inputs. The main deviation from the MP model is that learning (i.e., adjustment of weights) is incorporated in the operation of the unit. The desired or target output (b) is compared with the actual binary output (s), and the error (ϵ) is used to adjust the weights. The following equations describe the operation of the perceptron model of a neuron:

Activation:
$$x = \sum_{i=1}^M w_i a_i - \theta$$

Output signal:
$$s = f(x)$$

Error:
$$\epsilon = b - s$$

Weight change:
$$\Delta w_i = \eta \epsilon a_i$$

where η is the learning rate parameter.

There is a perceptron learning law which gives a step-by-step procedure for adjusting the weights. Whether the weight adjustment converges or not depends on the nature of the desired input-output pairs to be represented by the model. The perceptron convergence theorem enables us to determine whether the given pattern pairs are representable or not. If the weight values converge, then the corresponding problem is said to be represented by the perceptron network.

1.4.3 Adaline

ADAPtive LINear Element (ADALINE) is a computing model proposed by Widrow and is shown in Figure 1.6 [Widrow, 1962]. The main distinction between the Rosenblatt's perceptron model and the

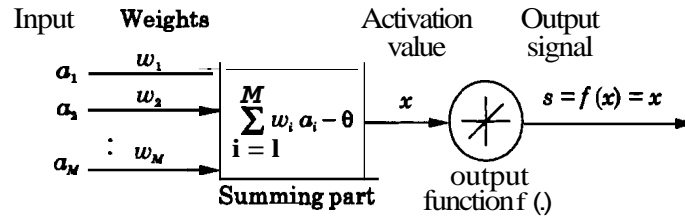


Figure 1.6 Widrow's Adaline model of a neuron.

Widrow's Adaline model is that, in the Adaline the analog activation value (x) is compared with the target output (b). In other words, the output is a linear function of the activation value (x). The equations that describe the operation of an Adaline are as follows:

$$\text{Activation:} \quad x = \sum_{i=1}^M w_i a_i - \theta$$

$$\text{Output signal:} \quad s = f(x) = x$$

$$\text{Error:} \quad \delta = b - s = b - x$$

$$\text{Weight change:} \quad \Delta w_i = \eta \delta a_i$$

where η is the learning rate parameter. This weight update rule **minimises** the mean squared error δ^2 , averaged over all inputs. Hence it is called Least Mean Squared (LMS) error learning law. This law is derived using the negative gradient of the error surface in the weight space. Hence it is also known as a gradient descent algorithm.

1.5 Topology

Artificial neural networks are useful only when the processing units are organised in a suitable manner to accomplish a given pattern recognition task. This section presents a few basic structures which will assist in evolving new architectures. The arrangement of the processing units, connections, and pattern **input/output** is referred to as *topology* [Simpson, 1990].

Artificial neural networks are normally organized into layers of processing units. The units of a layer **are similar** in the sense that they all have the same activation dynamics and output function. Connections can be made either from the units of one layer to the units of another layer (interlayer connections) or among the units within the layer (intralayer connections) or both interlayer and intralayer connections. Further, the connections across the layers and among the units within a layer can be organised either in a feedforward manner or in a feedback manner. In a feedback network the same processing unit may be visited more than once.

We will discuss a few basic structures which form building **blocks**

for more complex neural network architectures. Let us consider two layers F_1 and F_2 with M and N processing units, respectively. By providing connections to the j th unit in the F_2 layer from all the units in the F_1 layer, as shown in Figures 1.7a and 1.7b, we get two network structures *instar* and *outstar*, which have fan-in and fan-out geometries, respectively [Grossberg, 1982]. During learning, the normalised weight vector $\mathbf{w}_j = (w_{j1}, w_{j2}, \dots, w_{jM})^T$ in *instar* approaches the normalized input vector, when an input vector $\mathbf{a} = (a_1, a_2, \dots, a_M)^T$ is presented at the F_1 layer. Thus the activation $\mathbf{w}_j^T \mathbf{a} = \sum_{i=1}^M w_{ji} a_i$ of the j th unit in the F_2 layer will approach maximum value during learning. Whenever the input is given to F_1 , then the j th unit of F_2

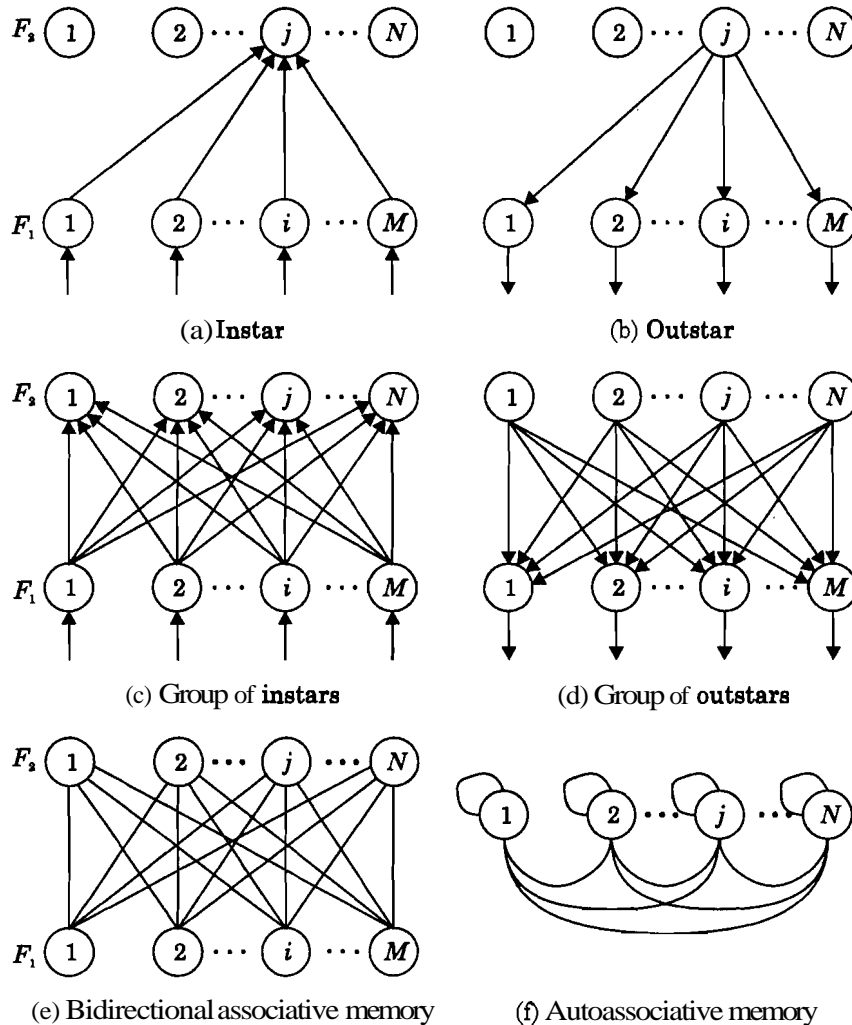


Figure 1.7 Some basic structures of artificial neural networks.

will be activated to the maximum extent. Thus the operation of an **instar** can be viewed as content addressing the memory. In the case of an **outstar**, during learning, the weight vector for the **connections** from the j th unit in F_2 approaches the activity pattern in F_1 , when an input vector a is presented at F_1 . During recall, whenever the unit j is activated, the signal pattern $(s_j w_{1j}, s_j w_{2j}, \dots, s_j w_{Mj})$ will be transmitted to F_1 , where s_j is the output of the j th unit. This signal pattern then produces the original activity pattern corresponding to the input vector a , although the input is absent. Thus the operation of an **outstar** can be viewed as memory addressing the contents.

When all the connections from the units in F_1 to F_2 are made as in Figure 1.7c, we obtain a heteroassociation network. This network can be viewed as a group of **instars**, if the flow is from F_1 to F_2 . On the other hand, if the flow is from F_2 to F_1 , then the network can be viewed as a group of **outstars** (Figure 1.7d).

When the flow is bidirectional, we get a bidirectional associative memory (Figure 1.7e), where either of the layers can be used as **input/output**.

If the two layers F_1 and F_2 coincide and the weights are symmetric, i.e., $w_{ji} = w_{ij}$, $i \neq j$, then we obtain an autoassociative memory in which each unit is connected to every other unit and to itself (Figure 1.7f).

1.6 Basic Learning Laws

The operation of a neural network is governed by neuronal dynamics. Neuronal dynamics consists of two parts: one corresponding to the dynamics of the activation state and the other corresponding to the dynamics of the synaptic weights. The Short Term Memory (**STM**) in neural networks is modelled by the activation state of the network. The Long Term Memory (**LTM**) corresponds to the encoded pattern information in the synaptic weights due to learning. We will discuss models of neuronal dynamics in Chapter 2. In this section we discuss some basic learning laws [Zurada, 1992, **Sec.** 2.5; Hassoun, 1995, Ch. 3]. Learning laws are merely implementation models of synaptic dynamics. Typically, a model of synaptic dynamics is described in terms of expressions for the first derivative of the weights. They are called **learning** equations.

Learning laws describe the weight vector for the i th processing unit at time instant $(t+1)$ in terms of the weight vector at time instant (t) as follows:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \Delta \mathbf{w}_i(t) \quad (1.1)$$

where $\Delta \mathbf{w}_i(t)$ is the change in the weight **vector**.

There are different methods for implementing the learning feature of a neural network, leading to several learning laws. Some

basic learning laws are discussed below. All these learning laws use only local information for adjusting the weight of the connection between two units.

1.6.1 Hebb's Law

Here the change in the weight vector is given by

$$\Delta \mathbf{w}_i = \eta f(\mathbf{w}_i^T \mathbf{a}) \mathbf{a} \quad (1.2)$$

Therefore, the j th component of $\Delta \mathbf{w}_i$ is given by

$$\begin{aligned} \Delta w_{ij} &= \eta f(\mathbf{w}_i^T \mathbf{a}) a_j \\ &= \eta s_i a_j, \quad \text{for } j = 1, 2, \dots, M \end{aligned} \quad (1.3)$$

where s_i is the output signal of the i th unit. The law states that the weight increment is proportional to the product of the input data and the resulting output signal of the unit. This law requires weight initialization to small random values around $w_{ij} = 0$ prior to learning. This law represents an unsupervised learning.

1.6.2 Perceptron Learning Law

Here the change in the weight vector is given by

$$\Delta \mathbf{w}_i = \eta [b_i - \text{sgn}(\mathbf{w}_i^T \mathbf{a})] \mathbf{a} \quad (1.4)$$

where $\text{sgn}(x)$ is sign of x . Therefore, we have

$$\begin{aligned} \Delta w_{ij} &= \eta [b_i - \text{sgn}(\mathbf{w}_i^T \mathbf{a})] a_j \\ &= \eta (b_i - s_i) a_j, \quad \text{for } j = 1, 2, \dots, M \end{aligned} \quad (1.5)$$

This law is applicable only for bipolar output functions $f(\cdot)$. This is also called discrete **perceptron** learning law. The expression for Δw_{ij} shows that the weights are adjusted only if the actual output s_i is incorrect, since the term in the square brackets is zero for the correct output. This is a supervised learning law, as the law requires a desired output for each input. In implementation, the weights can be initialized to any random initial values, as they are not critical. The weights converge to the final values eventually by repeated use of the input-output pattern pairs, provided the pattern pairs are representable by the system. These issues will be discussed in Chapter 4.

1.6.3 Delta Learning Law

Here the change in the weight vector is given by

$$\Delta \mathbf{w}_i = \eta [b_i - f(\mathbf{w}_i^T \mathbf{a})] f'(\mathbf{w}_i^T \mathbf{a}) \mathbf{a} \quad (1.6)$$

where $f'(x)$ is the **derivative** with respect to x . Hence,

$$\begin{aligned}\Delta w_{ij} &= \eta [b_i - f(\mathbf{w}_i^T \mathbf{a})] f'(\mathbf{w}_i^T \mathbf{a}) a_j \\ &= \eta [b_i - s_i] f'(x_i) a_j, \quad \text{for } j = 1, 2, \dots, M\end{aligned}\quad (1.7)$$

This law is valid only for a differentiable output function, as it depends on the derivative of the output function $f(\cdot)$. It is a supervised learning law since the change in the weight is based on the error between the desired and the actual output values for a given input. Delta learning law can also be viewed as a continuous **perceptron** learning law.

In implementation, the weights can be initialized to any random values as the values are not very critical. The weights converge to the final values eventually by repeated use of the input-output pattern pairs. The convergence can be more or less guaranteed by using more layers of processing units in between the input and output layers. The delta learning law can be generalized to the case of multiple layers of a **feedforward** network. We will discuss the generalized delta rule or the error backpropagation learning law in Chapter 4.

1.6.4 Widrow and Hoff LMS Learning Law

Here the change in the weight vector is given by

$$\Delta \mathbf{w}_i = \eta [b_i - \mathbf{w}_i^T \mathbf{a}] \mathbf{a} \quad (1.8)$$

Hence

$$\Delta w_{ij} = \eta [b_i - \mathbf{w}_i^T \mathbf{a}] a_j, \quad \text{for } j = 1, 2, \dots, M \quad (1.9)$$

This is a supervised learning law and is a special case of the delta learning law, where the output function is assumed linear, i.e., $f(x_i) = x_i$. In this case the change in the weight is made proportional to the negative gradient of the **error** between the desired output and the continuous activation value, which is also the continuous output signal due to linearity of the output function. Hence, this is also called the Least Mean Squared (LMS) error learning law. This is same as the learning law used in the Adaline model of neuron. In implementation, the weights may be initialized to any values. The input-output pattern pairs data is applied several times to achieve convergence of the weights for a given set of training data. The convergence is not **guaranteed** for any arbitrary training data set.

1.6.5 Correlation Learning Law

Here the change in the weight vector is given by

$$\Delta \mathbf{w}_i = \eta b_i \mathbf{a} \quad (1.10)$$

Therefore

$$\Delta w_{ij} = \eta b_i a_j, \quad \text{for } j = 1, 2, \dots, M \quad (1.11)$$

This is a special case of the Hebbian learning with the output signal (s_i) being replaced by the desired signal (b_i). But the Hebbian learning is an unsupervised learning, whereas the correlation learning is a supervised learning, since it uses the desired output value to adjust the weights. In the implementation of the learning law, the weights are initialised to small random values close to zero, i.e., $w_{ij} \approx 0$.

1.6.6 Instar (Winner-take-all) Learning Law

This is relevant for a collection of neurons, organized in a layer as shown in Figure 1.8. All the inputs are connected to each of the units

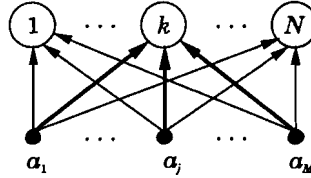


Figure 1.8 Arrangement of units for 'instar learning', where the adjusted weights are highlighted.

in the output layer in a **feedforward** manner. For a given input vector \mathbf{a} , the output from each unit i is computed using the weighted sum $\mathbf{w}_i^T \mathbf{a}$. The unit k that gives maximum output is identified. That is

$$\mathbf{w}_k^T \mathbf{a} = \max_i (\mathbf{w}_i^T \mathbf{a}) \quad (1.12)$$

Then the weight vector leading to the k th unit is adjusted as follows:

$$\Delta \mathbf{w}_k = \eta (\mathbf{a} - \mathbf{w}_k) \quad (1.13)$$

Therefore,

$$\Delta w_{kj} = \eta (a_j - w_{kj}), \quad \text{for } j = 1, 2, \dots, M \quad (1.14)$$

The final weight vector tends to represent a group of input vectors within a small neighbourhood. This is a case of unsupervised learning. In implementation, the values of the weight vectors are initialized to random values prior to learning, and the vector lengths are normalized during learning.

1.6.7 Outstar Learning Law

The **outstar** learning law is also related to a group of units arranged in a layer as shown in Figure 1.9. In this law the weights are adjusted so as to capture the desired output pattern characteristics. The adjustment of the weights is given by

$$\Delta w_{jk} = \eta (b_j - w_{jk}), \quad \text{for } j = 1, 2, \dots, M \quad (1.15)$$

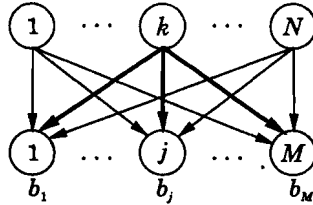


Figure 1.9 Arrangement of units for 'outstar learning', where the adjusted weights are highlighted.

where the k th unit is the only active unit in the input layer. The vector $\mathbf{b} = (b_1, b_2, \dots, b_M)^T$ is the desired response from the layer of M units. The **outstar** learning is a supervised learning law, and it is used with a network of **instars** to capture the characteristics of the input and output patterns for data compression. In implementation, the weight vectors are initialized to zero prior to **learning**.

1.6.8 Discussion on Basic Learning Laws

Table 1.2 gives a summary of the basic learning laws described so

Table 1.2 Summary of Basic Learning Laws (Adapted from [Zurada, 1992])

Learning law	Weight adjustment Δw_{ij}	Initial weights	Learning
Hebbian	$\Delta w_{ij} = \eta f(\mathbf{w}_i^T \mathbf{a}) a_j$ $= \eta s_i a_j,$ for $j = 1, 2, \dots, M$	Near zero	Unsupervised
Perceptron	$\Delta w_{ij} = \eta [b_i - \text{sgn}(\mathbf{w}_i^T \mathbf{a})] a_j$ $= \eta (b_i - s_i) a_j,$ for $j = 1, 2, \dots, M$	Random	Supervised
Delta	$\Delta w_{ij} = \eta [b_i - f(\mathbf{w}_i^T \mathbf{a})] \hat{f}'(\mathbf{w}_i^T \mathbf{a}) a_j$ $= \eta [b_i - s_i] f'(x_i) a_j,$ for $j = 1, 2, \dots, M$	Random	Supervised
Widrow-Hoff	$\Delta w_{ij} = \eta [b_i - \mathbf{w}_i^T \mathbf{a}] a_j,$ for $j = 1, 2, \dots, M$	Random	Supervised
Correlation	$\Delta w_{ij} = \eta b_i a_j,$ for $j = 1, 2, \dots, M$	Near zero	Supervised
Winner-take-all	$\Delta w_{kj} = \eta (a_j - w_{kj}),$ k is the winning unit, for $j = 1, 2, \dots, M$	Random but normalised	Unsupervised
Outstar	$\Delta w_{jk} = \eta (b_j - w_{jk}),$ for $j = 1, 2, \dots, M$	Zero	Supervised

Basics of Artificial Neural Networks

far. It shows the type of learning (supervised/unsupervised) and the nature of the output function (**sgn** for discrete $f(\cdot)$ for continuous) for which each law is applicable. The most important issue in the application of these laws is the convergence of the weights to some final limit values as desired. The convergence and the limit values of the weights depend on the initial setting of the weights prior to learning, and on the learning rate parameter.

The Hebb's law and the correlation law lead to the sum of the correlations between input and output (for Hebb's law) components and between input and desired output (for correlation law) components, **respectively**. But in order to achieve this, the starting initial weight values should be small random values near zero. The learning rate parameter η should be close to one. Typically, the set of patterns are applied only once in the training process. In some variations of these (as in the principal component learning to be discussed in Chapter 6), the learning rate parameter is set to a small value (< 1) and the training patterns are applied several times to achieve convergence.

The perceptron, delta and LMS learning laws lead to **final** steady state values (provided they converge), only when the weight adjustments are small. Since the correction depends on the error between the desired output and the actual output, only a small portion of the error is used for adjustment of the weights each time. Thus the learning rate parameter $\eta \ll 1$. The initial weights could be set to random values. The set of training patterns need to be applied several times to achieve convergence, if it exists. The convergence will naturally be faster if the starting weights are close to the final steady values.

The weights **in the instar** and **outstar** learning laws converge to the mean values of a set of input and desired output patterns, respectively. In these cases the learning rate parameter is typically set to a value less than one ($\eta < 1$). The weights in the case of **instar** can be initialized to any random values, and in the case of **outstar** to small random values near zero. The set of training patterns are applied several times to achieve convergence.

Besides these basic learning laws there are many other learning laws evolved primarily for application in different situations [Hassoun, 1995, Ch. 3]. Some of them will be discussed at appropriate places in the later chapters.

1.7 Summary

In this chapter we have seen the motivation and background for the current interest in the study of problems based on models using artificial neural networks. We have reviewed the features of the biological neural network and discussed the feasibility of realizing

some of these features through parallel and distributed processing (PDP) models (Appendix A). In particular, the associative memory, fault tolerance and concept learning features could be demonstrated through these PDP models. Some key developments in artificial neural networks were presented to show how the field has evolved to the present state of understanding.

An artificial neural network is built using a few basic building blocks. The building blocks were introduced starting with the models of artificial neurons and the topology of a few basic structures. While developing artificial neural networks for specific applications, the weights are adjusted in a systematic manner using learning laws. We have discussed some basic learning laws and their characteristics. But the full potential of a neural network can be exploited if we can incorporate in its operation the neuronal activation and synaptic dynamics of a biological neural network. Some features of these dynamics are discussed in the next chapter.

Review Questions

1. Describe some attractive features of the biological neural network that make it superior to the most sophisticated Artificial Intelligence computer system for pattern recognition tasks.
2. Explain briefly the terms cell body, axon, synapse, dendrite and neuron with reference to a biological neural network.
3. Explain briefly the operation of a biological neural network.
4. Compare the performance of a computer and that of a biological neural network in terms of speed of processing, size and complexity, storage, fault tolerance and control mechanism.
5. Give two examples of pattern recognition tasks to illustrate the superiority of the biological neural network over a conventional computer system.
6. What are the main differences among the three models of artificial neuron, namely, **McCulloch-Pitts**, **perceptron** and **adaline**?
7. What is meant by topology of artificial neural networks? Give a few basic topological structures of artificial neural networks.
8. What is the distinction between learning equation and learning law?
9. What are the basic learning laws?
10. Explain the significance of the initial values of weights and the learning rate parameter in the seven basic learning laws.
11. Identify supervised and unsupervised basic learning laws.
12. Compare LMS, perceptron and delta **learning** laws.

Problems

1. Explain the logic **functions** (using truth tables) performed by the following networks with MP neurons given in Figure P1.1.

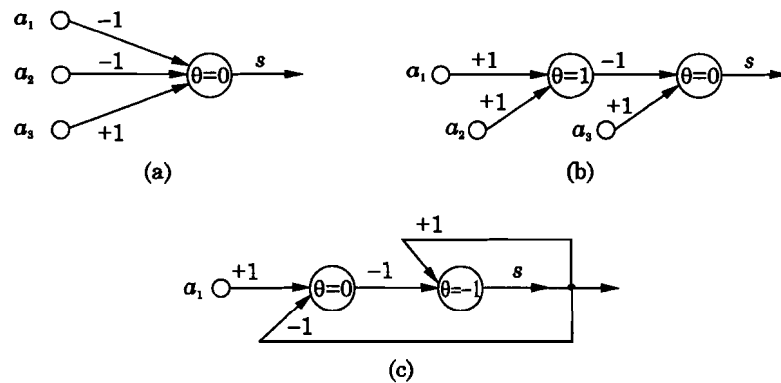


Figure P1.1 Three networks using MP neurons.

2. Design networks using M-P neurons to realize the following logic functions using ± 1 for the weights.
 - (a) $s(a_1, a_2, a_3) = a_1 a_3 + a_2 a_3 + \bar{a}_1 \bar{a}_3$
 - (b) $s(a_1, a_2, a_3) = a_1 a_2 a_3$
 - (c) $s(a_1, a_2, a_3) = \bar{a}_1 a_2 \bar{a}_3$
3. Give the output of the network in Figure P1.2 for the input $[111]^T$.

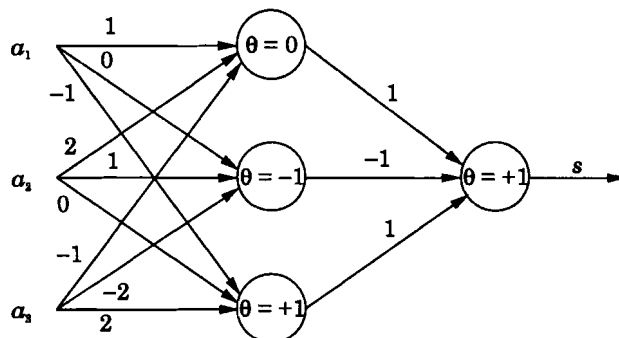


Figure P1.2 A feedforward network with MP neurons.

4. Determine the weights of the network in Figure **P1.3a** after one iteration using Hebb's law for the following set of input vectors for two different types of output **functions** shown in Figures **P1.3b** and **P1.3c**. Use suitable values for the **initial weights** and learning rate parameter. Input: $[1100]^T$, $[1001]^T$, $[0011]^T$ and $[0110]^T$. Choose $f(x) = 1/(1 + e^{-x})$ for Figure **P1.3c**.

Problems

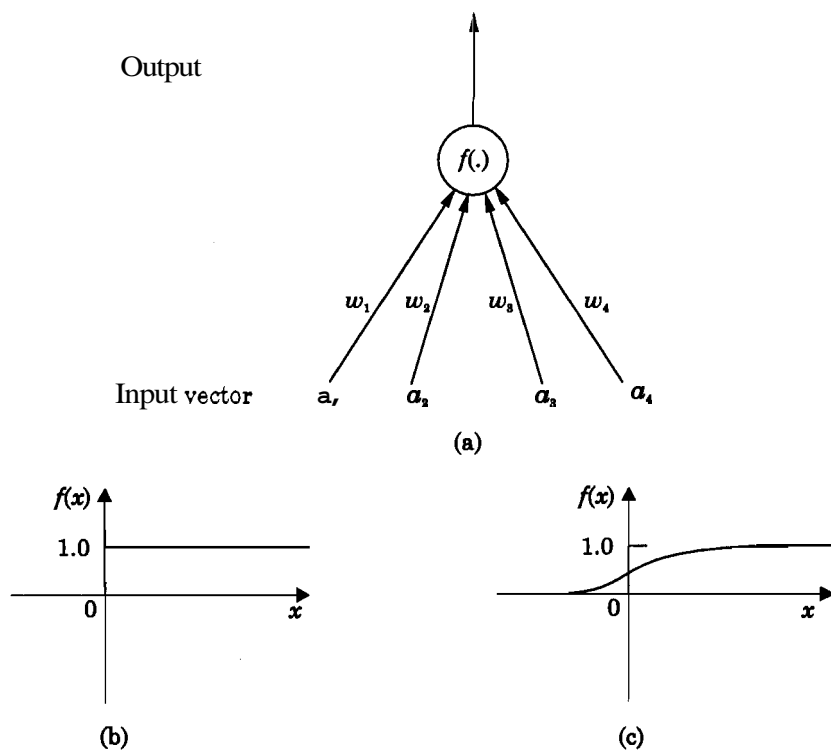


Figure P1.3 (a) A neuron with four inputs and one output, (b) Hard-limiting output function and (c) Sigmoid output function.

5. Determine the weights of a network with 4 input and 2 output units using (a) Perceptron learning law and (b) Delta learning law with $f(x) = 1/(1 + e^{-x})$ for the following **input-output** pairs:

Input: $[1100]^T$ $[1001]^T$ $[0011]^T$ $[0110]^T$

Output: $[11]^T$ $[10]^T$ $[01]^T$ $[00]^T$

Discuss your results for different choices of the learning rate parameters. Use suitable values for the initial weights. (Hint: Write a program to implement the learning laws.)

6. Using the **Instar** learning law, group all the sixteen possible binary **vectors** of length 4 into four different groups. Use suitable values for the initial weights and for the learning rate parameter. Use a 4-unit input and 4-unit output network. Select random initial weights in the range $[0, 1]$. (Hint: Write a program to implement the learning law.)

Chapter 2

Activation and Synaptic Dynamics

2.1 Introduction

An artificial neural network consists of several processing units (or artificial neurons) interconnected in a predetermined manner to accomplish a desired pattern recognition task. In the previous chapter we have seen some models of neurons and some basic topologies, using which it is possible to build complex structures. However, the structure of an artificial neural network is not useful, unless the rules governing the changes of the activation values and connection weight values are also specified. These rules are implied or specified in the activation and synaptic dynamics equations governing the behaviour of the network structure to accomplish the desired task.

In a neural network with N processing units, the set of activation values of the units at any given instant defines the activation state of the network. Typically, a problem is specified by a point in the activation state space. The trajectory of the activation states, leading to a solution state, reflects the dynamics of the network. The trajectory depends upon the activation dynamics built into the network. The activation dynamics is prescribed by a set of equations, which can be used to determine the activation state of the network at the next instant, given the activation state at the current instant.

For a given input data, the weights of the connecting links in a network are adjusted to enable the network to learn the pattern in the given input data. The set of weight values of all the links in a network at any given instant defines the weight state, which can be viewed as a point in the weight space. The trajectory of the weight states in the weight space is determined by the synaptic dynamics of the network.

A network is led to one of its steady activation states by the activation dynamics and the input pattern. Since the steady activation state depends on the input pattern, it is referred to as short term memory. The state will change if the input pattern changes. On the

other hand, the steady weight state of a network is determined by the synaptic dynamics for a given set of training inputs, and it does not change. Hence this steady weight state is referred to as *long term memory*.

Activation dynamics relates to the fluctuations at the neuronal level in a biological neural network. Typically, these fluctuations take place in average intervals of the order of a few milliseconds. Thus the neuronal level dynamics is **significantly faster** than the dynamics at the synaptic level, where significant changes in the synaptic weights take place at intervals of the order of a few seconds. Therefore, it can be assumed that during activation dynamics the synaptic weights do not change significantly, i.e., the weights can be assumed to be constants of the network.

The objective of this chapter is to discuss models for activation and synaptic dynamics. We must distinguish two situations here. *Models of neural networks* normally refer to the mathematical representation of our understanding and observed behaviour of the biological neural network. The purpose in this case is to capture the knowledge by the model. The model is not intended for detailed analysis of the network. Therefore a model of the neural network could be very complex, involving first and higher order derivatives of activations and weights, as well as several nonlinear interactions. In contrast, the purpose of *neural network models* is to provide a representation for the dynamics of an artificial network, incorporating features inspired by our understanding of the operation of the biological neural network. In other words, the neural network model is a mathematical model for analysis of gross characteristics of an artificial network. Typically, these models are described by an expression for the first order time derivative of the activation state for activation dynamics and an expression for the first order time derivative of the weight state for synaptic dynamics. **These** expressions are usually simple enough (although nonlinear) to enable us to predict the global characteristics of the network. An expression for the first derivative may contain time parameter explicitly, in which case such systems become nonautonomous dynamical systems. If the expression does not contain the time parameter explicitly, then the systems become autonomous dynamical systems, which are relatively easier to analyze. Throughout this chapter we use the terms models of neural networks and neural network models interchangeably, although they refer to the autonomous dynamical system models represented by an expression for the first derivative of the activation value of each unit in the network.

We discuss activation dynamics and synaptic dynamics, separately. The discussion on the activation and synaptic dynamics is adapted from [Kosko, 1992]. In Section 2.2 on the activation dynamics models we consider the additive, shunting (or

multiplicative) and stochastic models. We also discuss the equilibrium states of the networks with a specified activation dynamics. Since synaptic dynamics models lead to learning laws, in Section 2.3 we first consider the requirements of the learning laws for effective implementation. In this section we also discuss the distinction between the activation and synaptic dynamics models. In Section 2.4 several categories of learning are discussed, which include Hebbian, competitive, error correcting and stochastic learning. A brief discussion is included on the equilibrium of synaptic dynamics. In Section 2.5 we discuss the issues of stability and convergence in activation and synaptic dynamics, respectively. We shall review the general stability theorems and discuss briefly the issues of global and structural stability in neural networks. In Section 2.6 we discuss methods for neural network recall for both feedforward and feedback networks. In the final section we provide a brief summary of the issues discussed in this chapter.

2.2 Activation Dynamics Models

2.2.1 Issues in the Development of Activation Dynamics Models

Activation dynamics is described by the first derivative of the activation value of a neuron [Kosko, 1992]. For the i th neuron, it is expressed as

$$\dot{x}_i = \frac{dx_i}{dt} = h(.) \quad (2.1)$$

where $h(.)$ is a function of the activation state and synaptic weights of the network. Let us consider a network of N interconnected processing units, where the variables and constants of each unit are shown in Figure 2.1. The activation value is generally associated with

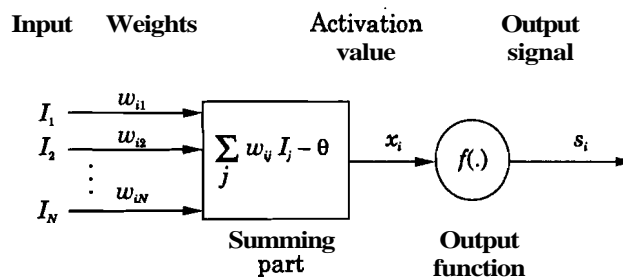


Figure 2.1 A typical processing unit i with associated parameters.

the cell membrane potential. The output function $f(.)$ determines the output signal generated at the axon hillock due to a given membrane potential. This function bounds the output signal, and it is normally

a nondecreasing function of the activation value. Thus the output is bounded as shown in **Figure 2.2a** for a typical output function. Although the activation value is shown to have a large range, in practice the membrane potential has to be bounded due to limitation of the current carrying capacity of a membrane. Thus there is a limit to the *operating range* of a processing unit, which corresponds to the difference between the maximum and minimum activation values.

The input values to a processing unit coming from external sources, especially through sensory inputs, may have a large dynamic range, as for example, the reflections **from** an object in a dim light and the same in a bright light. **Thus** the dynamic range of the external input values could be **very** large, and usually not in our control. If the neuron is made sensitive to smaller values of **inputs**, as in **Figure 2.2b**, its output signal will saturate for large input values, i.e., for $x > x_1$ in the figure. Moreover, even a noisy input could produce some output signal, which is not desirable. On the other hand, if the neuron is made sensitive to large values of the input by making the threshold θ large, as in **Figure 2.2c**, its activation value becomes insensitive to small values of the input. This is the familiar *noise-saturation dilemma* [Grossberg, 1982]. The problem is how a neuron with limited operating range for the activation values can be made sensitive to nearly unlimited range of the input values.

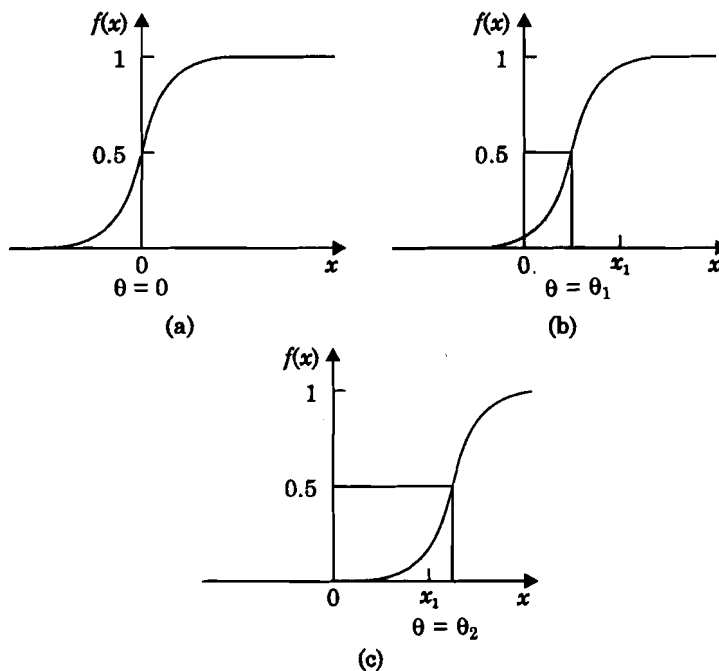


Figure 2.2 Output functions for three different bias values (θ).

The input to a processing unit may come from the outputs of the other neurons connected to it through synaptic weights, or from an external source such as a sensory input. Both of these types of inputs may have excitatory components which tend to increase the activation of the unit, or inhibitory components which tend to decrease the activation of the unit. The input, the activation value and the output could fall into one or more of the following categories of data depending on the nature of the external input and the nature of the output function: deterministic or stochastic, crisp or fuzzy and discrete or continuous.

In developing models for activation dynamics, it is necessary to take into account the known behaviour **from** the studies on biological neuronal dynamics, but at the same time, the models must be tractable for analysis to examine the global behaviour of a network consisting of a large number of interconnecting processing units. In particular, the model should be such that it should be possible to study the behaviour of the equilibrium states of the network to determine whether the network is globally and structurally stable. Structural stability refers to the state equilibrium situation where small perturbations of the state around the equilibrium brings the network back to the equilibrium state. This depends on the behaviour of the network in the neighbourhood of the equilibrium state, which in turn depends on the activation dynamics and the connection weights of the network. The model also should be able to learn (adjust the weights) while **satisfying** the requirements of storage capacity and stability characteristics. Global stability refers to the state equilibrium condition when both the synaptic and activation dynamics are simultaneously used. In the following discussion we will assume that the weights do not change while examining the activation dynamics.

We discuss models for activation dynamics starting from simple additive models and then moving to more general shunting or multiplicative models. Initially, we consider only the deterministic models and then extend the models to stochastic versions. We also provide a discussion on the equilibrium behaviour for different models of the network. It should be noted that each model takes into account a few features of the neuronal dynamics, which may be relevant for a **particular/limited** application.

2.2.2 Additive Activation Models

As mentioned before, the activation value x_i of the i th neuron can be interpreted as the cell membrane potential, and it is a function of time, i.e., $x_i = x_i(t)$. The activation models are described by an expression for the first derivative of the activation value of a neuron. Thus $\dot{x}_i(t)$ gives the rate of change of the activation value of the i th neuron of a neural network.

For the simplest case of a passive decay situation,

$$\dot{x}_i(t) = -A_i x_i(t) \quad (2.2)$$

where A_i (> 0) is a constant for the membrane and can be interpreted as the passive decay rate. The solution of this equation is given by

$$x_i(t) = x_i(0) e^{-A_i t} \quad (2.3)$$

In **electrical** circuit analogy, A_i can be interpreted as membrane conductance, which is inverse of the membrane resistance (R_i). The initial value of x_i is $x_i(0)$. The steady state value of x_i is given by $x_i(\infty) = 0$, which is also called the resting potential.

The passive decay time constant is altered by the membrane capacitance C_i which can also be viewed as a time scaling parameter. With C_i , the passive decay model is given by

$$C_i \dot{x}_i(t) = -A_i x_i(t) \quad (2.4)$$

and the solution is given by

$$x_i(t) = x_i(0) e^{-(A_i/C_i)t} \quad (2.5)$$

Without loss of generality, we can assume $C_i = 1$ throughout the following discussion. If we assume a nonzero resting potential, then the activation model can be expressed by adding a constant P_i to the passive decay term as

$$\dot{x}_i(t) = -A_i x_i(t) + P_i, \quad (2.6)$$

whose solution is given by

$$x_i(t) = x_i(0) e^{-A_i t} + \frac{P_i}{A_i} (1 - e^{-A_i t}) \quad (2.7)$$

The steady state activation value is given by $x_i(\infty) = P_i/A_i$, the resting potential.

Assuming the resting potential to be zero ($P_i = 0$), if there is a constant external excitatory input I_i , then the additive activation model is given by

$$\dot{x}_i(t) = -A_i x_i(t) + B_i I_i, \quad (2.8)$$

where B_i (> 0) is the weight given to I_i . The solution of this equation is given by

$$x_i(t) = x_i(0) e^{-A_i t} + \frac{B_i I_i}{A_i} (1 - e^{-A_i t}) \quad (2.9)$$

The steady state activation value is given by $x_i(\infty) = B_i I_i/A_i$, which shows that the activation value directly depends on the **external** input, and thus it is unbounded.

In addition to the external input, if there is an input from the outputs of the units, then the model becomes an additive autoassociative model, and is given by

$$\dot{x}_i(t) = -A_i x_i(t) + \sum_{j=1}^N f_j(x_j(t)) w_{ij} + B_i I_i \quad (2.10)$$

where $f_j(\cdot)$ is the output function of the j th unit. For inhibitory feedback connections or for inhibitory external input, the equations will be similar to the above except for the signs of the second and third terms in the above equation. The classical neural circuit described by **Perkel** is a special case of the additive autoassociative model, and is given by [**Perkel** et al, 1981]

$$\begin{aligned} C_i \dot{x}_i(t) &= \frac{-x_i(t)}{R_i} + \sum_{j=1}^N \frac{x_j(t) - x_i(t)}{R_{ij}} + B_i I_i \\ &= -\frac{x_i(t)}{R_i'} + \sum_{j=1}^N \frac{x_j(t)}{R_{ij}} + B_i I_i \end{aligned} \quad (2.11)$$

where R_{ij} is the resistance between the neurons i and j , and

$$\frac{1}{R_i'} = \frac{1}{R_i} + \sum_{j=1}^N \frac{1}{R_{ij}} \quad (2.12)$$

Perkel's model assumes a linear output function $f(x) = x$, thus resulting in a signal which is unbounded. If the output function $f(x)$ is strictly an increasing but bounded function, as in Figure 2.2, and the connection weights are symmetric, i.e., $w_{ij} = w_{ji}$, then the resulting model is called **Hopfield** model [**Hopfield**, 1982]. The **Hopfield** model belongs to the class of feedback neural network models, called autoassociative memory, that are globally stable. We will discuss further on this point in a later section.

A network consisting of two layers of processing units, where each unit in one layer (say layer 1) is connected to every unit in the other layer (say layer 2) and vice versa, is called a heteroassociative network. The additive activation model for a heteroassociative network is given by

$$\begin{aligned} \dot{x}_i(t) &= -A_i x_i(t) + \sum_{j=1}^N f_j(y_j(t)) v_{ij} + I_i, \quad i = 1, 2, \dots, M \\ \dot{y}_j(t) &= -A_j y_j(t) + \sum_{i=1}^M f_i(x_i(t)) w_{ji} + J_j, \quad j = 1, 2, \dots, N \end{aligned} \quad (2.13)$$

where I_i and J_j are the net external inputs to the units i and j , respectively. Note that A_i and $f_i(\cdot)$ could be different for each unit and

for each layer. In the above equations $V = [v_{ij}]$ is the matrix of weights from the units in the layer 2 to the units in the layer 1, and $W = [w_{ji}]$ is the matrix of weights from the units in the layer 1 to the units in the layer 2. These are coupled first order differential equations. Under special conditions, such as the weights in both the directions being identical, i.e., $W = V^T$, and the output function being bounded, the resulting hetroassociative model reduces to a bidirectional associative memory [Kosko, 1988]. Analogous to the Hopfield autoassociative memory, the bidirectional associative memory can also be proved to be globally stable. Table 2.1 gives a summary of the development of activation dynamics models discussed in this section.

Table 2.1 Summary of Development of Additive Activation Dynamics Models

- General form:

$$\dot{x}_i(t) = h(\cdot), \quad i = 1, 2, \dots, N$$

- Passive decay term:

$$C_i \dot{x}_i(t) = -A_i x_i(t),$$

where A_i is the membrane conductance and C_i is the membrane capacitance

- Nonzero resting potential (P_i/A_i):

$$\dot{x}_i(t) = -A_i x_i(t) + P_i$$

- With external input ($B_i I_i$):

$$\dot{x}_i(t) = -A_i x_i(t) + B_i I_i,$$

where B_i is a positive constant

- Additive **autoassociative** model:

$$\dot{x}_i(t) = -A_i x_i(t) + \sum_{j=1}^N f_j(x_j(t)) w_{ij} + B_i I_i$$

- **Perkel's** model:

$$C_i \dot{x}_i(t) = \frac{-x_i(t)}{R_i} + \sum_{j=1}^N \frac{x_j(t) - x_i(t)}{R_{ij}} + B_i I_i$$

- Hetroassociative model:

$$\dot{x}_i(t) = -A_i x_i(t) + \sum_{j=1}^N f_j(y_j(t)) v_{ij} + I_i, \quad i = 1, 2, \dots, M$$

$$\dot{y}_j(t) = -A_j y_j(t) + \sum_{i=1}^M f_i(x_i(t)) w_{ji} + J_j, \quad j = 1, 2, \dots, N$$

- Bidirectional associative memory:

$$[w_{ji}] = [v_{ij}]^T$$

2.2.3 Shunting Activation Models

Grossberg has proposed a shunting activation model to restrict the range of the activation values to a specified operating range irrespective of the dynamic range of the external inputs [Grossberg, 1982; Grossberg, 19881. We will first consider the saturation model, where the activation value is bounded to an upper limit. For an excitatory external input I_i , the shunting activation model is given by

$$\dot{x}_i(t) = -A_i x_i(t) + [B_i - x_i(t)] I_i \quad (2.14)$$

The steady state activation value is obtained by setting $\dot{x}_i(t) = 0$, and solving for x_i . The result is

$$x_i(\infty) = \frac{B_i I_i}{A_i + I_i} = \frac{B_i}{1 + A_i/I_i} \quad (2.15)$$

As the input $I_i \rightarrow \infty$, then $x_i(\infty) \rightarrow B_i$. That is, the steady state value of x_i saturates at B_i . In other words, if the initial value $x_i(0) \leq B_i$, then $x_i(t) \leq B_i$ for all t . If the input value refers to an intensity of reflected light $I_i = p_i I$, where I is the background intensity value, and p_i is the fraction of the background intensity that is input to the i th unit, then the above saturation model is insensitive to p_i for large background intensities.

In order to make the steady state activation value sensitive to reflectance, irrespective of the background intensity, Grossberg suggested an on-centre off-surround shunting activation model by providing inhibitory inputs from other input elements to the i th unit along with the excitatory input from the i th input element to i th unit as shown in Figure 2.3. Throughout the following discussion we

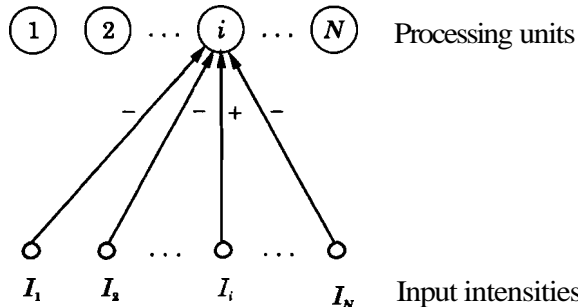


Figure 2.3 An on-center and off-surround configuration for shunting activation model.

assume a hard-limiting threshold function for the output function. That is, $f(x) = 0$, for $x \leq 0$ and, $f(x) = 1$, for $x > 0$. Assuming $\sum_{i=1}^N I_i = I$ and $I_i = p_i I$, for convenience, the shunting activation model with on-centre off-surround configuration is given by

$$\dot{x}_i(t) = -A_i x_i(t) + [B_i - x_i(t)] I_i - x_i(t) \sum_{j \neq i} I_j. \quad (2.16)$$

The steady state activation value is obtained by setting $\dot{x}_i(t) = 0$, and is given by

$$x_i(\infty) = \frac{B_i I_i}{A_i + I} = \frac{B_i p_i}{1 + A_i/I} \rightarrow B_i p_i, \quad \text{as } I \rightarrow \infty \quad (2.17)$$

From this we can see that, even if $I_i \rightarrow \infty$, as $I \rightarrow \infty$, the steady state activation value $x_i(\infty)$ does not saturate. Instead, $x_i(\infty)$ will still be sensitive to p_i , the input reflection. It can be seen that, since $p_i < 1$, the steady activation value is always less than B_i , the saturation limit, i.e., $x_i(t) < B_i$ for all t .

In order to make a unit insensitive to small positive inputs, may be due to noise, the shunting activation model can be modified to incorporate a lower limit (< 0) to the activation value. The following is the resulting model:

$$\dot{x}_i(t) = -A_i x_i(t) + [B_i - x_i(t)] I_i - [E_i + x_i(t)] \sum_{j \neq i} I_j \quad (2.18)$$

The steady state activation value is obtained by setting $\dot{x}_i(t) = 0$, and is given by

$$x_i(\infty) = \frac{(B_i + E_i) I_i - E_i I}{A_i + I} = \left(p_i - \frac{E_i}{B_i + E_i} \right) \left(\frac{B_i + E_i}{1 + A_i/I} \right) \quad (2.19)$$

Note that $x_i(\infty) \rightarrow p_i(B_i + E_i) - E_i$ as $I \rightarrow \infty$. This steady state activation value is negative as long as the input reflectance value to the i th unit, $p_i < E_i/(B_i + E_i)$. In that case the output signal of the unit will be zero, since we assume that $f(x) = 0$, for $x \leq 0$. That is, the i th processing unit will be sensitive to the input only if its input reflectance value is above a threshold. Thus it is possible to make the unit insensitive to random noise input within a specified threshold limit value. The above shunting activation model has therefore an operating range of $[-E_i, B_i]$ for the activation value, since the lowest value for $x_i(\infty) = -E_i$, which occurs when $p_i = 0$.

A shunting activation model with excitatory feedback from the same unit and inhibitory feedback from other units is given by

$$\begin{aligned} \dot{x}_i(t) = & -A_i x_i + (B_i - x_i) [I_i + f_i(x_i)] \\ & - (E_i + x_i) \left[J_i + \sum_{j \neq i} f_j(x_j) w_{ij} \right] \end{aligned} \quad (2.20)$$

where J_i is the inhibitory component of the external input. Note that, on the right hand side of Eq. (2.20) $x_i(t)$ is replaced by x_i for convenience. The inhibitory sign is taken out of the weights w_{ij} , and

hence $w_{ij} > 0$. The shunting model of (2.20) is a special case of Hodgkin-Huxley membrane equations [Hodgkin and Huxley, 1952].

Equation (2.20) can be written in the most general form as

$$\dot{x}_i(t) = -A_i x_i + (B_i - C_i x_i) [I_i + f_i(x_i)] - (E_i + D_i x_i) \left[J_i + \sum_{j \neq i} f_j(x_j) w_{ij} \right], \quad (2.21)$$

where all the constants are positive. The first term on the right hand side corresponds to the passive decay term; the second term corresponds to the excitatory term and the third term corresponds to the inhibitory term. If we consider the excitatory term $(B_i - C_i x_i) [I_i + f_i(x_i)]$, it shows the contribution of the excitatory (external and feedback) input in increasing the activation value $(x_i(t))$ of the input. If $C_i = 0$, then the contribution of this input reduces to an additive effect, as in the additive activation model. If $C_i > 0$, then the contribution of the excitatory input reduces to zero when the activation $x_i(t) = B_i/C_i$. This can be viewed as shunting effect in an equivalent electrical circuit, and hence the name shunting activation model. If the initial value $x_i(0) \leq B_i/C_i$, then the model ensures that $x_i(t) \leq B_i/C_i$, for all $t > 0$, thus showing the boundedness of the activation value within an upper limit. This can be proved by the following argument: If $x_i(t) > B_i/C_i$, then the second term becomes negative, since we assume that $f(x) = 1$, for all $x > 0$ and $I_i > 0$. Since the contribution due to the inhibitory third term is negative, if the excitatory second term is also negative, then the steady activation value, obtained by setting $\dot{x}_i(t) = 0$, will be negative. Thus there is a contradiction, since we started with the assumption that $x_i(t) > B_i/C_i$, which is positive. Hence $x_i(t) < B_i/C_i$ for all t .

Likewise, the inhibitory term $(E_i + D_i x_i) [J_i + \sum_{j \neq i} f_j(x_j) w_{ij}]$ shows the contribution of the inhibitory (external and feedback) input in decreasing the activation value $x_i(t)$ of the unit. In this case, if $D_i = 0$, then the contribution of this input reduces to an additive effect, as in the additive activation model. If $D_i > 0$, then the contribution of the inhibitory input reduces to zero when the activation $x_i(t) = -E_i/D_i$. This can be viewed as a shunting effect in an equivalent electrical circuit. If the initial value $x_i(0) \geq -E_i/D_i$, then the model ensures that $x_i(t) \geq -E_i/D_i$, for all $t > 0$. This can be proved by the following argument: For $x_i(t) < -E_i/D_i$, the contribution of the inhibitory third term will be positive, since $f(x) > 0$, for all x . Since the excitatory second term is always positive, the steady state activation value obtained by setting $\dot{x}_i(t) = 0$ is always positive. But we assumed that $x_i(t) < -E_i/D_i$, which is negative. Thus there is a contradiction. Hence $x_i(t) \geq -E_i/D_i$. Table 2.2 gives a summary of

the development of shunting activation models discussed in this section.

Table 2.2 Summary of Development of Shunting Activation Dynamics Models

- **Goal: To keep the operating range of activation value to a specified range**
- **General form:**

$$\dot{x}_i(t) = h(\cdot)$$
- **Saturation model: To restrict to an upper limit**

$$\dot{x}_i(t) = -A_i x_i(t) + [B_i - x_i(t)] I_i$$
- **On-centre off-surround configuration: To make it sensitive to changes in the external input**

$$\dot{x}_i(t) = -A_i x_i(t) + [B_i - x_i(t)] I_i - x_i(t) \sum_{j \neq i} I_j$$
- **Setting noise limit:**

$$\dot{x}_i(t) = -A_i x_i(t) + [B_i - x_i(t)] I_i - [E_i + x_i(t)] \sum_{j \neq i} I_j$$
- **With excitatory feedback from the same unit and inhibitory feedback from other units:**

$$\dot{x}_i(t) = -A_i x_i + (B_i - x_i) [I_i + f_i(x_i)] - (E_i + x_i) \left[J_i + \sum_{j \neq i} f_j(x_j) w_{ij} \right]$$

2.2.4 Stochastic Models

The activation models considered so far are deterministic models. In practice, the **input/output** patterns and the activation values may be considered as sample functions of random processes. The output signal of each processing unit may be a random function of the unit's activation value. In such cases the network activation state and output signal state can be viewed as vector stochastic processes. Each unit in turn behaves as a scalar stochastic process.

Stochastic activation models are represented in a simplified fashion by adding an additional noise component to the right side of the expression for $\dot{x}_i(t)$ for each of the deterministic activation models. The probability distribution of the noise component is assumed for analyzing the vector stochastic processes of the activation states. In particular, in stochastic equilibrium, the activation state vector hovers in a random fashion about a fixed (deterministic) equilibrium state, representing the average.

2.2.5 Discussion on Equilibrium

Normally the term equilibrium is used to denote the state of a network at which the network settles when small perturbations are

made to the state. In the deterministic models, the equilibrium states are also steady states. Hence these states satisfy the equations $\dot{\mathbf{x}}_i(t) = 0$, for $i = 1, 2, \dots, N$. Note that $\dot{\mathbf{x}}_i(t) = \mathbf{0}$ is a necessary condition for a state to be an equilibrium state, but not a sufficient condition. In stochastic models, the equilibrium states are defined by the equations $\dot{\mathbf{x}}_i(t) = \mathbf{n}_i(t)$, for $i = 1, 2, \dots, N$, where $\mathbf{n}_i(t)$ is the additive noise process. Note that in both the deterministic and stochastic models the transient due to the passive decay term is absent in the equilibrium state.

Equilibrium of a network depends on several other factors also besides the activation models. The most important among these is the update of the state change at each stage. The update could be synchronous, which means that the update of all the units is done at the same time. On the other hand, in an asynchronous update the change of state of any one unit changes the **overall** state of the network. Another factor is that the state update could be deterministic or stochastic. The equilibrium behaviour also depends on whether we are adopting a continuous time update or a discrete time update. A major issue in the study of equilibrium behaviour of a network is the speed at which the feedback signals from other units are received by the current unit.

2.3 Synaptic Dynamics Models

2.3.1 Learning

Synaptic dynamics is attributed to learning in a biological neural network. The synaptic weights are adjusted to learn the pattern information in the input samples. Typically, learning is a slow process, and the samples containing a pattern may have to be presented to the network several times before the pattern information is captured by the weights of the network. A large number of samples are normally needed for the network to learn the pattern implicit in the samples. Pattern information is distributed across all the weights, and it is difficult to relate the weights directly to the training samples. The only way to demonstrate the evidence of learning pattern information is that, given another sample from the same pattern source, the network would classify the new sample into the pattern class of the earlier trained samples. Another interesting feature of learning is that the pattern information is slowly acquired by the network from the training samples, and the training samples themselves are never stored in the network. That is why we say that we learn from examples, not store the examples themselves.

The adjustment of the synaptic weights is represented by a set of learning equations, which describe the synaptic dynamics of the network. The learning equation describing a synaptic dynamics model

is given as an expression for the first derivative of the synaptic weight w_{ij} connecting the unit j to the unit i . The set of equations for all the weights in the network determine the trajectory of the weight states in the weight space from a given initial weight state.

Learning laws refer to the specific manners in which the learning equations are implemented. Depending on the synaptic dynamics model and the manner of implementation, several learning laws have been proposed in the literature. The following are some of the requirements of the learning laws for effective implementation:

Requirements of learning laws:

- (a) The learning law should lead to convergence of weights.
- (b) The learning or training time for capturing the pattern information **from** samples should be as small as possible.
- (c) **An** on-line learning is preferable to an off-line learning. That is, the weights should be adjusted on presentation of each sample containing the pattern information.
- (d) Learning should use only the local information as far as possible. That is, the change in the weight on a connecting link between two units should depend on the states of these two units only. In such a case, it is possible to implement the learning law in parallel for all the weights, thus speeding up the learning process.
- (e) Learning should be able to capture complex nonlinear mapping between input-output pattern pairs, as well as between adjacent patterns in a temporal sequence of patterns.
- (f) Learning should be able to capture as many patterns as possible into the network. That is, the pattern information storage capacity should be as large as possible for a given network.

Categories of learning: Learning can be viewed **as** searching through the weight space in a systematic manner to determine the weight **vector** that leads to an optimum (minimum or maximum) value of an objective function. The search depends on the criterion used for learning. There are several criteria which include minimization of mean squared error, relative entropy, maximum likelihood, gradient descent, etc. [Hassoun, 1995]. There are several learning laws in use, and new laws are being proposed to suit a given application and architecture. Some of these **will** be discussed at appropriate places throughout the book, but there are some general categories that these laws fall into, based on the characteristics they are expected to possess. In the first place, the learning or weight adjustment could be *supervised* or *unsupervised*. In supervised learning the weight adjustment is determined based on the deviation

of the desired output from the actual output. Supervised learning may be used for *structural* learning or for *temporal* learning. Structural learning is concerned with capturing in the weights the relationship between the given input-output pattern pairs. Temporal learning is concerned with capturing in the weights the relationship between neighbouring patterns in a sequence of patterns.

Unsupervised learning discovers features in a given set of patterns, and organizes the patterns accordingly. There is no externally specified desired output in this case. Unsupervised learning uses mostly local information to update the weights. The local information consists of signal or activation values of the units at either end of the connection for which the weight update is being made.

Learning methods may be *off-line* or *on-line*. In an off-line learning all the given patterns are used together to determine the weights. On the other hand, in an on-line learning the information in each new pattern is incorporated into the network by incrementally adjusting the weights. Thus an on-line learning allows the neural network to update the information continuously. However, an off-line learning provides solutions better than an on-line learning since the information is extracted using all the training samples in the case of off-line learning.

In practice, the training patterns can be considered as samples of random processes. Accordingly, the activation and output states could also be considered as samples of random processes. Randomness in the output state could also result if the output function is implemented in a probabilistic manner rather than in a deterministic manner. These input, activation and output variables may also be viewed as fuzzy quantities instead of crisp quantities. Thus we can view the learning process as *deterministic* or *stochastic* or *fuzzy* or a *combination* of these characteristics.

Finally, in the implementation of the learning methods the variables may be *discrete* or *continuous*. Likewise the update of weight values may be in discrete steps or in continuous time. All these factors influence not only the convergence of weights, but also the ability of the network to learn from the training samples.

2.3.2 Distinction between Activation and Synaptic Dynamics Models

In order to appreciate the issues in evolving and implementing learning, it is necessary to clearly understand the distinction between the functions of the activation and synaptic dynamics models. This is discussed in this section. Both activation dynamics and synaptic dynamics models are expressed in terms of expressions for the first derivatives of the activation value of each unit and the strength of the connection between the i th unit and the j th unit, respectively. However, the purpose of invoking activation dynamics model is to

determine the equilibrium state that the network would reach for a given input. In this case, the input to the network is fixed throughout the dynamics. The dynamics model may have terms corresponding to passive decay, excitatory input (external and feedback) and inhibitory input (external and feedback). The passive decay term contributes to transients, which may eventually die, leaving only the steady state part. The transient part is due to the components representing the capacitance and resistance of the cell membrane. The steady state activation equations can be obtained by setting $\dot{\mathbf{x}}_i(t) = 0$, $i = 1, 2, \dots, N$. This results in a set of N coupled nonlinear equations, the solution of which will give the steady activation state as a function of time. This assumes that the transients decay faster than the signals coming from feedback, and the feedback signals do not produce any transients. It is in the movement of the steady activation state that we would be interested in the study of activation dynamics. Note that even a single unit network without feedback may have transient and steady parts, and the steady part in this case describes the stable state also. But in a network with feedback **from** other units, the steady activation states may eventually reach an equilibrium or a stable state, provided the conditions for the existence of stable states are satisfied by the parameters (especially the weights) in the activation dynamics model. **Thus**, in these cases we are not interested in the transient part of the solutions. We are only interested in the equilibrium stable states reached by the steady state activation values for a given input. The equilibrium states (\mathbf{x}) correspond to the locations of the minima of the Lyapunov energy function $V(\mathbf{x})$, and are given by $dV(\mathbf{x}(t))/dt = 0$, whereas the steady states are given by $\dot{\mathbf{x}}(t) = 0$, where $\mathbf{x}(t)$ is the activation vector with components $x_i(t)$, $i = 1, 2, \dots, N$. The equilibrium behaviour of the activation state of a neural network will be discussed in detail in Section 2.5.

The case of synaptic dynamics model is different from the activation dynamics model. The objective in synaptic dynamics is to capture the pattern information in the examples by incrementally adjusting the weights. Here the weights change due to input. If there is no input, the weights also do not change. Note that providing the same input at another instant again causes the weights to change, as it can be viewed as a sample given for further reinforcement of the weights. If the model contains a passive decay term in addition to the terms due to the varying external input, the network not only learns continuously, but also forgets what it had learnt initially. In discrete implementation, **i.e.**, determining the weight change at each discrete time step, suitable assumptions are made regarding the contribution of the initial weight state and also the contributions due to the samples given in the past. As an example, let us consider the following synaptic dynamics model, consisting of a passive decay term and a correlation term:

Activation and Synaptic Dynamics

$$\dot{w}_{ij}(t) = -w_{ij}(t) + f_i(x_i(t)) f_j(x_j(t)) \quad (2.22)$$

The solution to the equation is given by

$$w_{ij}(t) = w_{ij}(0) e^{-t} + e^{-t} \int_0^t f_i(x_i(\tau)) f_j(x_j(\tau)) e^{\tau} d\tau, \quad (2.23)$$

where $w_{ij}(0)$ is a constant initial value of the weight. The above solution shows that the weight accumulates the correlation of the output signals, i.e., $f_i(x_i(t)) f_j(x_j(t))$. Note that the activation values $x_i(t)$ and $x_j(t)$ depend on the external input given in the form of samples, continuous in time in this case. This is because the activation dynamics depends on the external input besides the network parameters like membrane capacitance and the connection topology like feedback. The activation values considered here are steady and stable, since it is assumed that the transients due to membrane parameters like capacitances have decayed down, and the steady activation state of the network has reached the stable state. This assumption is reasonable, since the adjustment of synaptic weights takes place at a much slower rate compared to the changes in the activation states.

The initial weight $w_{ij}(0)$ can be viewed as *a priori* knowledge. The term $w_{ij}(0) e^{-t}$ can be considered as a *forgetting* term. As $t \rightarrow \infty$, the contribution due to this term to the weight will be zero, i.e., the system would not remember the knowledge in the network at $t = 0$. The second term reflects *recency* effect. It shows the accumulation of the correlation term with time. There is an exponential weightage to this accumulation, which shows that recent correlation value is given more weight than the correlation values in the past. As mentioned above, these correlations depend on the input samples. The weights are expected to capture the patterns in the input samples as determined by the synaptic dynamics model.

Most of the time the learning laws ignore the passive decay term. Then the initial weight $w_{ij}(0)$ receives importance as can be seen below from the solution of the equation without the passive decay term. Let

$$\dot{w}_{ij}(t) = f_i(x_i(t)) f_j(x_j(t)) \quad (2.24)$$

The solution is given by

$$w_{ij}(t) = w_{ij}(0) + \int_0^t f_i(x_i(\tau)) f_j(x_j(\tau)) d\tau \quad (2.25)$$

Note that the recency effect also disappears, once the passive decay term is absent. That is, there is no exponential weighting on the accumulation of the correlation term.

In discrete-time implementation, the integral is replaced by

summation of the correlation terms, where each **term** is due to the application of one sample input pattern. That is

$$w_{ij}(t) = w_{ij}(0) + \sum_{\tau=1}^t f_i(x_i(\tau)) f_j(x_j(\tau)) \quad (2.26)$$

The initial weight at time $t = 0$ also influences the weight at any instant t . It does not decay with time. The change in the weight due to an input pattern at the time instant t is given by

$$\Delta w_{ij}(t) = w_{ij}(t) - w_{ij}(t-1) = f_i(x_i(t)) f_j(x_j(t)) \quad (2.27)$$

In summary, the distinction between the activation dynamics and synaptic dynamics models is highlighted by the following statements: For activation dynamics our interest is in the equilibrium states (\mathbf{x}) given by $\mathbf{V}(\mathbf{x}(t)) = \mathbf{0}$, which in turn uses the solution of equations for the steady activation states given by $\dot{\mathbf{x}}(t) = \mathbf{0}$, i.e., $\dot{x}_i(t) = 0$, for $i = 1, 2, \dots, N$. For synaptic dynamics, on the other hand, learning takes place when $\dot{w}_{ij}(t) \neq 0$.

2.4 Learning Methods

There **are** several methods of learning. For the purpose of discussion, the learning methods are organized into different groups. Table 2.3 gives a summary of the learning methods discussed in this section. The table also lists the categories of learning discussed in Section 2.3.1.

Table 2.3 Summary of Learning Methods

1. Categories of learning

- Supervised, reinforcement and unsupervised
- Off-line and on-line
- Deterministic, stochastic and **fuzzy**
- Discrete and continuous
- Criteria for learning

2. Hebbian learning

- Basic Hebbian learning
- Differential Hebbian learning
- Stochastic versions

3. Competitive learning—learning without a teacher

- Linear competitive learning
- Differential competitive learning
- Linear differential competitive learning
- Stochastic versions

Table 2.3 Summary of Learning Methods (Cont.)

4. Error correction learning—learning with a teacher

- Perceptron learning
- Delta learning
- LMS learning

5. Reinforcement learning—learning with a critic

- Fixed credit assignment
- Probabilistic credit assignment
- Temporal credit assignment

6. Stochastic learning

- In multilayer perceptron
- In Boltzmann machine

7. Other learning methods

- Sparse coding
- Min-max learning
- Principal component learning
- Drive-reinforcement learning

Details of implementation of some learning methods will be discussed at appropriate contexts in the later chapters.

2.4.1 Hebbian Learning

The basis for the class of *Hebbian learning* is that the changes in the synaptic strength is proportional to the correlation between the firing of the post- and pre-synaptic neurons [Hebb, 1949]. Figure 2.4

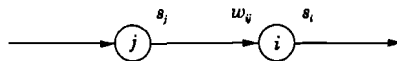


Figure 2.4 Topology for Hebbian learning, where i and j represent processing units.

shows the topology for Hebbian learning [Simpson, 1990]. The synaptic dynamics equation is given by a decay term ($-w_{ij}(t)$) and a correlation term ($s_i s_j$) as

$$\dot{w}_{ij}(t) = -w_{ij}(t) + s_i s_j \quad (2.28)$$

where $s_i s_j$ is the product of the post-synaptic and pre-synaptic neuronal variables for the i th unit. These variables could be activation values ($s_i s_j = x_i(t) x_j(t)$), or an activation value and an external input ($s_i s_j = x_i(t) a_j(t)$), or an output signal and an external input ($s_i s_j = f_i(x_i(t)) a_j(t)$), or output signals from two units ($s_i s_j = f_i(x_i(t)) f_j(x_j(t))$), or some other parameters related to the post-synaptic and pre-synaptic activity. If s_i and s_j represent variables which are deviations from their respective mean values (\bar{s}_i, \bar{s}_j), then the resulting

correlation term $(s_i - \bar{s}_i)(s_j - \bar{s}_j)$ is called covariance correlation term. **Thought** our discussion we will assume that $s_i = s_i(t) = f_i(x_i(t))$ and $s_j = s_j(t) = f_j(x_j(t))$, unless otherwise specified.

The solution of Eq. (2.28) is given by

$$w_{ij}(t) = w_{ij}(0) e^{-t} + e^{-t} \int_0^t s_i(\tau) s_j(\tau) e^{\tau} d\tau \quad (2.29)$$

where $w_{ij}(0)$ is the initial value of the weight at time $t = 0$. **The first term** shows that the past knowledge will decay exponentially to zero, which is equivalent to forgetting. The second term corresponds to correlation encoding with fading memory due to the exponential weight factor in the integral term. The Hebbian learning thus accumulates the correlation **terms**, giving more weightage to the recent terms.

Some variations of the Hebbian learning are as follows [Simpson, 1990]:

$$\dot{w}_{ij}(t) = -w_{ij}(t) + (s_i - \bar{s}_i)(s_j - \bar{s}_j) \quad [\text{Sejnowski, 1977}] \quad (2.30)$$

$$\dot{w}_{ij}(t) = -w_{ij}(t) + (s_i - \bar{s}_i) s_j \quad [\text{Sutton and Barto, 1981}] \quad (2.31)$$

$$\dot{w}_{ij}(t) = -w_{ij}(t) + w_{ij}(t) s_i s_j \quad [\text{Cheung and Omidvar, 1988}] \quad (2.32)$$

The stochastic version of the Hebbian learning given in Eq. (2.28) is approximated to the following stochastic differential equation:

$$\dot{w}_{ij}(t) = -w_{ij}(t) + s_i s_j + n_{ij}(t) \quad (2.33)$$

where $\{n_{ij}(t)\}$ is assumed to be a zero-mean Gaussian white noise process [Papoulis, 1991, independent of the signal process $-w_{ij}(t) + s_i s_j$.

Synaptic equilibrium in the deterministic case is given by the steady state condition:

$$\dot{w}_{ij}(t) = 0 \quad (2.34)$$

That is, there is no further change of weights. In the stochastic case, the weights reach **stochastic** equilibrium when the weight changes are only due to the noise component. That is

$$\dot{w}_{ij}(t) = n_{ij}(t) \quad (2.35)$$

In these cases the synaptic processes fluctuate randomly at stochastic equilibrium as the weights approach the asymptotic values. Note that the stochastic equilibrium corresponds to the deterministic equilibrium on the average. That is the average or expectation $\mathbf{E}[\dot{w}_{ij}(t)] = 0$, for all t **after** w_{ij} reaches equilibrium.

2.4.2 Differential Hebbian Learning

The deterministic differential Hebbian learning is described by

$$\dot{w}_{ij}(t) = -w_{ij}(t) + s_i s_j + \delta \delta. \quad (2.36)$$

or in a simpler classical version, it is given by [Klopf, 1986]

$$\dot{w}_{ij}(t) = -w_{ij}(t) + \dot{s}_i \dot{s}_j \quad (2.37)$$

That is, the differential equation consists of a passive decay term $-w_{ij}(t)$ and a correlation term $\dot{s}_i \dot{s}_j$, which is a result of the changes in the post- and pre-synaptic **neuronal activations**.

The stochastic versions of these laws are approximated by adding a noise term to the right hand side of these differential Hebbian learning equations.

2.4.3 Competitive Learning

Learning laws which modulate the difference between the output signal and the synaptic weight belong to the category of competitive learning. The general form of competitive learning is given by the following expression for the synaptic dynamics [Grossberg, 1969]:

$$\dot{w}_{ij}(t) = s_i [s_j - w_{ij}(t)] \quad (2.38)$$

where, $s_i = f_i(x_i(t))$ is the output signal of the unit i , and $s_j = f_j(x_j(t))$ is the output signal of the unit j . This is also called the deterministic competitive learning law. It can be written as

$$\dot{w}_{ij}(t) = -s_i w_{ij}(t) + s_i s_j \quad (2.39)$$

The above expression is similar to the deterministic Hebbian learning (see Eq. (2.28)), except that the forgetting term $(-s_i w_{ij}(t))$, is nonlinear in this case, whereas it was linear in the Hebbian case. Here learning or adjustment of weights takes place only when there is a nonzero post-synaptic signal (s_i). If $s_i = 0$, then the synaptic weights do not change. It is also interesting to note that, unlike in the Hebbian case, in the competitive learning case the system does not forget the past learning when the post-synaptic signal is zero. In the Hebbian case, for $s_i = 0$, $\dot{w}_{ij}(t) = -w_{ij}(t)$, which results in forgetting the knowledge already acquired.

The competitive learning works in a situation where an external input is presented to an input layer of units and these units feed signals to each of the units in the output layer. The signals from the units in the output layer compete with each other, leaving eventually one of the units (say i) as the winner. The weights leading to this unit are adjusted according to the learning law. This is also called the 'winner take-all' situation, since only one unit in the output layer will have a nonzero output eventually. The corresponding weights w_{ij}

from **all** the input units (j) are adjusted to match the input vector. While the **Hebbian** learning is generally distributed, **i.e.**, all the weights are adjusted for every input pattern, the competitive learning is not distributed. In fact the input vectors leading to the same **winning** unit in the competitive layer will produce a weight vector for that unit which is **an average of all** the corresponding input **vectors**.

If the **input layer** units are linear, **i.e.**, $s_j = x_j$, then the resulting learning is called linear competitive learning, and is given by

$$\dot{w}_{ij}(t) = s_i [x_j - w_{ij}(t)] \quad (2.40)$$

The stochastic versions of the competitive learning are approximated to the following stochastic differential equations [Kosko, 1992]:

Random competitive learning

$$\dot{w}_{ij}(t) = s_i [s_j - w_{ij}(t)] + n_{ij}(t) \quad (2.41)$$

Random linear competitive learning

$$\dot{w}_{ij}(t) = s_i [x_j - w_{ij}(t)] + n_{ij}(t) \quad (2.42)$$

where $\{n_{ij}(t)\}$ is assumed **to** be a zero-mean Gaussian white noise process, independent of the signal process.

If the input space is partitioned into K different nonoverlapping subspaces, D_1, D_2, \dots, D_K , **i.e.**,

$$D_i \cap D_j = 0, \quad \text{for } i \neq j, \quad (2.43)$$

then a reinforcement function for an input pattern \mathbf{a} is **defined as**

$$a = 1, \quad \text{if } \mathbf{a} \in D_i \quad (2.44)$$

$$= -1, \quad \text{if } \mathbf{a} \notin D_i \quad (2.45)$$

Using **this** reinforcement function, the following supervised learning laws are defined:

Random supervised competitive learning

$$\dot{w}_{ij}(t) = r_i(\mathbf{a}) s_i [s_j - w_{ij}(t)] + n_{ij}(t) \quad (2.46)$$

Random supervised linear competitive learning

$$\dot{w}_{ij}(t) = r_i(\mathbf{a}) s_i [x_j - w_{ij}(t)] + n_{ij}(t) \quad (2.47)$$

2.4.4 Differential Competitive Learning

Differential competition means that learning takes place only if there is a change in the post-synaptic neuronal activation. The deterministic differential competitive learning is described by

$$\dot{w}_{ij}(t) = \dot{s}_i [s_j - w_{ij}(t)] \quad (2.48)$$

This combines the competitive learning and differential Hebbian learning.

Linear differential competitive learning law is described by

$$\dot{w}_{ij}(t) = \dot{s}_i [x_j - w_{ij}(t)] \quad (2.49)$$

The stochastic versions of the above learning equations are obtained by adding a noise term to the **right** hand side of these differential competitive learning equations.

Random differential competitive *learning*

$$\dot{w}_{ij}(t) = \dot{s}_i [s_j - w_{ij}(t)] + n_{ij}(t) \quad (2.50)$$

Random linear differential competitive learning

$$\dot{w}_{ij}(t) = \dot{s}_i [x_j - w_{ij}(t)] + n_{ij}(t) \quad (2.51)$$

2.4.5 Error Correction Learning

Error correction learning uses the error between the desired output and the actual output for a given input pattern to adjust the weights. These are supervised learning laws, as they depend on the availability of the desired output for a given input. Let (\mathbf{a}, \mathbf{b}) be a sample of the input-output pair of vectors for which a network has to be designed by adjusting its weights so as to obtain minimum error between the desired and actual outputs. Let E be the error function and $\mathcal{E}[E]$ be the expected value of the error function for all the training data consisting of several input-output pairs. Since the joint probability density function of the pairs of random input-output vectors is not known, it is not possible to obtain the desired expectation $\mathcal{E}[E]$. Stochastic approximation estimates the expectation using the **observed** random input-output pairs of vectors (\mathbf{a}, \mathbf{b}) . These estimates are used in a discrete approximation algorithm like a stochastic gradient descent algorithm to adjust the weights of the network. This type of adjustment may not always result in the optimum set of weights, in the sense of minimizing $\mathcal{E}[E]$. It may result in some local minima of the expected error function. Stochastic gradient descent algorithms are discussed in detail in Chapter 4.

Most error correction learning methods use the instantaneous error $(\mathbf{b} - \mathbf{b}')$ to adjust the weights, where \mathbf{b}' is the actual output vector of the network for the input vector \mathbf{a} .

Rosenblatt's perceptron learning uses the instantaneous misclassification error to adjust the weights. It is given by

$$\dot{w}_{ij}(t) = \eta (b_i - s_i) a_j \quad (2.52)$$

where b_i is the desired output from the i th output unit for an input pattern $\mathbf{a} = (a_1, a_2, \dots, a_M)$, a_j is the j th component of the input

pattern to the i th unit, and η is a small positive learning constant. Here s_i is the actual output of the i th unit given by $s_i = \text{sgn}(\sum_j w_{ij} a_j)$. **Perceptron** learning for a bipolar (± 1) output unit produces an error value $b_i - s_i = \pm 2$. Note that $b_i - s_i = 0$ when there is no error. Thus the discrete perceptron learning adjusts weights only when there is **misclassification**.

The continuous perceptron learning uses a monotonically increasing nonlinear output function $f(\cdot)$ for each unit. The weights are adjusted so as to minimize the squared error between the desired and actual output at every instant. The corresponding learning equation is given by

$$\dot{w}_{ij}(t) = \eta (b_i - s_i) \dot{f}_i(x_i) a_j \quad (2.53)$$

where $s_i = f_i(x_i)$ and $x_i = \sum_{j=1}^M w_{ij} a_j$. Continuous perceptron learning is also called **delta learning**, and it can be generalized for a network consisting of several layers of feedforward units. The resulting learning method is called the **generalized delta rule**.

Widrow's least mean squared error (**LMS**) algorithm uses the instantaneous squared error between the desired and the actual output of a unit, assuming a linear output function for each unit, i.e., $f(x) = x$. The corresponding learning equation is given by

$$\dot{w}_{ij}(t) = \eta (b_i - x_i) a_j \quad (2.54)$$

Note that in all of the above error correction learning methods, we have assumed that the passive decay term to be zero. These methods require that the learning constant (η) is made as small as possible, and that the training samples are applied several times to the network until the weights lead to a minimum error. As stated earlier, the resulting weights may not correspond to a global minimum of the expected error function.

2.4.6 Reinforcement Learning

In error correction learning the desired output for a given input is known, and therefore the learning is based on the error between the desired output and the actual output. This supervised learning is called **learning with teacher**. On the other hand, there are many situations where the desired output for a given input is not known. Only the binary result that the output is right or wrong may be available. This output is called **reinforcement** signal. This signal only evaluates the output. The learning based on this evaluative signal is called **reinforcement learning** [Sutton, 1992]. Since this is evaluative and not instructive, it is also called **learning with critic** as opposed to learning with teacher in the supervised learning.

The reinforcement learning can be viewed as a credit assignment problem [Sutton, 1984]. Depending on the reinforcement signal, the credit or blame for the overall outcome is assigned to different units or weights of the network. This is called structural credit *assignment*, since it assigns credit to the internal structures of the system, whose actions generated the outcome. On the other hand, if the credit is assigned to the outcomes of series of actions based on the reinforcement signal received for the overall outcome, it is called temporal credit assignment. This happens for example in a game like chess, where the reinforcement signal (win or lose) is received only after a sequence of moves. The assignment of credit or blame in this case is to each of the moves in the sequence that led to the final outcome. The combined temporal and structural credit assignment problem is also relevant in situations involving temporally extended distributed learning systems [Williams, 1988; Williams, 1992].

The reinforcement signal can also be viewed as a feedback from the environment which provides input to the network and observes the output of the network. There are three types of reinforcement learning depending on the nature of the environment. If the reinforcement signal from the environment is the same for a given input-output pair, and if it does not change with time, it is called a fixed credit assignment problem. This is like supervised learning in classification problems. On the other hand, if the given input-output pair determines only the probability of positive reinforcement, then the network can be viewed as operating in a stochastic environment. In such a case it is called probabilistic credit assignment. Here the probabilities are assumed stationary. In both the fixed and probabilistic credit assignments the input patterns are chosen randomly and independently by the environment. That is, the input pattern does not depend on the past inputs or outputs. But in the general case where the environment itself is changing, then both the reinforcement signals and the input patterns may depend on the past history of the network outputs. In such cases temporal credit assignment is more appropriate.

The associative reward-penalty reinforcement learning by Barto and Anandan is applicable for a processing unit with probabilistic update (see Chapter 5), and is given by [Barto and Anandan, 1985]

$$\begin{aligned}\Delta w_{ij} &= \eta^+ (s_i - \langle s_i \rangle) (1 - \langle s_i \rangle^2) a_j \\ &= \eta^- (-s_i - \langle s_i \rangle) (1 - \langle s_i \rangle^2) a_j\end{aligned}\tag{2.55}$$

where $\langle s_i \rangle$ is the expected value of the output s_i for the i th unit, η^+ is the learning rate parameter for positive reinforcement (reward) and η^- is the learning rate parameter for negative reinforcement (penalty). Typically $\eta^+ \gg \eta^- > 0$. The term $(1 - \langle s_i \rangle^2)$ is a derivative term, which can be ignored without **affecting** the general behaviour

of this learning rule [Hassoun, 1995, p. 891. The above learning rule is **applicable** for **units** in a single layer only. In a multilayer network with hidden layer units, the error from the output layer units is propagated back to adjust the weights leading to the hidden layer units (see Chapter 4).

2.4.7 Stochastic Learning

Stochastic learning involves adjustment of weights of a neural network in a probabilistic manner [Ackley et al, 1985]. The adjustment uses a probability law, which in turn depends on the error. The error for a network is a positive scalar defined in terms of the external input, desired output and the weights connecting the units. In the learning process, a random weight change is made and the resulting change in the error is determined. If the resulting error is lower, then accept the random weight change. If the resulting error is not lower, then accept the random weight change with a predecided probability distribution. The acceptance of random change of weights despite increase in the **error** from the network allows the network to escape local minima in the search for the global minimum of the error surface.

Boltzmann learning uses stochastic learning along with simulated annealing to determine the weights of a feedback network to store a given set of patterns [Ackley et al, 1985; Szu, 1986]. Stochastic learning is also used in determining the optimum weights of a multilayer feedforward neural network to arrive at a set of weight values corresponding to the global minimum of the error surface, since stochastic learning helps to **overcome** the local minima problem [Wasserman, 1988]. However, all stochastic learning methods are slow in convergence and hence are time consuming.

2.4.8 Other Learning Methods

Sparse encoding: If $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 0, 1, 2, \dots, L - 1$ are the given set of input-output binary vector pairs, then the sparse encoding learning is given by the following logical OR and AND operations:

$$w_{ij}(l+1) = (\mathbf{a}_{ij} \text{ AND } \mathbf{b}_{li}) \text{ OR } w_{ij}(l) \quad (2.56)$$

with $w_{ij}(0) = 0$. This type of learning is used to store information in an associative memory [Steinbuch and Piske, 1963; Simpson, 1992].

Min-Max learning: This learning is used in the special case of providing connections to each processing unit from an input unit, one connection weight (v_{ij}) corresponds to the minimum of the inputs and the other connection weight (w_{ij}) corresponds to the maximum of the

inputs. The weight adjustments are made as follows [Simpson, 1991]:

$$v_{ij}(l+1) = \min(a_{ij}, v_{ij}(l)) \quad (2.57)$$

and

$$w_{ij}(l+1) = \max(a_{ij}, w_{ij}(l)) \quad (2.58)$$

with $v_{ij}(0) = w_{ij}(0) = 0$, where a_{ij} is the j th component of the input vector \mathbf{a}_i . The minimum and maximum values are treated as bounds for a given membership function, providing a mechanism to adjust and analyze classes being formed in a neural network [Simpson, 1992].

Principal component learning: Principal components of a set of input vectors are a minimal set of orthogonal vectors that span the space of the covariance matrix of the input data. Once these basis vectors are found, it is possible to express any vector as a linear combination of these basis vectors. Oja's principal component learning is given by [Oja, 1982]:

$$w_i(m+1) = w_i(m) + \eta y(m) (a_{mi} - y(m) w_i(m)) \quad (2.59)$$

where a_{mi} is the i th component of the given input vector, $y(m)$ is the actual output for the m th input vector. This learning extracts only the first principal component, and the output function of the unit is assumed to be linear. Note that the index m is used to indicate that a given input vector can be presented several times, even though there may be only a fixed number of vectors for training. The details of principal component learning are **discussed** in Chapter 6.

Drive-reinforcement learning: This law is given by

$$w_{ij}(t+1) = w_{ij}(t) + \Delta f_i(x_i(t)) \sum_{\tau=1}^t \alpha(t-\tau) |w_{ij}(t-\tau)| \Delta f_j(x_j(t-\tau)) \quad (2.60)$$

where $\alpha(t-\tau)$ is a decreasing function of time and Δf indicates the change in the output values of the units from the previous instant. The change in the weight uses a weighted sum of the changes in the past input values, multiplied by the current change in the output. The pre-synaptic changes $\Delta f_j(x_j(t-\tau))$, $j = 1, 2, \dots, t$ are referred to as drives, and the post-synaptic change $\Delta f_i(x_i(t))$ as the reinforcement, and hence the name drive-reinforcement learning. This learning law was proposed for control applications due to its ability to optimize temporal actions [Klopf, 1986].

2.4.9 Learning Functions

Learning laws are merely implementation methods for synaptic dynamics models. Typically, a synaptic dynamics model is described

in **terms** of expressions for the first derivative of the weights. **They** are called learning equations. One general way of expressing the learning feature in neural network studies is the following:

The change in the weight is proportional to the product of the input $\mathbf{a}(t)$ and a learning function $g(\cdot)$, and it is given by [Zurada, 1992]

$$\dot{\mathbf{w}}_i(t) = \eta g(\mathbf{w}_i(t), \mathbf{a}(t), b_i(t)) \mathbf{a}(t) \quad (2.61)$$

where,

η is the learning rate parameter

$\mathbf{w}_i = (w_{i1}, w_{i2}, \dots, w_{iM})^T$ is the weight vector for the i th unit with components w_{ij}

w_{ij} is the weight on the link connecting the j th input unit to the i th processing unit

$\mathbf{a} = (a_1, a_2, \dots, a_M)^T$ is the input vector with components a_j , $j = 1, 2, \dots, M$

$\mathbf{b} = (b_1, b_2, \dots, b_N)^T$ is the desired output vector with components b_i , $i = 1, 2, \dots, N$

Input units are assumed linear. Hence $\mathbf{a} = \mathbf{x}$ (activation) = \mathbf{s} (output). Output units are in general nonlinear. Hence $\mathbf{s}_i = f(\mathbf{w}_i^T \mathbf{a})$.

The function $g(\cdot)$ may be viewed as a *learning* function that depends on the type of learning. The increment in the weight vector in unit time interval is given by (see Eq. (2.27))

$$\Delta \mathbf{w}_i(t) = \eta g(\mathbf{w}_i(t), \mathbf{a}(t), b_i(t)) \mathbf{a}(t) \quad (2.62)$$

so that the weight at the time instant $(t + 1)$ in terms of the weight at the time instant t is given by

$$\mathbf{w}_i(t + 1) = \mathbf{w}_i(t) + \Delta \mathbf{w}_i(t) \quad (2.63)$$

There are different methods for implementing the learning feature of a neural network, leading to several learning laws. The different basic learning laws described in Section 1.6 differ in the expression for the learning function. *All* these learning laws use only local information for adjusting the weight of the connection between two units. The expression for the learning function for each of the basic learning laws is given below. The corresponding expression for the learning law is also given.

The learning function for Hebb's law is given by

$$g(\cdot) = f(\mathbf{w}_i^T \mathbf{a}) \quad (2.64)$$

where $f(\cdot)$ is the output function. Therefore the change in the weight is given by

$$\Delta \mathbf{w}_i = \eta f(\mathbf{w}_i^T \mathbf{a}) \mathbf{a} = \eta \mathbf{s}_i \mathbf{a} \quad (2.65)$$

where \mathbf{s}_i is the output signal of the i th unit.

In perceptron learning the learning function assumes the form

$$g(.) = b_i - s_i = b_i - \text{sgn}(\mathbf{w}_i^T \mathbf{a}) \quad (2.66)$$

which is the difference between the desired output and the actual output at the i th unit. The change in the weight is given by

$$\Delta \mathbf{w}_i = \eta [b_i - \text{sgn}(\mathbf{w}_i^T \mathbf{a})] \mathbf{a} \quad (2.67)$$

The learning function in delta learning law is given by

$$g(.) = [b_i - f(\mathbf{w}_i^T \mathbf{a})] \dot{f}(\mathbf{w}_i^T \mathbf{a}) \quad (2.68)$$

Therefore the resulting weight change is given by

$$\Delta \mathbf{w}_i = \eta [b_i - f(\mathbf{w}_i^T \mathbf{a})] \dot{f}(\mathbf{w}_i^T \mathbf{a}) \mathbf{a} \quad (2.69)$$

The learning function in **Widrow** and Hoff learning law is given by

$$g(.) = [b_i - \mathbf{w}_i^T \mathbf{a}] = [b_i - x_i] \quad (2.70)$$

Therefore the change in the weight is given by

$$\Delta \mathbf{w}_i = \eta [b_i - \mathbf{w}_i^T \mathbf{a}] \mathbf{a} \quad (2.71)$$

In correlation learning the learning function is given by

$$g(.) = b_i \quad (2.72)$$

and the weight change is given by

$$\Delta \mathbf{w}_i = \eta b_i \mathbf{a} \quad (2.73)$$

2.5 Stability and Convergence

Stability refers to the equilibrium behaviour of the activation state of a neural network, whereas convergence refers to the adjustment behaviour of the weights during learning, which will eventually lead to minimization of error between the desired and actual outputs. Thus convergence is typically associated with supervised learning, although it is relevant in all cases of learning, both supervised and unsupervised. The objective of any learning law is that it should eventually lead to a set of weights which will capture the pattern information in the training set data.

In this section we will discuss the global behaviour of artificial neural **networks** whose activation dynamics is described by the following set of equations [Cohen and Grossberg, 1983]:

$$\dot{x}_i = -\alpha_i(x_i) \left[b_i(x_i) - \sum_{k=1}^N c_{ik} d_k(x_k) \right], \quad i = 1, 2, \dots, N \quad (2.74)$$

where $\mathbf{x}_i = \mathbf{x}_i(t)$ and the **coefficients** $[c_{ik}]$ form a symmetric matrix.

These equations represent a class of N-dimensional competitive **dynamical** systems. **All** the previous activation models including the general shunting activation model form special cases of this system. In **general**, the activation state of the network starts from an initial state and follows a trajectory dictated by the dynamics of the equations. A network will be useful only if a trajectory leads eventually to an equilibrium state at which point there is no further change in the state. Such a state is also called a stable state, when a small perturbation of the state settles to the same state. Different initial states may follow different trajectories, all of which should terminate at some equilibrium states. There may be **several** trajectories that may **terminate** at the same equilibrium state.

The existence of such equilibrium states enables global pattern formation possible in a network. That is, an input pattern corresponding to a starting state will eventually lead to one of the global patterns, which can be interpreted as storage of the input pattern in long term memory. The global pattern thus formed will only change if there is a different external input. In some cases the network parameters such as weights may slowly change due to learning or self-organization. **If** the global pattern formation still occurs for **any** choice of these parameters, then the resulting pattern is said to be **absolutely** stable or **globally** stable.

Under certain conditions, which will be discussed later, the set of equations (2.74) describing activation dynamics do exhibit stable states which are also called **fixed point** equilibrium states. Such a network then can form global patterns at those states, and hence can be used for pattern storage. One of the conditions is that the weights $\{c_{ik}\}$ should be symmetric ($c_{ik} = c_{ki}$). If the weights are not exactly symmetric, then the network may exhibit periodic oscillations of states in certain regions of the state space. These oscillatory regions are also stable, and hence can be used for pattern storage. Oscillatory stable states may also arise when there is some delay in the feedback of the outputs from other processing units to the current unit, even though the weights are exactly symmetric.

For some other conditions, the network may display chaotic changes of states in the regions of equilibrium, also called **basins of attraction**. Such a network is said to exhibit **chaotic** stability. Thus pattern storage is possible in any network that exhibits either fixed point stability or oscillatory stability or chaotic stability. However, it is **difficult** to analyze and design a network suitable for the oscillatory and chaotic types of stabilities [Cohen and Grossberg, 1983; Hertz, 1995].

A general network is more likely to exhibit random chaotic changes of states throughout due to nonlinearly coupled set of equations with delayed feedback. One has to carefully choose the parameters of the activation dynamics models for ensuring stable points. In general, it is difficult to know whether a network will have

stable points, and if so, how many. It is even more difficult to determine the behaviour of the network near the stable points to examine the nature of stability. However, in a few cases it is possible to predict the global pattern behaviour, if it is possible to show the existence of an energy function called Lyapunov function [Amari, 1977]. It is a scalar function of the parameters of the network, denoted by $V(\mathbf{x})$, where x is the activation state vector of the network. $V(\mathbf{x})$ is said to be a Lyapunov function if $\dot{V}(\mathbf{x}) \leq 0$ for all x . It is sufficient if we can find a Lyapunov function for a network in order to demonstrate the existence of stable equilibrium states. It is not a necessary condition, as the network may still have stable points, even though a Lyapunov function could not be found. The existence of Lyapunov function makes it easy to analyze the stability of the network.

If the Lyapunov **function** is interpreted as an energy function, then the condition that $\dot{V}(\mathbf{x}) \leq 0$ means that any change in the energy due to change in the state of the network results in lowering the total energy. In other words, any change of the state of the network results in the trajectory of the state sliding along the energy surface in the state space towards lower energy. Eventually the **trajectory** leads to a state from where there is no further decrease in the energy due to changes in the state. Such a state corresponds to the energy minimum, at which $V(\mathbf{x}) = 0$. Normally there will be many states at which $V(\mathbf{x}) = 0$. All such states correspond to equilibrium points or stable states. All trajectories in the state space will eventually lead to one of these stable states.

In the following, three general theorems are given that describe the stability of a set of nonlinear **dynamical** systems. The first theorem, the Cohen-Grossberg theorem, is useful to show the stability of fixed weight autoassociative networks. The second theorem, the Cohen-Grossberg-Kosko theorem, is useful to show the stability of adaptive autoassociative networks. The third theorem, the adaptive bidirectional-associative memory theorem, is useful to show the stability of adaptive heteroassociative networks.

Cohen-Grossberg theorem: For a system of equations given by

$$\dot{x}_i = -a_i(x_i) \left[b_i(x_i) - \sum_{k=1}^N c_{ik} d_k(x_k) \right], \quad i = 1, 2, \dots, N \quad (2.75)$$

a global Lyapunov function is given by

$$V(\mathbf{x}) = \sum_{i=1}^N \int_0^{x_i} b_i(\xi_i) d_i(\xi_i) d\xi_i - \frac{1}{2} \sum_{i,k=1}^N c_{ik} d_i(x_i) d_k(x_k) \quad (2.76)$$

Since

$$\dot{V} = \frac{dV(\mathbf{x}(t))}{dt} = \sum_{i=1}^N \frac{\partial V}{\partial x_i} \frac{\partial x_i}{\partial t} \quad (2.77)$$

$$= \sum_{i=1}^N b_i(x_i) \dot{d}_i(x_i) \dot{x}_i - \sum_{i,k}^N c_{ik} \dot{d}_i(x_i) d_k(x_k) \dot{x}_i \quad (2.78)$$

$$= \sum_{i=1}^N \dot{x}_i \dot{d}_i(x_i) \left[b_i(x_i) - \sum_{k=1}^N c_{ik} d_k(x_k) \right] \quad (2.79)$$

$$= - \sum_{i=1}^N a_i(x_i) \dot{d}_i(x_i) \left[b_i(x_i) - \sum_{k=1}^N c_{ik} d_k(x_k) \right]^2 \quad (2.80)$$

we have

$$\dot{V} \leq 0, \quad (2.81)$$

if $a_i(x_i) \geq 0$, (i.e., $a_i(x_i)$ is nonnegative), $\dot{d}_i(x_i) \geq 0$ (i.e., $d_i(x_i)$ is monotonically nondecreasing function), c_{ik} are constant and do not change with time, and $[c_{ik}]$ is symmetric. This last property was used to obtain the simplified expression for the derivative of the second term of $V(\mathbf{x})$ in Eq. (2.78).

Note that the function $b_i(x_i)$ could be arbitrary, except that it should ensure the integrability of the **first term** in $V(\mathbf{x})$. Thus $V(\mathbf{x})$ is a global Lyapunov function, provided **these conditions are** satisfied.

Cohen-Grossberg-Kosko theorem: For a **dynamical** system where both the activation state and the synaptic weights are changing simultaneously, the equations describing the dynamics may be expressed **as** follows [Kosko, 19881:

$$\dot{x}_i = - a_i(x_i) \left[b_i(x_i) - \sum_{k=1}^N c_{ik} d_k(x_k) \right], \quad i = 1, 2, \dots, N \quad (2.82)$$

$$\dot{c}_{ik} = - c_{ik} + d_i(x_i) d_k(x_k) \quad (2.83)$$

where $[c_{ik}]$ is assumed to be a symmetric matrix. For such a system the following $V(\mathbf{x})$ is a Lyapunov function.

$$V(\mathbf{x}) = \sum_{i=1}^N \int_0^{x_i} b_i(\xi_i) \dot{d}_i(\xi_i) d\xi_i - \frac{1}{2} \sum_{i,k} c_{ik} d_i(x_i) d_k(x_k) + \frac{1}{4} \sum_{i,k} c_{ik}^2 \quad (2.84)$$

Adaptive bidirectional associative memory theorem: The system of equations describing the activation and synaptic dynamics of a neural network consisting of two layers of processing units, a unit in each layer feeding its output to all the units in the other layer, is given as follows [Kosko, 19881:

$$\dot{x}_i = -a_i(x_i) \left[b_i(x_i) - \sum_j c_{ij} d_j(y_j) \right] \quad (2.85)$$

$$\dot{y}_j = -a_j(y_j) \left[b_j(y_j) - \sum_i c_{ji} d_i(x_i) \right] \quad (2.86)$$

$$\dot{c}_{ij} = -c_{ij} + d_i(x_i) d_j(y_j) \quad (2.87)$$

The following is a Lyapunov function for the above system:

$$\begin{aligned} V(\mathbf{x}, \mathbf{y}) = & \sum_i \int_0^{x_i} b_i(\xi_i) d_i(\xi_i) d\xi_i + \sum_j \int_0^{y_j} b_j(\xi_j) d_j(\xi_j) d\xi_j \\ & - \sum_{i,j} c_{ij} d_i(x_i) d_j(y_j) + \frac{1}{2} \sum_{i,j} c_{ij}^2 \end{aligned} \quad (2.88)$$

2.6 Recall in Neural Networks

During learning, the weights are adjusted to store the information in a given pattern or a **pattern** pair. However, during performance, the weight changes are suppressed, and the input to the network determines the output activation x_i or the signal value s_i . This operation is called recall of the stored information. The recall techniques are different for feedforward and feedback networks.

The simplest feedforward network uses the following equation to compute the output signal from the input data vector \mathbf{a} .

$$s_i = f_i(\mathbf{w}_i^T \mathbf{a}) \quad (2.89)$$

where $f_i(\cdot)$ is the output function of the i th unit.

A recall equation for a network with feedback connections is given by the following additive model for activation dynamics:

$$x_i(t+1) = -(1-a)x_i(t) + \beta \sum_{j=1}^N w_{ij} f_j(x_j(t)) + a_i \quad (2.90)$$

where $x_i(t+1)$ is the activation of the i th unit in a single layer feedback network at time $(t+1)$. The function $f_j(\cdot)$ is the nonlinear output function of the j th unit, a (< 1) is a positive constant that regulates the amount of decay the unit has during the update interval, β is a positive constant that regulates the amount of feedback the other units provide to the i th unit, and a_i is the external input to the i th unit. This equation is same as the Eq. (2.10) except for a change of a few symbols. In general, stability is the main issue in feedback networks. If the network reaches a stable state in a finite number of iterations, then the resulting output signal vector represents the nearest neighbour stored pattern of the system for the approximate input pattern \mathbf{a} .

Cohen and Grossberg (1983) have shown that for a wide class of neural networks with certain constraints, the network with fixed weights reaches a stable state in a finite period of time from any initial condition. Later Kosko showed that a neural network could learn and recall at the same time, and yet remains stable [Kosko, 1988].

The response of a network due to recall could be the nearest neighbour or interpolative. In the nearest neighbour case, the stored pattern closest to the input pattern is recalled. This typically happens in the feedforward pattern classification or in the feedback pattern matching networks. In the interpolative case, the recalled pattern is a combination of the outputs corresponding to the input training patterns nearest to the given input test pattern. This happens in the feedforward pattern mapping networks,

2.7 Summary

In this chapter we have considered the issues in developing activation and synaptic dynamics models for artificial neural networks. The activation models of both additive and multiplication types are discussed in detail. The multiplicative or shunting type models are developed to limit the operating range of the activation value of a neuron. The synaptic dynamics model equations form the basis for learning in neural networks. Several learning methods are presented to indicate the variety of methods developed for different applications. The activation and synaptic dynamics models are useful only if global pattern formation is possible with these models. The global pattern formation is linked with stability and convergence of these models. The conditions to be met by a dynamical system for stability and convergence are discussed through stability theorems. Finally the issues in the recall of stored information are discussed briefly.

Having understood the basics of artificial neural networks, the next task is to determine what kind of problems these structures and models can solve. The next four chapters deal with pattern recognition tasks that can be solved by some basic structures of artificial neural networks.

Review Questions

1. Explain the following:
 - (a) Activation and synaptic dynamics
 - (b) Models of neural networks vs neural network models
 - (c) Autonomous and nonautonomous dynamical systems
 - (d) Additive and shunting models of activation models
 - (e) Stochastic models vs stochastic versions of models
 - (f) Stability and convergence
 - (g) Structural vs global stability

2. What is meant by each of the following:
 - (a) Transient state
 - (b) Steady state
 - (c) Equilibrium state
 - (d) Stable states
3. What is the noise-saturation dilemma in activation dynamics?
4. Explain the differences among the three different types of stability in neural networks: fixed-point, oscillatory and chaotic.
5. What is meant by global pattern formation in neural networks?
6. What are the requirements of learning laws for effective implementation?
7. What are forgetting and recency effects in learning?
8. What are the different categories of learning?
9. Explain the difference between short-term memory and **long-**terms memory with reference to dynamics models.
10. What is meant by operating range of a neuron?
11. What are the different types of Hebbian learning?
12. What are the **different** types of competitive learning?
13. What is reinforcement learning? In what way it is different from supervised learning?
14. Explain some criteria used for neural network learning.
15. Explain the distinction between stability and convergence.
16. What is meant by global stability?
17. Distinguish between an equilibrium state and a stable state.
18. What is the significance of Lyapunov function in neural networks?
19. Explain the significance of each of the following theorems:
 - (a) Cohen-Grossberg theorem
 - (b) Cohen-Grossberg-Kosko theorem
 - (c) Adaptive bidirectional associative memory theorem
20. What are the two general methods of recall of information in neural networks?
21. Explain with an example the distinction between nearest neighbour and interpolative recall of information.

Problems

1. Show that the **Perkel's** model given in Eq. (2.11) is a special case of the additive autoassociative model given by Eq. (2.10).

2. Show from Eq. (2.16) that if $x_i(0) \leq B_i$, then $x_i(t) \leq B_i$, for all $t > 0$.
3. Show from Eq. (2.21) that if $x_i(0) \leq B_i / C_i$, then $x_i(t) \leq B_i / C_i$ for all $t > 0$.
4. Show from Eq. (2.21) that if $x_i(0) > -E_i / D_i$, then $x_i(t) \geq -E_i / D_i$ for all $t > 0$.
5. Explain the meaning of 'shunting' with reference to the shunting model of Eq. (2.21).
6. Explain the significance of the following:
 - (a) $\dot{x}_i = 0$, for all i
 - (b) $\dot{V}(\mathbf{x}) \leq 0$
 - (c) $\dot{V}(\mathbf{x}) = 0$
 - (d) $\dot{w}_{ij} \neq 0$, for all i, j .
7. Give the expressions for the general functions $a_i(x_i)$, $b_i(x_i)$ and $d_i(x_i)$ in Eq. (2.74) with reference to specific activation models given in Eqs. (2.10) and (2.21).
8. Show that the Lyapunov function represents some form of energy of an electrical circuit.
9. Show that $\dot{V}(\mathbf{x}) \leq 0$ for Eq. (2.84).
10. Show that $\dot{V}(\mathbf{x}, y) \leq 0$ for Eq. (2.88).
11. Consider a stochastic unit with a bipolar $\{-1, 1\}$ output function. The probability distribution for the unit is given by

$$P(s = 1 | x) = 1 / (1 + \exp(-2\lambda x))$$

If the learning of the stochastic unit is based on gradient descent on the error between the desired and the average output, show that the resulting learning law is the same as the learning law obtained using delta learning for a deterministic unit with hyperbolic tangent as the output function.

12. Determine the weights and the threshold of a stochastic unit with bipolar $\{-1, 1\}$ output function to classify the following 2-class problem using reinforcement learning equation given in Eq. (2.55). Assume $P(s = 1 | x) = 1 / (1 + \exp(-2x))$ and $\eta^+ = 0.1$ and $\eta^- = 0.01$. Start with suitable values of initial weights and threshold. Use positive reinforcement when the classification is correct and negative reinforcement when the classification is wrong. Show the final decision surface. (Hint: Write a program to implement the learning.)

Class C_1 : $[0 \ 0]^T$, $[1 \ 0]^T$, $[0 \ 1]^T$, and $[1 \ 1]^T$
 Class C_2 : $[-1 \ -1]^T$, $[-1 \ -2]^T$, $[-2 \ -1]^T$, and $[-2 \ -2]^T$

Chapter 3

Functional Units of ANN for Pattern Recognition Tasks

So far we have considered issues in pattern recognition, and introduced the basics of artificial neural networks. In this chapter we discuss some functional units of artificial neural networks that can solve simple pattern recognition tasks. These functional units form building blocks for developing neural architectures to solve complex pattern recognition problems.

The pattern recognition problem to be addressed by a system is discussed in Section 3.1. Three fundamental functional units are identified to deal with the basic pattern association problem and some variations of this problem. These units are described in Section 3.2. The specific pattern recognition tasks that the various functional units can solve are discussed in Section 3.3. Table 3.1 gives the organization of the networks and the pattern recognition tasks to be discussed in this chapter.

Table 3.1 Basic Artificial Neural Network Models for Pattern Recognition Problems

-
1. Feedforward ANN
 - (a) Pattern association
 - (b) Pattern classification
 - (c) Pattern **mapping/classification**
 2. Feedback ANN
 - (a) Autoassociation
 - (b) Pattern storage (**LTM**)
 - (c) Pattern environment storage (**LTM**)
 3. Feedforward and Feedback (Competitive Learning) ANN
 - (a) Pattern storage (**STM**)
 - (b) Pattern clustering
 - (c) Feature mapping
-

Pattern Recognition Problem

3.1 Pattern Recognition Problem

In any pattern recognition task we have a set of input patterns and the corresponding output patterns. Depending on the nature of the output patterns and the nature of the task environment, the problem could be identified as one of association or classification or mapping. The given set of input-output pattern pairs form only a few samples of an **unknown** system. From these samples the pattern recognition model should capture the characteristics of the system. Without looking into the details of the system, let us assume that the input-output patterns are available or given to us. Without loss of generality, let us also assume that the patterns could be represented as vectors in multidimensional spaces. We first state the most straightforward pattern recognition problem, namely, the pattern association problem, and discuss its characteristics.

Pattern Association Problem: Given a set of input-output pattern pairs $(\mathbf{a}_1, \mathbf{b}_1), (\mathbf{a}_2, \mathbf{b}_2), \dots, (\mathbf{a}_l, \mathbf{b}_l), \dots, (\mathbf{a}_L, \mathbf{b}_L)$ where $\mathbf{a}_l = (a_{l1}, a_{l2}, \dots, a_{lM})$ and $\mathbf{b}_l = (b_{l1}, b_{l2}, \dots, b_{lN})$ are M and N dimensional vectors, respectively, design a neural network to associate each input pattern with the corresponding output pattern.

If \mathbf{a}_l and \mathbf{b}_l are distinct, then the problem is called **heteroassociation**. On the other hand, if $\mathbf{b}_l = \mathbf{a}_l$, then the problem is called **autoassociation**. In the latter case the input and the corresponding output patterns refer to the same point in an N -dimensional space, since $M = N$ and $a_{li} = b_{li}$, $i = 1, 2, \dots, N, l = 1, 2, \dots, L$.

The problem of storing the association of the input-output pattern pairs $(\mathbf{a}_l, \mathbf{b}_l), l = 1, 2, \dots, L$, involves determining the weights of a network to accomplish the task. This is the training part. Once stored, the problem of recall involves determining the output pattern for a given input pattern by applying the operations of the network on the input pattern.

The recalled output pattern depends on the nature of the input and the design of the network. If the input pattern is the same as one of those used in the training, then the recalled output pattern is the same as the associated pattern in the training. If the input pattern is a noisy version of the trained input pattern, then the pattern may not be identical to any of the patterns used in training the network. Let the input pattern is $\hat{\mathbf{a}} = \mathbf{a}_l + \mathbf{E}$, where \mathbf{E} is a (small amplitude) noise vector. Let us assume that $\hat{\mathbf{a}}$ is closer (according to some distance measure) to \mathbf{a}_l than any other $\mathbf{a}_k, k \neq l$. If the output of the **network** for this input $\hat{\mathbf{a}}$ is still \mathbf{b}_l , then the network is designed to exhibit an accretive behaviour. On the other hand, if the network produces an output $\hat{\mathbf{b}} = \mathbf{b}_l + \mathbf{G}$, such that $|\mathbf{G}| \rightarrow \mathbf{0}$ as $|\mathbf{E}| \rightarrow \mathbf{0}$, then the network is designed to exhibit an interpolative behaviour.

Depending on the interpretation of the problem, several pattern

recognition tasks can be viewed as variants of the pattern association problem. We will describe these tasks in Section 3.3. First we will consider three basic functional units of neural networks which perform the pattern association and related pattern recognition tasks.

3.2 Basic Functional Units

There are three types of artificial neural networks. They are: (i) feedforward, (ii) feedback and (iii) a combination of both. The simplest networks of each of these types form the basic functional units. They are *functional* because they can perform by themselves some simple pattern recognition tasks. They are *basic* because they form building blocks for developing neural network architectures for complex pattern recognition tasks to be described later in Chapter 7.

The simplest feedforward network (Figure 3.1) is a two layer network with M input units and N output units. Each input unit is

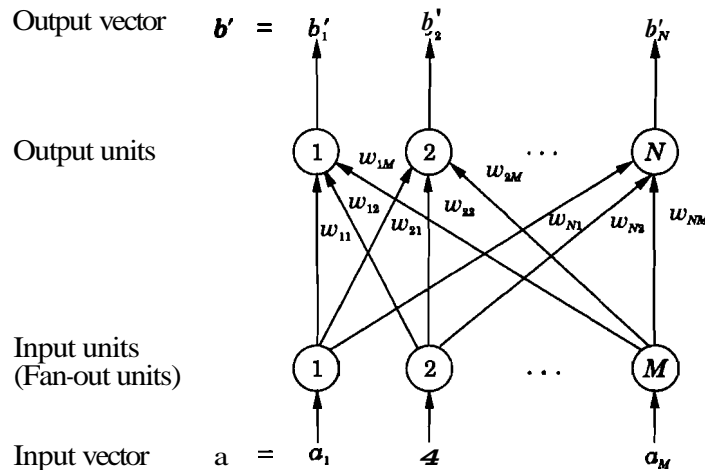


Figure 3.1 Basic feedforward neural network.

connected to each of the output units, and each connection is associated with a weight or strength of the connection. The input units are all linear, and they merely perform the task of fan-out, i.e., each unit is providing N outputs, one to each output unit. The output units are either linear or nonlinear depending on the task that the network should perform. Typically, feedforward networks are used for pattern association or pattern classification or pattern mapping.

The simplest feedback network, shown in Figure 3.2, consists of a set of N processing units, each connected to all other units. The connection strengths or weights are assumed to be symmetric, i.e., $w_{ij} = w_{ji}$, for $i \neq j$. Depending on the task, the units of the network could be linear or nonlinear. Typically feedback networks are used for autoassociation or pattern storage.

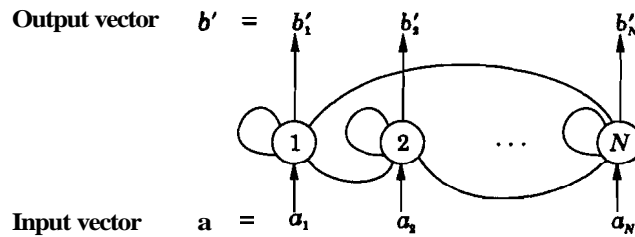


Figure 3.2 Basic feedback neural network.

The simplest combination network is called a competitive learning network, shown in Figure 3.3. It consists of an input layer of units

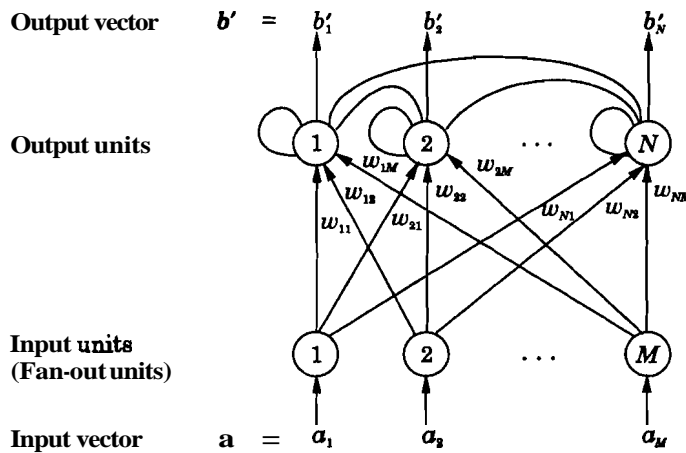


Figure 3.3 Basic competitive learning network.

feeding to the units in the output layer in a feedforward manner, and a feedback connection among the units in the output layer, including self-feedback. Usually the connection strengths or weights of the feedforward path are adjustable by training the network for a given pattern recognition task. The feedback connection strengths or weights in the output layer are usually fixed to specific values depending on the problem. The input units are **all** linear, and they merely perform the task of fan-out, i.e., each unit providing N outputs, one to each output unit. The output units are either linear or nonlinear depending on the task the network should perform. Typically the competitive learning network is used for pattern **grouping/clustering**.

3.3 Pattern Recognition Tasks by the Functional Units

Table 3.1 gives a summary of the pattern recognition **tasks** that can be performed by the three functional units described in the previous

section. All the pattern recognition tasks listed are simple, and can be viewed as variants of the pattern association problem. Each of these tasks can be described in terms of mapping of points from one multidimensional space onto another multidimensional space. In this section the geometrical interpretations of the pattern recognition tasks are given to obtain a clear understanding of the problems.

The input pattern space \mathcal{R}^M is an M-dimensional space, and the input patterns are points in this space. Likewise the output pattern space \mathcal{R}^N is an N-dimensional space, and the output patterns are points in this space. The pattern spaces are shown as circles in the figures used to illustrate the pattern recognition tasks.

3.3.1 Pattern Recognition Tasks by Feedforward Neural Networks

In this section we will discuss three pattern recognition tasks that can be performed by the basic feedforward neural network.

Pattern association problem: The pattern association problem is illustrated in Figure 3.4. The input patterns are shown as $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$

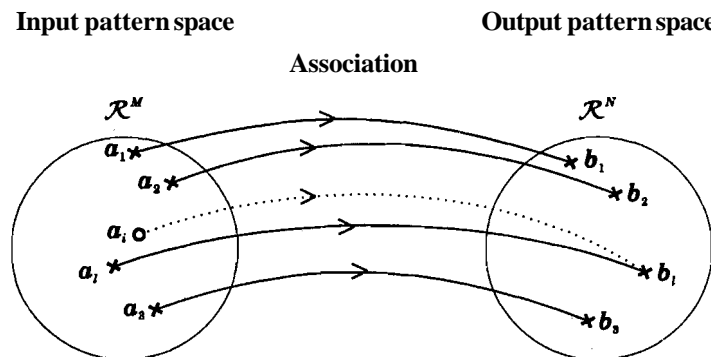


Figure 3.4 Illustration of pattern association task.

and the corresponding output patterns as $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$. The objective of designing a neural network is to capture the association between input-output pattern pairs in the given set of training data, so that when any of the inputs \mathbf{a}_i is given, the corresponding output \mathbf{b}_i is retrieved. Suppose an input pattern \mathbf{a}_i not used in the training set is given. If the training input pattern \mathbf{a}_j is the closest to \mathbf{a}_i , then the pattern association network should retrieve the output pattern \mathbf{b}_j for the input pattern \mathbf{a}_i . Thus the network should display accretive behaviour. The pattern \mathbf{a}_i can be viewed as a noisy version of the pattern \mathbf{a}_j . That is $\mathbf{a}_i = \mathbf{a}_j + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is a noise vector. If the amplitude of the noise added to \mathbf{a}_j is so large that the noisy input pattern is closer to some pattern (say \mathbf{a}_k) other than the correct one (\mathbf{a}_j), then the network produces an **incorrect** output pattern

$\mathbf{b}_k, k \neq l$. Thus an incorrect output pattern would be retrieved for the given noisy input.

An example of a pattern association problem is associating a unique binary code to a printed alphabet character, say $[00000]^T$ for A, $[00001]^T$ for B, etc. (See Figure 3.5). The input patterns A, B, etc.,

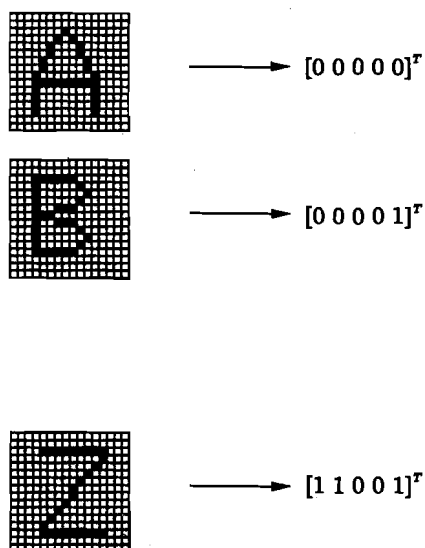


Figure 3.6 An example of pattern association problem.

could be represented as black and white pixels in a grid of size, say 16×16 points. Then the input pattern space is a binary **256-dimensional** space, and the output pattern space is a binary 5-dimensional space. Noisy versions of the input patterns are obtained when some of the pixels in the grid containing a character are transformed from black to white or vice versa.

Note that the performance of a network for the pattern association problem is mainly dictated by the distribution of the training patterns in the input space. **This** point will be discussed in detail in Chapter 4.

Pattern classification problem: In the pattern association problem if a group of input patterns correspond to the same output pattern, then typically there will be far fewer output patterns compared to the number of input patterns. In other words, if some of the output patterns in the pattern association problem are **identical**, then the number of distinct output **patterns** can be viewed as class labels, and the input patterns corresponding to each class can be viewed as samples of that class. The problem then becomes a pattern classification problem as illustrated in Figure 3.6.

In this case whenever a pattern belonging to a class is given as input, the network identifies the class label. During training, only a

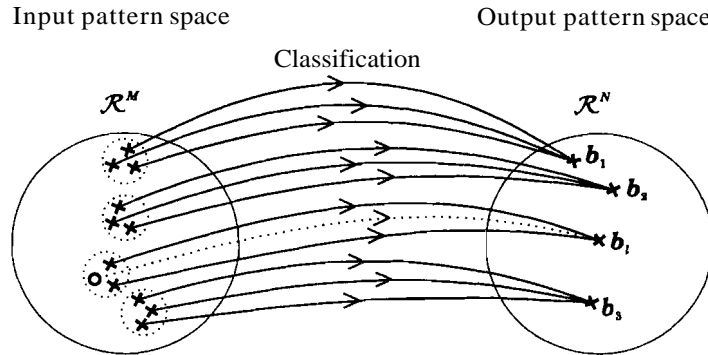


Figure 3.6 Illustration of pattern classification task.

few samples of patterns for each class are given. In testing, the input pattern is usually different from the patterns used in the training set for the class. The network displays an accretive behaviour in this case.

An example of pattern classification problem could be labelling hand printed characters within a specified grid into the corresponding printed character. Note that the printed character patterns are unique and fixed in number, and serve as class labels. These labels could be a unique 5-bit code as shown in Figure 3.7. For a given

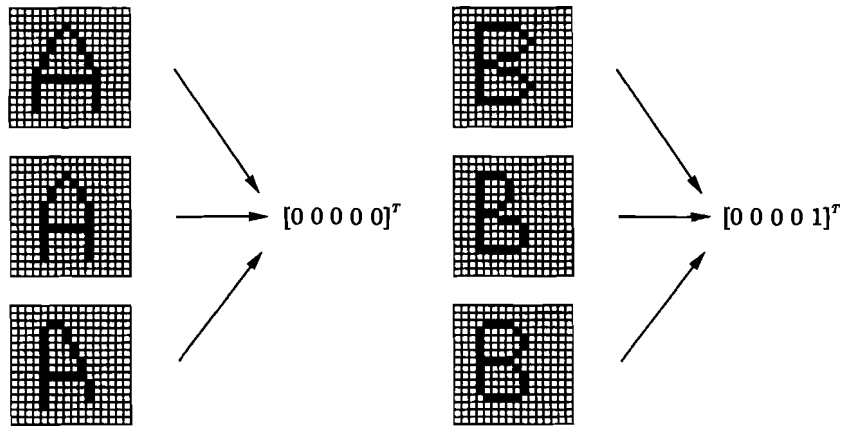


Figure 3.7 An example of pattern classification problem.

character, the samples of hand-printed versions of the character are not identical. In fact the dimensionality of the input pattern space will have to be very large in order to represent the hand-printed characters accurately. An input pattern not belonging to any class may be forced into one of the predetermined class labels by the network.

Note that the performance of a network for the pattern classification problem depends on the characteristics of the samples associated with each class. Thus grouping of the input patterns by

the class label dictates the performance. **This** point will be discussed in detail in Chapters 4 and 7.

Pattern mapping: Given a set of input-output pattern pairs as in the pattern association problem, if the objective is to capture the implied mapping, instead of association, then the problem becomes a pattern mapping problem (**Figure 3.8**). In a pattern mapping problem

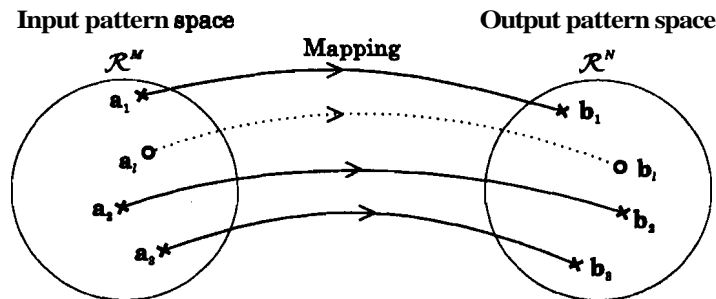


Figure 3.8 Illustration of pattern mapping task.

both the input and the output patterns are only samples from the mapping system. Once the system behaviour is captured by the network, the network would produce a possible output pattern for a new input **pattern**, not used in the training set. The possible output pattern would be **approximately** an interpolated version of the output patterns corresponding to the input training patterns close to the given test input pattern. Thus the network displays an interpolative behaviour. Typically the input and output pattern spaces are continuous in **this** case, and the mapping function must be smooth for the interpolation to work satisfactorily.

An example of the data for a pattern mapping problem could be the input data given to a complex physical system and the corresponding output data from the system for a number of trials. The objective is to capture the unknown system behaviour from the samples of the input-output pair data.

A pattern mapping problem is the most general case, from which the pattern classification and pattern association problems can be derived as special cases. The network for pattern mapping is expected to perform generalization. The details of how well a given network can do generalization will be discussed in Chapter 7.

3.3.2 Pattern Recognition Tasks by Feedback Neural Networks

In this section we **will** discuss three pattern recognition tasks **that** can be performed by the basic feedback neural networks.

Autoassociation problem: If each of the output patterns b_i in a

pattern association problem is identical to the corresponding input patterns \mathbf{a}_l , then the output pattern space is identical to the input pattern space (Figure 3.9). In such a case the problem becomes an

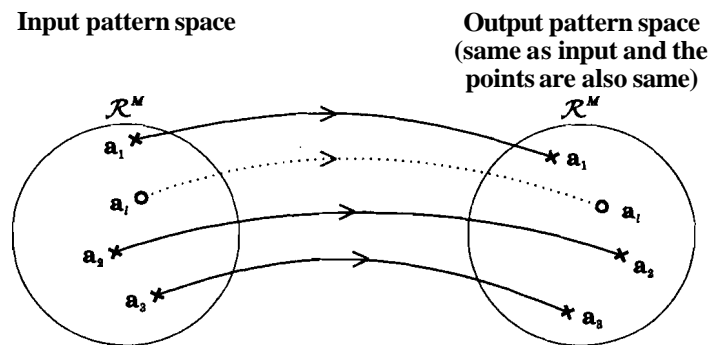


Figure 3.9 Illustration of autoassociation task.

autoassociation problem. This is a trivial case where the network merely stores the given set of input patterns. When a noisy input pattern is given, the network retrieves the same noisy pattern. Thus there is an absence of accretive behaviour.

A detailed analysis of the autoassociation problem is given in Chapter 5. Note that the special case of $\mathbf{b}_l = \mathbf{a}_l$, $l = 1, 2, \dots, L$ in the pattern association task is considered as a problem of heteroassociation task to be addressed by a feedforward network. The term autoassociation task is thus used exclusively in the context of feedback networks.

Pattern storage problem: In the autoassociation problem, if a given input pattern is stored in a network for later recall by an approximate input pattern, then the problem becomes a pattern storage problem (Figure 3.10). Any input vector close to a stored input pattern will

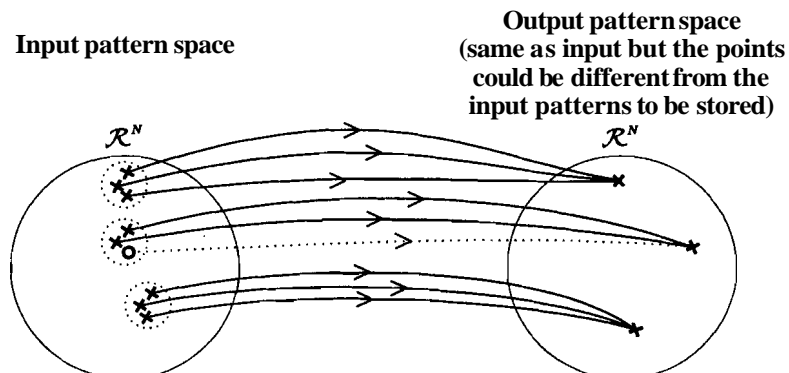


Figure 3.10 Illustration of pattern storage task.

recall that input pattern exactly from the network, and thus the network displays accretive behaviour. The stored patterns could be

the same as the input patterns given during training. In such a case the input pattern space is a continuous one, and the output space consists of a fixed finite set of (stored) patterns corresponding to some of the points in the input pattern space. The stored patterns could also be some transformed versions of the input patterns, but of the same dimension as the input space. In such a case the stored patterns may correspond to different points in the input space.

Due to its accretive behaviour, the pattern storage network is very useful in practice. A detailed analysis of this network is given in Chapter 5.

Pattern environment storage problem: If a set of patterns together with their probabilities of occurrence are specified, then the resulting specification is called pattern environment. The design of a network to store a given pattern environment aims at recall of the stored patterns with the lowest probability of error. This is called a pattern environment storage problem. A detailed analysis of this problem together with the network design is given in Chapter 5.

3.3.3 Pattern Recognition Tasks by Competitive Learning Neural Networks

In this section we will discuss three pattern recognition tasks that can be performed by a combination neural network consisting of feedforward and feedback parts. The network is also called the competitive learning network.

Temporary pattern storage: If a given input pattern is stored in a network, even in a transformed form, in such a way that the pattern remains only until a new pattern input is given, then the problem becomes that of a short term memory or temporary storage problem. This is only of academic interest. However, a detailed analysis of this problem is given in Chapter 6.

Pattern clustering problem: Given a set of patterns, if they are grouped according to similarity of the patterns, then the resulting problem is called pattern clustering. It is illustrated in Figure 3.11. There are two types of problems here. In one case the network displays an accretive behaviour (Figure 3.11a). That is, if an input pattern not belonging to any group is presented, then the network will force it into one of the groups. The input pattern space is typically a continuous space. The test input patterns could be the same as the ones used in the training or could be different. The output pattern space consists of a **set** of cluster centres or labels.

The second type of problem displays interpolative behaviour as shown in Figure 3.11b. In this case, a test input pattern not belonging

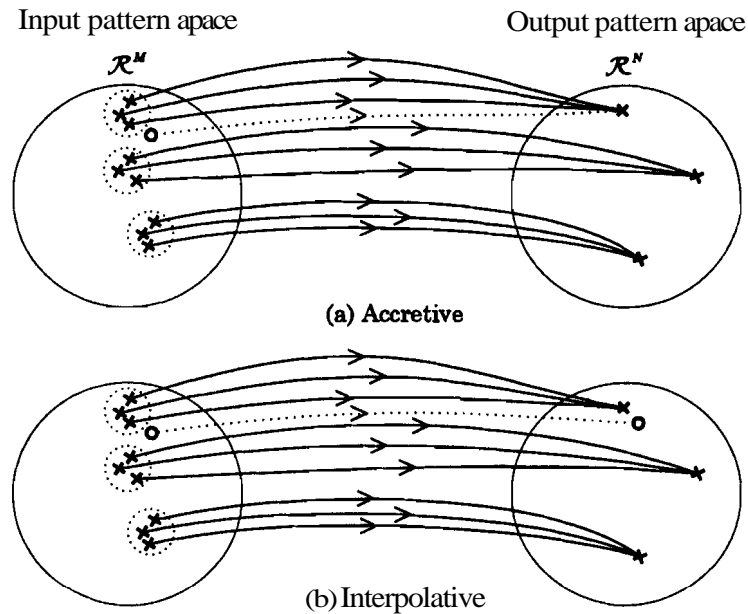


Figure 3.11 Illustration of two types of pattern clustering tasks.

to any group produces an output which is some **form** of interpolation of the output patterns or cluster centers, depending on the proximity of the test input pattern to the input pattern groups formed during training.

Pattern clustering also leads to the problem of vector quantization. A detailed analysis of these problems is given in Chapter 6.

Feature mapping problem: In the pattern clustering problem a group of approximately similar input patterns are identified with a fixed output pattern or a group label. On the other hand, if similarities of the features of the input patterns have to be retained in the output, the problem becomes one of feature mapping. In this, a given set of input patterns **are** mapped onto output patterns in such a way that the **proximity** of the output patterns reflect the similarity of the features of the corresponding input patterns. When a test input pattern is given, it will generate an output which is in the neighbourhood of the outputs for similar patterns. Note that typically the number of output patterns are fixed, but they are much larger than in the pattern clustering case, and they are organized physically in the network in such a way that the neighbourhood pattern labels reflect closeness of features. A detailed analysis of the feature mapping problem is given in Chapter 6.

In summary, this chapter dealt with some basic functional units of neural networks and a description of the pattern recognition tasks that these units can perform. In particular, we have identified three

basic networks: **feedforward**, feedback and competitive learning networks. We have defined the pattern association problem as a basic problem, and we have seen how several other pattern recognition tasks could be interpreted as **variants** of this problem. We have discussed each of the pattern recognition tasks in the form of a mapping problem. What we have not discussed is how the basic functional units perform the corresponding pattern recognition tasks mentioned in this chapter. The next three chapters deal with a detailed analysis of these tasks by the networks.

Review Questions

1. What are the three functional units? Why are they called functional units?
2. Explain the meaning of (a) accretive behaviour and (b) interpolative behaviour.
3. Distinguish between pattern association, pattern classification and pattern mapping tasks.
4. Give a real life example of a pattern mapping problem.
5. Explain the difference between autoassociation problem and heteroassociation problem.
6. What is meant by a pattern environment storage problem? Give a real life example to illustrate the problem.
7. Explain the difference between the accretive and interpolative type of clustering problems.
8. Explain what is meant by feature mapping? Explain the problem with a real life example from speech production.
9. Explain how recognition of handwritten **digits** is closer to a classification type problem, whereas recognition of vowel sounds in continuous speech is closer to a feature mapping type of problem.

Chapter 4

Feedforward Neural Networks

4.1 Introduction

This chapter presents a detailed analysis of the pattern recognition tasks that can be performed by a **feedforward** artificial neural network. As mentioned earlier, a feedforward artificial neural network consists of layers of processing units, each layer feeding input to the next layer in a feedforward manner through a set of connection strengths or weights. The simplest such network is a two layer network.

By a suitable choice of architecture for a feedforward network, it is possible to perform several pattern recognition tasks. The simplest task is a pattern association task, which can be realized by a two layer feedforward network with linear processing units. A detailed analysis of the linear association network shows that the network is limited in its capabilities. In particular, the number of input-output pattern pairs to be associated are limited to the dimensionality of the input pattern, and also the set of input patterns must be linearly independent. The constraint on the number of input patterns is overcome by using a two layer feedforward network with nonlinear processing units in the output layer. This modification automatically leads to the consideration of pattern classification problems. While this modification overcomes the constraints of number and linear independence on the input patterns, it introduces the limitations of linear separability of the functional relation between input and output patterns. Classification problems which are not linearly separable are called hard problems. In order to overcome the constraint of linear separability for pattern classification problems, a multilayer feedforward network with nonlinear processing units in all the intermediate hidden layers and in the output layer is proposed. While a multilayer feedforward architecture could solve representation of the hard problems in a network, it introduces the problem of hard learning, i.e., the difficulty in adjusting the weights of the network to capture the implied functional relationship between the given input-output pattern pairs. The hard learning problem is solved by using the backpropagation learning algorithm. The resulting network provides a solution to the pattern mapping problems. The generaliza-

tion (ability to learn a mapping function) feature of a multilayer feedforward network with the backpropagation learning law depends on several factors such as the architectural details of the network, the learning rate parameter of the training process and the training samples themselves.

Table 4.1 shows the summary of the topics to be discussed in this chapter. The pattern association problem is discussed in Section 4.2.

Table 4.1 Pattern Recognition Tasks by Feedforward Neural Networks

Pattern association	
• Architecture:	Two layers, linear processing units, single set of weights
• Learning:	Hebb's (orthogonal) rule, Delta (linearly independent) rule
• Recall:	Direct
• Limitation:	Linear independence, number of patterns restricted to input dimensionality
• To overcome:	Nonlinear processing units, leads to a pattern classification problem
Pattern classification	
• Architecture:	Two layers, nonlinear processing units, geometrical interpretation
• Learning:	Perceptron learning
• Recall:	Direct
• Limitation:	Linearly separable functions, cannot handle hard problems
• To overcome:	More layers, leads to a hard learning problem
Pattern mapping or classification	
• Architecture:	Multilayer (hidden), nonlinear processing units, geometrical interpretation
• Learning:	Generalized delta rule (backpropagation)
• Recall:	Direct
• Limitation:	Slow learning, does not guarantee convergence
• To overcome:	More complex architecture

This section gives a detailed analysis of a linear associative network, and shows the limitations of the network for pattern association problems. Section 4.3 describes the pattern classification problem. An analysis of a two layer feedforward network with nonlinear processing units in the output layer brings out the limitations of the network for pattern classification task. The section also discusses the problems of classification, representation, learning and convergence in the context of perceptron networks. In Section 4.4 the problem of pattern mapping by a multilayer neural network is discussed. The chapter concludes with a discussion on the backpropagation learning law and its implications for generalization in a pattern mapping problem.

4.2 Analysis of Pattern Association Networks

4.2.1 Linear Associative Network

The objective in pattern association is to design a network that can represent the association in the pairs of vectors $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \dots, L$, through a set of weights to be determined by a learning law. The given set of input-output pattern pairs is called training data. The input **patterns** are typically generated synthetically, like machine printed characters. The input patterns used for recall may be corrupted by external noise.

The following vector and matrix notations are used for the analysis of a linear associative network:

Input vector	$\mathbf{a}_l = [a_{l1}, a_{l2}, \dots, a_{lM}]^T$
Activation vector of input layer	$\mathbf{x} = [x_1, x_2, \dots, x_M]^T$
Activation vector of output layer	$\mathbf{y} = [y_1, y_2, \dots, y_N]^T$
Output vector	$\mathbf{b}_l = [b_{l1}, b_{l2}, \dots, b_{lN}]^T$
Input matrix	$A = [\mathbf{a}_1 \mathbf{a}_2 \dots \mathbf{a}_L]$ is an $M \times L$ matrix
Output matrix	$B = [\mathbf{b}_1 \mathbf{b}_2 \dots \mathbf{b}_L]$ is an $N \times L$ matrix
Weight matrix	$W = [\mathbf{w}_1 \mathbf{w}_2 \dots \mathbf{w}_N]^T$ is an $N \times M$ matrix
Weight vector for j th unit of output layer	$\mathbf{w}_j = [w_{j1}, w_{j2}, \dots, w_{jM}]^T$

The network consists of a set of weights connecting two layers of processing units as shown in Figure 4.1. The output function of each

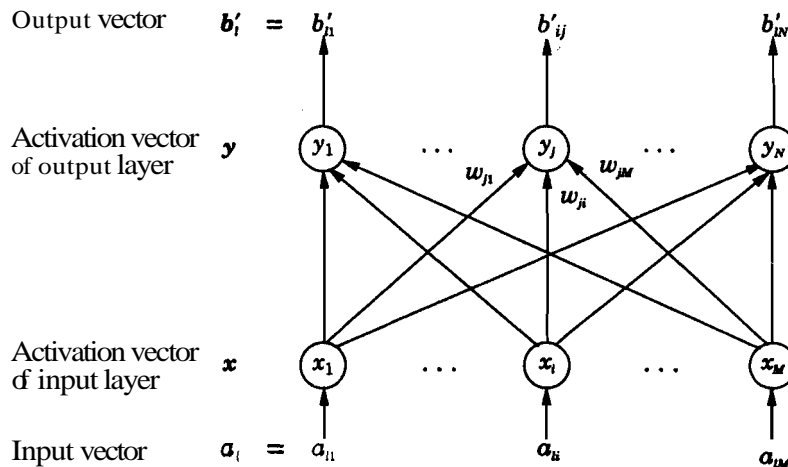


Figure 4.1 Linear associative network.

unit in these layers is linear. Each output unit **receives inputs** from the M input units corresponding to the M -dimensional input vectors. The number (N) of output units corresponds to the dimensionality of the output vectors. Due to linearity of the output function, the activation values (x_i) and the signal values of the units in the input layer are the same as the input data values a_{li} . The activation value of the j th unit in the output layer is given by

$$y_j = \sum_{i=1}^M w_{ji} a_{li} = \mathbf{w}_j^T \mathbf{a}_l, \quad j = 1, 2, \dots, N. \quad (4.1)$$

The output (b'_{ij}) of the j th unit is the same as its activation value y_j , since the output function of the unit is linear, i.e., $b'_{ij} = y_j$. The network is called linear since the output of the network is simply a linear weighted sum of the component values of the input pattern.

The objective is to determine a set of weights $\{w_{ji}\}$ in such a way that the actual output b'_{ij} is equal to the desired output b_{ij} for all the given L pattern pairs. The weights are determined by using the criterion that the total mean squared error between the desired output and the actual output is to be minimized. The weights can be determined either by computing them from the training data set or by learning. Computation of weights makes use of all the training set data together. On the other hand, in learning, the weights are updated after presentation of each of the input-output pattern pairs in the training set.

4.2.2 Determination of Weights by Computation

For a linear associative network Mecht-Nielsen, 19901,

$$x_i = a_{li}, \quad i = 1, 2, \dots, M \quad (4.2)$$

$$y_j = \sum_{i=1}^M w_{ji} x_i, \quad j = 1, 2, \dots, N \quad (4.3)$$

$$b'_{ij} = y_j = \mathbf{w}_j^T \mathbf{x} = \mathbf{w}_j^T \mathbf{a}_l, \quad j = 1, 2, \dots, N \quad (4.4)$$

Actual output vector

$$\mathbf{b}'_l = \mathbf{y} = \mathbf{W}\mathbf{x} = \mathbf{W}\mathbf{a}_l \quad (4.5)$$

Error in the output is given by the distance between the desired output vector and the actual output vector. The total error $E(\mathbf{W})$ over all the L input-output pattern pairs is given by

$$\begin{aligned} E(\mathbf{W}) &= \frac{1}{L} \sum_{l=1}^L \sum_{j=1}^N (b_{lj} - b'_{lj})^2 \\ &= \frac{1}{L} \sum_{l=1}^L \|\mathbf{b}_l - \mathbf{W}\mathbf{a}_l\|^2 \end{aligned} \quad (4.6)$$

We can write

$$E(W) = \frac{1}{L} \|B - WA\|^2 \quad (4.7)$$

where the square norm

$$\|B - WA\|^2 = \sum_{l=1}^L \sum_{j=1}^N (b_{lj} - \mathbf{w}_j^T \mathbf{a}_l)^2 \quad (4.8)$$

Using the definition that the trace of a square matrix S is the sum of the main diagonal entries of S , it is easy to see that

$$E(W) = \frac{1}{L} \text{tr}(S), \quad (4.9)$$

where the matrix S is given by

$$S = (B - WA)(B - WA)^T \quad (4.10)$$

and $\text{tr}(S)$ is the trace of the matrix S .

Using the definition for pseudoinverse of a matrix [Penrose, 1955], i.e., $A^+ = A^T(AA^T)^{-1}$, we get the matrix identities $A^+AA^T = A^T$ and $AA^T(A^+)^T = A$. Using these matrix identities we get

$$S = (W - BA^+)AA^T(W - BA^+)^T + B(I - A^+A)B^T \quad (4.11)$$

It can be seen that the trace of the first term in Eq. (4.11) is always nonnegative, as it is in a quadratic form of the real symmetric matrix AA^T . It becomes zero for $W = BA^+$. The trace of the second term is a constant, independent of W . Since the trace of sum of matrices is the sum of traces of the individual matrices, the error $E(W)$ is minimum when $W = BA^+$. The minimum error is obtained by substituting $W = BA^+$ in Eq. (4.7), and is given by

$$\begin{aligned} E_{\min} &= \frac{1}{L} \|B - BA^+A\|^2 \\ &= \frac{1}{L} \text{tr}[(B(I - A^+A))(B(I - A^+A))^T] \\ &= \frac{1}{L} \text{tr}[B(I - A^+A)B^T]. \end{aligned} \quad (4.12)$$

where I is an $L \times L$ identity matrix. The above simplification is obtained by using the following matrix identities: $(A^+A)^T = A^T(A^+)^T$ and $AA^T(A^+)^T = A$.

The following singular value decomposition (SVD) of an $M \times L$ matrix A is used to compute the pseudoinverse and to evaluate the minimum error. Assuming $L \leq M$, the SVD of a matrix A is given by [Strang, 1980]

$$A = \sum_{i=1}^L \lambda_i^{1/2} \mathbf{p}_i \mathbf{q}_i^T, \quad (4.13)$$

where $AA^T\mathbf{p}_i = \lambda_i\mathbf{p}_i$, $A^T\mathbf{A}\mathbf{q}_i = \lambda_i\mathbf{q}_i$, and the sets $(\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_M)$ and $(\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_L)$ are each orthogonal. The eigenvalues λ_i of the matrices AA^T and $A^T\mathbf{A}$ will be real and nonnegative, since the matrices are symmetric. The eigenvalues are ordered, i.e., $\lambda_i \geq \lambda_{i+1}$. Note that the \mathbf{p}_i 's are M -dimensional vectors and \mathbf{q}_i 's are L -dimensional vectors.

The pseudoinverse A^+ of A is given by

$$A^+ = \sum_{i=1}^r \lambda_i^{-1/2} \mathbf{q}_i \mathbf{p}_i^T, \quad (4.14)$$

where r is the rank (maximum number of linearly independent columns) of A . Also r turns out to be the number of nonzero eigenvalues λ_i . Note that if $r = L$, then all the L column vectors are linearly independent.

Using the SVD, it can be shown that $A^+A = I_{L \times L}$, if L is the number of linearly independent columns of A . In such a case $I - A^+A = \mathbf{0}$ (null vector), and hence $E_{\min} = 0$ (See Eq. (4.12)). Therefore, for a linearly independent set of input pattern vectors, the error in the recall of the associated output pattern vector is zero, if the optimum choice of $W = BA^+$ is used.

If the rank r of the matrix A is less than L , then the input vectors are linearly dependent. In this case also the choice of the weight matrix as $W = BA^+$ still results in the least error E_{\min} . But this least error is not zero in this case. The value of the error depends on the rank r of the matrix. The matrix A^+A will have a sub-matrix $I_{r \times r}$, and all the other four sub-matrices will be zero. That is

$$[A^+A]_{L \times L} = \begin{bmatrix} I_{r \times r} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (4.15)$$

The expression for minimum error is given from Eq. (4.12) as

$$\begin{aligned} E_{\min} &= \frac{I}{L} \text{tr}[B(I_{L \times L} - A^+A)B^T] \\ &= \frac{I}{L} \sum_{i=r+1}^L \|B\mathbf{q}_i\|^2 \end{aligned} \quad (4.16)$$

The next issue is how to achieve the minimum error retrieval from the linear associative network, when there is noise \mathbf{n}_i added to the input vectors. The noisy input vectors are

$$\mathbf{c}_i = \mathbf{a}_i + \mathbf{n}_i \quad (4.17)$$

It is assumed that each component of the noise vector \mathbf{n}_i is uncorrelated with the other components and also with the components of the pattern vectors, and has the same standard deviation σ . Let C be an $M \times L$ matrix of the noisy input vectors. The objective is to find a W that minimizes

$$E(W) = \frac{1}{L} \|B - WC\|^2 \quad (4.18)$$

Murakami has shown that, if $W = BA^+$, then the expression for error $E(W)$ is given by [Murakami and Aibara, 19871

$$E(W) = \frac{1}{L} \left[\sum_{i=r+1}^L \|B\mathbf{q}_i\|^2 + L\sigma^2 \sum_{i=1}^r \lambda_i^{-1} \|B\mathbf{q}_i\|^2 \right] \quad (4.19)$$

The first term in the square brackets can be attributed to the linear dependency part of the column vectors of A. If the rank of the matrix A is L, then $r = L$, and the first term will be zero. Therefore, the error is determined by the noise power σ^2 . If in addition, there is no noise, i.e., $\sigma = 0$, then the error $E(W) = 0$. In that case error-free recall is possible.

To minimize $E(W)$ in the presence of noise in the input pattern vectors, choose $W = B\hat{A}^+$, where

$$\hat{A}^+ = \sum_{i=1}^s \lambda_i^{-1/2} \mathbf{q}_i \mathbf{p}_i^T. \quad (4.20)$$

The value of s is determined in such a way that

$$\frac{L\sigma^2}{\lambda_s} \leq 1 \leq \frac{L\sigma^2}{\lambda_{s+1}} \quad (4.21)$$

That is, the noise power σ^2 will decide how many terms should be considered in the SVD expression for the pseudoinverse. If the eigenvalue λ_i is so small that the noise power dominates, then that eigenvector could as well be included in the first term of the expression in Eq. (4.19) for the error corresponding to the linear dependence. This will reduce the error.

Note that this analysis is valid only if the legal inputs are corrupted by noise. It is not valid if the input consists of only noise. The expression for the error $E(W)$ is applicable for the closed set of the column vectors in A [Murakami and Aibara, 19871.

4.23 Determination of Weights by Learning

It is desirable to determine the weights of a network in an incremental manner, as and when a new training input-output pattern pair is available. This is called *learning*. Each update of the weights with a new input data can be interpreted as network learning. Computationally also learning is preferable because it does not require information of all the training set data at the same time. As will be seen later in this section, it is also preferable to have learning confined to a local operation. That is, the update of a weight connecting two processing units depends only on the connection

weight and the activations of the units on either side of the connection. Two learning laws and their variations, as applicable to a linear associative network, **are** discussed in this section.

Hebb's law: Let the input pattern vector \mathbf{a}_l and the corresponding desired output pattern vector \mathbf{b}_l be applied to the linear associative network. According to the Hebb's law, the updated weight value of a connection depends only on the activations of the processing **units** on either side of the connecting link. That is

$$\begin{aligned} w_{ji}(l) &= w_{ji}(l-1) + x_i y_j \\ &= w_{ji}(l-1) + a_{li} b_{lj}, \\ &\text{for } i = 1, 2, \dots, M; \quad j = 1, 2, \dots, N \end{aligned} \quad (4.22)$$

Note that the computation of the increment $x_i y_j = a_{li} b_{lj}$ is purely local for the processor unit and the input-output pattern pair. The updated weight matrix for the application of the l th pair $(\mathbf{a}_l, \mathbf{b}_l)$ is given by

$$\mathbf{W}(l) = \mathbf{W}(l-1) + \mathbf{b}_l \mathbf{a}_l^T, \quad (4.23)$$

where $\mathbf{W}(l-1)$ refers to the weight matrix after presentation of the first $(l-1)$ pattern pairs, and $\mathbf{W}(l)$ refers to the weight matrix after presentation of the first l pattern pairs. Note that $\mathbf{b}_l \mathbf{a}_l^T$ is the outer product of the two vectors, which results in an $N \times M$ matrix. Each element of this matrix is an increment of the corresponding element in the weight matrix.

If the initial values of the elements of the weight matrix are assumed to be zero, then the weight matrix resulting after application of the L input-output pattern vector pairs $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \dots, L$, is given by

$$\mathbf{W} = \sum_{l=1}^L \mathbf{b}_l \mathbf{a}_l^T = \mathbf{B} \mathbf{A}^T, \quad (4.24)$$

where the element w_{ji} of \mathbf{W} is given by

$$w_{ji} = \sum_{l=1}^L a_{li} b_{lj} \quad (4.25)$$

To verify whether the network has learnt the association of the given set of input-output pattern vector pairs, apply the input pattern \mathbf{a}_k and determine the actual output vector \mathbf{b}'_k .

$$\begin{aligned} \mathbf{b}'_k &= \mathbf{W} \mathbf{a}_k = \sum_{l=1}^L \mathbf{b}_l \mathbf{a}_l^T \mathbf{a}_k \\ &= \mathbf{b}_k (\mathbf{a}_k^T \mathbf{a}_k) + \sum_{l \neq k} \mathbf{b}_l (\mathbf{a}_l^T \mathbf{a}_k) \end{aligned} \quad (4.26)$$

It is obvious from the above equation that the actual output \mathbf{b}'_k is not the same as the desired output \mathbf{b}_k . Only if some restrictions are imposed on the set of input pattern vectors $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_L\}$, we can get the recall of the correct output pattern \mathbf{b}_k for the input pattern \mathbf{a}_k . The restriction is that the set of input vectors must be **orthonormal**. That is

$$\begin{aligned} \mathbf{a}_k^T \mathbf{a}_l &= \mathbf{a}_l^T \mathbf{a}_k = \delta_{kl} = 1 & \text{if } l = k \\ &= 0 & \text{if } l \neq k \end{aligned} \quad (4.27)$$

In such a case the first term in the expression for \mathbf{b}'_k in Eq. (4.26) becomes \mathbf{b}_k , and the second term becomes zero, as each of the products $\mathbf{a}_l^T \mathbf{a}_k$ is zero for $l \neq k$. The restriction of orthogonality limits the total number (L) of the input patterns in the set A to M, i.e., the dimensionality of the input vectors, as there can be only M or less than M mutually orthogonal vectors in an M-dimensional space.

If the restriction of orthogonality on the set of input vectors is relaxed to mere linear independence, then the expression in Eq. (4.26) for recall reduces to

$$\mathbf{b}'_k = \mathbf{b}_k + \mathbf{e} \quad (4.28)$$

where it is assumed that the vectors are of unit magnitude, so that $\mathbf{a}_k^T \mathbf{a}_k = 1$. This leaves an error term \mathbf{e} indicating that the recall is not perfect, if the weight matrix is derived using the Hebb's law.

However, it was shown in the previous Section 4.2.2 that, for linearly independent set of input vectors, exact recall can be achieved if the weight matrix W is chosen as $W = \mathbf{B}\mathbf{A}^+$, where \mathbf{A}^+ is the pseudoinverse of the matrix \mathbf{A} . If the set of input vectors are not linearly independent, then still the best choice of W is $\mathbf{B}\mathbf{A}^+$, as this yields, on the average, the least squared error in the recall of the associated pattern. The error is defined as the difference between the desired and the actual output patterns from the associative network. If there is noise in the input, the best choice of the weight matrix W is $\mathbf{B}\hat{\mathbf{A}}^+$, where $\hat{\mathbf{A}}^+$ includes fewer (s) terms in the singular value decomposition expansion of \mathbf{A} than the rank (r) of the matrix, the choice of s being dictated by the level of the noise (See Eq. (4.21)).

For all these best choices of W , the weight values have to be computed from the knowledge of the complete input pattern matrix \mathbf{A} , since all of them need the SVD of \mathbf{A} to compute the pseudoinverse \mathbf{A}^+ . However, it is possible, at least in some cases, to develop learning algorithms which can approach the best choices for the weight matrices. The purpose of these learning algorithms is to provide a procedure for incremental update of the weight matrix when an input-output pattern pair is presented to the network. Most of these learning algorithms are based on gradient descent along an error surface (See Appendix C). The most basic among them is **Widrow** and

Hoff's least mean square (LMS) algorithm [Widrow and Hoff, 1960]. The gradient descent algorithms are discussed in detail later in the section on pattern mapping tasks.

Widrow's law: A form of Widrow learning can be used to obtain $W = BA^+$ recursively. Let $W(l-1)$ be the weight matrix after presentation of $(l-1)$ samples. Then $W(l-1) = B(l-1)A^+(l-1)$, where the matrices $B(l-1)$ and $A(l-1)$ are composed of the first $(l-1)$ vectors of \mathbf{b}_k and the first $(l-1)$ vectors of \mathbf{a}_k , respectively. When the pair $(\mathbf{a}_l, \mathbf{b}_l)$ is given to the network, then the updated matrix is given by (See [Hecht-Nielsen, 1990])

$$W(l) = W(l-1) + (\mathbf{b}_l - W(l-1)\mathbf{a}_l)\mathbf{p}_l^T \quad (4.29)$$

where

$$\mathbf{p}_l = \frac{[\mathbf{I} - A(l-1)A^+(l-1)]\mathbf{a}_l}{|[\mathbf{I} - A(l-1)A^+(l-1)]\mathbf{a}_l|^2}, \quad \text{if the denominator is } \neq 0$$

$$- \frac{A^T(l-1)A^+(l-1)\mathbf{a}_l}{1 + |A^+(l-1)\mathbf{a}_l|^2}, \quad \text{otherwise} \quad (4.30)$$

By starting with zero initial values for all the weights, and successively adding the pairs $(\mathbf{a}_1, \mathbf{b}_1), (\mathbf{a}_2, \mathbf{b}_2), \dots, (\mathbf{a}_L, \mathbf{b}_L)$, we can obtain the final pseudoinverse-based weight matrix $W = BA^+$. The problem with this approach is that the recursive procedure **cannot** be implemented locally because of the need to calculate \mathbf{p}_l in Eq. (4.29).

The same eventual effect can be approximately realized using the following variation of the above learning law,

$$W(l) = W(l-1) + \eta (\mathbf{b}_l - W(l-1)\mathbf{a}_l)\mathbf{a}_l^T, \quad (4.31)$$

where η is a small positive constant called the learning rate parameter. This Widrow's learning law can be implemented locally by means of the following equation,

$$w_{ji}(l) = w_{ji}(l-1) + \eta (b_{lj} - \mathbf{w}_j^T(l-1)\mathbf{a}_l) a_{li}, \quad (4.32)$$

where $\mathbf{w}_j(l-1)$ is the weight vector associated with the j th processing unit in the output layer of the linear associative network at the $(l-1)$ th iteration. With this scheme, it is **often** necessary to apply the pairs $(\mathbf{a}_l, \mathbf{b}_l)$ of the training set data several times, with each pair chosen at random.

The convergence of the Widrow's learning law in Eq. (4.32) depends on the choice of the learning rate parameter η . For sufficiently low values of η , a linear associative network can adaptively form only an approximation to the desired weight matrix $W = BA^+$. There is no known method for adaptively learning the best

choice of the weight matrix $W = BA^+$. Note also that no method is known to adaptively learn even an approximation to the best choice of the weight matrix $W = \hat{BA}^+$ in the case of additive noise in the input pattern vectors. Therefore, in the case of noisy patterns, the best weight matrix has to be computed using the expressions for \hat{A}^+ in terms of the components of the singular value decomposition of A , depending on the estimated noise level in the input patterns. This is obvious from the fact that noise effects can be reduced only when its statistics are observed over several patterns.

4.2.4 Discussion on Pattern Association Problem

Table 4.2 gives a summary of the results of linear associative networks.

Table 4.2 Summary of Results of Linear Associative Networks

Pattern association **problem**

Given a set $\{(\mathbf{a}_i, \mathbf{b}_i)\}$ of L pattern pairs, the objective is to determine the weights of a linear associative network so as to minimize the error between the desired and actual outputs. If $A = [\mathbf{a}_1 \ \mathbf{a}_2 \ \dots \ \mathbf{a}_L]$, $B = [\mathbf{b}_1 \ \mathbf{b}_2 \ \dots \ \mathbf{b}_L]$ and W are the input, output and weight matrices, respectively, then the optimum weights are given by

- (a) $W = BA^T$ for orthogonal set of input vectors
- (b) $W = BA^{-1}$ for linearly independent set of input vectors (full rank square matrix: $r = L = M$)
- (c) $W = BA^+$ for linearly independent set of input vectors (full rank matrix: $r = L < M$)
- (d) $W = BA^+$ for linearly dependent set of input vectors (reduced rank: $r < L \leq M$)
- (e) $W = \hat{BA}^+$ for noisy input vectors

For the cases (a), (b) and (c), the minimum error is zero. For the case (d) the minimum error is determined by the rank of the input matrix. For the case (e) the minimum error is determined by both the rank of the input matrix and the noise power.

Determination of weights by learning

- (a) For orthogonal input vectors the optimum weights $W = BA^T$ can be obtained using Hebb's learning law.
 - (b) For linearly independent or dependent input vectors an approximation to the optimum weights $W = BA^+$ can be learnt using a form of Widrow's learning law.
 - (c) For noisy input vectors there is no known learning law that can provide even an approximation to the optimum weights $W = \hat{BA}^+$.
-

It is **often** useful to allow the processing units in the output layer of the network to have a bias input. In such a case the input matrix A to this layer is augmented with an additional column vector, whose values are always -1 . Addition of this bias term results in a weight matrix W that performs an *affine transformation*. With the affine transformation, any arbitrary rotation, scaling and translation operation on patterns can be handled, whereas linear transformations of the previous associative network can carry out only arbitrary rotation and scaling operations on the input patterns [Hecht-Nielsen, 1990].

In many applications the linkage between the dimensionality (M) of the input data and the number (L) of data items that can be associated and recalled is an unacceptable restriction. By means of coding schemes, the dimensionality of the input data can sometimes be increased artificially, thus allowing more ($L > M$) pairs of items to be associated [Pao, 1989].

But, as we will see in the next section, the dependence of the number of input patterns on the dimensionality of the pattern vector can be removed completely by using nonlinear processing units in the output layer. Thus the *artificial neural networks* can capture the association among the pairs $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \dots, L$, even when the number of input patterns is greater than the dimensionality of the input vectors, i.e., $L > M$. While the constraint of dimensionality on the number of input patterns is removed in the artificial neural networks, some other restrictions will be placed which involve the functional relation between an input and the corresponding output. In particular, the implied mapping between the input and output pattern pairs can be captured by a two layer artificial neural network, provided the mapping function belongs to a linearly separable class. But the number of linearly separable functions decrease rapidly as the dimensionality of the input and output pattern vectors increases. These issues will be discussed in the following section.

4.3 Analysis of Pattern Classification Networks

In an M -dimensional space if a set of points could be considered as input patterns, and if an output pattern, not necessarily distinct from one another, is assigned to each of the input patterns, then the number of distinct output patterns can be viewed as distinct classes or class labels for the input patterns. There is no restriction on the number of input patterns. The input-output pattern vector pairs $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \dots, L$, in this case can be considered as a training set for a *pattern classification* problem. Typically, for pattern classification problems, the output patterns are points in a discrete (normally binary) N -dimensional space. The input patterns are usually from natural sources like speech and hand-printed characters. The input patterns may be corrupted by external noise. Even a noisy

input will be mapped onto one of the distinct pattern classes, and hence the recall displays an accretive behaviour.

4.3.1 Pattern Classification Network: Perceptron

A two layer **feedforward** network with nonlinear (**hard-limiting**) output functions for the units in the output layer can be used to perform the task of pattern classification. The number of units in the input layer corresponds to the dimensionality of the input pattern vectors. The units in the input layer are all linear, as the input layer merely contributes to fan out the input to each of the output units. The number of output units depends on the number of distinct classes in the pattern classification task. We assume for this discussion that the output units are binary. Each output unit is connected to all the input units, and a weight is associated with each connection. Since the output function of a unit is a hard-limiting threshold function, for a given set of input-output patterns, the weighted sum of the input values is compared with the threshold for the unit to determine whether the sum is greater or less than the threshold. Thus in this case a set of inequalities are generated with the given data. Thus there is no unique solution for the weights in this case, as in the case of linear associative network. It is necessary to determine a set of weights to satisfy all the inequalities. Determination of such weights is usually accompanied by means of incremental adjustment of the weights using a learning law.

A detailed analysis of pattern classification networks is presented here assuming M input units and a single binary output unit. The output unit uses a hard-limiting threshold function to decide whether the output signal should be 1 or 0. **Typically**, if the weighted sum of the input values to the output unit exceeds the threshold, the output signal is labelled as **1**, otherwise as **0**. Extension of the analysis for a network consisting of multiple binary units in the output layer is trivial [Zurada, 1992]. Multiple binary output units are needed if the number of pattern classes exceeds 2.

Pattern classification problem: If a subset of the input patterns belong to one class (say class A,) and the remaining subset of the input patterns to another class (say class A₂), then the objective in a pattern classification problem is to determine a set of weights w_1, w_2, \dots, w_M such that if the weighted sum

$$\sum_{i=1}^M w_i a_i > \theta, \text{ then } \mathbf{a} = (a_1, a_2, \dots, a_M)^T \text{ belongs to class A,} \quad (4.33)$$

and if

$$\sum_{i=1}^M w_i a_i \leq \theta, \text{ then } \mathbf{a} = (a_1, a_2, \dots, a_M)^T \text{ belongs to class A}_2, \quad (4.34)$$

Note that the dividing surface between the two classes is given by

$$\sum_{i=1}^M w_i a_i = \theta \tag{4.35}$$

This equation represents a linear hyperplane in the **M-dimensional** space. The hyperplane becomes a point if $M = 1$, a straight line if $M = 2$, and a plane if $M = 3$.

Since the solution of the classification problem involves determining the weights and the threshold value, the classification network can be depicted as shown in Figure 4.2, where the input \mathbf{a}_0

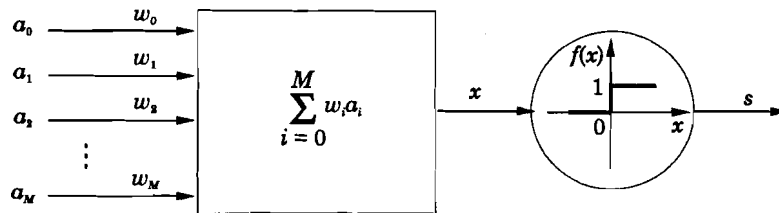


Figure 4.2 A single unit pattern classification network (perceptron).

to the connection involving the threshold value $w_0 = \theta$ is always -1 . Defining the augmented input and weight vectors as $\mathbf{a} = (-1, a_1, \dots, a_M)^T$ and $\mathbf{w} = (w_0, w_1, \dots, w_M)^T$, respectively, the perceptron classification problem can be stated as follows:

- If $\mathbf{w}^T \mathbf{a} > 0$, then \mathbf{a} belongs to class A_1 , and
- if $\mathbf{w}^T \mathbf{a} \leq 0$, then \mathbf{a} belongs to class A_2 .

The equation for the dividing linear hyperplane is $\mathbf{w}^T \mathbf{a} = 0$.

Perceptron learning law: In the above perceptron classification problem, the input space is an **M-dimensional** space and the number of output patterns are two, corresponding to the two classes. Note that we use the $(M + 1)$ -dimensional vector to denote a point in the M -dimensional space, as the \mathbf{a}_0 component of the vector is always -1 . Suppose the subsets A_1 and A_2 of points in the M -dimensional space contain the sample patterns belonging to the classes A_1 and A_2 , respectively. The objective in the perceptron learning is to systematically adjust the weights for each presentation of an input vector belonging to A_1 or A_2 , along with its class identification. The perceptron learning law for the two-class problem may be stated as follows:

$$\begin{aligned} \mathbf{w}(m+1) &= \mathbf{w}(m) + \eta \mathbf{a}, \text{ if } \mathbf{a} \in A_1 \text{ and } \mathbf{w}^T(m) \mathbf{a} \leq 0 \\ &= \mathbf{w}(m) - \eta \mathbf{a}, \text{ if } \mathbf{a} \in A_2 \text{ and } \mathbf{w}^T(m) \mathbf{a} > 0 \end{aligned} \tag{4.36}$$

where the index m is used to denote the learning process at the m th step. The vectors \mathbf{a} and $\mathbf{w}(m)$ are the input and weight vectors, respectively, at the m th step, and η is a positive learning rate parameter. η can be varying at each learning step, although it is assumed as constant in the perceptron learning. Note that no adjustment of weights is made when the input vector is correctly classified. That is,

$$\begin{aligned}\mathbf{w}(m+1) &= \mathbf{w}(m), \text{ if } \mathbf{a} \in A_1 \text{ and } \mathbf{w}^T(m)\mathbf{a} > 0 \\ &= \mathbf{w}(m), \text{ if } \mathbf{a} \in A_2 \text{ and } \mathbf{w}^T(m)\mathbf{a} \leq 0\end{aligned}\quad (4.37)$$

The initial value of the weight vector $\mathbf{w}(0)$ could be random. Figure 4.3 shows an example of the decision boundaries at different

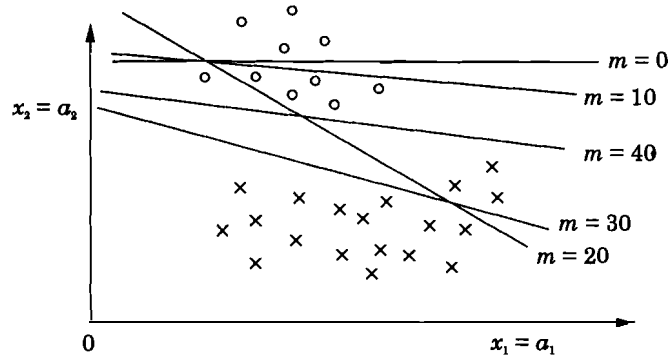


Figure 4.3 Illustration of decision boundaries formed during implementation of perceptron learning for linearly separable classes.

times for a 2-dimensional input vector. The equation of the straight line is given by

$$w_1 a_1 + w_2 a_2 = \theta \quad (4.38)$$

For different values of the weights during learning, the position of the line changes. Note that in this example the two classes can be separated by a straight line, and hence they are called linearly separable classes. On the other hand consider the example of the pattern classification problem in Figure 4.4. In this case the straight line wanders in the plane during learning, and the weights do not converge to a final stable value, as the two classes cannot be separated by a single straight line.

Perceptron convergence theorem: This theorem states that the perceptron learning law converges to a final set of weight values in a finite number of steps, if the classes are linearly separable. The proof of this theorem is as follows:

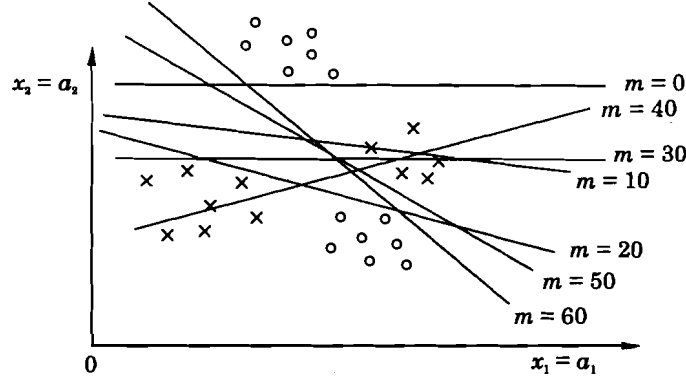


Figure 4.4 Illustration of decision boundaries formed during implementation of perceptron learning for linearly inseparable classes.

Let \mathbf{a} and \mathbf{w} be the augmented input and weight vectors, respectively. Assuming that there exists a solution \mathbf{w}^* for the classification problem, we have to show that \mathbf{w}^* can be approached in a finite number of steps, starting from some initial random weight values. We know that the solution \mathbf{w}^* satisfies the following inequality as per the Eq. (4.37):

$$\mathbf{w}^{*T} \mathbf{a} > \alpha > 0, \quad \text{for each } \mathbf{a} \in A_1 \quad (4.39)$$

where

$$\alpha = \min_{\mathbf{a} \in A_1} (\mathbf{w}^{*T} \mathbf{a})$$

The weight vector is updated if $\mathbf{w}^T(m) \mathbf{a} \leq 0$, for $\mathbf{a} \in A_1$. That is,

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \eta \mathbf{a}(m), \quad \text{for } \mathbf{a}(m) = \mathbf{a} \in A_1, \quad (4.40)$$

where $\mathbf{a}(m)$ is used to denote the input vector at step m . If we start with $\mathbf{w}(0) = \mathbf{0}$, where $\mathbf{0}$ is an all zero column vector, then

$$\mathbf{w}(m) = \eta \sum_{i=0}^{m-1} \mathbf{a}(i) \quad (4.41)$$

Multiplying both sides of Eq. (4.41) by \mathbf{w}^{*T} , we get

$$\mathbf{w}^{*T} \mathbf{w}(m) = \eta \sum_{i=0}^{m-1} \mathbf{w}^{*T} \mathbf{a}(i) > q m \alpha \quad (4.42)$$

since $\mathbf{w}^{*T} \mathbf{a}(i) > \alpha$ according to Eq. (4.39). Using the Cauchy-Schwartz inequality

$$\|\mathbf{w}^{*T}\|^2 \cdot \|\mathbf{w}(m)\|^2 \geq [\mathbf{w}^{*T} \mathbf{w}(m)]^2 \quad (4.43)$$

we get from Eq. (4.42)

$$\|\mathbf{w}(m)\|^2 > \eta^2 m^2 \alpha^2 / \|\mathbf{w}^{*T}\|^2 \quad (4.44)$$

We also have from Eq. (4.40)

$$\begin{aligned}\|\mathbf{w}(m+1)\|^2 &= (\mathbf{w}(m) + \eta \mathbf{a}(m))^T (\mathbf{w}(m) + \eta \mathbf{a}(m)) \\ &= \|\mathbf{w}(m)\|^2 + \eta^2 \|\mathbf{a}(m)\|^2 + 2\eta \mathbf{w}^T(m) \mathbf{a}(m) \\ &\leq \|\mathbf{w}(m)\|^2 + \eta^2 \|\mathbf{a}(m)\|^2\end{aligned}\quad (4.45)$$

since for learning $\mathbf{w}^T(m)\mathbf{a}(m) \leq 0$ when $\mathbf{a}(m) \in A_1$. Therefore, starting from $\mathbf{w}(0) = \mathbf{0}$, we get from Eq. (4.45)

$$\|\mathbf{w}(m)\|^2 \leq \eta^2 \sum_{i=0}^{m-1} \|\mathbf{a}(i)\|^2 < \eta^2 m \beta \quad (4.46)$$

where $\beta = \max_{\mathbf{a}(i) \in A_1} \|\mathbf{a}(i)\|^2$. Combining Eqs. (4.44) and (4.46), we obtain the optimum value of m by solving

$$\frac{m^2 \alpha^2}{\|\mathbf{w}^{*T}\|^2} = \beta m \quad (4.47)$$

or,

$$m = \frac{\beta}{\alpha^2} \|\mathbf{w}^{*T}\|^2 = \frac{\beta}{\alpha^2} \|\mathbf{w}^*\|^2 \quad (4.48)$$

Since β is positive, Eq. (4.48) shows that the optimum weight value can be approached in a finite number of steps using the perceptron learning law.

Alternate proof of the convergence theorem: Assume that a solution vector \mathbf{w}^* exists. Then using the following perceptron behaviour

$$\mathbf{w}^{*T} \mathbf{a} > \alpha > 0 \quad \text{for } \mathbf{a} \in A_1 \quad (4.49)$$

and

$$\mathbf{w}^{*T} \mathbf{a} < -\alpha < 0 \quad \text{for } \mathbf{a} \in A_2 \quad (4.50)$$

we can show that the magnitude of the cosine of the angle ϕ between the weight vectors \mathbf{w}^* and $\mathbf{w}(m)$ is given by

$$|\cos \phi| = \frac{|\mathbf{w}^{*T} \mathbf{w}(m)|}{\|\mathbf{w}^*\| \|\mathbf{w}(m)\|} > \frac{\sqrt{m} \alpha}{\|\mathbf{w}^*\| \sqrt{\beta}} \quad (4.51)$$

where

$$\alpha = \min_{\mathbf{a}} |\mathbf{w}^{*T} \mathbf{a}| \quad (4.52)$$

and

$$\beta = \max_{\mathbf{a}} \|\mathbf{a}\|^2 \quad (4.53)$$

Using the perceptron learning law in Eq. (4.36), and Eqs. (4.49) and (4.52), we get the following:

$$\begin{aligned}\mathbf{w}^{*T}\mathbf{w}(m+1) &= \mathbf{w}^{*T}(\mathbf{w}(m) + \eta \mathbf{a}(m)) \\ &> \mathbf{w}^{*T}\mathbf{w}(m) + qa, \text{ for } w^T(m)a(m) \leq 0\end{aligned}\quad (4.54)$$

Starting from $\mathbf{w}(0) = \mathbf{0}$, we get

$$\mathbf{w}^{*T}\mathbf{w}(m) > m\eta\alpha, \text{ for } w^T(m)a(m) \leq 0, \text{ and } \mathbf{a}(m) \in A_1 \quad (4.55)$$

Likewise, using the perceptron learning law in Eq. (4.36), and Eqs. (4.50) and (4.52), we get

$$\begin{aligned}\mathbf{w}^{*T}\mathbf{w}(m+1) &= \mathbf{w}^{*T}(\mathbf{w}(m) - \eta \mathbf{a}(m)) \\ &< \mathbf{w}^{*T}\mathbf{w}(m) - \eta\alpha, \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) > 0\end{aligned}$$

Starting from $\mathbf{w}(0) = \mathbf{0}$, we get

$$\mathbf{w}^{*T}\mathbf{w}(m) < -mqa, \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) > 0, \text{ and } \mathbf{a}(m) \in A_2 \quad (4.56)$$

That is

$$|\mathbf{w}^{*T}\mathbf{w}(m)| > mqa, \text{ for } w^T(m)a(m) > 0, \text{ and } \mathbf{a}(m) \in A_2 \quad (4.57)$$

Therefore from Eqs. (4.55) and (4.57), we get

$$|\mathbf{w}^{*T}\mathbf{w}(m)| > mqa, \text{ for all } a \quad (4.58)$$

Similarly, using Eq. (4.36), we get

$$\begin{aligned}\|\mathbf{w}(m+1)\|^2 &= (\mathbf{w}(m) + \eta \mathbf{a}(m))^T(\mathbf{w}(m) + \eta \mathbf{a}(m)) \\ &< \|\mathbf{w}(m)\|^2 + 2\eta \mathbf{w}^T(m)\mathbf{a}(m) + \eta^2\beta \\ &< \|\mathbf{w}(m)\|^2 + \eta^2\beta, \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) \leq 0\end{aligned}\quad (4.59)$$

and

$$\begin{aligned}\|\mathbf{w}(m+1)\|^2 &= (\mathbf{w}(m) - \eta \mathbf{a}(m))^T(\mathbf{w}(m) - \eta \mathbf{a}(m)) \\ &< \|\mathbf{w}(m)\|^2 - 2\eta \mathbf{w}^T(m)\mathbf{a}(m) + \eta^2\beta \\ &< \|\mathbf{w}(m)\|^2 + \eta^2\beta, \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) > 0\end{aligned}\quad (4.60)$$

Starting from $\mathbf{w}(0) = \mathbf{0}$, we get for both (4.59) and (4.60)

$$\|\mathbf{w}(m)\|^2 < m\eta^2\beta \quad (4.61)$$

Therefore, from Eqs. (4.51), (4.58) and (4.61), we get

$$1 \geq \frac{|\mathbf{w}^{*T}\mathbf{w}(m)|}{\|\mathbf{w}^*\| \|\mathbf{w}(m)\|} > \frac{\sqrt{m}\alpha}{\|\mathbf{w}^*\| \sqrt{\beta}} \quad (4.62)$$

$$\frac{\sqrt{m}\alpha}{\|\mathbf{w}^*\| \sqrt{\beta}} < 1 \quad (4.63)$$

$$m < \frac{\beta}{\alpha^2} \|\mathbf{w}^*\|^2 \quad (4.64)$$

Discussion on the convergence theorem: The number of iterations for convergence depends on the relation between $\mathbf{w}(0)$ and \mathbf{w}^* . Normally the initial value $\mathbf{w}(0)$ is set to 0. The initial setting of the weight values does not affect the proof of the perceptron convergence theorem.

The working of the perceptron learning can be viewed as follows: At the $(m + 1)$ th iteration we have

$$\begin{aligned} \mathbf{w}(m + 1) &= \mathbf{w}(m) + \eta \mathbf{a}(m), \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) \leq 0 \\ &\text{and } \mathbf{a}(m) \in A_1 \end{aligned} \quad (4.65)$$

From this we get

$$\mathbf{w}^T(m + 1)\mathbf{a}(m) = \mathbf{w}^T(m)\mathbf{a}(m) + \eta \mathbf{a}^T(m)\mathbf{a}(m) \quad (4.66)$$

Notice that if $\mathbf{w}^T(m)\mathbf{a}(m) \leq 0$, then $\mathbf{w}^T(m + 1)\mathbf{a}(m) > 0$, provided η is chosen as the smallest positive real number (< 1) such that

$$\eta \mathbf{a}^T(m)\mathbf{a}(m) > |\mathbf{w}^T(m)\mathbf{a}(m)| \quad (4.67)$$

Thus the given pattern $\mathbf{a}(m)$ is classified correctly if it is presented to the perceptron with the new weight vector $\mathbf{w}(m + 1)$. The weight vector is adjusted to enable the pattern to be classified correctly.

The perceptron convergence theorem for the two class problem is applicable for both binary $\{0, 1\}$ and bipolar $\{-1, +1\}$ input and output data. By considering a two-class problem each time, the perceptron convergence theorem can be proved for a multiclass problem as well. The perceptron learning law and its proof of convergence are applicable for a single layer of nonlinear processing units, also called a single layer perceptron. Note that convergence takes place provided an optimal solution \mathbf{w}^* exists. Such a solution exists for a single layer perceptron, only if the given classification problem is *linearly separable*. In other words, the perceptron learning law converges to a solution only if the class boundaries are separable by linear hyperplanes in the M-dimensional input pattern space.

Perceptron learning as gradient descent: The perceptron learning law in Eq. (4.36) can also be written as

$$\mathbf{w}(m + 1) = \mathbf{w}(m) + \eta (b(m) - s(m)) \mathbf{a}(m) \quad (4.68)$$

where $b(m)$ is the desired output, which for the binary case is given by

$$b(m) = 1, \text{ for } \mathbf{a}(m) \in A_1, \quad (4.69)$$

$$= 0, \text{ for } \mathbf{a}(m) \in A_2 \quad (4.70)$$

and $s(m)$ is the actual output for the input vector $\mathbf{a}(m)$ to the perceptron. The actual output is given by

$$s(m) = 1, \text{ if } \mathbf{w}^T(m)\mathbf{a}(m) > 0 \quad (4.71)$$

$$= 0, \text{ if } \mathbf{w}^T(m)\mathbf{a}(m) \leq 0 \quad (4.72)$$

From Eq. (4.68) we note that if $\mathbf{s}(m) = \mathbf{b}(m)$, then $\mathbf{w}(m+1) = \mathbf{w}(m)$, i.e., no correction takes place. On the other hand, if there is an error, $\mathbf{s}(m) \neq \mathbf{b}(m)$, then the update rule given by (4.68) is same as the update rule given in Eq. (4.36).

Note that Eq. (4.68) is also valid for a bipolar output function, i.e., when $\mathbf{s}(m) = f(\mathbf{w}^T(m)\mathbf{a}(m)) = \pm 1$. Therefore Eq. (4.68) can be written as

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \eta e(m) \mathbf{a}(m) \quad (4.73)$$

where $e(m) = \mathbf{b}(m) - \mathbf{s}(m)$ is the error signal. If we use the instantaneous correlation (product) between the output error $e(m)$ and the activation value $x(m) = \mathbf{w}^T(m)\mathbf{a}(m)$ as a measure of performance $E(m)$, then

$$E(m) = -e(m) x(m) = -e(m) \mathbf{w}^T(m)\mathbf{a}(m) \quad (4.74)$$

The negative derivative of $E(m)$ with respect to the weight vector $\mathbf{w}(m)$ can be defined as the negative gradient of $E(m)$ and is given by

$$-\frac{\partial E(m)}{\partial \mathbf{w}(m)} = e(m) \mathbf{a}(m) \quad (4.75)$$

Thus the weight update $\eta e(m) \mathbf{a}(m)$ in the perceptron learning **il**, Eq. (4.73) is proportional to the negative gradient of the performance measure $E(m)$.

Perceptron representation problem: Convergence in the perceptron learning takes place only if the pattern classes are linearly separable in the pattern space. Linear separability requires that the convex hulls of the pattern sets of the classes are disjoint. A convex hull of a pattern set \mathbf{A}_0 is the smallest convex set in \mathcal{R}^M that contains \mathbf{A}_0 . A convex set is a set of points in the M-dimensional space such that a line joining any two points in the set lies entirely in the region enclosed by the set. For linearly separable classes, the perceptron convergence theorem ensures that the final set of weights will be reached in a finite number of steps. These weights define a linear hyperplane separating the two classes. But in practice the number of linearly separable functions will decrease rapidly as the dimension of the input pattern space is increased [Cameron, 1960; Muroga, 1971]. Table 4.3 shows the number of linearly separable functions for a two-class problem with binary input patterns for different dimensions of the input pattern space. For binary pattern classification problems ($M = 2$), there are 14 functions which are linearly separable. The problem in **Figure 4.5a** is one of the linearly separable functions. There are two functions which are linearly inseparable, one of which is shown in **Figure 4.5b**. These linearly inseparable problems do not lead to convergence of weights through the perceptron learning

law, indicating that these problems are not *representable* by a single layer of perceptron discussed so far.

Table 4.3 . Number of Linearly Separable Functions for a Two-class Problem

Dimension of input data M	Number of possible functions 2^{2^M}	Number of linearly separable functions
1	4	4
2	16	14
3	256	104
4	65536	1882
5	$- 4.3 \times 10^9$	94572
6	$- 1.8 \times 10^{19}$	15028134

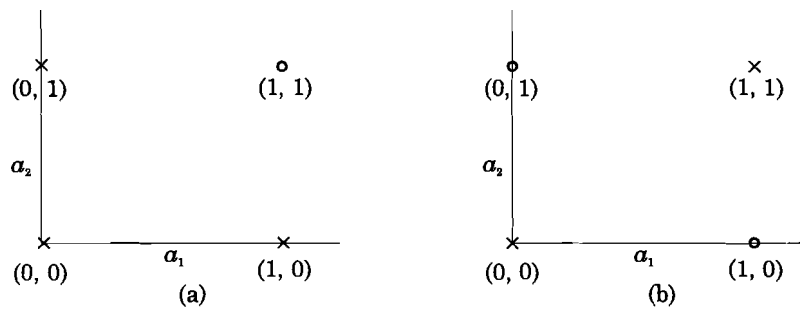


Figure 4.5 Examples of (a) linearly separable and (b) linearly inseparable classification problems. The classes are indicated by 'x' and 'o'.

4.3.2 Linear Inseparability: Hard Problems

A two-layer feedforward network with hard-limiting threshold units in the output layer can solve linearly separable pattern classification problems. This is also called a single layer perceptron, as there is only one layer of nonlinear units. There are many problems which are not linearly separable, and hence are not representable by a single layer perceptron. These unrepresentable problems are called *hard problems*. Some of these problems are illustrated using the perceptron model consisting of sensory units, association units and the output layer as shown in Figure 4.6. The output unit of the perceptron computes a logical predicate, based on the information fed to it by the association units connected to it. The association units form a family of local predicates, computing a set of local properties or features. The family of local predicates are looking at a 'retina', which consists of points on a plane, **which** in the figure corresponds to the sensory input. In the simple 'linear' perceptron the output unit looks at the local predicates from the association units, takes their weighted sum, compares with a threshold, and then responds with a value for the overall logical predicate, which is either true or false. If it were

Analysis of Pattern Classification Networks

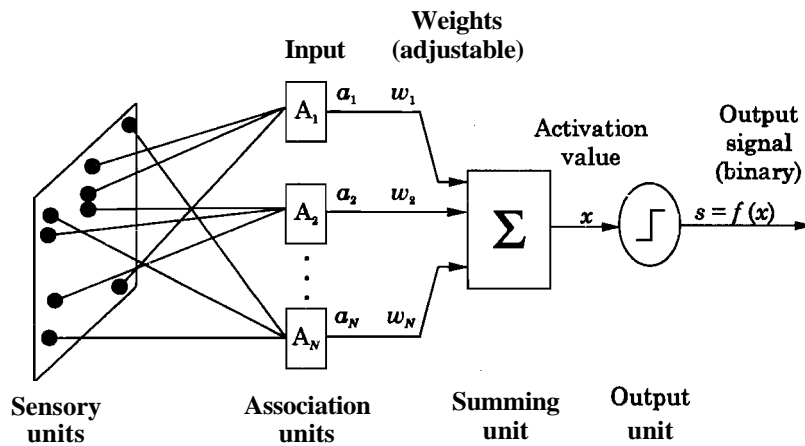


Figure 4.6 Rosenblatt's perceptron model of a neuron.

possible for the local predicates to look at every point in the entire retina, any possible logical predicate could be computed. This would be impractical, since the number of possible patterns on the retina grows exponentially with the size of the retina [Minsky and Papert, 1990]. Two important limitations on the local predicates are: order-limited, where only a certain maximum order of retinal points could be connected to the local decision unit computing the local predicate, and diameter-limited, where only a geometrically restricted region of retina could be connected to the local predicate. The order-limited perceptron cannot compute the parity problem examples shown in Figure 4.7, where images with an even number of distinct

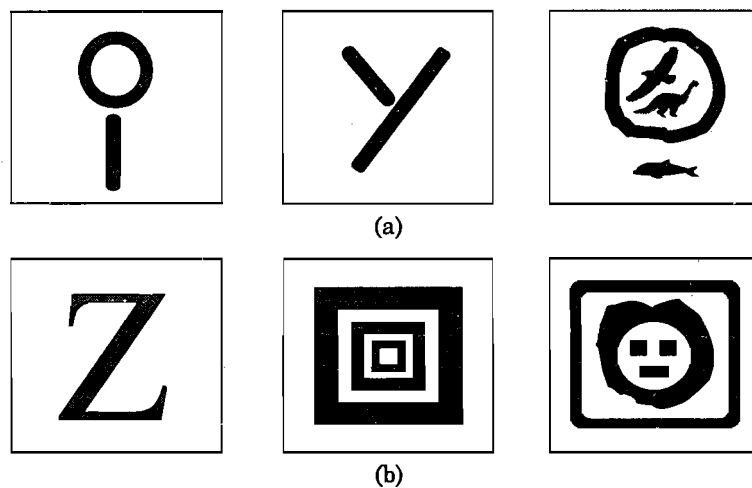


Figure 4.7 A parity problem illustrating the order-limited perceptron: (a) Even parity and (b) Odd parity.

unconnected patterns (Figure 4.7a) should produce an output 1, otherwise the output for the odd number of patterns in Figure 4.7b should be 0. Likewise the diameter-limited perceptron cannot handle the connectedness problem examples shown in Figure 4.8, where to detect connectedness the output of the perceptron should be 1 if there is only one **connected** pattern as in Figure 4.8a, otherwise it should be 0 for patterns shown in Figure 4.8b [Aleksander and Morton, 1990].

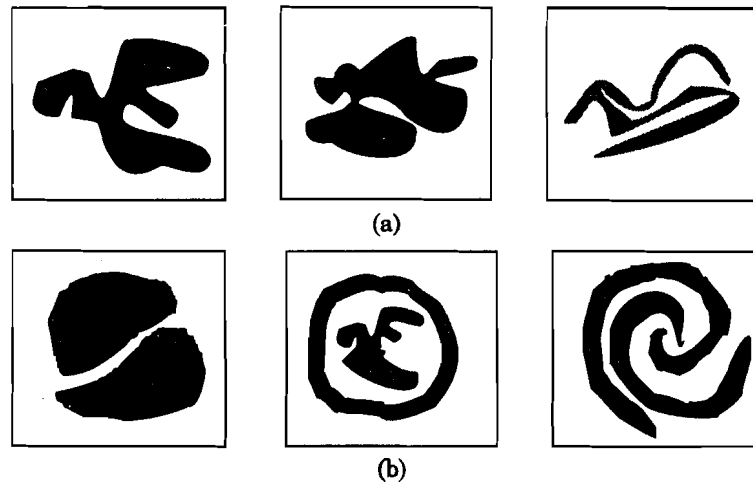


Figure 4.8 A connectedness problem illustrating the diameter-limited perceptron: (a) Connected class and (b) Disconnected class.

4.3.3 Geometrical Interpretation of Hard Problems: Multilayer Perceptron

In this section the problem of pattern classification and the performance of feedforward neural networks are discussed in geometric terms. A pattern classification problem can be viewed as determining the **hypersurfaces** separating the multidimensional patterns belonging to different classes. For convenience throughout this section we consider a **2-dimensional** pattern space. If the pattern classes are linearly separable then the hypersurfaces reduce to straight lines as shown in Figure 4.9. A two-layer network consisting of two input units and N output units can produce N distinct lines in the pattern space. These lines can be used to separate different classes, provided the regions formed by the pattern classification problem are linearly separable. As mentioned earlier (See Table 4.3), linearly separable problems are in general far fewer among all possible problems, especially as the dimensionality of the input space increases. If the outputs of the second layer **are** combined by a set of units forming another layer, then it can be shown that any convex region can be formed by the separating surfaces [Lippmann, 1987; Wasserman, 1989]. A convex region is one in which a line joining any

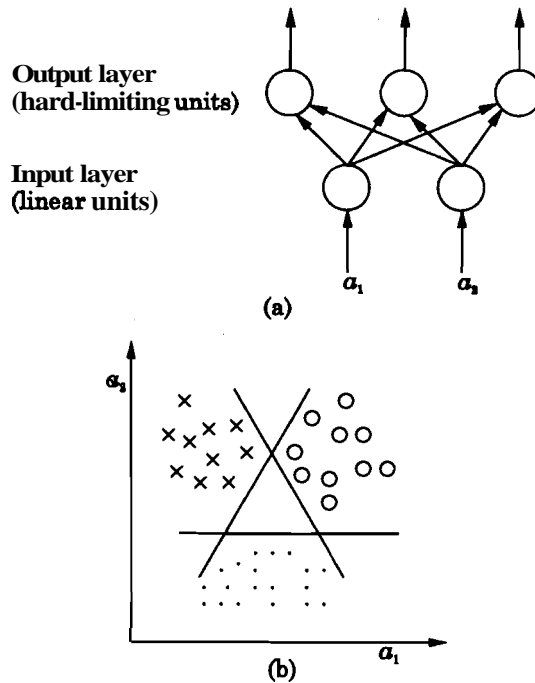


Figure 4.9 An example of linearly separable classes: (a) Network and (b) Linearly separable classes.

two points is entirely confined to the region itself. Figure 4.10b illustrates the regions that can be created by a three layer network. In this case the number of units in the second layer determines the shape of the dividing surface. The number of units in the third layer decides the number of classes. It can be seen that the three-layer network (Fig. 4.10b) is not general enough, as it is not guaranteed that the class regions in the pattern space form convex regions in **all** cases. In fact one could have a situation as shown for the classes with meshed regions, where the desired classes are enclosed by complicated nonconvex regions. Note that intersection of linear hyperplanes in the three layer network **can** only produce convex surfaces.

However, intersection of the convex regions may produce any nonconvex region also. Thus adding one more layer of units to combine the outputs of the third layer can yield surfaces which can separate even the nonconvex regions of the type shown in Figure 4.10c. In fact it can be shown that a four-layer network with the input layer consisting of linear units, and the other three layers consisting of hard-limiting nonlinear units, can perform any complex pattern classification tasks. Thus all the hard problems mentioned earlier can be handled by a multilayer feedforward neural network, with nonlinear units. Such a network is also called a multilayer perceptron. Note that the two-layer network in Figure 4.10a is a

single-layer perceptron, and the three-layer and four-layer networks in the Figures 4.10b and 4.10c are two-layer and three-layer perceptrons, respectively.


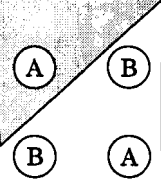
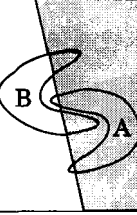

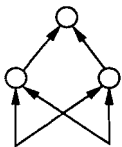
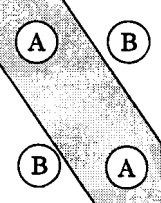
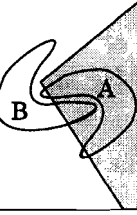

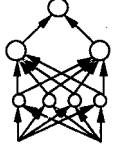
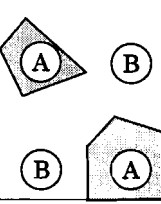
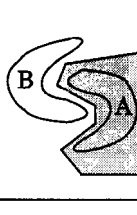
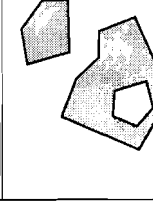
Structure	Types of decision regions	Exclusive OR problem	Classes with meshed regions	Most general region shapes
Two-layer network 	Half Plane (Bounded by hyperplane)			
Three-layer network 	Convex (Open or closed regions)			
Four-layer network 	Arbitrary (Complexity limited by number of neurons)			

Figure 4.10 Geometrical interpretation of pattern classification. The figure shows decision regions for different layers of perceptron networks. [Adapted from Lippmann, 1987].

The above discussion is focussed primarily on a multilayer perceptron network with units having hard-limiting nonlinear output functions. Similar behaviour is expected from a multilayer feed-forward neural network when the output functions of the units are continuous nonlinear functions, such as **sigmoid** functions. In these cases the decision regions are typically bounded by smooth surfaces instead of linear hyperplanes, and hence geometrical visualization and interpretation is difficult.

Table 4.4 gives a summary of the discussion on the perceptron network. The main difficulty with a multilayer perceptron network is that it is not straightforward to adjust the weights leading to the units in the intermediate layers, since the desired output values of the units in these layers are not known. The perceptron learning uses the knowledge of the error between the desired output and the actual output to adjust the weights. From the given data only the desired

Table 4.4 Summary of Issues in Perceptron Learning

<p>1. Perceptron network</p> <p>Weighted sum of the input to a unit with hard-limiting output function</p> <p>2. Perceptron classification problem</p> <p>For a two class (A_1 and A_2) problem, determine the weights (\mathbf{w}) and threshold (θ) such that</p> $\mathbf{w}^T \mathbf{a} - \theta > 0, \text{ for } \mathbf{a} \in A_1 \text{ and } \mathbf{w}^T \mathbf{a} - \theta \leq 0, \text{ for } \mathbf{a} \in A_2.$ <p>3. Perceptron learning law</p> <p>The weights are determined in an iterative manner using the following learning law at the $(m + 1)^{\text{th}}$ iteration:</p> $\mathbf{w}(m + 1) = \mathbf{w}(m) + \eta \mathbf{a}(m), \text{ for } \mathbf{w}^T(m) \mathbf{a}(m) \leq \theta \text{ and } \mathbf{a}(m) \in A_1$ $= \mathbf{w}(m) - \eta \mathbf{a}(m), \text{ for } \mathbf{w}^T(m) \mathbf{a}(m) > \theta \text{ and } \mathbf{a}(m) \in A_2$ <p>where η is a (positive) learning rate parameter.</p> <p>4. Perceptron learning as gradient descent</p> <p>The perceptron learning law can be rewritten as a single equation:</p> $\mathbf{w}(m + 1) = \mathbf{w}(m) + \eta e(m) \mathbf{a}(m), \text{ where } e(m) = b(m) - s(m).$ <p>Denoting</p> $\Delta \mathbf{w}(m) = \eta e(m) \mathbf{a}(m),$ <p>we have</p> $\Delta \mathbf{w}(m) = \eta \frac{-\partial E(m)}{\partial \mathbf{w}(m)}$ <p>where $E(m) = -e(m) \mathbf{w}^T(m) \mathbf{a}(m)$.</p> <p>5. Perceptron convergence theorem</p> <p>The perceptron learning law converges in a finite number of steps, provided that the given classification problem is representable.</p> <p>6. Perceptron representation problem</p> <p>A classification problem is representable by a single layer perceptron if the classes are linearly separable, i.e., separable by linear hyperplanes in the input feature space. Classification problems that are not linearly separable are called hard problems.</p> <p>7. Multilayer perceptron</p> <p>Any pattern classification problem, including the hard problems, can be represented by a multilayer perceptron network.</p>	<hr/>
---	-------

output values of the units in the final output layer are known. Thus, although a multilayer perceptron network can handle hard problems, the problem of learning or training such a network, called *hard learning* problem, remains. This problem is discussed in detail in the following section.

4.4 Analysis of Pattern Mapping Networks

4.4.1 Pattern Mapping Problem

If a set of input-output pattern pairs is given corresponding to an

arbitrary function transforming a point in the M -dimensional input pattern space to a point in the N -dimensional output pattern space, then the problem of capturing the implied functional relationship is called a *mapping* problem. The network that accomplishes this task is called a mapping network. Since no restriction such as linear separability is placed on the set of input-output pattern pairs, the pattern mapping problem is a more general case of pattern classification problem.

Note that the objective in the **pattern** mapping problem is to capture the implied function, *i.e.*, the generalization implied in the given input-output pattern pairs. This can also be viewed as an approximation of the function from a given data. For a complex system with multiple (M) inputs and multiple (N) outputs, if several input-output pattern pairs are collected during experimentation, then the objective in the **pattern** mapping problem is to capture the system characteristics from the observed data. For a new input, the captured system is expected to produce an output close to the one that would have been obtained by the real system. In terms of function approximation, the approximate mapping system should give an output which is close to the values of the real function for inputs close to the current input used during learning. Note that the approximate system does not produce strictly an interpolated output, as the function finally captured during learning may not fit any of the points given in the training set. This is illustrated in Figure 4.11.

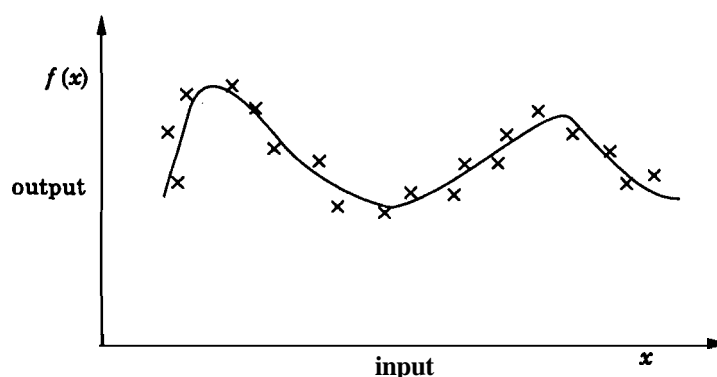


Figure 4.11 Function approximation in pattern mapping problem.

4.4.2 Pattern Mapping Network

Earlier we have seen that a multilayer feedforward neural network with at least two intermediate layers in addition to the input and output layers can perform any pattern classification task. Such a network can also perform a pattern mapping task. The additional layers are called hidden layers, and the number of units in the hidden layers depends on the nature of the mapping problem. For any

arbitrary problem, generalization may be difficult. With a sufficiently large size of the network, it is possible to (virtually) store all the input-output pattern pairs given in the training set. Then the network will not be performing the desired mapping, because it will not be capturing the implied functional relationship between the given input-output pattern pairs.

Except in the input layer, the units in the other layers must be nonlinear in order to provide generalization capability for the network. In fact it can be shown that, if all the units are linear, then a **multilayer** network can be reduced to an equivalent two-layer network with a set of $N \times M$ weights.

Let W_1 , W_2 and W_3 be the weight matrices of appropriate sizes between the input layer and the first hidden layer, the first hidden layer and the second hidden layer, and the second hidden layer and the output layer, respectively. Then if all the units are linear, the output and input patterns are related by the weight matrix containing $N \times M$ weight elements. That is,

$$W_{N \times M} = W_3 W_2 W_1 \quad (4.76)$$

As can be seen easily, such a network reduces to a linear associative network. But if the units in the output layer are nonlinear, then the network is limited by the linear separability constraint on the function relating the input-output pattern pairs. If the units in the hidden layers and in the output layer are nonlinear, then the number of unknown weights depend on the number of units in the hidden layers, besides the number of units in the input and output layers. The pattern mapping problem involves determining these weights, given a training set consisting of input-output pattern pairs. We need a systematic way of updating these weights when each input-output pattern pair is presented to the network. In order to do this updating of weights in a supervisory mode, it is necessary to know the desired output for each unit in the hidden and output layers. Once the desired output is known, the error, *i.e.*, the difference between the desired and actual outputs from each unit may be used to guide the updating of the weights leading to the unit from the units in the previous layer. We know the desired output only for the **units** in the final output layer, and not for the units in the hidden layers. Therefore a straightforward application of a learning rule, that depends on the difference between the desired and the actual outputs, is not feasible in this case. The problem of updating the weights in this case is called a hard learning problem.

The hard learning problem is solved by using a differentiable nonlinear output function for each unit in the hidden and output layers. The corresponding learning law is based on propagating the error from the output layer to the hidden layers for updating the weights. This is an error correcting learning law, also called the

generalized delta rule. It is based on the **principle** of gradient descent along the error surface.

Appendix C gives the background information needed for understanding the gradient descent methods. Table 4.5 gives a summary of the gradient search methods discussed in the Appendix C. In the following section we derive the generalized delta rule applicable for a **multilayer** feedforward network with nonlinear units.

Table 4.5 Summary of Basic Gradient Search Methods

1. Objective

Determine the optimal set of weights for which the expected error $E(\mathbf{w})$ between the desired and actual outputs is minimum.

For a linear network the error surface is a quadratic function of the weights

$$E(\mathbf{w}) = \mathcal{E}[e^2] = E_{\min} + (\mathbf{w} - \mathbf{w}^*)^T R (\mathbf{w} - \mathbf{w}^*)$$

The **optimum weight** vector \mathbf{w}^* is given by

$$\mathbf{w}^* = \mathbf{w} - \frac{1}{2} R^{-1} \nabla, \text{ where } \nabla = dE/d\mathbf{w}$$

and R is the autocorrelation matrix of the input data.

2. Gradient Search Methods

We can write the equation for adjustment of weights as

$$\mathbf{w}(m+1) = \mathbf{w}(m) - \frac{1}{2} R^{-1} \nabla_m$$

- If R and ∇_m are known exactly, then the above adjustment **gives \mathbf{w}^*** in one step starting from any initial weights $\mathbf{w}(m)$.
- If R and ∇_m are known only approximately, then the optimum weight vector can be obtained in an iterative manner by writing

$$\mathbf{w}(m+1) = \mathbf{w}(m) - \eta R^{-1} \nabla_m,$$

where $\eta < 1/2$ for convergence. This is Newton's method. The error moves approximately along the path from $\mathbf{w}(m)$ to \mathbf{w}^* . Here η is a dimensionless quantity.

- If the weights are adjusted in the direction of the negative gradient at each step, it becomes method **σ** steepest descent.

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \mu (-\nabla_m),$$

where $\mu < 1/(2\lambda_{\max})$ for convergence and λ_{\max} is the largest eigenvalue of R. The learning rate parameter μ has the dimensions of inverse of signal power. Here convergence is slower than in the Newton's method.

- In general, the gradient cannot be computed, but can only be estimated. Hence convergence of the gradient descent methods is not guaranteed. The estimate depends on our knowledge of the error surface.

Table 4.5 (Cont.)

3. Nature of error surface

- Error surface may not be quadratic if R is to be estimated from a small set of samples.
- Error surface is not quadratic for instantaneous measurement of error.
- Error surface is also not quadratic if the **processing** units are nonlinear.
- Error surface is not predictable for **nonstationary** input data, **since** R will be varying with time.

4. Estimation of gradient

- Derivative measurement: Uses general **knowledge** of the error surface.
- **Instantaneous** measurement (linear units): Uses **specific** knowledge of the error surface.
LMS algorithm. Leads to convergence in the mean (**stochastic** gradient descent).
- Instantaneous measurement (nonlinear units): **Uses** specific knowledge of the error surface.
Delta rule. No guarantee of **convergence** even in the mean **as** in the LMS algorithm.

4.4.3 Generalized Delta Rule: Backpropagation learning

The objective is to develop a learning algorithm for a **multilayer** feedforward neural network, so that the network can be trained to capture the mapping implicit in the given set of input-output pattern pairs. The approach to be followed is basically a gradient descent along the error surface to arrive at the optimum set of weights. The error is defined as the squared difference between the desired output (i.e., given output pattern) and the actual output obtained at the output layer of the network due to application of an input pattern from the given input-output pattern pair. The output is calculated using the current setting of the weights in all the layers. The optimum weights may be obtained if the weights are adjusted in such a way that the gradient descent is made along the total error surface. But the desirable characteristic of any learning law is to specify the incremental update of the weights of the network for each presentation of an input-output pattern pair. While this may result in a suboptimal solution, in most cases of practical significance the result is acceptable.

A learning law, called generalized delta rule or backpropagation law, is derived in this section [Werbos, 1974; Rumelhart et al, 1986a]. Let $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \dots, L$ be the set of training pattern pairs. It is not necessary to have all the training data set at one time, nor the training data set to be a finite set. The objective is to determine the weight update for each presentation of an input-output pattern pair.

Since the given data may be used several times during training, let us use the index m to indicate the presentation step for the training pair at step m .

For training a multilayer feedforward neural network, we use the following estimate of the gradient descent along the error surface to determine the increment in the weight connecting the units j and i :

$$\Delta w_{ij}(m) = -\eta \frac{\partial E(m)}{\partial w_{ij}}, \quad (4.77)$$

where $\eta > 0$ is a learning rate parameter, which may also vary for each presentation of the training pair. The weight update is given by

$$w_{ij}(m+1) = w_{ij}(m) + \Delta w_{ij}(m) \quad (4.78)$$

The generalized delta rule to be derived below consists of deriving expressions for Δw_{ij} for the connections at different layers. Let us consider the multilayer **feedforward** neural network given in **Figure 4.12**. The network consists of three layers of units, the first

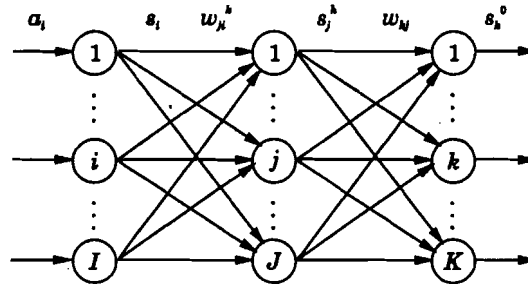


Figure 4.12 A three layer feedforward neural network.

layer has I linear input units indexed by i , the second layer has J nonlinear units indexed by j , and the third layer has K nonlinear units indexed by k . For simplicity only one layer (the second layer) of hidden units is considered here. Extension of learning to a network consisting of several hidden layers is trivial.

Since the input vector $\mathbf{a}(m)$ is given at the input layer and the desired output $\mathbf{b}(m)$ is available only at the output layer, the error between the desired output vector $\mathbf{b}(m)$ and the actual output vector $\mathbf{b}'(m)$ is available only at the output layer. Using this error it is necessary to adjust the **weights** (w_{ji}^h) from the input units to the hidden units, and the weights (w_{kj}) from the hidden units to the output units.

Let $(\mathbf{a}(m), \mathbf{b}(m))$ be the current sample of the function mapping the input space to the output space $\mathcal{R}^I \rightarrow \mathcal{R}^K$. Let $\mathbf{b}'(m)$ be the actual output of the network for the input $\mathbf{a}(m)$ at the step m . The mean squared error at the m th step is given by

$$E(m) = \frac{1}{2} \sum_{k=1}^K [b_k(m) - b'_k(m)]^2 = \frac{1}{2} \sum_{k=1}^K [b_k(m) - s_k^o]^2 \quad (4.79)$$

$$= \frac{1}{2} \sum_{k=1}^K [b_k(m) - f_k^o(x_k^o)]^2, \quad (4.80)$$

where

$$x_k^o = \sum_{j=1}^J w_{kj} s_j^h \quad (4.81)$$

$$s_j^h = f_j^h(x_j^h) \quad (4.82)$$

$$x_j^h = \sum_{i=1}^I w_{ji}^h s_i \quad (4.83)$$

$$s_i = x_i = a_i(m). \quad (4.84)$$

The superscript 'o' refers to the output units quantities, the superscript 'h' refers to the hidden units quantities, and a_i , x_i , and s_i refer to the input, activation and output values for the unit i , respectively. For the weights leading to the units in the output layer:

$$\Delta w_{kj}(m) = -\eta \frac{\partial E(m)}{\partial w_{kj}} \quad (4.85)$$

$$\frac{\partial E(m)}{\partial w_{kj}} = \frac{1}{2} \frac{\partial}{\partial w_{kj}} \left[b_k - f_k^o \left(\sum_{j=1}^J w_{kj} s_j^h \right) \right]^2 \quad (4.86)$$

$$= -(b_k - f_k^o) \dot{f}_k^o s_j^h \quad (4.87)$$

$$= -\delta_k^o s_j^h, \quad (4.88)$$

where $\delta_k^o = (b_k - f_k^o) \dot{f}_k^o$. Here the iteration index is omitted in all the functions and variables on the right hand side for convenience. Therefore

$$\Delta w_{kj}(m) = \eta \delta_k^o s_j^h \quad (4.89)$$

and

$$w_{kj}(m+1) = w_{kj}(m) + \Delta w_{kj}(m) \quad (4.90)$$

$$= w_{kj}(m) + \eta \delta_k^o s_j^h. \quad (4.91)$$

For the weights leading to the units in the hidden layer:

$$\Delta w_{ji}^h(m) = -\eta \frac{\partial E(m)}{\partial w_{ji}^h} \quad (4.92)$$

$$\frac{\partial E(m)}{\partial w_{ji}^h} = - \sum_{k=1}^K (b_k - f_k^o) \frac{\partial f_k^o \left(\sum_{j=1}^J w_{kj} s_j^h \right)}{\partial w_{ji}^h} \quad (4.93)$$

$$= - \sum_{k=1}^K (b_k - f_k^o) \dot{f}_k^o w_{kj} \frac{\partial s_j^h}{\partial w_{ji}^h}, \quad (4.94)$$

Since $s_j^h = f_j^h(x_j^h)$, we get

$$\frac{\partial s_j^h}{\partial w_{ji}^h} = \dot{f}_j^h \frac{\partial x_j^h}{\partial w_{ji}^h}$$

Since $x_j^h = \sum_{i=1}^I w_{ji}^h s_i$, we get

$$\frac{\partial x_j^h}{\partial w_{ji}^h} = s_i$$

Therefore

$$\frac{\partial E(m)}{\partial w_{ji}^h} = - \sum_{k=1}^K (b_k - f_k^o) \dot{f}_k^o w_{kj} \dot{f}_j^h s_i \quad (4.95)$$

$$= - \delta_j^h s_i \quad (4.96)$$

where

$$\delta_j^h = \dot{f}_j^h \sum_{k=1}^K w_{kj} \delta_k^o \quad (4.97)$$

Hence

$$\Delta w_{ji}^h(m) = \eta \delta_j^h s_i = \eta \delta_j^h a_i(m) \quad (4.98)$$

since $s_i = x_i = a_i(m)$. Therefore

$$\begin{aligned} w_{ji}^h(m+1) &= w_{ji}^h(m) + \Delta w_{ji}^h(m) \\ &= w_{ji}^h(m) + \eta \delta_j^h a_i(m) \end{aligned} \quad (4.99)$$

where $\delta_j^h = \dot{f}_j^h \sum_{k=1}^K w_{kj} \delta_k^o$ represents the error propagated back to the output of the hidden units from the next layer, hence the name *backpropagation* for this learning algorithm. Table 4.6 gives a summary of the backpropagation learning algorithm.

4.4.4 Discussion on Backpropagation Law

There are several issues which are important for understanding and implementing the **backpropagation** learning in practice [Haykin, 1994; Russo, 1991; Guyon, 1991; Hush and Horne, 1993; Werbos, 1994]. A summary of the issues is given in Table 4.7. A few of these issues will be discussed in this section.

Table 4.6 Backpropagation Algorithm (Generalized Delta Rule)

Given a set of input-output patterns $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \dots, L$,
 where the l th input vector $\mathbf{a}_l = (a_{l1}, a_{l2}, \dots, a_{lI})^T$ and the l th output vector $\mathbf{b}_l = (b_{l1}, b_{l2}, \dots, b_{lK})^T$.
 Assume only one hidden layer and initial setting of weights to be arbitrary.
 Assume input layer with only linear units.
 Then the output signal is equal to the input activation value for each of these **units**. Let η be the learning rate parameter.

Let $\mathbf{a} = \mathbf{a}(m) = \mathbf{a}_l$ and $\mathbf{b} = \mathbf{b}(m) = \mathbf{b}_l$.

Activation of unit i in the input layer, $x_i = a_i(m)$

Activation of unit j in the hidden layer, $x_j^h = \sum_{i=1}^I w_{ji}^h x_i$

Output signal from the j th unit in the **hidden layer**, $s_j^h = f_j^h(x_j^h)$

Activation of unit k in the output layer, $x_k^o = \sum_{j=1}^J w_{kj} s_j^h$

Output signal from unit k in the output layer, $s_k^o = f_k^o(x_k^o)$

Error term for the k th output unit, $\delta_k^o = (b_k - s_k^o) f_k^o$

Update weights on output layer, $w_{kj}(m+1) = w_{kj}(m) + \eta \delta_k^o s_j^h$

Error term for the j th hidden unit, $\delta_j^h = f_j^h \sum_{k=1}^K \delta_k^o w_{kj}$

Update the weights on the hidden layer, $w_{ji}^h(m+1) = w_{ji}^h(m) + \eta \delta_j^h a_i$

Calculate the error for the l th pattern, $E_l = \frac{1}{2} \sum_{k=1}^K (b_{lk} - s_k^o)^2$

Total error for all patterns, $E = \sum_{l=1}^L E_l$

Apply the given patterns one by one, may be several times, in some random order and update the weights until the total error reduces to an acceptable value.

Table 4.7 Issues in Backpropagation Learning

Description **and** features of backpropagation

- Significance of error backpropagation
- Forward computation (inner product and nonlinear **function**)
- Backward operation (error calculation and derivative of output function)
- Nature of output function (semilinear)
- Stochastic gradient descent
- Local computations
- Stopping criterion

Performance of backpropagation learning

- Initialization of weights
- Presentation of training patterns: Pattern and batch **modes**

Table 4.7 Issues in Backpropagation Learning (Cont.)

-
- Learning rate parameter η
 - Range and value of η for stability and convergence
 - Learning rate adaptation for better convergence
 - Momentum term for faster convergence
 - Second order methods for better and faster convergence

Refinement of backpropagation learning

- Stochastic gradient descent, not an optimization method
- Nonlinear system identification: Extended Kalman-type algorithm
- Unconstrained optimization: Conjugate-gradient methods
- Asymptotic estimation of a *posteriori* class probabilities
- Fuzzy backpropagation learning

Interpretation of results of learning

- Ill-posed nature of solution
- Approximation of functions
- Good estimation of decision surfaces
- Nonlinear feature detector followed by linearly separable classification
- Estimation of a *posteriori* class probabilities

Generalization

- VC dimension
- Cross-validation
- Loading problem
- Size and efficiency of training set data
- Architectures of network
- Complexity of problem

Tasks with backpropagation network

- Logic function
- Pattern classification
- Pattern mapping
- Function approximation
- Probability estimation
- Prediction

Limitations of backpropagation learning

- Slow convergence (no proof of convergence)
- Local minima problem
- Scaling
- Need for teacher: Supervised learning

Extensions to backpropagation

- Learning with critic
 - Regularization
 - Radial basis functions
 - Probabilistic neural networks
 - Fuzzy neural networks
-

Description and features of backpropagation: The training patterns are applied in some random order one by one, and the weights are adjusted using the backpropagation learning law. Each application of the training set patterns is called a cycle. The patterns may have to be applied for several training cycles to obtain the output error to an acceptable low value. Once the network is trained, it can be used to recall the appropriate pattern (in this case some interpolated output pattern) for a new input pattern. The computation for recall is straightforward, in the sense that the weights and the output functions of the units in different layers are used to compute the activation values and the output signals. The signals from the output layer correspond to the output.

Backpropagation learning emerged as the most significant result in the field of artificial neural networks. In fact it is this learning law that led to the resurgence of interest in neural networks, nearly after 15 years period of lull due to exposition of limitations of the perceptron learning by Minsky and Papert (1969). In this section we will discuss various features including limitations of the **backpropagation** learning. We will also discuss the issues that determine the performance of the network resulting from the learning law. We will discuss these issues with reference to specific applications, and also with reference to some potential applications of the multilayer feedforward neural networks.

As noted earlier, the backpropagation learning involves propagation of the error backwards from the output layer to the hidden layers in order to determine the update for the weights leading to the units in a hidden layer. The error at the output layer itself is computed using the difference between the desired output and the actual output at each of the output units. The actual output for a given input training pattern is determined by computing the outputs of units for each hidden layer in the forward pass of the input data. Note that the error in the output is propagated backwards only to determine the weight updates. There is no feedback of the signal itself at any stage, as it is a feedforward neural network.

Since the backpropagation learning is exactly the same as the delta learning (see Section 1.6.3) at the output layer and is similar to the delta learning with the propagated error at the hidden layers, it is also called **generalized delta rule**. The term 'generalized' is used because the delta learning could be extended to the hidden layer units. Backpropagation of error is possible only if the output functions of the nonlinear processing units are differentiable. Note that if these output functions are linear, then we cannot realize the advantage of a multilayer network to generate complex decision boundaries for a nonlinearly separable (hard) classification problems. In fact a multilayer feedforward network with linear processing units is equivalent to a linear associative network, as discussed in Eq. (4.76), which, in

turn, is limited to solving simple pattern association problems. On the other hand, hard-limiting output function as in a multilayer perceptron cannot be used for learning the weights. A common differentiable output function used in the backpropagation learning is one which possesses a sigmoid nonlinearity. Two examples of **sigmoidal** nonlinear function are the logistic function and hyperbolic tangent function (See Figure 4.13):

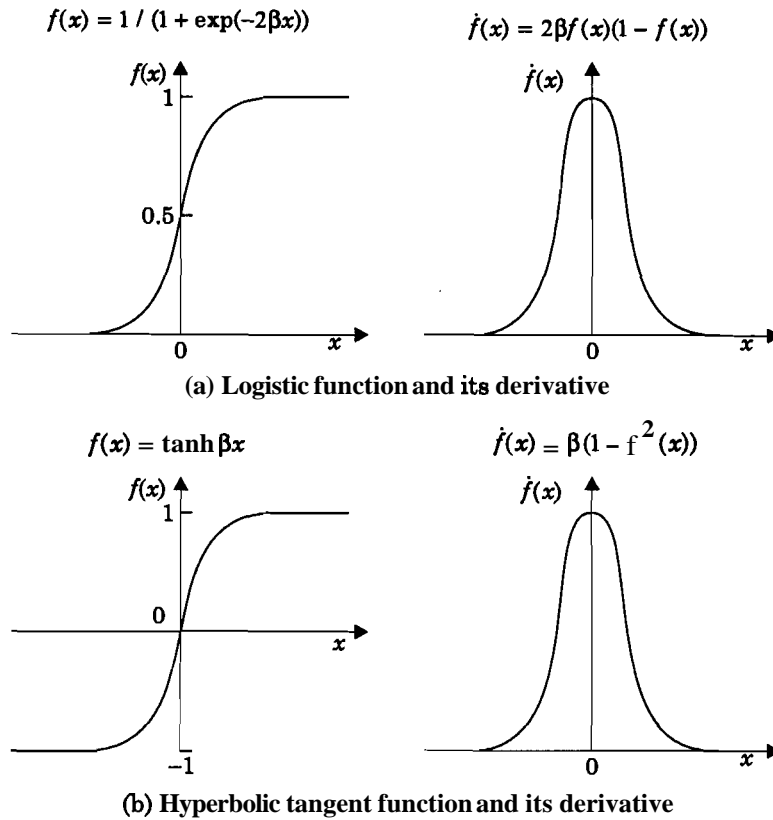


Figure 4.13 Logistic and hyperbolic tangent functions and their derivatives for $\beta = 0.5$.

Logistic function

$$f(x) = \frac{1}{1 + e^{-x}}, \quad -\infty < x < \infty \quad (4.100)$$

Hyperbolic function

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad -\infty < x < \infty. \quad (4.101)$$

For the logistic function the limits are $0 \leq f(x) \leq 1$, and for the hyperbolic tangent function the limits are $-1 \leq f(x) \leq 1$.

Analysis of Pattern *Mapping* Networks

Let us consider the derivative of the logistic function

$$\dot{f}(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x)[1-f(x)] \quad (4.102)$$

It can be seen from Eq. (4.102) that $\dot{f}(x)$ has the maximum value of 0.25 when $f(x) = 0.5$, and has the minimum value of 0 when $f(x) = 0$ or 1. Since the amount of change in the weight value leading to any unit i in the network is proportional to $\dot{f}_i(x)$, the change is maximum in the midrange of the activation value. This feature of the learning law contributes to its stability [Rumelhart et al, 1986a].

Note that the hyperbolic tangent function can be viewed as a biased and scaled version of the logistic function. That is

$$a \tanh(bx) = \frac{2a}{1+e^{-bx}} - a. \quad (4.103)$$

The asymmetry of the hyperbolic tangent function seems to make the learning faster by reducing the number of iterations required for training [Guyon, 1991].

The backpropagation learning is based on the gradient descent along the error surface. That is, the weight adjustment is proportional to the negative gradient of the error with respect to the weight. The error is the instantaneous error between the desired and the actual values of the output of the network. This instantaneous error is due to a given training pattern, which can be assumed to be a sample function of a random process. **Thus** the error can be assumed to be a random variable. Therefore this gradient descent method is a stochastic gradient learning method. Due to this stochastic nature, the path to the minimum of the error surface will be zigzag. The error surface itself will be an approximation to the true error surface **determined** by the entire training set of patterns. Moreover, even the true error surface is not a smooth quadratic surface as in the case of the Adaline. In fact the error surface may contain several local minima besides the global minimum. Hence the stochastic approximation of the gradient descent used in the backpropagation learning need not converge. There is no proof of convergence even in the mean as in the case of the LMS algorithm. The issues in the convergence of gradient descent methods are summarized in Table 4.8.

Since there is no proof of convergence, some heuristic criteria are used to stop the process of learning. They are based on the values of the gradient and the error in successive iterations and also on the total number of iterations. The average gradient value over each training cycle (presentation of all the training patterns once) is observed, and if this average value is below a preset threshold value for successive cycles, then the training process may be stopped. Likewise, the training process may be stopped using a threshold for the average error and observing the average error in successive cycles.

Table 4.8 Gradient Descent and Convergence

-
1. Let the input-output vector pair (\mathbf{a}, \mathbf{b}) be the sample function of a random process.

True ensemble average of the error

$$E(\mathbf{w}) = \mathbf{E}[e^2(m)]$$

where $e(m)$ is the instantaneous error for a given sample function. For linear units the error surface $E(\mathbf{w})$ is a smooth bowl-shaped in the weight space and hence the gradient descent $\partial E/\partial w_{ij}$ converges to the optimal weight vector \mathbf{w}^* .

2. Estimation of the error from a finite set of input-output pairs:

$$E(\mathbf{w}) = \sum_{m=1}^M e^2(m)$$

For linear units, this error surface is an approximation to the bowl-shape in the weight space and hence convergence of the gradient descent is only approximate.

3. Instantaneous error (Linear units):

$$E(\mathbf{w}) = e^2(m)$$

For linear units, the gradient descent converges only in the mean (stochastic convergence)

4. Instantaneous error (Nonlinear units):

$$E(\mathbf{w}) = e^2(m)$$

For nonlinear units, there is no proof of convergence even in the stochastic sense.

Sometimes both the average gradient as well as the average error may be used in the stopping criterion. But the main objective is to capture the implicit pattern behaviour in the training set data so that adequate generalization takes place in the network. The generalization feature is verified by testing the performance of the network for several new (test) patterns.

Performance of the backpropagation learning law: The performance of the backpropagation learning law depends on the initial setting of the weights, learning rate parameter, output functions of the units, presentation of the training data, besides the specific pattern recognition task (like classification, mapping, etc.) or specific application (like function approximation, probability estimation, prediction, logic function, etc.). It is important to initialize the weight values properly before applying the learning law for a given training set [Hush et al, 1991; Lee et al, 1991]. Initial weights **correspond** to a priori knowledge. If we have the knowledge and also if we know how to present the knowledge in the form of initial weights, then the overall performance of the resulting trained network in terms of speed

of learning and generalization would improve significantly. In general it is not known how to collect the relevant knowledge a priori. The more difficult part is to know how to include it in the form of weights. Therefore all the weights in the network are initialized to random numbers that are **uniformly** distributed in a small range of values. The range is typically $[-a/\sqrt{N_i}, +a/\sqrt{N_i}]$ where N_i is the number of inputs to the i th unit. Thus the range can be different for each unit. The value of a is typically in the range (1 to 3) [Wessels and Barnard, 1992]. Initial weights that are in very small range will result in long learning times. On the other hand, large initial weight values may result in the network output values in the saturation region of the output function. In the saturation region the gradient value is small. If the saturation is at the incorrect level, it may result in slow learning due to small changes made in the weights in each iteration. Incorrect saturation rarely occurs if the unit operates in the linear range of the output function.

Adjustment of the weights using backpropagation learning law is done by presenting the given set of training patterns several times. Randomizing the presentation of these patterns tends to make the search in the weight space stochastic, and **thus** reduces the possibility of limit cycles' in the trajectory in the weight space during learning [Haykin, 1994, p. 151]. Presentation of the training data pattern by pattern for **adjustment** of the weights makes it possible to have the learning online. This pattern mode also reduces the problem of local minima. But to speed up the learning process it is preferable to update the weights in a batch mode, in which the gradient of the error, computed over all the training patterns, is used. The batch mode gives a better estimation of the gradient of the overall error surface.

Learning rate parameter η plays a crucial role in the backpropagation learning. The order of values for η depends on the variance of the input data. For the case of **Adaline**, the learning rate parameter $\eta < 1/(2\lambda_{\max})$, where λ_{\max} is the largest eigenvalue of the autocorrelation matrix of the input data. This gives an indication for the choice of η , since the derivation in the backpropagation does not **suggest** any clue for this choice. Since it is a stochastic gradient descent learning, too small an η will result in a smooth trajectory in the weight space, but takes long time to converge. On the other hand, too large an η may increase the speed of learning, but **will** result in large random fluctuations in the weight space, which in turn may lead to an unstable situation in the sense that the network weights may not converge.

It is desirable to adjust the weights in such a way that all the units learn nearly at the same rate. That is, the net change in all the weights leading to a unit should be nearly the same. To accomplish this, the learning rate parameters should be different for

different weights. The weights leading to a unit with many inputs should have smaller η compared to the η for the weights leading to a unit with fewer inputs. Also, the gradient of the error with respect to the weights leading to the output layer will be **larger** than the gradient of the **error** with respect to the weights leading to the hidden layers. Therefore the learning rate parameters η should be typically smaller for the weights at the output layer and larger for the weights leading to the units in the hidden layers. This will ensure that the net change in the weights remains nearly the same for all layers.

Better convergence in learning can be achieved by adapting the learning rate parameter η suitably for each iteration. For this the change in η is made proportional to the negative gradient of the instantaneous error with respect to η [Haykin, 1994, p. 195]. That is

$$\Delta\eta_{ji}(m+1) = -\gamma \frac{\partial e^2(m)}{\partial \eta_{ji}(m)} \quad (4.104)$$

where γ is a proportionality constant.

It was shown in [Haykin, 1994] that

$$\frac{\partial e^2(m)}{\partial \eta_{ji}(m)} = \frac{\partial e^2(m)}{\partial w_{ji}(m)} \frac{\partial e^2(m-1)}{\partial w_{ji}(m-1)} \quad (4.105)$$

This is called delta-delta learning rule [Jacobs, 1988]. The change in the learning rate parameter depends on the instantaneous gradients at the previous two iterations. In this learning it is difficult to **choose** suitable values for the proportionality constant γ if the magnitudes of the two gradients in the product are either too small or too large. To overcome this limitation a modification of the above learning rule, namely, delta-bar-delta learning rule was proposed [Jacobs, 1988; Minai and Williams, 1990].

The adaptation of the learning rate parameter using the delta-delta learning rule or the delta-bar-delta learning rule slows down the backpropagation learning significantly due to additional complexity in computation at each iteration. It is possible to reduce this complexity by using the idea of the gradient reuse method, in which the gradient estimate is obtained by averaging the gradient values corresponding to several training patterns. **Thus**

$$w_{ji}(m+1) = w_{ji}(m) + \eta_{ji}(m) \sum_{l=1}^L \delta_j^l(m) s_i^l(m) \quad (4.106)$$

where l is the index for training pattern and $\delta_j^l(m)$ is the propagated error. The learning rate parameter $\eta_{ji}(m)$ is **also** computed using the averaged gradient for several training patterns.

The values of the learning rate parameters computed using any of the above methods are very low, thus resulting in slow learning.

One way to increase the rate of learning is by using a momentum term in the weight change as follows [Plaut et al, 1986; Fahlman, 1989; Rumelhart et al, 1986a]:

$$\Delta w_{ji}(m) = \alpha \Delta w_{ji}(m-1) + \eta \delta_j(m) s_i(m) \quad (4.107)$$

where $0 \leq \alpha < 1$ is the momentum constant. The use of the momentum term accelerates the descent to the minimum of the **error** surface. It will also help in reducing the effects of local minima of the error surface.

The expression for the updated weight which includes momentum **term** as well as the learning rate adaptation is given by

$$w_{ji}(m+1) = w_{ji}(m) + \alpha \Delta w_{ji}(m-1) + \eta_{ji}(m) \sum_{l=1}^L \delta_j^l(m) s_i^l(m) \quad (4.108)$$

Normally the backpropagation learning uses the weight change proportional to the negative gradient of the instantaneous error. Thus it uses only the first derivative of the instantaneous error with **respect** to the weight. If the weight change is made using the information in the second derivative of the error, then a better estimate of the optimum weight change towards the minimum may be obtained. The momentum method is one such method where both the weight change at the previous step and the gradient at the current step are used to determine the weight change for the current step.

More effective methods [Battiti, 1992] can be derived starting with the following Taylor series expression of the error as a function of the weight **vector**

$$E(\mathbf{w} + \Delta \mathbf{w}) = E(\mathbf{w}) + \mathbf{g}^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T H \Delta \mathbf{w} + \dots \quad (4.109)$$

where $\mathbf{g} = \frac{\partial E}{\partial \mathbf{w}}$ is the gradient vector, and $H = \frac{\partial^2 E}{\partial \mathbf{w}^2}$ the Hessian matrix. For small $\Delta \mathbf{w}$, the higher order terms can be neglected, so that we get

$$\Delta E = E(\mathbf{w} + \Delta \mathbf{w}) - E(\mathbf{w}) \quad (4.110)$$

$$= \mathbf{g}^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T H \Delta \mathbf{w} \quad (4.111)$$

Taking the derivative of E with **respect** to \mathbf{w} gives the **gradient**. That is

$$\frac{\partial E}{\partial \mathbf{w}} = \mathbf{g} \quad (4.112)$$

On the other hand, taking the derivative of ΔE with respect to $\Delta \mathbf{w}$ gives

$$\frac{\partial \Delta E}{\partial \Delta \mathbf{w}} = \mathbf{g} + H \Delta \mathbf{w} \quad (4.113)$$

Setting this to zero gives an optimum value of $\Delta \mathbf{w}$, taking **upto** the second order term into account. Therefore

$$\Delta \mathbf{w}^* = -H^{-1} \mathbf{g} \quad (4.114)$$

Thus the new weight **vector** taking the optimal value of $\Delta \mathbf{w}$ is given by

$$\mathbf{w}(m+1) = \mathbf{w}(m) - H^{-1} \mathbf{g} \quad (4.115)$$

This is the Newton's method. Note that this is similar to the expression (C.17) in Appendix-C.

For the quadratic error function $E(\mathbf{w})$, the optimal step $\Delta \mathbf{w}^*$ will lead to the final weight value \mathbf{w}^* starting from any initial weight vector $\mathbf{w}(0)$. That is

$$\mathbf{w}^* = \mathbf{w}(0) - H^{-1} \mathbf{g} \quad (4.116)$$

provided $H^{-1} \mathbf{g}$ is known at $\mathbf{w} = \mathbf{w}(0)$. For a nonquadratic error surface, as in the network with nonlinear units, the Newton's method gives the optimal weight change if the variation of the error is considered only **upto** the second derivative. Note that the Newton's method is different from the gradient descent. Since the Newton's method uses more information of the error surface than the gradient descent, it is expected to converge faster. But there is no guarantee that this choice of the weight change will converge.

Implementation of Newton's method is cumbersome due to the need for computation of the Hessian matrix. Methods were proposed which will avoid the need for the computation of the Hessian matrix. The conjugate gradient method is one such method, where the increment in the weight at the m th step is given by

$$\Delta \mathbf{w} = \mathbf{w}(m+1) - \mathbf{w}(m) = \eta(m) \mathbf{d}(m) \quad (4.117)$$

where the direction of the increment $\mathbf{d}(m)$ in the weight is a linear combination of the current gradient vector and the previous direction of the increment in the weight. That is

$$\mathbf{d}(m) = -\mathbf{g}(m) + \alpha(m-1) \mathbf{d}(m-1) \quad (4.118)$$

where the value of $\alpha(m)$ is obtained in terms of the gradient by one of the following formulae [Fletcher and Reeves, 1964; Polak and Ribiere, 1969].

$$\alpha(m) = \frac{\mathbf{g}^T(m+1) \mathbf{g}(m+1)}{\mathbf{g}^T(m) \mathbf{g}(m)} \quad (4.119)$$

or

$$\alpha(m) = \frac{\mathbf{g}^T(m+1) [\mathbf{g}(m+1) - \mathbf{g}(m)]}{\mathbf{g}^T(m) \mathbf{g}(m)} \quad (4.120)$$

Computation of the learning rate parameters $\eta(m)$ in Eq. (4.117) requires line minimization for each iteration [Johansson et al, 1990].

The objective is to determine the value of η for which the error $E[\mathbf{w}(m) + \eta \mathbf{d}(m)]$ is minimized for given values of $\mathbf{w}(m)$ and $\mathbf{d}(m)$. Performance of the conjugate-gradient method depends critically on the choice of $\eta(m)$ and hence on the line minimization. But generally the conjugate-gradient method converges much faster than the standard backpropagation learning, although there is no proof of convergence in this case also due to the nonquadratic nature of the error surface [Kramer and Sangiovanni-Vincentelli, 1989].

Refinements of the backpropagation learning: The backpropagation learning is based on the steepest descent along the surface of the instantaneous error in the weight space. It is only a first order approximation of the descent as the weight change is assumed to be proportional to the negative gradient. The instantaneous error is a result of a single training pattern, which can be viewed as a sample function of a random process. The search for the global minimum of the error surface is stochastic in nature as it uses only the instantaneous error at each step. The stochastic nature of the gradient descent results in a zig-zag path of the trajectory in the weight space in our search for the global minimum of the error surface. Note that the zig-zag path is also due to the nonquadratic nature of the error surface, which in turn is due to the nonlinear output functions of the units. Note also that the backpropagation learning is based only on the gradient descent and not on any optimization criterion.

A better learning in terms of convergence towards the global minimum may be achieved if the information from the given training patterns are used more effectively. One such approach is based on posing the supervised learning problem as a nonlinear system identification problem [Haykin, 1991]. The resulting learning algorithm is called an extended Kalman-type learning [Singhal and Wu, 1989] which uses piecewise linear approximation to the nonlinear optimal filtering problem.

Better learning can also be achieved if the supervised learning is posed as an unconstrained optimization problem, where the cost function is the error function $E(\mathbf{w})$ [Battiti, 1992]. In this case the optimal value of the increment in the weight is obtained by considering only upto second order derivatives of the error function. The resulting expression for the optimal $\Delta \mathbf{w}$ requires computation of the second derivatives of $E(\mathbf{w})$ with respect to all the weights, namely, the Hessian matrix. The convergence will be faster than the gradient descent, but there is no guarantee for convergence in this case also.

A multilayer **feedforward** neural network with backpropagation learning on a finite set of independent and identically distributed samples leads to an asymptotic approximation of the underlying a posteriori class probabilities provided that the size of the training set

data is large, and the learning algorithm does not get stuck in a local minima [Hampshire and Pearlmutter, 1990].

If the a posteriori conditional probabilities are used as the desired response in a learning algorithm based on an information theoretic measure for the cost function [Kullback, 1968; Haykin, 1994, Sec. 6.201, then the network captures these conditional probability distributions. In particular, the output of the network can be interpreted as estimates of the a posteriori conditional probabilities for the underlying distributions in the given training data.

Yet another way of formulating the learning problem for a multilayer neural network is by using the fuzzy representation for input or output or for both. This results in a fuzzy backpropagation learning law [Ishibuchi et al, 1993]. The convergence of the fuzzy backpropagation learning is significantly faster, and the resulting minimum mean squared error is also significantly lower than the usual backpropagation learning.

Interpretation of the result of learning: A trained multilayer feed-forward neural network is expected to capture the functional relationship between the input-output pattern pairs in the given training data. It is implicitly assumed that the mapping function corresponding to the data is a smooth one. But due to limited number of training samples, the problem becomes an ill-posed problem, in the sense that there will be many solutions satisfying the given data, but none of them may be the **desired/correct** one [Tikhonov and Arsenin, 1977; Wieland and Leighton, 1987]. Figure 4.14 illustrates the basic

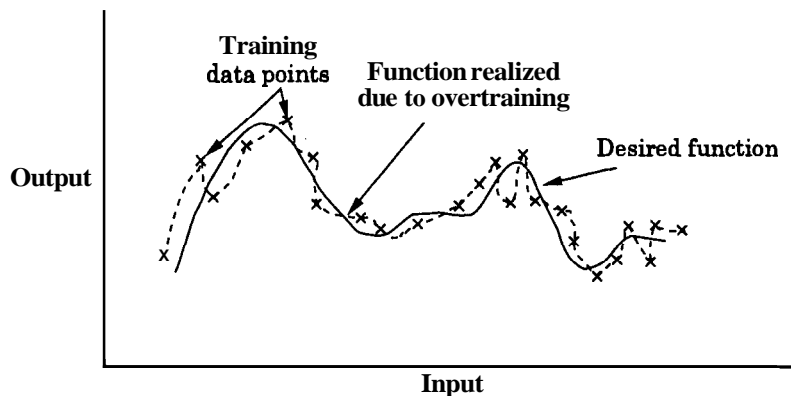


Figure 4.14 Illustration of an ill-posed problem for a function of one variable.

idea of an ill-posed problem for a function of one variable. Given the samples marked 'x', the objective is to capture the function represented by the solid curve. But depending on the size of the network, several solutions are possible, including the overtraining situations (shown by dotted curve) in which for all the training data

the error is zero. In fact there could be several functions passing through the given set of points, none of which is the desired one. This happens if the number of free parameters (weights) of the network is very large. Such a situation results in a large error when some other (test) samples **are** given to validate the network model for the function. This is called 'poor generalization' by the network. On the other hand, fewer number of the free parameters may result in a large error even for the training data, and hence a poor approximation to the desired function. The function approximation interpretation of a multilayer **feedforward** neural network enables us to view different hidden layers of the network performing different functions. For example, the first hidden layer can be interpreted as capturing some local features in the input space. The second hidden layer can be interpreted as capturing some global features. This two-stage approximation has been shown to realize any continuous vector-valued function [Sontag, 1992b]. The universal approximation theorem of Cybenko seems to suggest that even a single layer of nonlinear units would suffice to realize any continuous function [Cybenko, 1989]. But this result assumes that a hidden layer of unlimited size is available, and that the continuous function to be approximated is also available. Thus Cybenko's theorem gives only an existence proof, but it is not useful to realize the function by training a single hidden layer network.

A trained multilayer neural network can be interpreted as a classifier, with complex decision surfaces separating the classes. These decision surfaces are due to multiple layers of nonlinear units. In the limiting case of hard-limiting nonlinear units, the geometrical arguments for the creation of the complex decision surfaces in a multilayer perceptron discussed in Section 4.3.3 are applicable.

It is also possible to view that the hidden layers perform a nonlinear feature extraction to map the input data into linearly separable classes in the feature space. At the output layer the unit with the largest output is considered as the class to which the input belongs.

As mentioned earlier, the output of a trained multilayer neural network can also be considered as an approximation to the a posteriori class probabilities.

Generalization: A backpropagation learning network is expected to generalize from the training set data, so that the network can be used to determine the output for a new test input. As mentioned earlier, 'generalization' is different from 'interpolation', since in generalization the network is expected to model the unknown system or function from which the training set data has been obtained. The problem of determination of weights **from** the training set data is called the loading' problem [Judd, 1990; Blum and Rivest, 1992]. The

generalization performance depends on the size and efficiency of the training set, besides the architecture of the network and the complexity of the problem [Hush and Horne, 1993]. Testing the performance of the network with new data is called cross-validation. If the performance for the test data is as good as for the training data, then the network is said to have generalized from the training data. Further discussion on generalization is given later in Section 7.3 and in Appendix D.

Tasks with backpropagation network: A backpropagation network can be used for several applications such as realization of logic functions, pattern classification, pattern mapping, function approximation, estimation of probability distribution and prediction [Hush and Horne, 1993]. These tasks were demonstrated in several real world applications such as in speech, character recognition, system identification, passive sonar detection/classification, speech synthesis, etc. [Sejnowski and Rosenberg, 1987; Cohen et al, 1993; LeCun et al, 1990; Narendra and Parthasarathy, 1990; Casselman et al, 1991].

Limitations of backpropagation: The major limitation of the backpropagation learning is its slow convergence. Moreover, there is no proof of convergence, although it seems to perform well in practice. Due to stochastic gradient descent on a nonlinear error surface, it is likely that most of the time the result may converge to some local minimum on the error surface [Gori and Tesi, 1992]. There is no easy way to eliminate this effect completely, although stochastic learning algorithms were proposed to reduce the effects of local minima [Wasserman, 1988]. Another major problem is the problem of scaling. When the complexity of the problem is increased, there is no guarantee that a given network would converge, and even if it converges, there is no guarantee that good generalization would result. The complexity of a problem can be defined in terms of its size or its predicate order [Minsky and Papert, 1990; Hush and Horne, 1993]. Effects of scaling can be handled by using the prior information of the problem, if possible. Also, modular architectures can also reduce the effects of the scaling problem [Ballard, 1990; Jacobs et al, 1991; Haykin, 1994].

For many applications, the desired output may not be known precisely. In such a case the backpropagation learning cannot be used directly. Other learning laws have been developed based on the information whether the response is correct or wrong. This mode of learning is called reinforcement learning or learning with critic [Sutton et al, 1991; Barto, 1992] as discussed in Section 2.4.6.

Extensions of backpropagation: Principles analogous to the ones used in the backpropagation network have been applied to extend the

scope of the network in several directions as in the case of probabilistic neural networks, **fuzzy** backpropagation networks, regularization networks and radial basis function networks [Wasserman, 1993].

4.5 Summary and Discussion

We have presented a detailed analysis of feedforward networks in this chapter with emphasis on the pattern recognition tasks that can be realized using these networks. A network with linear units (Adaline units) performs a pattern association task provided the input patterns are linearly independent. Linear independence of input patterns also limits the number of patterns to the dimensionality of the input pattern space. We have seen that this limitation is overcome by using hard-limiting threshold units (perceptron units) in the feedforward network. Since threshold units in the output layer results in a discrete set of states, the resulting network performs pattern classification task. The hard-limiting threshold units provide a set of inequalities to be satisfied by the network. Thus the weights of the network are not unique any more and hence they are determined by means of the perceptron learning law.

A single layer perceptron is limited to linearly separable classes only. For an arbitrary pattern classification problem, a multilayer perceptron (MLP) is needed. But due to absence of desired output at the units in the intermediate layers of units, the MLP network cannot be trained by the simple perceptron learning law. This hard learning problem can be solved by using nonlinear units with differentiable output functions. Since the output functions are now continuous, the multilayer feedforward neural network can perform pattern mapping task. The output error **backpropagation** is used in the learning algorithm for these multilayer networks.

Since the backpropagation learning is based on stochastic gradient descent along a rough error surface, there is no guarantee that the learning law converges towards the desired solution for a given pattern mapping task. Several variations of the backpropagation learning have been suggested to improve the convergence as well as the result of convergence. Although there is no proof of convergence, the backpropagation learning algorithm seems to perform effectively for many tasks such as pattern classification, function approximation, time series prediction, etc.

How well a trained feedforward network performs a given task is discussed both theoretically and experimentally in the literature on generalization. The issue of generalization is an important topic, but it is not discussed in this book. There are excellent treatments of this topic in [Vidyasagar, 1997; Valiant, 1994]. Appendix D gives an overview of generalization in neural networks.

Some of the limitations of backpropagation such as convergence

can be addressed with reference to specific tasks, exploiting the knowledge of the task domain. Thus architectures developed for specific tasks are more useful than the general **feedforward** neural networks. Some of these architectures will be discussed in Chapter 7.

Review Questions

1. What is a linear associative network?
2. What is pseudoinverse of a matrix?
3. Explain the significance of (a) determination of weights by computation and (b) determination of weights by learning.
4. What is the difference between linearly independent set and orthogonal set of vectors?
5. What does the rank of an input matrix indicate?
6. Explain the nature of the input vectors in each of the following cases of the optimal choice of weight matrix. (a) $W = BA^T$, (b) $W = BA^{-1}$ and (c) $W = BA^+$.
7. Explain the choice of $W = BA^+$ for linearly independent and linearly dependent cases of input vectors.
8. Why the choice of $W = BA^+$ need not be the best choice for noisy input vectors? Discuss your answer with reference to the Murakami result given in Eq. (4.19).
9. What is the significance of the **Widrow's** learning for linear associative networks?
10. Why is it that there is no learning law for obtaining the best choice of the weights for the case of noisy input vectors?
11. Why is it that the number of input patterns are linked to the dimensionality of the input vectors in the case of linear associative network?
12. Why learning is essential for a network with nonlinear units?
13. What is perceptron learning for pattern classification?
14. Explain the significance of perceptron convergence theorem.
15. Explain how to interpret perceptron learning as a gradient descent algorithm. What is the gradient term here?
16. What is meant by perceptron representation problem?
17. Distinguish between linearly separable and linearly inseparable problems.
18. Why a single layer of perceptron cannot be used to solve linearly inseparable problems?
19. Give two examples of linearly inseparable problems.
20. Show by geometrical arguments that with 3 layers of nonlinear units, any hard classification problem can be solved.

21. Distinguish between multilayer perceptron and a general multilayer feedforward neural network.
22. Explain how a multilayer feedforward neural network with linear units in all the layers is equivalent to a linear associative network.
23. What is meant by gradient descent methods?
24. Explain the difference between method of steepest descent and Newton's method.
25. Explain the difference between LMS learning and delta learning.
26. Why LMS learning is called a stochastic gradient descent method?
27. Comment on the nature of the **error** surface for a multilayer feedforward neural network.
28. Why backpropagation learning is also called generalized delta rule?
29. Why convergence is not guaranteed for the backpropagation learning algorithm?
30. Discuss the significance of semilinear function in the **backpropagation** learning.
31. How 'pattern' mode and 'batch' mode of training affect the result of backpropagation learning?
32. Explain why it is preferable to have **different** values for η for weights leading to the units in different layers in a feedforward neural network.
33. What is the significance of momentum term in backpropagation learning?
34. What is conjugate gradient method?
35. Discuss various interpretations of the results of backpropagation learning.
36. What is an ill-posed problem in the context of training a multilayer feedforward network?
37. What is meant by generalization in feedforward networks?
38. Why should generalization depend on the size and efficiency of the training set, architecture of the network and the complexity of the problem?
39. Discuss a few tasks that can be performed by a backpropagation network.
40. How can we interpret the results of backpropagation learning as an estimation of a posteriori class probabilities?
41. Explain the limitations of backpropagation learning.

Problems

1. A p th order polynomial threshold function is defined as [Hassoun, 1995, p. 8]

$$s = \begin{cases} 1, & \text{if } \sum_{i_1=1}^M w_{i_1} a_{i_1} + \sum_{i_1=1}^M \sum_{i_2=i_1}^M w_{i_1 i_2} a_{i_1} a_{i_2} + \dots \\ & + \sum_{i_1=1}^M \sum_{i_2=i_1}^M \dots \sum_{i_p=i_{p-1}}^M w_{i_1 i_2 \dots i_p} a_{i_1} a_{i_2} \dots a_{i_p} \geq \theta \\ 0, & \text{otherwise} \end{cases}$$

Show that the-number of weights is given by

$$r = \sum_{i=0}^p \binom{M}{i} \text{ for } \mathbf{a} \in \{0, 1\}^M$$

and

$$r = \binom{M+p}{p} \text{ for } \mathbf{a} \in \mathcal{R}^M$$

where $\binom{M}{i}$ is the number of combinations of M different items taken i at a time without repetition. (Hint: Count the number of weights in each case. Note that for the binary case $\mathbf{a} \in \{0, 1\}^M$ the indices in the summation are $i_p > i_{p-1}$ for $p \geq 2$.)

2. In the equation in Problem 1 above $p = 1$ corresponds to the case of linear threshold function, and $p = 2$ for a quadratic threshold function. Larger values of p gives higher flexibility to realize a Boolean function by threshold gates. In fact it can be shown that any Boolean function of M variables can be realized using a polynomial threshold function of order $p \leq M$ [Hassoun, 1995, p. 9]. Show that the most difficult M -variable Boolean function to implement by a polynomial threshold function requires 2^M parameters in the worst case. (Hint: Use the result of Problem 1 for $p = M$.)
3. A set of N points in \mathcal{R}^M is said to be in 'general position' if every subset of M or fewer points is linearly dependent. A 'dichotomy' is labelling of N points into two distinct categories. The number of linearly separable dichotomies of N points in general position in \mathcal{R}^M is given by [Hassoun, 1995, p. 181]

$$C(N, M) = \begin{cases} 2 \sum_{i=0}^M \binom{N-1}{i} & \text{for } N > M + 1 \\ 2^N & \text{for } N \leq M + 1 \end{cases}$$

Derive the above result using repeated iteration of the recursive relation

$$C(N+1, M) = C(N, M) + C(N, M-1)$$

and noting that $C(1, M) = 2$. (Hint: See [Hertz, 1991, p. 113-114].)

4. A p th order **polynomial** threshold function with labeled inputs $\mathbf{a} \in \mathcal{R}^M$ may be viewed as a linear threshold function with $(r-1)$ preprocessed inputs, where

$$\mathbf{r} = \begin{pmatrix} M+p \\ p \end{pmatrix}$$

The mapping $\mathcal{S}(\mathbf{a})$ from the input space, \mathcal{R}^M to the \mathcal{R}^{r-1} space is called '\$-mapping'. A dichotomy of N points is ' **ϕ -separable**' if there exist a $(r-2)$ -dimensional hyperplane in the '\$-space' which correctly classifies the N points [Hassoun, 1995, p. 20]. Show that the number of Boolean functions of M -variables that can be realized by an M -input p th order polynomial threshold function is less than $C(N, r-1)$.

5. Using the matrix identities $\mathbf{A} = \mathbf{A}\mathbf{A}^T(\mathbf{A}^+)^T$ and $\mathbf{A}^T = \mathbf{A}^+ \mathbf{A}\mathbf{A}^T$, derive the expression given in Eq. (4.11) from the definition of the matrix \mathbf{S} given in Eq. (4.10).
6. Derive the expression for E_{\min} given in Eq. (4.16) using the expression for the pseudoinverse given in Eq. (4.14).
7. Compute the weight **matrix** for the following pattern association task

$$\mathbf{a}_1 = \begin{bmatrix} \frac{1}{6} & -\frac{5}{6} & -\frac{1}{6} & \frac{1}{2} \end{bmatrix}^T, \quad \mathbf{b}_1 = [1 \ 0 \ 0]^T$$

$$\mathbf{a}_2 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix}^T, \quad \mathbf{b}_2 = [0 \ 1 \ 1]^T$$

$$\mathbf{a}_3 = \begin{bmatrix} -\frac{5}{6} & \frac{1}{6} & -\frac{1}{6} & \frac{1}{2} \end{bmatrix}^T, \quad \mathbf{b}_3 = [0 \ 0 \ 0]^T$$

8. Using the **perceptron** learning law design a classifier for the following problem:
 Class \mathbf{C}_1 : $[-2 \ 2]^T$, $[-2 \ 1.5]^T$, $[-2 \ 0]^T$, $[1 \ 0]^T$, and $[3 \ 0]^T$
 Class \mathbf{C}_2 : $[1 \ 3]^T$, $[3 \ 3]^T$, $[1 \ 2]^T$, $[3 \ 2]^T$, and $[10 \ 0]^T$
9. Design and train a **feedforward** network for the following problems:
- (a) Parity: Consider a 4-input and 1-output problem, where the output should be 'one' if there are odd number of **1s** in the input pattern and 'zero' otherwise. The difficulty of the problem is due to the **fact** that the input patterns differing in only one bit require opposite **outputs**.

- (b) Encoding: Consider an 8-input and 8-output problem, where the output should be equal to the input for any of the 8 combinations of seven 0s and one 1.
- (c) Symmetry: Consider a 4-input and 1-output problem where the output is required to be 'one' if the input configuration is symmetrical and 'zero' otherwise.
- (d) Addition: Consider a 4-input and 3-output problem, where the output should be the result of the sum of two 2-bit input numbers.

(Hint: Write a program to implement the algorithm. In all the cases start with a hidden layer of 8 units and progressively reduce the number of units.)

10. Generalize the XOR problem to a parity problem for $N (> 2)$ variables by considering a network for the two variables first and then extending the network considering the output of the first network as one variable and the third variable as another. Repeat this for $N = 4$ and design a network for solving the parity problem for 4 variables. (See [Bose and Liang, 1996, p. 214].)
11. For the following 2-class problem determine the decision boundaries obtained by LMS and perceptron learning laws. Comment on the results.

Class C_1 : $[-2 \ 2]^T$, $[-2 \ 3]^T$, $[-1 \ 1]^T$, $[-1 \ 4]^T$, $[0 \ 0]^T$,
 $[0 \ 1]^T$, $[0 \ 2]^T$, $[0 \ 3]^T$ and $[1 \ 1]^T$

Class C_2 : $[1 \ 0]^T$, $[2 \ 1]^T$, $[3 \ -1]^T$, $[3 \ 1]^T$, $[3 \ 2]^T$,
 $[4 \ -2]^T$, $[4 \ 1]^T$, $[5 \ -1]^T$ and $[5 \ 0]^T$

12. Study the classification performance of a **MLFFNN** for a 2-class problem, where the 2-dimensional data for each class is derived from the Gaussian distributions with the following means and variances, and the class probabilities:

Class C_1 : $\mu_1 = [0 \ 0]^T$, $\sigma_1^2 = 1$, and $P(C_1) = 0.4$

Class C_2 : $\mu_2 = [3 \ 0]^T$, $\sigma_2^2 = 4$, and $P(C_2) = 0.6$

Assume a single hidden layer and a sigmoid output function for the units in the hidden and output layers. Study the performance for different number of hidden units (say 2, 4, and 6), and for different learning rate parameters (say 0.01, 0.1, and 0.9). Study also the effect of momentum term by considering two different values for the momentum parameter (say 0.1 and 0.5).

- 13.** Compare the classification performance of the best network in Problem 12 above with the performance of the Bayesian classification result. The Bayes classification result is obtained by computing the probability of error for optimum Bayes classifier for the given distributions. The Bayes classifier is defined as follows:

For a given input \mathbf{x} , decide C_1 if

$$P(C_1|\mathbf{x}) > P(C_2|\mathbf{x})$$

otherwise decide C_2 , where

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{p(\mathbf{x})}, \quad i = 1, 2.$$

and

$$p(\mathbf{x}) = \sum_{k=1}^2 p(\mathbf{x}|C_k)P(C_k).$$

Chapter 5

Feedback Neural Networks

5.1 Introduction

This chapter presents a detailed analysis of the pattern recognition tasks that can be performed by feedback artificial neural networks. In its most general form a feedback network consists of a set of processing units, the **output** of each unit is fed as input to all other units including the same unit. With each link connecting any two units, a weight is associated which determines the amount of output a unit feeds as input to the other unit. A general feedback network does not have any structure, and hence is not likely to be useful for solving any pattern recognition task.

However, by appropriate choice of the parameters of a feedback network, it is possible to perform several pattern recognition tasks. The simplest one is an autoassociation task, which can be performed by a feedback network consisting of linear processing units. A detailed analysis of the linear autoassociative network shows that the network is severely limited in its capabilities. In particular, a linear autoassociative network merely gives out what is given to it as input. That is, if the input is noisy, it comes out as noisy output, thus giving an error in recall even with optimal setting of weights. Therefore a linear autoassociative network does not have any practical use. By using a nonlinear output function for each processing unit, a feedback network can be used for pattern storage. The function of a feedback network with nonlinear units can be described in terms of the **trajectory** of the state of the network with time. By associating an energy with each state, the trajectory describes a traversal along the energy landscape. The minima of the energy landscape correspond to stable states, which can be used, to store the given input patterns. The number of patterns that can be stored in a given network depends on the number of units and the strengths of the connecting links. It is quite possible that the number of available energy minima is less than the number of patterns to be stored. In such a case the given pattern storage problem becomes a hard problem for the network. If on the other hand, the number of energy minima in the energy landscape of a network is greater than the required number of

patterns to be stored, then there is likely to be an error in the recall of the stored patterns due to the additional false minima. The hard problem can be solved by providing additional (hidden) units in a feedback network, and the errors in recall of the stored patterns due to false minima can be reduced using probabilistic update for the output function of a unit. A feedback network with hidden units and probabilistic update is called a Boltzmann machine. It can be used to store a pattern environment, described by a set of patterns to be stored, together with the probability of occurrence of each of these patterns.

Table 5.1 shows the organization of the topics to be discussed in this chapter. A detailed analysis of linear autoassociative feedforward networks is considered first in Section 5.2. The pattern storage problem is analyzed in detail in Section 5.3. In particular, the **Hopfield** energy analysis, and the issues of hard problem and false minima are discussed in this section. The Boltzmann machine is introduced in Section 5.4. This section also deals with the details of the pattern environment storage problem and the Boltzmann learning law. Some practical issues in the implementation of learning laws for feedback networks including simulated annealing are discussed in Section 5.5.

Table 5.1 Pattern Recognition Tasks by Feedback Neural Networks

Autoassociation

- **Architecture:** Single **layer** with feedback, linear processing units
- **Learning:** Not important
- **Recall:** Activation dynamics until stable states are reached
- **Limitation:** No accretive behaviour
- **To overcome:** Nonlinear processing units, leads to a pattern storage problem

Pattern Storage

- **Architecture:** Feedback neural network, nonlinear processing units, states, **Hopfield** energy analysis
- **Learning:** Not important
- **Recall:** Activation dynamics until stable states are reached
- **Limitation:** Hard problems, limited number of patterns, false minima
- **To overcome:** Stochastic update, hidden units

Pattern Environment Storage

- **Architecture:** Boltzmann machine, nonlinear processing units, hidden units, stochastic update
 - **Learning:** Boltzmann learning law, simulated annealing
 - **Recall:** Activation dynamics, simulated annealing
 - **Limitation:** Slow learning
 - **To Overcome:** Different architecture
-

5.2 Analysis of Linear Autoassociative FF Networks

First we consider the realization of an autoassociative task with a feedforward network as shown in Figure 5.1. Analogous to the hetero-

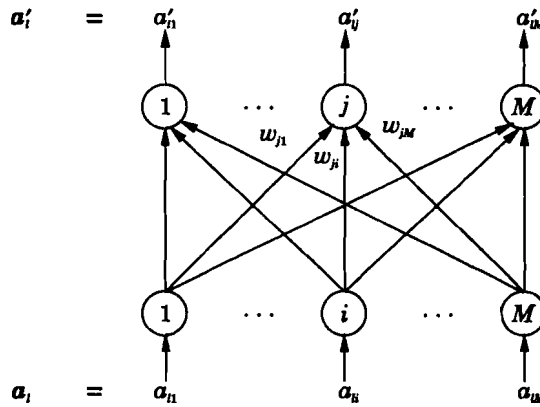


Figure 5.1 Linear autoassociative feedforward network.

association, in autoassociation the objective is to associate a given pattern with itself during training, and then to recall the associated pattern when an **approximate/noisy** version of the same pattern is given during testing. In other words, in **autoassociation** the associated output pattern **b_l** is same as the input pattern **a_l** for the **l**th pattern. That is, with reference to the pattern association task, **b_l = a_l**, **l = 1, 2, ..., L** in the case of autoassociation. In recall it is desired to obtain **b_l** as output for an approximate input **a_l + ε**. The weight matrix **W = [w_{ij}]** of a linear autoassociative network (Figure 5.1) can be determined as in the case of the linear heteroassociator, for a fixed set of input pattern vectors **{a_l}**. Since we want **WA = A**, the optimal weights are given by (see Section 4.2.2)

$$W = AA^+ \tag{5.1}$$

where **A⁺** is the pseudoinverse of the **M x L** matrix **A** consisting of the input vectors **{a_l}**. The pseudoinverse is given in terms of the components of singular value decomposition of the matrix **A** as follows:

$$A = \sum_{i=1}^L \lambda_i^{1/2} \mathbf{p}_i \mathbf{q}_i^T \tag{5.2}$$

where **λ_i** are the eigenvalues, and **p_i** and **q_i** are the eigenvectors of the matrices **AA^T** and **A^TA**, respectively. That is,

$$AA^T \mathbf{p}_i = \lambda_i \mathbf{p}_i \tag{5.3}$$

and

$$A^T A \mathbf{q}_i = \lambda_i \mathbf{q}_i \tag{5.4}$$

The sets of **eigenvectors** $\{\mathbf{p}_i\}$ and $\{\mathbf{q}_i\}$ are orthonormal sets. The **eigenvalues** λ_i are real and nonnegative, since the matrices $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$ are symmetric. The eigenvalues λ_i are ordered, i.e., $\lambda_i \geq \lambda_{i+1}$. If the **rank** of the matrix \mathbf{A} is r ($\leq L$), then the eigenvalues $\lambda_i, i > r$ will be zero. Therefore

$$\mathbf{A} = \sum_{i=1}^r \lambda_i^{1/2} \mathbf{p}_i \mathbf{q}_i^T \quad (5.5)$$

and the **pseudoinverse**

$$\mathbf{A}^+ = \sum_{i=1}^r \lambda_i^{-1/2} \mathbf{q}_i \mathbf{p}_i^T \quad (5.6)$$

The minimum error for the choice of the optimum weight matrix $\mathbf{W} = \mathbf{A}\mathbf{A}^+$ is given from Eq. (4.16) as

$$\begin{aligned} E_{\min} &= \frac{1}{L} \sum_{i=r+1}^L \|\mathbf{A}\mathbf{q}_i\|^2 \\ &= \frac{1}{L} \sum_{i=r+1}^L \left\| \sum_j \lambda_j^{1/2} \mathbf{p}_j \mathbf{q}_j^T \mathbf{q}_i \right\|^2 \\ &= \frac{1}{L} \sum_{i=r+1}^L \lambda_i \text{ since } \mathbf{q}_j^T \mathbf{q}_i = 0, \quad j \neq i \\ &= 1, \quad j = i \end{aligned} \quad (5.7)$$

But since $\lambda_i = 0$ for $i > r$, $E_{\min} = 0$. Thus in the case of linear autoassociative network there is no error in the recall due to linear dependency of the input patterns, unlike in the case of linear heteroassociative network. In other words, in this case the input comes out as output without any error.

When noise is added to the input vector, the noisy input vectors are given by

$$\mathbf{c}_l = \mathbf{a}_l + \boldsymbol{\varepsilon}, \quad l = 1, 2, \dots, L \quad (5.8)$$

where the noise **vector** $\boldsymbol{\varepsilon}$ is **uncorrelated** with the input vector \mathbf{a}_l , and has the average power or variance σ^2 . For the choice of $\mathbf{W} = \mathbf{A}\mathbf{A}^+$, the error in recall is given from Eq. (4.19) as [Murakami and Aibara, 1987]

$$\begin{aligned} E(\mathbf{W}) &= \frac{1}{L} \left[\sum_{i=r+1}^L \|\mathbf{A}\mathbf{q}_i\|^2 + L\sigma^2 \sum_{i=1}^r \lambda_i^{-1} \|\mathbf{A}\mathbf{q}_i\|^2 \right] \\ &= \sigma^2 r \end{aligned} \quad (5.9)$$

Thus the error in the recall is mainly due to noise, as the linear dependence component of the error is zero in the case of **auto-association**. Note that this is because a noisy input vector comes out

as a noisy output and hence its difference from the true vector in the recall is only due to noise.

The error in recall due to noise can be reduced by the choice of $W = A\hat{A}^+$, where

$$\hat{A}^+ = \sum_{i=1}^s \lambda_i^{-1/2} \mathbf{q}_i \mathbf{p}_i^T \quad (5.10)$$

where s is given by

$$\frac{L \sigma^2}{\lambda_s} < 1 < \frac{L \sigma^2}{\lambda_{s+1}} \quad (5.11)$$

That is, for a given noise power σ^2 , the error can be reduced by moving the error into the linear dependency term, which is realized by an appropriate choice of the number of terms in the expression for the pseudoinverse. The resulting error for the optimal choice of $W = A\hat{A}^+$ is

$$E_{\min} = \frac{1}{L} \left[\sum_{i=s+1}^r \lambda_i + L \sigma^2 s \right] \quad (5.12)$$

The linear **autoassociation** task can also be realized by a single layer feedback network with linear processing units shown in Figure 5.2. The

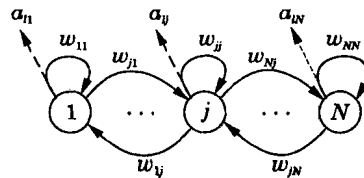


Figure 5.2 Linear autoassociation by a feedback network.

condition for autoassociation, namely, $W\mathbf{a}_i = \mathbf{a}_i$, is satisfied if $W = \mathbf{I}$, an identity matrix. This trivial choice of the weight matrix is realized if the input vectors are linearly independent, so that $W = A A^{-1} = \mathbf{I}$. For this choice of W , the output for a noisy input $\mathbf{a}_i + \boldsymbol{\varepsilon}$ is given by $W(\mathbf{a}_i + \boldsymbol{\varepsilon}) = \mathbf{a}_i + \boldsymbol{\varepsilon}$, which is the noisy input itself. This is due to lack of accretive **behaviour** during recall, and such a feedback network is not useful for storing information. It is possible to make a feedback network useful, especially for pattern storage, if the linear processing units are replaced with processing units having nonlinear output functions. We discuss this case in the next section and give a detailed analysis of pattern storage networks.

5.3 Analysis of Pattern Storage Networks

5.3.1 Pattern Storage Networks

The objective in a pattern storage task is to store a given set of

patterns, so that any of them can be recalled exactly when an approximate version of the corresponding pattern is presented to the network. For this purpose, the features and their spatial relations in the patterns need to be stored. The pattern recall should take place even when the features and their spatial relations are slightly disturbed due to noise and distortion or due to natural variation of the pattern generating process. The approximation of a pattern refers to the closeness of the features and their spatial relations to the original stored pattern.

Sometimes the data itself is actually stored through the weights, as in the case of binary patterns. In this case the approximation can be measured in terms of some distance, like Hamming distance, between the patterns. The distance is automatically captured by the threshold feature of the output functions of the processing units in a feedback network Freeman and Skapura, 1991.

Pattern storage is generally accomplished by a feedback network consisting of processing units with nonlinear output functions. The outputs of the processing units at any instant of time define the output state of the network at that instant. Likewise, the activation values of the units at any instant determine the activation state of the network at that instant.

The state of the network at successive instants of time, i.e., the trajectory of the state, is determined by the activation dynamics model used for the network. Recall of a stored pattern involves starting at some initial state of the network depending on the input, and applying the activation dynamics until the trajectory reaches an equilibrium state. The final equilibrium state is the stored pattern resulting from the network for the given input.

Associated with each output state is an energy (to be defined later) which depends on the network parameters like the weights and bias, besides the state of the network. The energy as a function of the state of the network corresponds to something like an *energy landscape*. The shape of the energy landscape is determined by the network parameters and states. The feedback among the units and the nonlinear processing in the units may create basins of attraction in the energy landscape, when the weights satisfy certain constraints. Figure 5.3 shows energy landscapes as a function of the output state for the two cases of with and without the basins of attraction. In the latter case the energy fluctuates quickly and randomly from one state to another as shown in Figure 5.3b. But in the energy landscape with basins of attraction as in Figure 5.3a, the states around the stable state correspond to small deviations from the stable state. The deviation can be measured in some suitable distance measure, such as Hamming distance for binary patterns. The Hamming distance between two binary patterns each of length N is defined as the number of bit positions in which the patterns differ. Thus the states

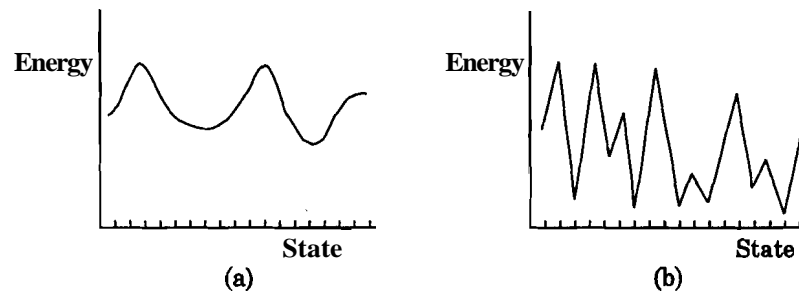


Figure 5.3 Energy landscapes (a) with basins of attraction and (b) without basins of attraction.

closer to the stable states correspond to patterns with smaller Hamming distance.

The basins of attraction in the energy landscape tend to be the regions of stable equilibrium states [Cohen and Grossberg, 1983]. If there is a fixed state in each of the basins where the energy is minimum, then that state corresponds to a fixed point of equilibrium. The basins could also be periodic (oscillatory) regions or chaotic regions of equilibria. For an oscillatory region, the state of the network changes continuously in a periodic manner. For a chaotic region, the state of the network is not predictable, but it is confined to the equilibrium region. Throughout the subsequent discussion we consider only the fixed points of equilibrium in the energy landscape.

It is the existence of the basins of attraction or regions of equilibrium states that is exploited for the pattern storage task. The fixed points in these regions correspond to the states of the energy minima, and they are used to store the desired patterns. These stored patterns can be recalled even with approximate patterns as inputs. **An** erroneous pattern is more likely to be closer to the corresponding true pattern than to the other stored patterns according to some distance measure. Each input pattern results in an initial state of the network, which may be closer to the desired true state in the sense that it may lie near the basin of attraction corresponding to the true state. **An** arbitrary state may not correspond to an equilibrium or a stable state. As the dynamics of the network evolves, the network may eventually settle at a stable state, from which the pattern may be read or derived.

Given a network specified by the number of processing units, their connection strengths and the activation dynamics, it is not normally possible to determine exactly the number of basins of attraction in the energy landscape as well as their relative spacings and depths in the state space of the network. The spacing between two states can be measured by a suitable distance measure, such as the Hamming distance for binary patterns. The number of patterns that can be stored is called the capacity of the network. It is possible to estimate

the capacity of the network and also the average probability of error in recall. The probability of error in recall can be reduced by adjusting the weights in such a way that the resulting energy landscape is matched to the probability distribution of the desired patterns.

Typically the capacity of a fully connected network is of the order of N , the number of processing units. Although there are 2^N different states for a network with binary state units, the network can be used to store only of the order of N binary **patterns**, as there will be only that many fixed points or energy minima in the energy landscape.

In general, the number of desired patterns is independent of the number of basins of attractions. The latter depends only on the network units and their interconnections. If the number of patterns is more than the number of basins of attraction, then the pattern storage problem becomes a hard problem, in the sense that the patterns cannot be stored in the given network. On the other hand, if the number of patterns is less than the number of basins of attraction, then there will be the so called false wells or minima due to the additional basins of attraction. During recall, it is likely that the state of the network, as it evolves from the initial state corresponding to the input pattern, may settle in a false well. The recalled pattern **corresponding** to the false well may not be the desired pattern, thus resulting in an **error** in the recall.

In the next subsection we will consider the **Hopfield** model of a feedback network for the pattern storage and discuss the working of a discrete **Hopfield** model. The **Hopfield** model is a fully connected feedback network with symmetric weights. In the discrete **Hopfield** network the state update is asynchronous and the units have **binary/bipolar** output functions. In the continuous **Hopfield** model the state update is dictated by the activation dynamics, and the units have continuous nonlinear output functions.

5.3.2 The Hopfield Model

Consider the **McCulloch-Pitts** neuron model for the units of a feedback network, where the output of each unit is fed to all the other units with weights w_{ij} , for all i and j . Let the output function of each of the units be bipolar (± 1) so that

$$s_i = f(x_i) = \text{sgn}(x_i), \quad (5.13)$$

and

$$x_i = \sum_{j=1}^N w_{ij} s_j - \theta_i \quad (5.14)$$

where θ_i is the threshold for the unit i . We will assume $\theta_i = 0$ for convenience. The state of each unit is either $+1$ or -1 at any given instant of time. Due to feedback, the state of a unit depends on the

states of the other units. The updating of the state of a unit can be done synchronously or asynchronously. In the synchronous update all the units are simultaneously updated at each time instant, assuming that the state of the network is frozen until update is made for **all** the units. In the asynchronous update a unit is **selected** at random and its new state is computed. Another unit is selected at random and its state is updated using the current state of the network. The **updating** using the random choice of a unit is continued until no further change in the state takes place for all the units. That is, the state at time $(t + 1)$ is the same as the state at time t for all the units. That is

$$s_i(t+1) = s_i(t), \quad \text{for all } i \quad (5.15)$$

In this situation we can say that the network activation dynamics reached a stable state. We assume **asynchronous** update throughout the following discussion. Note that the asynchronous update ensures that the next state is at most unit Hamming distance from the current state.

If the network is to store a pattern $\mathbf{a} = (a_1, a_2, \dots, a_N)^T$, then in a stable state we must have the updated state value to be the same as the current state value. That is

$$\text{sgn} \left(\sum_{j=1}^N w_{ij} a_j \right) = a_i, \quad \text{for all } i \quad (5.16)$$

This can happen if $w_{ij} = (1/N) a_i a_j$, because

$$\sum_{j=1}^N w_{ij} a_j = \frac{1}{N} \sum_{j=1}^N a_i a_j a_j = \frac{a_i}{N} \sum_{j=1}^N a_j^2 = a_i \quad (5.17)$$

where $a_j^2 = 1$ for bipolar (± 1) states.

For storing L patterns, we could choose a general Hebbian rule given by the **summation** of the Hebbian terms for each pattern. That is,

$$w_{ij} = \frac{1}{N} \sum_{l=1}^L a_{li} a_{lj} \quad (5.18)$$

Then the state \mathbf{a}_k will be stable if

$$\text{sgn} \left(\frac{1}{N} \sum_{j=1}^N \sum_{l=1}^L a_{li} a_{lj} a_{kj} \right) = a_i, \quad \text{for all } i \quad (5.19)$$

Taking out the $l = k$ term in the summation and simplifying it using $a_{kj}^2 = 1$, we get

$$\text{sgn} \left(a_{ki} + \frac{1}{N} \sum_{j=1}^N \sum_{l \neq k}^L a_{li} a_{lj} a_{kj} \right) = a_i, \quad \text{for all } i \quad (5.20)$$

Since $a_{ki} = \pm 1$, the above is true for all a_{ki} , provided the **crossterm**

in Eq. (5.20) does not change the sign of a_{ki} plus the crossterm. Table 5.2 gives an algorithm for storing and recall of patterns in a **Hopfield** network.

Table 6.2 **Hopfield** Network Algorithm to Store and Recall a Set of Bipolar Patterns

Let the network **consist** of N fully connected **units** with each unit having hard-limiting bipolar threshold output function. Let $\mathbf{a}_l, l = 1, 2, \dots, L$ be the vectors to be stored. The vectors $\{\mathbf{a}_l\}$ are assumed to have bipolar components, i.e., $a_{li} = \pm 1, i = 1, 2, \dots, N$.

1. **Assign** the connection weights

$$w_{ij} = \frac{1}{N} \sum_{l=1}^L a_{li} a_{lj}, \text{ for } i \neq j$$

$$= 0, \text{ for } i = j, 1 \leq i, j \leq N$$

2. Initialize the network output with the given **unknown** input pattern a

$$s_i(0) = a_i, \text{ for } i = 1, 2, \dots, N$$

where $s_i(0)$ is the output of the unit i at time $t = 0$

3. Iterate until convergence

$$s_i(t+1) = \text{sgn} \left[\sum_{j=1}^N w_{ij} s_j(t) \right], \text{ for } i = 1, 2, \dots, N$$

The process is repeated until the outputs **remain** unchanged with further iteration. The steady outputa of the **units** represent the stored pattern that best matches the given input.

In general, the **crossterm** in Eq. (5.20) is negligible if $L \ll N$. Eq. (5.20) is satisfied if the number of patterns L is limited to the storage capacity of the network, i.e., the maximum number of patterns that can be stored in the network.

5.3.3 Capacity of Hopfield Model

We consider the discrete **Hopfield** model to derive the capacity of the network. Let us consider the following quantity [Hertz et al, 1991]

$$c_i^k = -a_{ki} \frac{1}{N} \sum_{j=1}^N \sum_{l \neq k} a_{li} a_{lj} a_{kj} \quad (5.21)$$

If c_i^k is negative then the cross term and a_{ki} have the same sign in Eq. (5.20) and hence the pattern \mathbf{a}_k is stable. On the other hand, if c_i^k is positive and greater than 1, then the sign of the cross term changes the sign of a_{ki} plus the cross term in Eq. (5.20). The result is that the pattern \mathbf{a}_k turns out to be unstable, and hence the desired pattern cannot be stored.

Therefore the probability of **error** is given by

$$P_e = \text{Prob}(c_i^k > 1) \quad (5.22)$$

To compute this probability, let us assume that the probability of a_{ij} equal to **+1** or **-1** is 0.5. For random **patterns**, the cross term corresponds to $1/N$ times the sum of about NL independent random numbers, each of which is **+1** or **-1**. Thus c_i^k is a sum of random variables having a binomial distribution with zero mean and **variance** $\sigma^2 = L/N$. If NL is assumed large, then the distribution of c_i^k can be approximated by a Gaussian distribution with zero mean and σ^2 variance [Papoulis, 1991]. Therefore,

$$\begin{aligned} P_e &= \frac{1}{\sqrt{2\pi}\sigma} \int_1^{\infty} e^{-x^2/(2\sigma^2)} dx \\ &= \frac{1}{2} \left[1 - \text{erf} \left(\sqrt{\frac{N}{2L}} \right) \right] \end{aligned} \quad (5.23)$$

where $\text{erf}(x)$ is error function given by

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2} dx \quad (5.24)$$

This gives a value of $L_{\max}/N = 0.105$ for $P_e = 0.001$. Thus the maximum number of patterns that can be stored for a probability of error of 0.001 is $L_{\max} = 0.105 N$.

A more sophisticated calculation [Amit et al, 1987; Amit, 1989] using probabilistic update leads to a capacity of $L_{\max} = 0.138 N$.

5.3.4 Energy Analysis of Hopfield Network

Discrete Hopfield model: Associated with each state of the network, **Hopfield** proposed an energy function whose value always either reduces or remains the same as the state of the network changes. Assuming the threshold value of the unit i to be θ_i , the energy function is given by [Hopfield, 1982]

$$V(\mathbf{s}) = V = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i \theta_i s_i \quad (5.25)$$

The energy $V(\mathbf{s})$ as a function of the state \mathbf{s} of the network describes the energy landscape in the state space. The energy landscape is determined only by the network architecture, i.e., the number of units, their output functions, threshold values, connections between units and the strengths of the connections. **Hopfield** has shown that for symmetric weights with no self-feedback, i.e., $w_{ij} = w_{ji}$, and with bipolar **-1, +1** or binary **{0, 1}** output functions, the dynamics of the

network using the asynchronous update always leads towards energy minima at equilibrium. The states corresponding to these energy minima turn out to be stable states, which means that small perturbations around it lead to unstable states. Hence the dynamics of the network takes the network back to a stable state again. It is the existence of these stable states that enables us to store patterns, one at each of these states.

To show that $\Delta V \leq 0$, let us consider the change of state due to update of one unit, say k , at some instant. All other units remain unchanged. We can write the expressions for energy before and after the change as follows [Freeman and Skapura, 1991]:

$$\begin{aligned} V^{\text{old}} &= -\frac{1}{2} \sum_i \sum_j w_{ij} s_i^{\text{old}} s_j^{\text{old}} + \sum_i \theta_i s_i^{\text{old}} \\ V^{\text{new}} &= -\frac{1}{2} \sum_i \sum_j w_{ij} s_i^{\text{new}} s_j^{\text{new}} + \sum_i \theta_i s_i^{\text{new}} \end{aligned} \quad (5.26)$$

The change in energy due to update of the k th unit is given by

$$\begin{aligned} \Delta V &= V^{\text{new}} - V^{\text{old}} \\ &= -\frac{1}{2} \sum_{i \neq k} \sum_{j \neq k} w_{ij} (s_i^{\text{new}} s_j^{\text{new}} - s_i^{\text{old}} s_j^{\text{old}}) + \sum_{i \neq k} \theta_i (s_i^{\text{new}} - s_i^{\text{old}}) \\ &\quad - \frac{1}{2} \sum_i w_{ik} s_i^{\text{new}} s_k^{\text{new}} - \frac{1}{2} \sum_j w_{kj} s_k^{\text{new}} s_j^{\text{new}} + \theta_k s_k^{\text{new}} \\ &\quad + \frac{1}{2} \sum_i w_{ik} s_i^{\text{old}} s_k^{\text{old}} + \frac{1}{2} \sum_j w_{kj} s_k^{\text{old}} s_j^{\text{old}} - \theta_k s_k^{\text{old}} \end{aligned} \quad (5.27)$$

Since $s_i^{\text{new}} = s_i^{\text{old}}$, for $i \neq k$, the first two terms on the right hand side of Eq. (5.27) will be zero. Hence,

$$\begin{aligned} \Delta V &= s_k^{\text{new}} \left[-\frac{1}{2} \sum_i w_{ik} s_i^{\text{new}} - \frac{1}{2} \sum_j w_{kj} s_j^{\text{new}} + \theta_k \right] \\ &\quad + s_k^{\text{old}} \left[+\frac{1}{2} \sum_i w_{ik} s_i^{\text{old}} + \frac{1}{2} \sum_j w_{kj} s_j^{\text{old}} - \theta_k \right] \end{aligned} \quad (5.28)$$

If the weights are assumed symmetric, i.e., $w_{ij} = w_{ji}$, then we get

$$\Delta V = -s_k^{\text{new}} \left[\sum_i w_{ki} s_i^{\text{new}} - \theta_k \right] + s_k^{\text{old}} \left[\sum_i w_{ki} s_i^{\text{old}} - \theta_k \right] \quad (5.29)$$

If, in addition, $w_{kk} = 0$, then since $s_i^{\text{new}} = s_i^{\text{old}}$ for $i \neq k$, the terms in both the parentheses are equal. Therefore,

$$\Delta V = (s_k^{\text{old}} - s_k^{\text{new}}) \left[\sum_i w_{ki} s_i^{\text{old}} - \theta_k \right] \quad (5.30)$$

The update rule for each unit k is as follows:

Case A: If $\sum_i w_{ki} s_i^{\text{old}} - \theta_k > 0$, then $s_k^{\text{new}} = +1$

Case B: If $\sum_i w_{ki} s_i^{\text{old}} - \theta_k < 0$, then $s_k^{\text{new}} = -1$

Case C: If $\sum_i w_{ki} s_i^{\text{old}} - \theta_k = 0$, then $s_k^{\text{new}} = s_k^{\text{old}}$

For case A, if $s_k^{\text{old}} = +1$, then $\Delta V = 0$, and if $s_k^{\text{old}} = -1$, then $\Delta V \leq 0$.
For case B, if $s_k^{\text{old}} = +1$, then $\Delta V < 0$, and if $s_k^{\text{old}} = -1$, then $\Delta V = 0$.
For case C, irrespective of the value of s_k^{old} , $\Delta V = 0$.

Thus we have $\Delta V \leq 0$. Therefore the energy decreases or remains the same when a unit, selected at random, is updated, provided the weights are symmetric, and the self-feedback is zero. This is the energy analysis for discrete **Hopfield** model.

That the expression for V in Eq. (5.25) does indeed represent some form of energy can be seen from the following arguments based on **Hebb's** law:

If a given pattern vector \mathbf{a}_l is to be stored in the network state vector \mathbf{s} , then the match will be perfect if both the vectors coincide. That is, the magnitude of their inner product is maximum. Alternatively, the negative of the magnitude of their inner product is minimum. Thus we can choose a quantity [**Hertz** et al, 1991]

$$V = -\frac{1}{2N} \left(\sum_{i=1}^N s_i a_{li} \right)^2 \quad (5.31)$$

to be minimized for storing a pattern vector \mathbf{a}_l in the network. For storing L pattern vectors we can write the resulting V as a summation of the contributions due to each pattern vector. That is

$$\begin{aligned} V &= -\frac{1}{2N} \sum_{l=1}^L \left(\sum_{i=1}^N s_i a_{li} \right)^2 \\ &= -\frac{1}{2N} \sum_{i=1}^N \sum_{j=1}^N s_i s_j \sum_{l=1}^L a_{li} a_{lj} \end{aligned} \quad (5.32)$$

If we identify the weights w_{ij} with the term $(1/N) \sum_{l=1}^L a_{li} a_{lj}$, then we get

$$V = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N w_{ij} s_i s_j \quad (5.33)$$

which is same as the first term in the **Hopfield** energy expression given in Eq. (5.25).

This method of identifying w_{ij} from an energy function is useful, especially to solve several optimization problems. Given an energy function or a cost function or an objective function for a problem in terms of its variables and constraints, if we can identify the coefficients associated with s_i, s_j, s_k and constant terms in the function, then a feedback network can be built with weights corresponding to these coefficients. Then using an activation dynamics for the network, the equilibrium state or states can be found. These states correspond to the minima or maxima of the energy function. Higher order terms consisting of product of three (s_i, s_j, s_k) or more variables cannot be handled by the feedback model with pairwise connections.

Continuous Hopfield model. In this subsection we will consider the energy analysis for a continuous Hopfield model [Hopfield, 1984; Hertz et al, 1991; Freeman and Skapura, 1991]. A continuous model is a fully connected feedback network with a continuous nonlinear output function in each unit. The output function is typically a sigmoid function $f(\lambda x) = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$ which is shown in Figure 5.4 for different

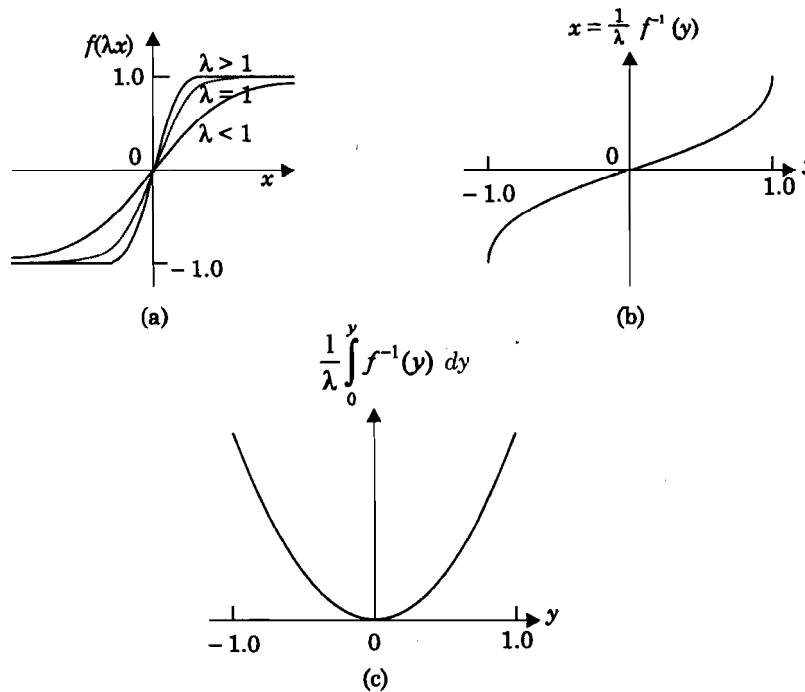


Figure 5.4 (a) Sigmoid function $f(x) = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$ for different values of gain parameter λ . (b) The inverse function. (c) Contribution of $f(\cdot)$ to the energy function.

values of the gain parameter λ . In the continuous model all the units will change their output signals (s_i) continuously and **simultaneously** towards values determined by the output function. The activation values (x_i) will also change continuously according to $x_i = \sum_j w_{ij} s_j$. This is reflected in the following equation for the activation dynamics:

$$\tau_i \dot{x}_i = -x_i + \sum_j w_{ij} s_j \quad (5.34)$$

where τ_i is the time constant and $s_i = f(x_i)$.

Consider the following energy function [Hopfield, 1984]:

$$V = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i \int_0^{s_i} f^{-1}(s) ds \quad (5.35)$$

We can show that in this case $(dV/dt) \leq 0$.

$$\frac{dV}{dt} = -\frac{1}{2} \sum_i \sum_j w_{ij} \frac{ds_i}{dt} s_j - \frac{1}{2} \sum_i \sum_j w_{ij} s_i \frac{ds_j}{dt} + \sum_i f^{-1}(s_i) \frac{ds_i}{dt} \quad (5.36)$$

Assuming symmetry of weights, i.e., $w_{ij} = w_{ji}$, we get

$$\begin{aligned} \frac{dV}{dt} &= -\sum_i \frac{ds_i}{dt} \left[\sum_j w_{ij} s_j - f^{-1}(s_i) \right] \\ &= -\sum_i \frac{ds_i}{dt} \left[\sum_j w_{ij} s_j - x_i \right] \end{aligned} \quad (5.37)$$

Using the relation in Eq. (5.34), we get

$$\begin{aligned} \frac{dV}{dt} &= -\sum_i \tau_i \frac{ds_i}{dt} \frac{dx_i}{dt} \\ &= -\sum_i \tau_i f'(x_i) \left(\frac{dx_i}{dt} \right)^2 \end{aligned} \quad (5.38)$$

Since $f(x)$ is a monotonically increasing function, $f'(x) > 0$. Hence $dV/dt \leq 0$.

Note that $dV/dt = 0$ when $dx_i/dt = 0$, for **all** i . This shows that the activation dynamics eventually leads to a state where the energy function has a local minimum value, i.e., $dV/dt = 0$. This happens when the activation state reaches an equilibrium steady state at which there is no further change in the activation values, i.e., $dx_i/dt = 0$. The above result, namely, $dV/dt \leq 0$, shows that the energy always decreases as the state of the network changes.

Let us examine the differences between the continuous model and the discrete model. In the discrete model only one unit is considered

at a time for update. The choice of the unit for update is random and the dynamics is that of the steady activation values ($\dot{\mathbf{x}}_i = 0$), since the transients are assumed to have died down at each update of the state of the unit. Hence in the discrete case $\dot{\mathbf{x}}_i = \mathbf{0}$ and $\dot{V}(\mathbf{x}) = \mathbf{0}$ are different conditions. In the continuous case the states of all the units and hence the state of the network change continuously, as dictated by the differential equations for the activation dynamics. Hence, in this case $\dot{\mathbf{x}}_i = \mathbf{0}$ for all i implies that $\dot{V} = \mathbf{0}$. The energy function V is also called the **Lyapunov** function of the dynamical system.

The **difference** in the energy functions for the discrete and continuous case is due to the extra term $\sum_i \int_0^{s_i} f^{-1}(s) ds$ in Eq. (5.35).

This expression is for a gain value $\lambda = 1$. For a general gain value this term is given by $(1/\lambda) \sum_i \int_0^{s_i} f^{-1}(s) ds$. The integral term is $\mathbf{0}$ for

$s_i = \mathbf{0}$ and becomes very large as s_i approaches ± 1 (see Figure 5.4c). But for high gain values ($h \gg 1$), this term in the energy function becomes negligibly small, and hence the energy function approaches that of the discrete case. In fact when $\lambda \rightarrow \infty$, the output function becomes a bipolar function, and hence is equal to the discrete case. In the discrete case the energy minima are at some of the corners of the hypercube in the N -dimensional space, since all the states are at the corners of the hypercube. On the other hand, for moderate or small values of h , the integral term contributes to large positive values near the surfaces, edges and corners of the hypercube, and it contributes small values interior to the hypercube. This is because the value of s_i is $\mathbf{1}$ at the surfaces, edges and corners. Thus the energy minima will be **displaced** to the interior of the hypercube. As $\lambda \rightarrow 0$, the minima of the energy function disappear one by one, since all the states will tend to have the same energy value.

The energy analysis so far shows that, for symmetric weights on the connections, there exist basins of attraction with a fixed point or a stable point for each basin corresponding to an energy minimum. If the connections are not symmetric, then the basins of attraction may correspond to oscillatory or chaotic states regions. In the case of purely random connections, with mean $\mathbf{0}$ and variance σ^2 , there will be a transition from stable to chaotic behaviour as σ^2 is **increased** [Sompolinsky et al, 1988; Hertz, 1995; Hertz et al, 1991].

We can summarize the behaviour of feedback networks in relation to the complexity of the network as follows: To make a network useful for pattern storage, the output functions of the units are made **hard-limiting** nonlinear units. For analysis in terms of storage capacity, as well as for the recall of information from the stable states, we have imposed **Hopfield** conditions of symmetry of weights and asynchronous update. A more natural situation will be to use continuous

output functions, so that any type of pattern can be stored. But the analysis of the performance of the network will be more difficult. In addition, if we relax the conditions on the symmetry of weights, we may still get stable regions, but it is not possible to analyse the network in terms of its storage capacity and retrieval of information. If we further relax the constraints to make the feedback system more closer to the natural biological system, then we may be able to get better functionality, but it is almost impossible to analyse such complex networks. For example it is not possible to predict the global pattern behaviour of a feedback network with random weights. Thus, although the networks may get more and more powerful by relaxing the constraints on the network, they become less useful, if we cannot predict and control the pattern storage and recall of the desired information.

5.3.5 State Transition Diagram

Derivation of state transition diagram: The energy analysis of the Hopfield network in the previous section shows that the energy of the network at each state either decreases or remains the same as the network dynamics evolves. In other words, the network either remains in the same state or moves to a state having a lower energy. This can also be demonstrated by means of a state transition diagram which gives the states of the network and their energies, together with the probability of transition from one state to another. In this section we will illustrate the state transition diagram (adapted from [Aleksander and Morton, 1990]) for a 3-unit feedback network with symmetric weights $w_{ij} = w_{ji}$. The units have a threshold value of θ_i , $i = 1, 2, 3$ and a binary $\{0, 1\}$ output function. A binary output function is assumed for convenience, although the conclusions are equally valid for the bipolar $\{-1, +1\}$ case.

Figure 5.5 shows a 3-unit feedback network. The state update for the unit i is governed by the following equation:

$$\begin{aligned} s_i(t+1) &= 1, & \text{if } \sum_j w_{ij} s_j(t) > \theta_i \\ &= 0, & \text{if } \sum_j w_{ij} s_j(t) \leq \theta_i \end{aligned} \quad (5.39)$$

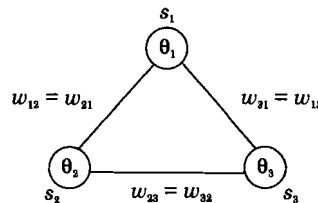


Figure 5.5 A 3-unit feedback network with symmetric weights w_{ij} , threshold values θ_i and the output states s_i , $i = 1, 2, 3$.

Analysis of Pattern Storage Networks

The energy at any state $s_1 s_2 s_3$ of the network is given by

$$V(s_1 s_2 s_3) = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i s_i \theta_i \quad (5.40)$$

There are eight different states for the 3-unit network, as each of the s_i may assume a value either 0 or 1. Thus the states are: 000, 001, 010, 100, 011, 101, 110 and 111. Assuming the values

$$\begin{aligned} w_{12} = w_{21} = -0.5, \quad w_{23} = w_{32} = 0.4, \quad w_{31} = w_{13} = 0.5 \\ \theta_1 = -0.1, \quad \theta_2 = -0.2, \quad \text{and} \quad \theta_3 = 0.7, \end{aligned}$$

we get the following energy values, for each state.

$$\begin{aligned} V(000) = 0.0, \quad V(001) = 0.7, \quad V(010) = -0.2, \quad V(100) = -0.1, \\ V(011) = 0.1, \quad V(101) = 0.1, \quad V(110) = 0.2, \quad \text{and} \quad V(111) = 0.0. \end{aligned}$$

The transition from any state to the next state can be computed using the state update Eq. (5.39). For example, if the current state is 000, by selecting any one unit, say unit 2, at random, we can find its next state by computing the activation value $x_2 = \sum_{j=1}^3 w_{2j} s_j$ and comparing it with the threshold θ_2 . Since $x_2 (= 0) > \theta_2 (= -0.2)$ the state of the unit 2 changes from 0 to 1. Thus if we select this unit, there will be a transition from the state 000 to 010. Since we can select any one of the three units with equal probability, i.e., U3, the probability of making a transition from 000 to 010 is thus U3. Likewise by selecting the unit 1 for update, the network makes a transition from 000 to 100 with a probability U3. Selecting the unit 3 for update results in a transition from 000 to itself, since the activation $x_3 (= 0) < \theta_3 (= 0.7)$. By computing the transition probabilities for all the states, we get the state transition diagram shown in Figure 5.6. Note that while computing the transitions, only asynchronous update of each unit selected at random was used. Table 5.3 shows the computation of the state transitions by comparing the weighted inputs with the threshold value for each unit. The entries in the parenthesis are $(\sum_j w_{ij} s_j < > \theta_i)$.

From the state transition diagram we observe the following points: The diagram is drawn in such a way that the higher energy states are shown above the lower energy states. The transition is always from a higher energy state to a state with equal or lower energy. Thus the Hopfield result $AV \leq 0$ is satisfied. There are some states 010, 100 and 111 which have a self-transition probability of 1. That means, once these states are reached, the network remains in these states, which is equivalent to saying that the activation dynamics equation is such that

$$f\left(\sum_{j=1}^3 w_{ij} s_j - \theta_i\right) = s_i, \quad i = 1, 2, 3 \quad (5.41)$$

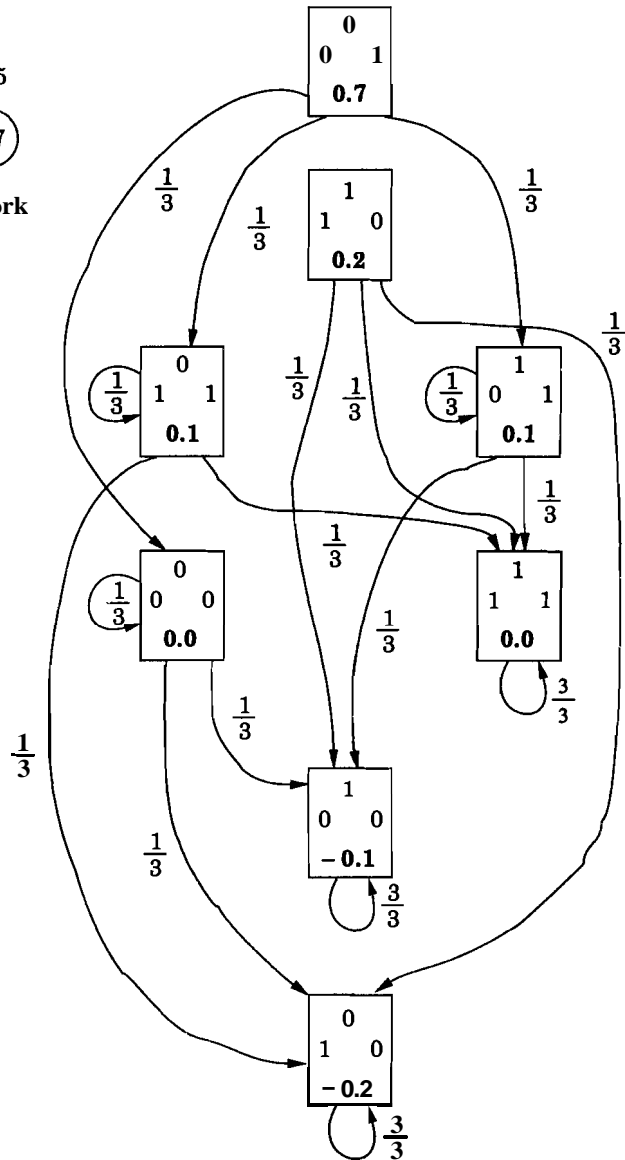
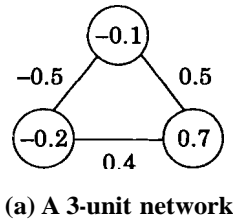


Figure 5.6 A 3-unit network and the corresponding state transition diagram. (Adapted from [Aleksander and Morton, 1990]).

where $f(\cdot)$ is the binary output function, i.e., $f(x) = 0$, for $x \leq 0$ and $f(x) = 1$, for $x > 0$. Since there is no transition from these states to other states, these are stable states. Note that only three out of the total eight are stable states. As per the approximate capacity calculations made in Section 5.3.3 for a Hopfield network, the number

Table 6.3 Computation of State Transitions for Figure 5.6

	Unit 1	Unit 2	Unit 3
000	(0 > - 0.1) 100	(0 > - 0.2) 010	(0 < 0.7) 000
001	(0.5 > - 0.1) 101	(0.4 > - 0.2) 011	(0 < 0.7) 000
010	(- 0.5 < - 0.1) 010	(0 > - 0.2) 010	(0.5 < 0.7) 010
100	(0 > - 0.1) 100	(- 0.5 < - 0.2) 100	(0.5 < 0.7) 100
011	(0 > - 0.1) 111	(0.4 > - 0.2) 011	(0.4 < 0.7) 010
101	(0.5 > - 0.1) 101	(- 0.1 > - 0.2) 111	(0.5 < 0.7) 100
110	(- 0.5 < - 0.1) 010	(- 0.5 < - 0.2) 100	(0.9 > 0.7) 111
111	(0 > - 0.1) 111	(- 0.1 > - 0.2) 111	(0.9 > 0.7) 111

of stable states will be much fewer than the number of possible states, and in fact the number of stable states are of the order N . The stable states are always at the energy minima, so that the transition to any of these states is always from a state with a higher energy value than the energy value of the stable state.

Computation of weights for pattern storage: So far we have considered the analysis of a given feedback network and studied its characteristics. But patterns can be stored at the stable states by design. That is, it is possible to determine the weights of a network by calculation in order to store a given set of patterns in the network. Let 010 and 111 be the two patterns to be stored in a 3-unit binary network. Then at each of these states the following activation dynamics equations must be satisfied:

$$f\left(\sum_j w_{ij} s_j - \theta_i\right) = s_i, \quad i = 1, 2, 3 \quad (5.42)$$

This will result in the following inequalities for each of the states: For the state $s_1 s_2 s_3 = 010$

$$w_{12} - \theta_1 \leq 0, \quad w_{22} - \theta_2 > 0, \quad w_{32} - \theta_3 \leq 0,$$

and for the state $s_1 s_2 s_3 = 111$

$$w_{11} + w_{12} + w_{13} - \theta_1 > 0, \quad w_{21} + w_{22} + w_{23} - \theta_2 > 0,$$

$$w_{31} + w_{32} + w_{33} - \theta_3 > 0$$

Since we assume symmetry of the weights ($w_{ij} = w_{ji}$) and $w_{ii} = 0$, the above inequalities reduce to

$$w_{12} \leq \theta_1, \theta_2 < 0, w_{23} \leq \theta_3, w_{12} + w_{31} > \theta_1, w_{12} + w_{23} > \theta_2, w_{31} + w_{23} > \theta_3$$

The following choice of the thresholds and weights, namely,

$$\theta_1 = 0.1, \theta_2 = -0.2, \theta_3 = 0.7, w_{12} = -0.5, w_{23} = 0.4, w_{31} = 0.7$$

satisfies the above inequalities and hence the resulting network given in Figure 5.7a stores the given two patterns. These two patterns correspond to the stable states 010 and 111 in the network as can be seen from the state transition diagram in Figure 5.7b. The energy values for different states are as follows:

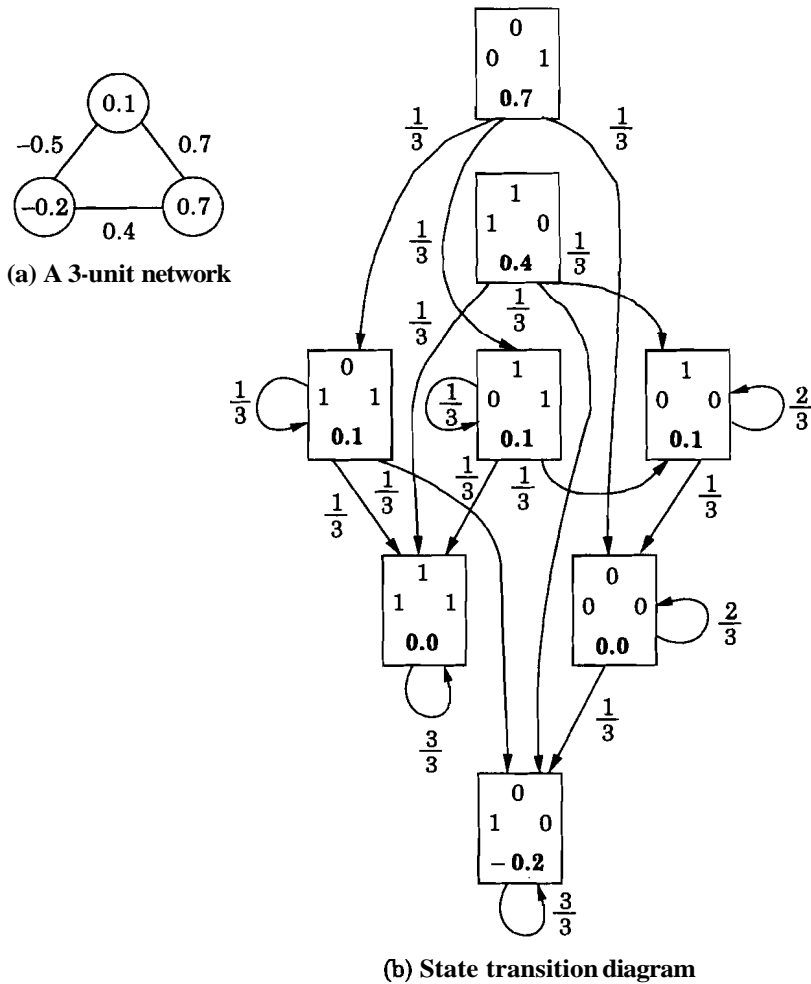


Figure 5.7 A 3-unit network and the corresponding state transition diagram. (Adapted from [Aleksander and Morton, 1990]).

Analysis of Pattern Storage Networks

The energies of different states are:

$$V(000) = 0.0, \quad V(001) = 0.7, \quad V(010) = -0.2, \quad V(100) = 0.1 \\ V(011) = 0.1, \quad V(101) = 0.1, \quad V(110) = 0.4, \quad V(111) = 0.0$$

Table 5.4 shows the computation of the state transitions by comparing the weighted inputs with the threshold values for each unit in each state. The entries in the parenthesis are $(\sum_j w_{ij} s_j \leq \theta_i)$.

Table 6.4 Computation of State Transitions for Figure 5.7

	unit 1	Unit 2	Unit 3
000	(0 < 0.1) 000	(0 > -0.2) 010	(0 < 0.7) 000
001	(0.7 > 0.1) 101	(0.4 > -0.2) 011	(0 < 0.7) 000
010	(-0.5 < 0.1) 010	(0 > -0.2) 010	(0.4 < 0.7) 010
100	(0 < 0.1) 000	(-0.5 < -0.2) 100	(0.7 = 0.7) 100
011	(0.2 > 0.1) 111	(0.4 > -0.2) 011	(0.4 < 0.7) 010
101	(0.7 > 0.1) 101	(-0.1 > -0.2) 111	(0.7 = 0.7) 100
110	(-0.5 < 0.1) 010	(-0.5 < -0.2) 100	(1.1 > 0.7) 111
111	(0.2 > 0.1) 111	(-0.1 > -0.2) 111	(1.1 > 0.7) 111

5.3.6 Pattern Storage by Computation of Weights-Problem of False Energy Minima

For another choice of the values of θ_i and w_{ij} which satisfies all the inequalities for the above problem of storage of the patterns 010 and 111 in a 3-unit network, there may be more than two energy minima or stable states in the network. Two of them correspond to the desired patterns and the other extra states correspond to false minima. This is illustrated for the choice of the thresholds and weights shown in **Figure 5.6a**. The corresponding state **transition** diagram is given in the **Figure 5.6b**. Here there are three energy minima corresponding to the three stable states 010, 100, 111.

The presence of the extra stable state may result in recalling a pattern not in the set of the desired patterns to be stored. If an approximate input is given to the units in the network, so that the network is forced into the state, say $s_1 s_2 s_3 = 000$ initially, then since this state is unstable, the dynamics of the network will eventually lead to either the state 010 (the desired pattern) or to the state 100

(See the state transition diagram in **Figure 5.6b**). Both these states are stable states and have equal probability of transition from the initial state 000. While the state 010 is the desirable pattern to be recalled, there is an equal chance that the pattern 100 may be recalled. Likewise, if the initial state is 110, then there is an equal chance that any one of the stable states 010, 100 and 111 may be recalled. The recall of the pattern 111 results in an undetectable error, as the desired pattern is 010 for the approximate input 110. The recall of the pattern 100 at any time will give as output a pattern which was not stored in the network intentionally, since in our pattern storage task we have specified only 010 and 111 as the desired patterns to be stored in the network. The stable state 100 in this case corresponds to a false (undesirable) energy minimum.

Errors in recall due to false minima can be reduced in two ways:

1. By designing the energy minima for the given patterns in an optimal way, so that the given patterns correspond to the lowest energy minima in the network.
2. By using a stochastic update of the state for each unit, instead of the deterministic update dictated by the activation values and the output function.

The issue of stochastic update will be discussed in Section 5.4, and the issue of designing energy wells by learning in Section 5.5.

Pattern storage—Hard problems: In the previous subsection we have discussed the **effect** of having more minima in the energy landscape than the number required to store the given patterns. In this section we consider the case of the so called *hard* problems of pattern storage. Let us consider the problem of storing the patterns say 000, 011, 101 and 110. By using the condition $f\left(\sum_j w_{ij}s_j - \theta_i\right) = s_i$ for each unit i , the inequalities to be satisfied to make these states stable in a 3-unit feedback network can be derived. In this case no choice of thresholds and weights can satisfy all the constraints in the inequalities. The reason is that the number of desired patterns is more than the capacity of the network, and hence they cannot be **represented/stored** in a feedback network with 3 units. In some cases, even if the number of desired patterns is within the capacity limit of a network, the **specific** patterns may not be representable in a given type (binary) of a feedback network. For example, for storing the patterns 000 and 100, the following inequalities have to be satisfied by the type of network we have been considering so far.

$$\theta_1 \geq 0, \theta_2 \geq 0, \theta_3 \geq 0, \text{ and } \theta_1 < 0, w_{21} \leq \theta_2, w_{13} \leq \theta_3 \quad (5.43)$$

The conditions on $\theta_1 < 0$ and $\theta_1 \geq 0$ cannot obviously be satisfied simultaneously by any choice of θ_1 . In fact any pair of patterns within a Hamming distance of 1 cannot be stored in a 3-unit network.

Pattern storage problems which **cannot** be represented by a feedback network of a given size, can be called hard problems. This is analogous to the hard problems in the pattern classification task for a single layer perceptron network. Hard problems in the pattern storage task are handled by introducing additional units in the feedback network. These units are called hidden units. But with hidden units it is difficult to write a set of inequalities **as** before to make the given patterns correspond to stable states in the feedback network. **Thus** the design of a network with hidden units becomes difficult due to lack of a straightforward approach for determining the weights of the network. In other words, this may be viewed as hard learning problems. We will see in Section 5.5 how this problem is addressed in **Boltzmann** learning law. To store a given number of patterns, a network with **sufficiently** large number of units may have to be considered. But in general it is difficult to know the required number of units exactly for a given number of patterns to be stored.

5.4 Stochastic Networks and Simulated Annealing

5.4.1 Stochastic Update

Error in pattern recall due to false minima can be reduced significantly if initially the desired patterns are stored (by careful training) at the lowest energy minima of a network. The error can be reduced further by using suitable activation dynamics. Let us assume that by training we have achieved a set of weights which will enable the desired patterns to be stored at the lowest energy minima. The activation dynamics is modified so that the network can also move to a state of higher energy value initially, and then to the nearest deep energy minimum. This way errors in recall due to false minima can be reduced.

It is possible to realize a transition to a higher energy state from a lower energy state by using a stochastic update in each unit instead of the deterministic update of the output function as in the **Hopfield** model. In a stochastic update the activation value of a unit does **not** decide the next output state of the unit by directly using the output function $f(x)$ as shown in Figure 5.8a. Instead, the update is expressed in probabilistic terms, like the probability of firing by the unit being greater than 0.5 if the activation value exceeds a threshold, and less than 0.5 if the activation value is less than the threshold. Note that the output function $f(x)$ is still a nonlinear function, either a hard-limiting threshold logic function or a semilinear sigmoidal function, but the function itself is applied in a stochastic manner.

Figure 5.8b shows a typical probability function that can be used for stochastic update of units. The output function itself is the binary logic function $f(x)$ shown in Figure 5.8a.

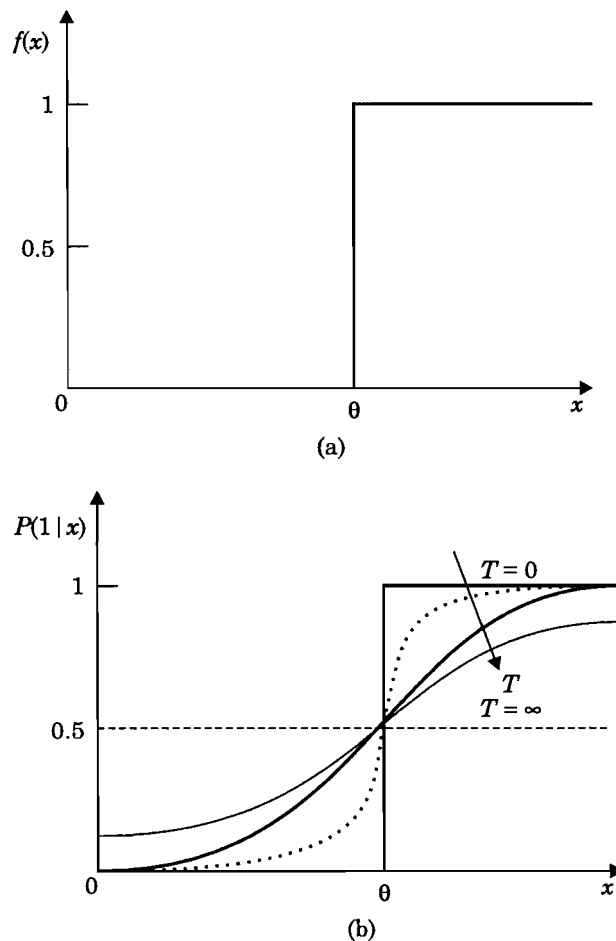


Figure 5.8 Stochastic update of a unit using the probability law $P(s=1|x) = 1/(1 + e^{-(x-\theta)/T})$. (a) Binary output function and (b) Probability function for stochastic update for different values of T .

The probability of firing for an activation value of x can be expressed as

$$P(s=1|x) = \frac{1}{1 + e^{-(x-\theta)/T}} \quad (5.44)$$

The probability function is defined in terms of a parameter called temperature T . At $T = 0$, the probability function is sharp with a discontinuity at $x = \theta$. In this case the stochastic update reduces to the deterministic update used in the Hopfield analysis. As the temperature is increased, the uncertainty in making the update according to $f(x)$ increases, giving thus a chance for the network to go to a higher energy state. Therefore the result of the Hopfield

energy analysis, namely $\Delta V \leq 0$, will be no longer true for nonzero temperatures. Finally, when $T = \infty$, then the update of the unit does not depend on the activation value (x) any more. The state of a unit changes randomly from $\mathbf{1}$ to $\mathbf{0}$ or vice versa. It is important to note that the stochastic update for different T does not change the energy landscape itself, since the shape of the landscape depends on the network, its weights and the output function, which are fixed. Only the traversal in the landscape will be changing. In contrast, in the continuous **Hopfield** model, the output function is different for different values of the gain parameter λ , and hence the energy landscape itself is different for different λ (See **Sec. 5.3.4**).

5.4.2 Equilibrium of Stochastic Networks

A feedback neural network with N binary units and stochastic update of the units is described by the following set of equations:

Assuming the threshold value $\theta_i = 0$,

$$x_i = \sum_j w_{ij} s_j, \quad i = 1, 2, \dots, N \quad (5.45)$$

$$\begin{aligned} f(x_i) &= 0, \quad \text{for } x_i \leq 0 \\ &= 1, \quad \text{for } x_i > 0 \end{aligned} \quad (5.46)$$

A unit i is selected at random for updating. The output is updated according to the stochastic update law, specified by the probability that the output $s_i = 1$ given the activation x_i . It is given by

$$P(s_i = 1 | x_i) = \frac{1}{1 + \exp(-x_i/T)} \quad (5.47)$$

A network with the above dynamics is called a stochastic network.

A stochastic network will evolve differently each time it is **run**, in the sense that the trajectory, of the state of the network becomes a sample function of a random process. In the case of deterministic update the trajectories will eventually reach an equilibrium corresponding to a stable state. The equilibrium is a static equilibrium. In contrast, there will never be a static stable state for a stochastic network, as the state of the network is always changing due to stochastic update for each unit. However, one could talk of a dynamic equilibrium for stochastic networks, if the ensemble average state of the network does not change with time [Papoulis, 1991]. By ensemble average we mean that for several (infinitely large) runs of the network the average value of the state ($\langle \mathbf{s} \rangle$) of the network is computed. The average value of the state is described in terms of the average value ($\langle s_i \rangle$) of the output of each unit (i) of the network. That is

$$\langle s_i \rangle = \int s_i p(s_1, s_2, \dots, s_i, \dots, s_N) ds_1 ds_2 \dots ds_i \dots ds_N \quad (5.48)$$

where $p(\mathbf{s}) = p(s_1, s_2, \dots, s_i, \dots, s_N)$ is the joint probability density function of the components of the state vector \mathbf{s} .

First of all, the probability distribution of states should be stationary or independent of time for a network state to be in a stochastic equilibrium. If stationarity of the probability distribution of states is achieved at a given temperature, then the network is said to be in thermal equilibrium [Muller and Reinhardt, 1990, p. 1471. At thermal equilibrium the average value of the output of the i th unit is given by

$$\langle s_i \rangle = \int s_i p(s_i) ds_i \quad (5.49)$$

where $p(s_i)$ the probability density function for the i th unit. For binary units

$$\begin{aligned} \langle s_i \rangle &= 1 \times P(s_i = 1 | x_i) + 0 \times P(s_i = 0 | x_i) \\ &= P(s_i = 1 | x_i) \\ &= \frac{1}{1 + \exp(-x_i/T)} \end{aligned} \quad (5.50)$$

Thus for a given temperature, the average value of the output unit is a **continuous** function of the activation value of the unit. **Figure 5.9**

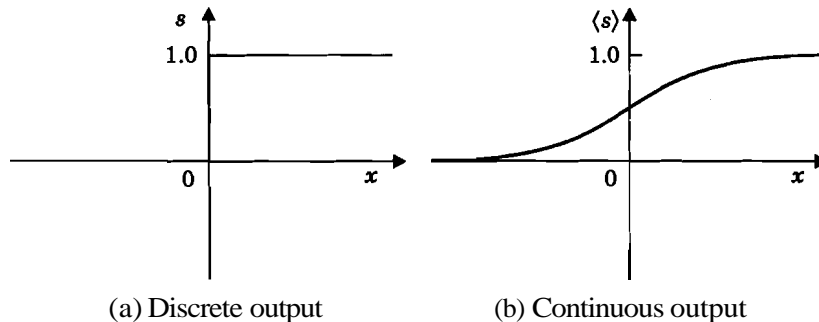


Figure 5.9 Instantaneous (discrete) and average (continuous) outputs of a unit in a stochastic network.

shows the discrete output and the average continuous output of a unit.

At stochastic equilibrium the average value of the output state of a unit does not change due to stationarity of probability distribution of the state of the network. For the average value to remain constant, the flow of activity of a **unit** i between the active ($s_i = 1$) to the inactive ($s_i = 0$) state should be balanced by the corresponding flow of activity from the inactive ($s_i = 0$) to the active ($s_i = 1$) state. This will ensure that the average value of the state due to state transitions over a period of time remains constant. In other words, for a binary

unit, the probability of a unit that is currently active ($s_i = 1$) to become inactive ($s_i = 0$) **must** be equal to the probability of the unit when it is inactive ($s_i = 0$) to become active ($s_i = 1$). That is

$$\begin{aligned} & P(s_1, s_2, \dots, s_i = 1, \dots, s_N) P(s_i \rightarrow 0 | s_i = 1) \\ &= P(s_1, s_2, \dots, s_i = 0, \dots, s_N) P(s_i \rightarrow 1 | s_i = 0) \end{aligned} \quad (5.51)$$

Since the stochastic update rule for each unit is assumed to be independent of the state of the unit, we have from Eq. (5.47)

$$P(s_i \rightarrow 1 | s_i = 0) = P(s_i = 1 | x_i) = \frac{1}{1 + \exp(-x_i/T)} \quad (5.52)$$

and

$$\begin{aligned} P(s_i \rightarrow 0 | s_i = 1) &= P(s_i = 0 | x_i) = 1 - \frac{1}{1 + \exp(-x_i/T)} \\ &= \frac{1}{1 + \exp(x_i/T)} \end{aligned} \quad (5.53)$$

Therefore from Eqs. (5.51), (5.52) and (5.53), we get

$$\begin{aligned} \frac{P(s_1, s_2, \dots, s_i = 1, \dots, s_N)}{P(s_1, s_2, \dots, s_i = 0, \dots, s_N)} &= \frac{P(s_i = 1 | x_i)}{P(s_i = 0 | x_i)} \\ &= \frac{1 + \exp(x_i/T)}{1 + \exp(-x_i/T)} \\ &= \exp(x_i/T) \end{aligned} \quad (5.54)$$

From the **Hopfield** analysis we have the global energy for the state \mathbf{s} as

$$E(\mathbf{s}) = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j \quad (5.55)$$

Difference in the energy for change of state in the k th unit from $s_k = 0$ to $s_k = 1$ is given by

$$\Delta E_k = E(s_k = 1) - E(s_k = 0) = -\sum_j w_{kj} s_j = -x_k \quad (5.56)$$

Note that this is true only if the weights are symmetric, i.e., $w_{ij} = w_{ji}$. The ratio of probabilities of the states of the network before and after an update in the i th unit is then given by (see Eq. (5.54))

$$\frac{P(s_1, s_2, \dots, s_i = 1, \dots, s_N)}{P(s_1, s_2, \dots, s_i = 0, \dots, s_N)} = \exp(-\Delta E_i/T) \quad (5.57)$$

Let $\Delta E_i = E_\beta - E_\alpha$ where E_β is the energy for the state $\mathbf{s}_\beta = (s_1, s_2, \dots, s_i = 1, \dots, s_N)$, and E_α is the energy for the state $\mathbf{s}_\alpha = (s_1, s_2, \dots, s_i = 0, \dots, s_N)$. Therefore we get

$$\frac{P(\mathbf{s}_\beta)}{P(\mathbf{s}_\alpha)} = \frac{e^{E_\alpha/T}}{e^{E_\beta/T}} \quad (5.58)$$

From Eq. (5.58) we find that the probability of a state at thermal equilibrium is inversely proportional to the exponential of the energy of the state. That is

$$P(\mathbf{s}_\alpha) \propto e^{-E_\alpha/T} \quad (5.59)$$

where E_α is the energy of the network in any state \mathbf{s}_α , and is given by the Hopfield energy equation (5.55).

Since $\sum_{\alpha} P(\mathbf{s}_\alpha) = 1$, we get

$$P(\mathbf{s}_\alpha) = \frac{1}{Z} e^{-E_\alpha/T}, \quad (5.60)$$

where $1/Z$ is the proportionality constant. Therefore,

$$\sum_{\alpha} P(\mathbf{s}_\alpha) = \frac{1}{Z} \sum_{\alpha} e^{-E_\alpha/T} = 1 \quad (5.61)$$

$$Z = \sum_{\alpha} e^{-E_\alpha/T} \quad (5.62)$$

Therefore the probability of a state at thermal equilibrium is given by

$$P(\mathbf{s}_\alpha) = \frac{e^{-E_\alpha/T}}{\sum_{\beta} e^{-E_\beta/T}} \quad (5.63)$$

This is called the Boltzmann-Gibb's probability distribution. The normalization factor Z is called the partition function in statistical mechanics [Hertz et al, 1991; Muller and Reinhardt, 1990]. The partition function plays a central role in statistical mechanics, as it can be used to compute averages at thermal equilibrium. At high temperature ($T \rightarrow \infty$), the stationary probabilities of the states are nearly equal and are independent of the energies of the states. On the other hand, at low temperatures ($T \rightarrow 0$), the stationary probabilities are dictated by the energies of the states, and the states with lower energy will have higher probabilities.

5.4.3 Thermal Averages

From the stationary probabilities $P(\mathbf{s}_\alpha)$ of the states at thermal equilibrium at a given temperature, the average (A) of any quantity A pertaining to the states can be computed as follows:

$$\langle A \rangle = \sum_{\alpha} A_{\alpha} P(\mathbf{s}_\alpha) \quad (5.64)$$

where A_{α} is the value of the quantity for the state \mathbf{s}_α . This is called thermal average of the quantity A.

In many cases the thermal averages can be computed from the partition function itself, instead of using the stationary probabilities. But, in practice, obtaining the partition function becomes the main issue. Assuming that the partition function Z is known, the averages of some relevant quantities are computed as follows:

In order to compute the averages, it is convenient to define a term, called **free** energy of the system, given by [Hertz et al, 1991, Appendix].

$$F = -T \log Z = -T \log \left(\sum_{\alpha} e^{-E_{\alpha}/T} \right) \quad (5.65)$$

where \log stands for the natural logarithm. Then

$$-\frac{\partial F}{\partial T} = \frac{1}{T} \sum_{\alpha} E_{\alpha} \frac{e^{-E_{\alpha}/T}}{Z} = \frac{1}{T} \sum_{\alpha} E_{\alpha} P(\mathbf{s}_{\alpha}) \quad (5.66)$$

The average energy (E) of the states is given by

$$\langle E \rangle = \sum_{\alpha} P(\mathbf{s}_{\alpha}) E_{\alpha} = -T \frac{\partial F}{\partial T} = \frac{T^2}{Z} \frac{\partial Z}{\partial T} \quad (5.67)$$

To compute the averages $\langle s_i \rangle$ and $\langle s_i s_j \rangle$, let us consider the expression for the **Hopfield** energy with the bias term as in Eq. (5.25) [Muller and Reinhardt, 1990, p. 149].

$$E(\mathbf{s}) = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i \theta_i s_i \quad (5.68)$$

Taking the derivative with respect to θ_i , we get

$$\frac{\partial E(\mathbf{s})}{\partial \theta_i} = s_i \quad (5.69)$$

We can show using Eqs. (5.65) and (5.69) that

$$\langle s_i \rangle = \frac{\partial F}{\partial \theta_i} = \frac{T}{Z} \frac{\partial Z}{\partial \theta_i} \quad (5.70)$$

If we take $-\partial F / \partial w_{ij}$, then we get

$$\begin{aligned} -\frac{\partial F}{\partial w_{ij}} &= \frac{T}{Z} \frac{\partial Z}{\partial w_{ij}} \\ &= \frac{T}{Z} \left[\sum_{\alpha} \frac{\partial e^{-E_{\alpha}/T}}{\partial w_{ij}} \right] \\ &= \frac{T}{Z} \left[-\frac{1}{T} \sum_{\alpha} e^{-E_{\alpha}/T} (-s_i s_j) \right] \\ &= \sum_{\alpha} \frac{e^{-E_{\alpha}/T}}{Z} s_i s_j = \langle s_i s_j \rangle \end{aligned} \quad (5.71)$$

Thus.

$$\langle s_i s_j \rangle = -\frac{\partial F}{\partial w_{ij}} = \frac{T}{Z} \frac{\partial Z}{\partial w_{ij}} \quad (5.72)$$

Equations (5.67), (5.70) and (5.72) are three thermal average expressions in terms of the free energy F and partition function Z . The free energy F can be interpreted as follows:

Since $e^{-F/T} = Z = \sum_{\mathbf{a}} e^{-E_{\mathbf{a}}/T}$, we have

$$\frac{e^{-FIT}}{Z} = \sum_{\mathbf{a}} P(\mathbf{s}_{\mathbf{a}}) \quad (5.73)$$

The sum of probabilities over all possible states is 1, and hence the left hand side of Eq. (5.73) above is equal to 1. But the above equation applies even if the summation is taken over a subset of the states. Then the left hand side gives the probability of finding the system in that subset.

5.4.4 Stability in Stochastic Networks

In stochastic networks, equilibrium refers to the thermal equilibrium at which the averages over all possible realizations of the states are independent of time. This is because the probability distribution of the states does not change with time. It can be proved that networks with symmetric weights do indeed reach **thermal** equilibrium at a given temperature. Since the state of the network changes due to stochastic update, it is not possible to talk about absolutely stable states in the sense that, once such a state is reached, it should remain there. On the other hand, one can still study stability of states at thermal equilibrium in which the average values do not change in time. Like in the deterministic case, for a stable state in the stochastic case we invoke the condition that the average value of the **output** is proportional to one of the stored patterns, say the k th one (\mathbf{a}_k). That is, for each component of the vector,

$$\langle s_i \rangle = m a_{ki}, \quad i = 1, 2, \dots, N \quad (5.74)$$

where m is the proportionally constant. In the deterministic ($T = 0$) case, these stable states exist for $m = 1$ as seen in Section 5.3.2. In the stochastic case we have from Eq. (5.50)

$$\langle s_i \rangle = \frac{1}{1 + \exp(\Delta E_i/T)}, \quad (5.75)$$

where $\Delta E_i = -x_i = -\sum_j w_{ij} \langle s_j \rangle$. Here the activation value is determined using the average of the fluctuations of the outputs from the other units. This is called the *mean-field approximation* by which

the actual variables are replaced by their averaged quantities [Soukoulis et al, 1983; Bilbro et al, 1992].

We can obtain an approximate solution of Eq. (5.74) by substituting for $\langle s_i \rangle$ in Eq. (5.74) and simplifying. In doing this analysis it is convenient to assume units with bipolar $\{-1, +1\}$ outputs. Then the average value $\langle s_i \rangle$ is given by

$$\langle s_i \rangle = 1 \times P(s_i = 1 | x_i) - 1 \times P(s_i = -1 | x_i) \quad (5.76)$$

Assuming for convenience the probability of update in this case as

$$P(s_i = 1 | x_i) = \frac{1}{1 + \exp(-2x_i/T)} \quad (5.77)$$

we get

$$\begin{aligned} \langle s_i \rangle &= \frac{1}{1 + \exp(-2x_i/T)} - \left[1 - \frac{1}{1 + \exp(-2x_i/T)} \right] \\ &= \tanh\left(\frac{x_i}{T}\right) \end{aligned} \quad (5.78)$$

where x_i is given by

$$x_i = \sum_j w_{ij} \langle s_j \rangle$$

using the mean-field approximation. From the Hebb's law given in Eq. (5.18), we have

$$w_{ij} = \frac{1}{N} \sum_{l=1}^L a_{li} a_{lj} \quad (5.79)$$

Therefore from Eqs. (5.74), (5.78) and (5.79), we have

$$m a_{ki} = \tanh\left(\frac{1}{TN} \sum_j \sum_l a_{li} a_{lj} m a_{kj}\right) \quad (5.80)$$

Ignoring the cross terms, which is valid if the number of patterns stored (L) is much less than N , we get

$$m a_{ki} = \tanh\left(\frac{m}{T} a_{ki}\right) \quad (5.81)$$

Since $a_{ki} = \pm 1$, and $\tanh(-x) = -\tanh(x)$, we get

$$m = \tanh\left(\frac{m}{T}\right) \quad (5.82)$$

Solutions of this equation are shown in Figure 5.10. It shows that the solutions for m are the **points** of intersection of the straight line $y = Tx$ and the sigmoid curve $y = \tanh(x)$. If $T \geq 1$, there is only one solution at $y = 0$. For $T < 1$, there are three solutions.

From Figure 5.10 we can obtain the values of y for different

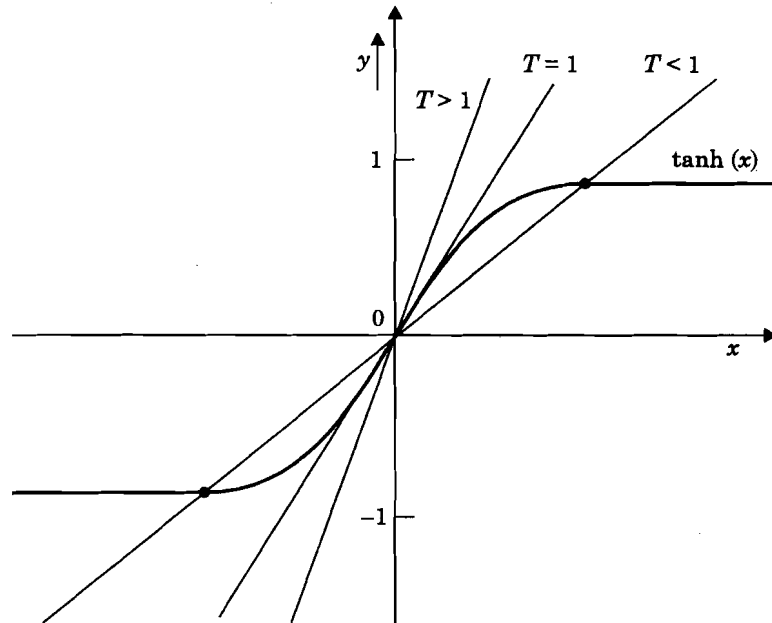


Figure 5.10 Solution of the equation $y = \tanh(m/T)$ as points of intersection of $y = Tx$ and $y = \tanh(x)$.

temperatures. Figure 5.11 shows the positive value of (s) of Eq. (5.78) as a function of the temperature T . It shows that nonzero solutions to (s) exist only when $T < T_c$. As $T \rightarrow 0$, (s) approaches ± 1 . The critical temperature $T_c = 1$ for stochastic networks with $L \ll N$, since the crossterms in Eq. (5.80) are negligible under this condition.

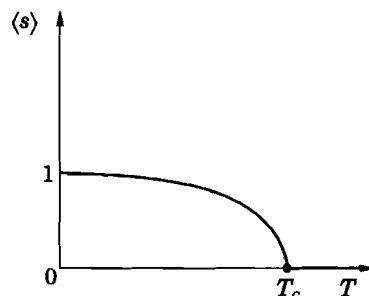


Figure 5.11 The positive solution (s) as a function of temperature.

The above analysis shows that a stochastic network with symmetric weights will have stable equilibrium states, i.e., will satisfy Eq. (5.81), only at temperatures below a critical temperature, provided $L \ll N$. The number of such states is very small compared to N , and the actual number depends on the temperature. But the maximum value of L will be less than $0.138 N$, which is the limit for the deterministic case, i.e., for $T = 0$ [Amit et al, 1987].

The maximum number of stable states for a stochastic network is referred to as the capacity of the network. The capacity will be different for different temperatures. Above a critical temperature for any given L/N (See Figure 5.12), the network will not have any stable

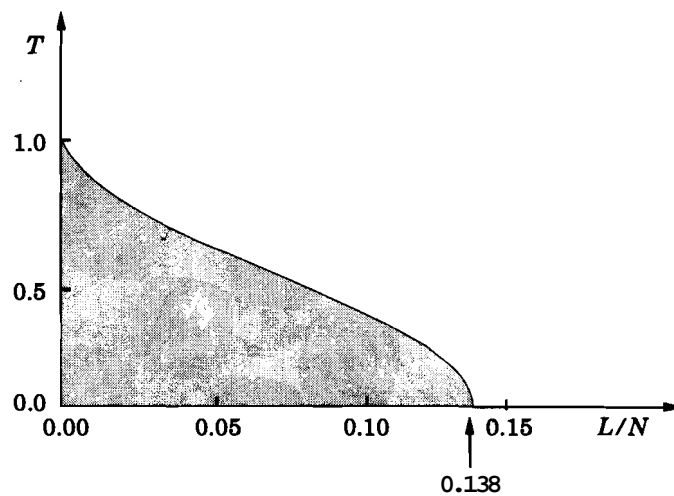


Figure 5.12 The region (shaded) for the existence of stable states for a stochastic network.

equilibrium state and hence cannot be used as memory. The critical temperature is lower for higher values of L/N . For $L/N > 0.138$, however, there are no stable states for any temperature, including $T = 0$ [Amit et al, 1987; Hertz et al, 1991, p. 391.

5.4.5 Operation of a Stochastic Network

Having seen that a stochastic network exhibits stable states for temperatures below a critical temperature, we shall discuss the operation of a stochastic network for memorizing a given set of patterns. Throughout we assume that $L \ll N$, and that we are operating at temperatures lower than the critical temperature, so that the network has stable states at thermal equilibrium at a given temperature.

Given a feedback network with symmetric connections, there exists an energy landscape with a unique value of energy for each state of the network. There are two aspects of the network when used as memory: Designing a network to store a given set of patterns (training) and recalling a pattern stored in the network (recall). To understand these aspects, let us **first** discuss the operation of a stochastic network in detail. The operation involves monitoring the trajectory of the states of the feedback network and studying the characteristics of the resulting random process in terms of probability distributions of states and the relation of these distributions with the energy.

Note that the energy landscape is fixed for a given network, as the energy depends only on the output state s_i of each unit i and on the weights w_{ij} on the **connecting** link between units j and i . The activation dynamics including the asynchronous or synchronous operation and the stochastic update decide the **trajectory** of the states and hence the traversal along the energy landscape. Since provision for stochastic update is available, the trajectory may move along a path which may include movement to states with higher energies in the energy landscape.

Figure 5.13 illustrates the regions for trajectories of states at different temperatures. Note that at $T = 0$, the trajectory can only

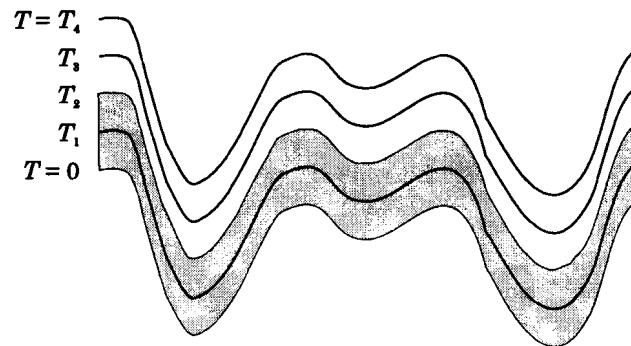


Figure 5.13 Regions of traversal in the energy landscape. The shaded area indicates region of traversal for a temperature of T_2 .

slide along the downward slope of the energy and reach a fixed point equilibrium state. Thus at $T = 0$, there are as many stable states as there are energy minima in the landscape. At higher temperatures there is greater mobility, thus resulting sometimes in a movement towards higher energy states. In such a case all the energy minima regions covered in the region of movement cease to be stable regions. Only regions with deep energy minima are likely to be stable. Thus the number of stable regions decreases with increase in temperature. At a given temperature several trajectories are possible depending on the update at each unit. These trajectories may be viewed as sample functions of a random process. When the temperature is changed, the trajectories correspond to the transient phenomenon during which the random process is nonstationary. Consequently the probability distribution of states changes with time as shown in **Figure 5.14** for three time instants (t_0, t_1, t_2), where t_0 is the instant at which the temperature parameter is changed, t_1 is an instant in the transient region and t_2 is an instant in the steady region after the random process became stationary. Note that the probability distributions may not be related to the energy landscape during the transient phenomenon. But once the thermal equilibrium is reached,

Stochastic Networks and Simulated Annealing

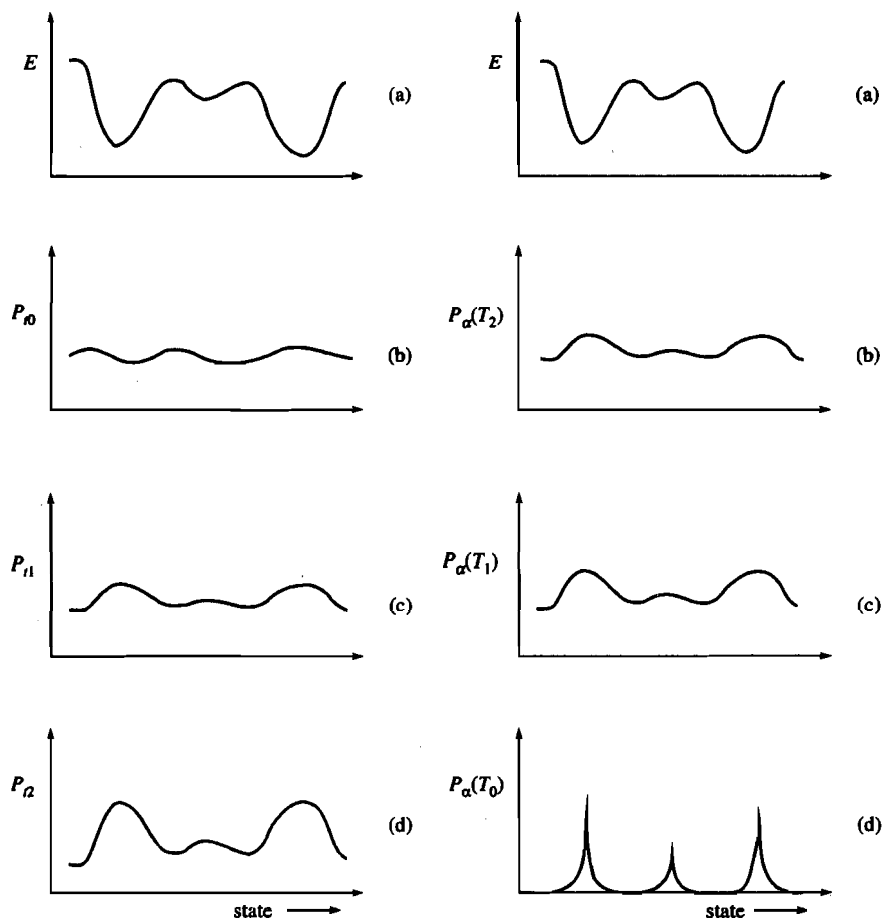


Figure 5.14 Probability distribution of states at a given new temperature during transient phenomenon at three instants of time ($t_2 > t_1 > t_0$) (a) Energy landscape. (b), (c) and (d) are probability distributions at times t_0 , t_1 and t_2 , respectively.

Figure 5.15 Stationary probability distribution at three different temperatures for a stochastic network ($T_0 < T_1 < T_2$). (a) Energy landscape. (b), (c) and (d) are stationary probability distributions at temperatures T_2 , T_1 and T_0 , respectively, with $T_0 \approx 0$.

the stationary probability distribution shows some relation to the energy landscape. In particular, the peaks in the probability distribution correspond typically to deeper valleys of the energy landscape.

Figure 5.15 shows the stationary probability distributions at

different temperatures. At higher temperatures the probability distribution is more flat, indicating that several higher energy states are also likely with high probability. At $T = 0$, the probability distribution shows impulses at the states which are fixed stable points and at which the energy is minimum. Probabilities of nonminimum energy states are zero. Moreover, the probability of a state is inversely proportional to the energy at that state. Hence the average energy at $T = 0$ is minimum. The average energy at thermal equilibrium at $T \neq 0$ is higher for higher temperatures. This is because the energy landscape is fixed, but the probability distribution of states is flatter at higher temperatures than at lower temperatures.

5.4.6 Simulated Annealing

From the above analysis we can see that the matching probability distribution of states for a given network gives the lowest average energy at $T = 0$. When the network is used as a memory to store a given set of patterns, the average error in the recall will be minimum if the probability distribution of the given patterns matches the optimal probability distribution of the stable states of the network at $T = 0$. This can happen only by determining an optimal size of the network and the connection weights. Determination of a suitable architecture and adjustment of the weights are discussed in Section 5.5. But the adjustment of weights or learning involves determining the probability distribution at $T = 0$ for each presentation of a pattern input, by going through a sequence of temperature values starting from a high temperature to finally $T = 0$. Thus for each application of a training pattern the network is said to be annealed to obtain the probability distribution matching the energy landscape. To reinforce the given pattern, the weights are adjusted in accordance with the statistics of the outputs of the units collected at the end of the annealing process. The **objective** is to ultimately shape the energy landscape for the given distribution of the input patterns so as to obtain a low probability of error in the recall. But if the number of patterns to be stored is smaller than the number of energy minima, then during recall the network may settle in a state corresponding to an energy minimum not used for storage. This is called the local minima problem.

The existence of local minima may result in an error in the recall, even though the training attempts to match the given distribution of patterns with the energy landscape of the network. For recall, when an approximate input is given, the network is allowed to reach an equilibrium state near $T = 0$, following an annealing process starting from a high temperature. This will reduce the effects of local minima and thus **reduces** the probability of error in the recall. Using the given approximate input as a constraint, we want to arrive at a state

that corresponds to a minimum energy of the given network. The information corresponding to the state gives the desired pattern to be recalled.

Both training and recall of patterns in a stochastic network use the process of annealing controlled by the temperature parameter. The rate of change of temperature, called *annealing schedule*, becomes critical in realizing the desired probability distribution of states near $T = 0$. This process is called *simulated annealing* [Kirkpatrick et al, 1983].

5.4.7 Example of Simulated Annealing

In this section we consider an example to illustrate the ideas of stochastic update, thermal equilibrium and simulated annealing. The example is adapted from [Aleksander and Morton, 1990]. For this we take the three unit binary network with symmetric weights shown in Figure 5.7. For each unit, the probability of firing for an activation \mathbf{x} is assumed to be

$$P(s = 1 | \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{x} - \theta)/T}} \quad (5.83)$$

For each temperature T a separate state transition diagram can be derived, which indicates the transition from each state to other states. Let us illustrate the calculation of the transition probabilities for one state, say $s_1 s_2 s_3 = 011$. Assuming only one unit is allowed to change at a time, for each unit we can compute the activation value $(\mathbf{x} - \theta)$ using $\sum_j w_{ij} s_j - \theta_i$. For a temperature $T = 0.25$, the values of $P(\mathbf{1} | \mathbf{x})$ for units 1, 2 and 3 are 0.6, 0.92 and 0.23, respectively. The corresponding probabilities for not firing $P(\mathbf{0} | \mathbf{x}) = 1 - P(\mathbf{1} | \mathbf{x})$ are 0.4, 0.08 and 0.77, respectively for the units 1, 2 and 3. A change of state from 011 to 111 occurs if the first unit fires. Since each unit can be chosen with a probability of $1/3$, and the probability of unit 1 firing is 0.6, the transition probability from 011 to 111 is 0.613. Likewise the transition probability from 011 to 001 is 0.0813 and from 011 to 010 is 0.7713. Since only these transitions are possible from 011, besides self-transition, the probability of self-transition is given by $1 - 0.613 - 0.0813 - 0.7713$. The transition probabilities for the state 011 for three different temperatures ($T = 0.0, 0.25$ and 0.5) are shown in Figure 5.16. It can be seen that in the deterministic case ($T = 0$) the transitions to the state 001 are not possible because it is at a higher energy level. For nonzero temperature there is a nonzero probability of transition to all possible states including to a state with higher energy value. These transition probabilities get distributed more evenly at higher temperatures.

The complete state transition diagram for $T = 0.25$ shows a transition with nonzero probability from every state to its neighbouring (Hamming distance = 1) states. The transition probabilities for all

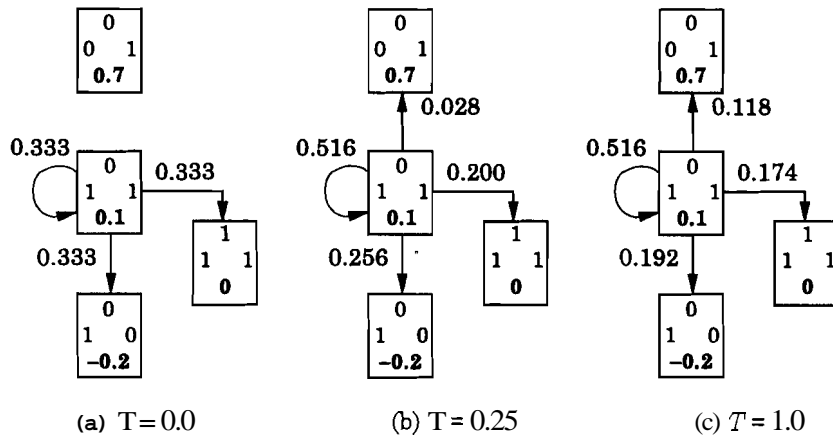


Figure 5.16 State transition probabilities for the state 011 at different temperatures. (a) $T = 0$, (b) $T = 0.25$ and (c) $T = 1.0$.

the eight states for $T = 0.25$ case are shown in the form of a matrix in Table 5.5. Note there are four nonzero entries in each row, indicating the four possible transitions for each state, and these entries in each row add up to 1.

Table 5.5 State Transition Probabilities $P(i|j)$ at $T = 0.25$ for the 3-unit Network in Figure 5.7(a)

j	i	000	001	010	011	100	101	110	111
000		0.617	0.019	0.230	0.000	0.134	0.000	0.000	0.000
001		0.314	0.074	0.000	0.306	0.000	0.306	0.000	0.000
010		0.103	0.000	0.792	0.077	0.000	0.000	0.028	0.000
011		0.000	0.028	0.256	0.516	0.000	0.000	0.000	0.200
100		0.200	0.000	0.000	0.000	0.557	0.166	0.077	0.000
101		0.000	0.028	0.000	0.000	0.166	0.606	0.000	0.200
110		0.000	0.000	0.306	0.000	0.256	0.000	0.161	0.277
111		0.000	0.000	0.000	0.134	0.000	0.134	0.056	0.676

For a given state transition probability matrix, it is possible to determine the probability distribution of the states $\{P_i(t)\}$ at each instant of time t , starting with some assumed distribution at time $t=0$. Let us assume equal probability for each state as the initial state probability distribution. That is $P_i(0) = 1/8 = 0.125$, as there are eight states.

Let $P(i|j)$ be the probability of transition from the state j to the state i . Since each state j occurs with probability $P_j(t)$, the probability of reaching state i in the next instant from the state j at the current

instant is $P_j(t)P(i|j)$. Summing this over all states will give the probability of the state i at the next instant and is given by

$$P_i(t+1) = \sum_j P_j(t)P(i|j), \quad \text{for } i = 0, 1, \dots, 7 \quad (5.84)$$

In matrix notation the probability distribution of states at time $t+1$ is given by

$$\mathbf{p}(t+1) = \mathbf{P}\mathbf{p}(t),$$

where $\mathbf{p}(t) = [P_i(t)]^T$ is a column matrix with the state probabilities as elements, and $\mathbf{P} = [P(i|j)]^T$, is the transition probability matrix with $P(i|j)$ as elements. Therefore,

$$\mathbf{p}(t) = \mathbf{P}^t \mathbf{p}(0), \quad (5.85)$$

where \mathbf{P}^t denotes matrix multiplication t times. From Eq. (5.85) the steady probability distribution can be obtained when $t \rightarrow \infty$. Note that the time to reach the steady state depends on the initial state probabilities and the state transition probability matrix.

The state transition probability matrix \mathbf{P} depends on the temperature T . As the temperature is varied, these transition probabilities also will change. Therefore the steady state probability distribution depends on the temperature, and hence the corresponding steady state is called thermal equilibrium. When the temperature is changed say from T_2 to T_1 , then the network moves from one thermal equilibrium condition to another thermal equilibrium condition after going through a transient phase during which the state probability distribution will be changing. This is illustrated in Table 5.6 for the three unit network example in Fig. 5.7(a). In the table, the notation $P_i(t)$ indicates the probability of the state i (the integer value of the corresponding binary state) at time t . Starting with equal initial probabilities for all the states at $T = 1$, the probabilities during transient phase are calculated until thermal equilibrium is reached. Thermal equilibrium is indicated when there is no change in the state probabilities for subsequent updating instants. At this stage the temperature is changed to $T = 0.25$, and the state probabilities are again calculated until thermal equilibrium is reached again. **Finally** the temperature is set to zero, i.e., $T = 0$, and the state probabilities are updated until thermal equilibrium is reached. At this stage, we notice that there are only two states with nonzero probabilities, and these probabilities are inversely related to their state energies. These states correspond to the stable states of the network.

In general the rate of change of the temperature parameter is critical to arrive at the final stable states **after** passing through several stages of thermal equilibrium. This rate of change of temperature is called annealing schedule.

Table 5.6 Illustration of State Probabilities during Simulated Annealing for the 3-unit Network in Figure 5.7(a). (Adapted from [Aleksander and Morton, 1990])

State	Probability	$P_n(t)$	$P_1(t)$	$P_2(t)$	$P_3(t)$	$P_4(t)$	$P_5(t)$	$P_6(t)$	$P_7(t)$
Temp.	Time t								
1.0	0	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
1.0	1	0.137	0.087	0.147	0.129	0.129	0.135	0.098	0.137
1.0	2	0.139	0.076	0.158	0.128	0.130	0.135	0.093	0.141
1.0	3	0.140	0.073	0.163	0.127	0.129	0.133	0.093	0.142
1.0	4	0.140	0.071	0.166	0.127	0.129	0.132	0.093	0.142
1.0	5	0.140	0.071	0.168	0.127	0.128	0.131	0.094	0.142
1.0	6	0.140	0.070	0.169	0.127	0.128	0.130	0.094	0.142
1.0	7	0.141	0.070	0.172	0.127	0.128	0.128	0.094	0.141
1.0	8	0.141	0.070	0.172	0.127	0.127	0.128	0.094	0.141
1.0	9	0.141	0.070	0.172	0.127	0.127	0.128	0.094	0.141
0.25	10	0.152	0.015	0.230	0.119	0.135	0.139	0.038	0.172
0.25	11	0.149	0.011	0.259	0.107	0.128	0.134	0.033	0.178
0.25	12	0.148	0.010	0.277	0.103	0.122	0.130	0.032	0.178
0.25	13	0.148	0.010	0.289	0.101	0.118	0.126	0.032	0.176
0.25	14	0.148	0.010	0.299	0.101	0.115	0.123	0.032	0.173
0.25	15	0.148	0.010	0.306	0.102	0.112	0.120	0.032	0.171
0.25	16	0.149	0.010	0.321	0.102	0.108	0.113	0.032	0.165
0.25	17	0.150	0.010	0.325	0.103	0.107	0.111	0.032	0.163
0.25	18	0.150	0.010	0.328	0.103	0.106	0.110	0.031	0.162
0.25	19	0.151	0.009	0.330	0.103	0.106	0.109	0.031	0.161
0.25	20	0.151	0.009	0.332	0.103	0.105	0.108	0.031	0.160
0.25	21	0.152	0.009	0.334	0.103	0.105	0.107	0.031	0.159
0.25	23	0.153	0.009	0.338	0.103	0.104	0.105	0.031	0.156
0.25	24	0.153	0.009	0.340	0.103	0.104	0.104	0.031	0.155
0.25	25	0.153	0.009	0.340	0.103	0.104	0.104	0.031	0.155
0.25	26	0.153	0.009	0.341	0.103	0.103	0.104	0.031	0.155
0.25	27	0.153	0.009	0.341	0.103	0.103	0.104	0.031	0.155
0.25	28	0.153	0.009	0.341	0.103	0.103	0.104	0.031	0.155
0.25	29	0.154	0.009	0.341	0.103	0.103	0.104	0.031	0.155
0.25	30	0.154	0.009	0.342	0.103	0.103	0.103	0.031	0.155
0.00	31	0.140	0.000	0.438	0.037	0.045	0.106	0.000	0.233
0.00	32	0.108	0.000	0.498	0.012	0.015	0.086	0.000	0.281
0.00	33	0.077	0.000	0.538	0.004	0.005	0.062	0.000	0.314
0.00	35	0.036	0.000	0.583	0.000	0.001	0.029	0.000	0.351
0.00	36	0.024	0.000	0.595	0.000	0.000	0.020	0.000	0.360
0.00	37	0.016	0.000	0.603	0.000	0.000	0.013	0.000	0.367
0.00	38	0.011	0.000	0.609	0.000	0.000	0.009	0.000	0.372
0.00	39	0.007	0.000	0.612	0.000	0.000	0.006	0.000	0.374
0.00	40	0.005	0.000	0.615	0.000	0.000	0.004	0.000	0.376
0.00	41	0.003	0.000	0.616	0.000	0.000	0.003	0.000	0.378
0.00	42	0.002	0.000	0.618	0.000	0.000	0.002	0.000	0.379
0.00	43	0.001	0.000	0.619	0.000	0.000	0.001	0.000	0.380
0.00	44	0.001	0.000	0.619	0.000	0.000	0.001	0.000	0.380
0.00	45	0.000	0.000	0.620	0.000	0.000	0.000	0.000	0.380

Boltzmann Machine

5.5 Boltzmann Machine

5.5.1 Problem of Pattern Environment Storage

The relation between probabilities and energies of the stable states suggests that the probability of error in the recall of stored patterns in a feedback neural network can be reduced if the weights are chosen appropriately. If the probability distribution of the given (desired) patterns, called *pattern environment*, is known, then this knowledge can be used for determining the weights of the network while storing the patterns. The training procedure should try to capture the pattern environment in the network in an optimal way. Boltzmann learning law to be discussed in Section 5.5.3 gives a procedure to accomplish this pattern environment storage in an optimal way for a given feedback network. But first we shall discuss considerations in the choice of a suitable feedback network for the pattern environment storage problem.

5.5.2 Architecture of a Boltzmann Machine

Given a set of L patterns, each pattern described by a point in an N -dimensional space, it is not clear how many processing units would be needed for a feedback network. It may not be possible to store them in a network consisting of N units, if the resulting number of stable states (for a given set of weights) is less than the number of patterns L . That is, the capacity of the network is less than L . Such problems are called hard problems. In general it is difficult to say whether a given pattern storage problem is a hard problem or not for a given network. To be on the safe side, one can add extra units to the feedback network. These extra units are called hidden units, whereas the remaining N units, to which the input patterns are applied during training, are called visible units. A fully **connected** network consisting of both hidden and visible units (Figure 5.17) and operating asynchronously with stochastic update for each unit is called a *Boltzmann machine*. Since the steady state probabilities at thermal equilibrium follow the Boltzmann-Gibb's distribution, the network architecture is called a Boltzmann machine [Ackley et al, 1985].

Since the network architecture is so chosen that the number of stable states is more than the desired number of patterns, the additional stable states become spurious stable states. Existence of the spurious stable states results in a nonzero probability of **error** in the recall, even though the network is trained to capture the pattern environment in an optimal way. Pattern recall from a Boltzmann machine uses simulated annealing to reduce the **effects** of these additional stable states, which correspond to local minima in the energy landscape of the network.

Feedback Neural Networks

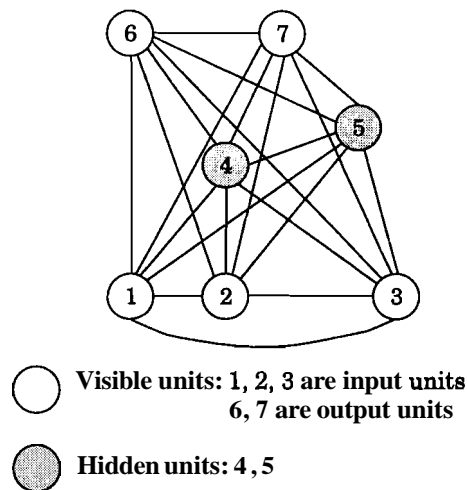


Figure 5.17 Illustration of a Boltzmann machine.

A Boltzmann machine can also be used for a pattern association task. This is accomplished by **identifying** a subset of the visible units with the inputs and the remaining visible units with the outputs of the given pattern pairs. In other words, each input-output pair is considered as a pattern, and these patterns are stored as in the pattern environment storage problem. For recall, the input is presented only to the input subset of the visible units. The output is read out **from** the output subset of the visible units, **after** the network reached thermal equilibrium at $T \approx 0$ using a simulated annealing schedule to reduce the local minima **effects**.

In general a Boltzmann machine architecture can be used for any pattern completion task, in which the stored (input-output) pattern can be recalled by providing a partial information about the pattern.

5.5.3 Boltzmann Learning Law

The Boltzmann learning law gives a procedure to represent a given pattern environment by a Boltzmann machine. The law uses an information theoretic measure to evaluate how well the environment is represented by the network. If a perfect representation is obtained, then there will be as many energy minima as there are desired patterns in the environment. Moreover, these energy minima are inversely related to the probabilities of occurrence of the corresponding patterns in the environment. Normally since only an approximate representation of the environment is accomplished after training, there will be a residual mismatch of the probabilities of the patterns in the environment with the probabilities of the stable states of the resulting network. This mismatch, together with the inevitable existence of spurious stable states, results in some nonzero probability

of error in recalling the stored patterns. In this section we will derive the Boltzmann learning law based on an information theoretic measure [Gray, 1990]. We will discuss the implementation issues in the next section.

Let $P^+(V_a)$, $a = 1, 2, \dots, L$ be the probability distribution of the set of given L patterns $\{V_a\}$, where V_a is a point in the N -dimensional space. The superscript '+' denotes the desired states and their probabilities for the visible units of the network. Note that $\sum P^+(V_a) = 1$. During recall it is desirable to have the network settle at one of the **training** patterns only. Let $P^-(V_a)$ be the actual probability distribution of the desired patterns at the visible units at equilibrium for a given network. The objective of training is to adjust the weights of the network so that the **difference** in the network behaviour for these two probability distributions is negligible. Ideally one would like to have $P^+(V_a) = P^-(V_a)$, $a = 1, 2, \dots, L$.

The distribution $P^+(V_a)$ corresponds to the desired situation of the states at the visible units and the distribution $P^-(V_a)$ corresponds to the probability of occurrence of these states when the network is running freely. The difference between these conditions is represented by an error criterion derived based on information theoretic measure [Gray, 1990]. The error function is given by

$$G = \sum_a P^+(V_a) \log \frac{P^+(V_a)}{P^-(V_a)} \quad (5.86)$$

It is easy to show that $G \geq 0$, using the relation $\log x \geq 1 - (1/x)$. The error function $G = 0$, only when $P^+(V_a) = P^-(V_a)$. In other words, when $G = 0$ the given pattern environment is represented by the network exactly. Using the gradient descent along the error surface G in the weight space to adjust the weights, we have

$$\Delta w_{ij} \propto - \frac{\partial G}{\partial w_{ij}} \quad (5.87)$$

To compute the gradient $\partial G / \partial w_{ij}$, we will use the following relations for the probabilities and energies of the states:

$$P^-(V_a) = \sum_b P^-(V_a \wedge H_b), \quad (5.88)$$

where H_b is a vector of states of the hidden units, V_a is a vector of states of the visible units and $(V_a \wedge H_b)$ represents the state of the entire network. $P^-(.)$ represents the probabilities of the states when the network is free running.

Likewise

$$P^+(V_a) = \sum_b P^+(V_a \wedge H_b) \quad (5.89)$$

where H_b is a vector of states of the hidden units. $P^+(\cdot)$ represent the probabilities of the states when the network is forced or clamped with the given patterns as states of the visible units.

The energy of the network in the state $V_a \wedge H_b$ is given by

$$E_{ab} = -\frac{1}{2} \sum_{i,j} w_{ij} s_i^{ab} s_j^{ab} \quad (5.90)$$

where s_i^{ab} is the output of the unit i when the network is in the state $V_a \wedge H_b$.

Since $P^-(V_a \wedge H_b)$ represents the probability distribution of states at thermal equilibrium at some temperature T , the probability of the state $(V_a \wedge H_b)$ at equilibrium is related to the energy E_{ab} through Boltzmann-Gibbs law as follows:

$$P^-(V_a \wedge H_b) = \frac{e^{-E_{ab}/T}}{\sum_{mn} e^{-E_{mn}/T}} = \frac{e^{-E_{ab}/T}}{Z} \quad (5.91)$$

Therefore,

$$P^-(V_a) = \frac{1}{Z} \sum_b e^{-E_{ab}/T} \quad (5.92)$$

The gradient $\partial G/\partial w_{ij}$ is given by

$$\frac{\partial G}{\partial w_{ij}} = -\sum_a \frac{P^+(V_a)}{P^-(V_a)} \frac{\partial P^-(V_a)}{\partial w_{ij}} \quad (5.93)$$

since $P^+(V_a)$ is constant for a given pattern environment. We have

$$\begin{aligned} \frac{\partial P^-(V_a)}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \sum_b P^-(V_a \wedge H_b) \\ &= \frac{\partial}{\partial w_{ij}} \left[\left(\frac{1}{Z} \sum_b e^{-E_{ab}/T} \right) \right] \\ &= -\frac{1}{T} \sum_b \frac{e^{-E_{ab}/T}}{Z} \frac{\partial E_{ab}}{\partial w_{ij}} - \sum_b \frac{e^{-E_{ab}/T}}{Z^2} \frac{\partial Z}{\partial w_{ij}} \end{aligned} \quad (5.94)$$

From Eq. (5.90) we have

$$\frac{\partial E_{ab}}{\partial w_{ij}} = -s_i^{ab} s_j^{ab}, \quad (5.95)$$

Therefore we get

$$\begin{aligned} \frac{\partial Z}{\partial w_{ij}} &= \sum_{mn} \left(-\frac{1}{T} \frac{\partial E_{mn}}{\partial w_{ij}} e^{-E_{mn}/T} \right) \\ &= \frac{1}{T} \sum_{mn} s_i^{mn} s_j^{mn} e^{-E_{mn}/T} \end{aligned} \quad (5.96)$$

Therefore,

$$\begin{aligned} \frac{\partial P^-(V_a)}{\partial w_{ij}} &= \frac{1}{T} \sum_b P^-(V_a \wedge H_b) s_i^{ab} s_j^{ab} \\ &\quad - \frac{P^-(V_a)}{T} \sum_{mn} P^-(V_m \wedge H_n) s_i^{mn} s_j^{mn} \end{aligned} \quad (5.97)$$

and

$$\begin{aligned} \frac{\partial G}{\partial w_{ij}} &= -\frac{1}{T} \sum_{ab} \frac{P^+(V_a)}{P^-(V_a)} P^-(V_a \wedge H_b) s_i^{ab} s_j^{ab} \\ &\quad + \frac{\sum_a P^+(V_a)}{T} \sum_{mn} P^-(V_m \wedge H_n) s_i^{mn} s_j^{mn} \end{aligned} \quad (5.98)$$

We have the following relations:

$$\begin{aligned} \sum_a P^+(V_a) &= 1 \\ P^+(V_a \wedge H_b) &= P^+(H_b | V_a) P^+(V_a) \\ P^-(V_a \wedge H_b) &= P^-(H_b | V_a) P^-(V_a) \end{aligned} \quad (5.99)$$

Also we know that the probability that the state H_b will occur on the hidden units should not depend on whether the state V_a on the visible units got there by being forced by environment or by free running. That is

$$P^+(H_b | V_a) = P^-(H_b | V_a) \quad (5.100)$$

Hence from Eq. (5.99), we get

$$\frac{P^-(V_a \wedge H_b)}{P^+(V_a \wedge H_b)} = \frac{P^-(V_a)}{P^+(V_a)} \quad (5.101)$$

Therefore,

$$\begin{aligned} \frac{\partial G}{\partial w_{ij}} &= -\frac{1}{T} \sum_{ab} P^+(V_a \wedge H_b) s_i^{ab} s_j^{ab} + \frac{1}{T} \sum_{mn} P^-(V_m \wedge H_n) s_i^{mn} s_j^{mn} \\ &= -\frac{1}{T} [p_{ij}^+ - p_{ij}^-] \end{aligned} \quad (5.102)$$

where

$$p_{ij}^+ = \sum_{ab} P^+(V_a \wedge H_b) s_i^{ab} s_j^{ab}$$

is the average of the product of the outputs of the units i and j , when the network is clamped and

$$p_{ij}^- = \sum P^-(V_m \wedge H_n) s_i^{mn} s_j^{mn}$$

is the average value of the product of the outputs of the units i and j when the network is free running. The change in the weight is given from Eqs. (5.87) and (5.102) as

$$\Delta w_{ij} = \frac{\eta}{T} [p_{ij}^+ - p_{ij}^-] \quad (5.103)$$

where η is a learning rate parameter and T is the temperature at which the equilibrium statistics are computed. This is called Boltzmann learning.

5.5.4 Issues In Implementation of Boltzmann Learning

Discussion on Boltzmann learning: The Boltzmann learning law in Eq. (5.103) is a result of using three principles: (a) **Hopfield** model with symmetric weights, no self-feedback and asynchronous update, (b) Boltzmann-Gibb's distribution for probability of states at thermal equilibrium under stochastic update of the neurons, and (c) an information theoretic measure for error criterion for matching the probability distribution of the states at thermal equilibrium with the distribution specified for the pattern environment. In the derivation of the law we have implicitly used the features of the Boltzmann machine, namely, the concept of hidden and visible units, stochastic update of the units, thermal equilibrium of the network at each temperature and simulated annealing with a specified schedule for annealing. The final objective is to adjust the weights in a systematic way so that the feedback network will have stable states at the desired input patterns. These states will have energies related to the probabilities of the input patterns through the Gibb's law. Table 5.7 gives a summary of Boltzmann learning law. Table 5.8 lists some of the issues in Boltzmann learning which will be discussed in some detail in this section.

The expression in Eq. (5.103) for Boltzmann learning shows that the learning has the local property, namely, the change in the weight connecting the units i and j depends on the values of the variables associated with those units only. It is interesting that the gradient descent along the information-theoretic based error surface leads to this desirable property of a learning law. The terms p_{ij}^+ and p_{ij}^- correspond to the terms in a Hebb's learning law. The term p_{ij}^+ is the average of the product of the output state values for the units i and j , averaged over all possible states of the network when the visible units of the network are clamped with the patterns to be stored. Thus p_{ij}^+ can be interpreted as correlation between the output values of the i th and j th units. Likewise p_{ij}^- is the correlation between the units when the network is in free running condition. The contribution to the weight change due to p_{ij}^+ can be viewed as Hebbian learning and that due to p_{ij}^- can be viewed as Hebbian unlearning. The second term

Table 5.7 Summary of Boltzmann Learning Law

-
- The objective is to adjust the weights of a Boltzmann machine so as to store a pattern environment described by the set of vectors $\{V_a\}$ and their probabilities of occurrence. These vectors should appear **as** outputs of the visible units. Let $\{H_b\}$ be the set of vectors appearing on the hidden units.
 - Let $P^+(V_a)$ be the probability that the outputs of the visible units will be clamped (indicated by '+' superscript) to the vector V_a . Then, $P^+(V_a) = \sum_b P^+(V_a \wedge H_b)$, where $P^+(V_a \wedge H_b)$ is the probability of the state of the network when the outputs of the visible units **are** clamped to the vector V_a , and the outputs of the hidden units **are** H_b .
 - Likewise the probability that V_a will appear on the visible units when none of the visible units **are** clamped (indicated by '-' superscript) is given by

$$P^-(V_a) = \sum_b P^-(V_a \wedge H_b).$$

- Note that $P^+(V_a)$ is given **by** the pattern environment description, and $P^-(V_a)$ **depends** on the network dynamics and is given by

$$P^-(V_a) = \sum_b \exp(-E_{ab}/T) / \sum_{mn} \exp(-E_{mn}/T),$$

where the total energy of the system in the state $V_a \wedge H_b$ is given by

$$E_{ab} = -\frac{1}{2} \sum_{i,j} w_{ij} s_i^{ab} s_j^{ab},$$

s_i^{ab} refers to the output of the i th unit in the state $V_a \wedge H_b$.

- The Boltzmann learning law is derived using the negative gradient descent of the functional

$$G = \sum_a P^+(V_a) \log [P^+(V_a)/P^-(V_a)]$$

- It can be shown that

$$-\partial G / \partial w_{ij} = (1/T) (p_{ij}^+ - p_{ij}^-),$$

where

$$p_{ij}^+ = \sum_{ab} P^+(V_a \wedge H_b) s_i^{ab} s_j^{ab}, \quad p_{ij}^- = \sum_{ab} P^-(V_a \wedge H_b) s_i^{ab} s_j^{ab}$$

The weight updates are calculated according to

$$\Delta w_{ij} = -\eta (\partial G / \partial w_{ij}) = (\eta/T) (p_{ij}^+ - p_{ij}^-).$$

- The Boltzmann learning law is implemented using an annealing schedule for the network during clamped and unclamped phases of the visible units of the network to determine p_{ij}^+ and p_{ij}^- , respectively.
-

can also be interpreted as a forgetting term. When the two correlations are equal, then we can interpret that the resulting network with the learned weights has absorbed the given pattern environment.

Table 5.8 Issues in Boltzmann Learning

Expression for Boltzmann learning: $\Delta w_{ij} = \frac{\eta}{T} [p_{ij}^+ - p_{ij}^-]$

- Significance of p_{ij}^+ and p_{ij}^-
- Learning and unlearning
- Local property
- Choice of η and initial weights

Implementation of Boltzmann learning

- Algorithm for **learning** a pattern environment
- Algorithm for recall of a pattern
- Implementation of simulated annealing
- Annealing schedule

Pattern recognition tasks by Boltzmann machine

- Pattern completion
- Pattern association
- Recall from noisy or partial input

Interpretation of Boltzmann learning

- Markov property of simulated annealing
- Clamped-free energy and full-free energy

Variations of Boltzmann learning

- Deterministic Boltzmann machine
 - Mean-field approximation
-

In the implementation of the Boltzmann learning law, there are two distinct phases, one for determining p_{ij}^+ by clamping the input **pattern** to the visible units and the **other** for determining p_{ij}^- corresponding to the free running condition. In each phase the network is subjected to an annealing process, starting with some high temperature and using an annealing schedule. At each temperature in the schedule the network is allowed to reach **thermal** equilibrium. The Metropolis algorithm [Metropolis et al, **1953**] may be used to arrive at the thermal equilibrium of states at each temperature. The algorithm uses the probability law given in Eq. (5.47) for updating the state of a unit based on the activation value (net input to the unit). The probability law is implemented using a random number uniformly distributed in the interval 0 to 1, and comparing the number with the probability [Binder and **Heerman**, **1981**]. If the difference between the generated random number and the computed probability is positive, then the state of the unit is updated to the new value. Otherwise the state is unaltered. This is repeated for each unit **selected** at random, and for several cycles. Each cycle consists of N iterations, where N is the number of **units** in the network. After a certain number of cycles the network reaches thermal equilibrium at that temperature. At that stage the temperature is lowered to the next value in the schedule.

At the thermal equilibrium, achieved at the lowest value of the temperature in the annealing schedule, the products $s_i s_j$ are computed for all i and j . This process is repeated for each presentation of an input pattern in the clamped phase. The input patterns are presented one by one several times according to their frequency of occurrence in the pattern environment. **Find** the average of the $s_i s_j$ from all these trials. The resulting average value is an estimate of p_{ij}^+ . The same operations are repeated on the network in the free running condition for the same number of trials with the same number of cycles at each stage in each trial. The resulting average of $s_i s_j$ gives p_{ij}^- .

To start with, the weights are set to some random initial values in the range -1 to $+1$, assuming that the state of each unit in the network is either 0 or 1 (binary units). The value of the learning rate parameter η is chosen in the range of 0 to 1 , preferably a small value of 0.1 . The range of temperatures for annealing also could be from $T=1$ to $T=0.1$. The weights are adjusted according to Eq. (5.103). The algorithm for implementing the Boltzmann learning law is given in Table 5.9.

Table 5.9 Boltzmann Machine Learning Algorithm for Binary Units

-
1. Clamp one training vector to the visible units.
 2. Anneal until equilibrium is reached at desired minimum temperature.
 3. Continue to run the network for several processing cycles. **After** each cycle determine the connected units whose states are '1' simultaneously.
 4. Average the cooccurrence results from Step 3.
 5. Repeat Steps 1 to 4 for all training vectors to get p_{ij}^+ .
 6. Unclamp the visible units, and anneal until equilibrium at the desired minimum temperature
Continue to run the network for several processing cycles. **After** each cycle determine the connected units whose states are '1' simultaneously.
 8. Average the cooccurrence results **from** Step 7.
 9. Repeat Steps 6 to 8 for the same number of times as in Step 5 to get p_{ij}^- .
 10. Calculate and apply the appropriate weight changes.
 11. Repeat Steps 1 to 10 until $p_{ij}^+ - p_{ij}^-$ is **sufficiently** small.
-

To recall a stored pattern, the given partial input is clamped to the appropriate visible units, and the network is subjected to an annealing process according to a schedule to reach thermal equilibrium at the minimum temperature. The output state of the visible units at this stage corresponds to the pattern to be recalled. Table 5.10 gives an algorithm for recalling a stored pattern.

The Boltzmann learning is a very slow process, since a large number of cycles **are** needed to obtain **sufficient** amount of data to estimate the desired averages p_{ij}^+ and p_{ij}^- reasonably well. The learning

Table 5.10 Boltzmann Machine: Recall from Partial Input

-
1. Force outputs of visible units to specified initial input binary vector.
 2. Assign unknown visible units and hidden units to random value (0, 1).
 3. Select a unit k at random and calculate the activation value x_k .
 4. Assign the output $s_k = 1$ with probability $1/(1 + e^{-x_k/T})$.
 5. Repeat Steps 3 and 4 until all units have a chance to update (one processing cycle).
 6. Repeat Step 5 for several processing cycles until thermal equilibrium is reached at the temperature T .
 7. Lower T and repeat Steps 3 to 6.
 8. Once the temperature has been reduced to a small value, the network will stabilize.
 9. The final result will be the outputs of the visible units.
-

rate parameter η should be small in order to take care of the inaccuracies in the estimates of these averages. If η is large, then there is a possibility of taking a large step along a wrong direction due to approximation in the computation of the gradient of the error measure. But a small value of η further slows down the learning process.

Success of annealing in the learning process critically depends on the annealing schedule. The probability distribution of the states converges asymptotically to the distribution corresponding to the minimum average energy value, provided the temperature at the k th step in the annealing schedule satisfies the following inequality [Geman and Geman, 1984; Aarts and Korst, 1989; Salamon et al, 1988]:

$$T_k \geq \frac{T_s}{1 + \log k} \quad (5.104)$$

where T_s is the initial high temperature. This annealing schedule is too slow for implementation in practice. Several ad hoc schedules were suggested to speed up the process of annealing. One such method uses $T_k = T_s/(1 + k)$, which is known as fast annealing schedule or a Cauchy machine [Szu, 1986]. But there is no proof of convergence towards the minimum average energy value in these ad hoc methods.

Boltzmann machine can be used for recalling a stored pattern from partial input, by clamping the known input at the corresponding visible units. This is called pattern completion task. Boltzmann machine can also be used for pattern association task. In this case the visible units are split into two parts, one part corresponding to the input pattern and the other to the output pattern. During training both the input and output patterns are given as a pair to the visible units. Thus all the given pattern pairs are used in the training. While recalling, the input part of the visible units are clamped and the recall is implemented as in the pattern completion task. The state at the output part of the visible units gives the associated pattern.

It is also possible to recall a pattern from a noisy version of it. In this case the noisy input pattern is presented initially to the visible units and subsequently the network is allowed to anneal as in a **free** running condition. The initial presentation of the noisy input pattern will bias the state of the network towards the true state and the annealing process will be helpful to overcome the local minima states to reach the deep minimum corresponding to the stored pattern.

In the operation of the Boltzmann machine the state of the network due to a transition depends on the previous state only and not on the states prior to the previous state (See Eq. (5.84)). This is called Markov property of the simulated annealing [van Laarhoven and Aarts, 1988; **Haykin**, 1994, p. 316]. Note that the transition probabilities are derived assuming a probability distribution for the update of the state of a unit and using an asynchronous update in which only one unit is considered at a time for updating. The probability distribution of the states of the network at a given instant together with the transition probabilities will enable us to **determine** the probability distribution of the states at the next instant in the simulated annealing process (See Eq. (5.84)). This Markov property will eventually lead to a stationary probability distribution of the states at thermal equilibrium. Moreover, the stationary probability distribution in turn is related to the energy distribution of the states through the Boltzmann-Gibb's law.

The Boltzmann learning law can be interpreted in terms of the energy and probability distribution of the states as follows: Let the partition **function** Z be expressed as

$$Z = \sum_{ab} e^{-E_{ab}/T} = e^{-F/T} \quad (5.105)$$

where F is the free energy of the system. Then we have from Eqs. (5.65) and (5.72)

$$F = -T \log Z \quad (5.106)$$

and

$$p_{ij}^- = -\frac{\partial F}{\partial w_{ij}} \quad (5.107)$$

Let $Z_{\text{clamped}}^a = \sum_b e^{-E_{ab}/T}$ and $Z_{\text{unclamped}} = \sum_{ab} e^{-E_{ab}/T}$ Then

$$\begin{aligned} P^-(V_a) &= \sum_b P^-(V_a \wedge H_b) \\ &= \sum_b \frac{e^{-E_{ab}/T}}{Z} = \frac{Z_{\text{clamped}}^a}{Z_{\text{unclamped}}} \\ &= (e^{-F^a/T}) / (e^{-F/T}) \end{aligned} \quad (5.108)$$

where F^a is the clamped **free** energy, i.e., the free energy when the visible units are clamped with the state V_a .

The error function G of Eq. (5.86) can be written as

$$G = G_0 - \sum_a P^+(V_a) \log P^-(V_a) \quad (5.109)$$

where $G_0 = \sum_a P^+(V_a) \log P^+(V_a)$ is independent of the network parameters **such** as weights. Therefore, from Eqs. (5.108) and (5.109) we get

$$\begin{aligned} G &= G_0 + \frac{1}{T} \sum_a P^+(V_a) F^a - \frac{1}{T} \sum_a P^+(V_a) F \\ &= G_0 + \frac{1}{T} [\bar{F}^a - F] \end{aligned} \quad (5.110)$$

where \bar{F}^a is the average clamped free energy and F is the full free energy [Hertz et al, 1991, Appendix]. We can show that (See Problem 5 in Ch. 5)

$$p_{ij}^+ = \frac{-\partial \bar{F}^a}{\partial w_{ij}} \quad (5.111)$$

Therefore,

$$\begin{aligned} \Delta w_{ij} &= -\eta \frac{\partial G}{\partial w_{ij}} \\ &= \frac{\eta}{T} (p_{ij}^+ - p_{ij}^-) \end{aligned} \quad (5.112)$$

where p_{ij}^+ and p_{ij}^- are given by Eqs. (5.111) and (5.107), respectively. Thus we can view the error function ($G - G_0$) as the difference between the average clamped free energy and the full free energy. The full free energy will be lower since under free running condition the energy landscape is perfectly matched to the probability distribution of states at thermal equilibrium through the **Gibb's** law. In the clamped condition the stationary probabilities do not match the energy landscape **perfectly** due to the constraint of the clamping. Note that the energy landscape depends on the connection weights and states of the network.

Computation of the average values of the correlations requires a very large number of iterations in the Boltzmann learning law. The implementation of the Metropolis algorithm by Monte **Carlo** method [Binder et al, 1988] for state update, together with the simulated annealing according to an annealing schedule, results in an extremely slow learning of the Boltzmann machine. A fast learning procedure is to simply run the machine only at zero temperature. This is called deterministic Boltzmann machine [LeCun, 1986]. This has the

disadvantage of getting stuck in shallow minima. Consequently, the pattern environment cannot be exactly matched with the network.

A better approach for learning which retains some of the advantages of the stochastic nature of the network is called mean-field annealing [Peterson and Anderson, 1987]. In the mean-field annealing the stochastic nature of the neurons is replaced by mean values of the outputs of the units. That is, according to the Eq. (5.78) for bipolar units we have

$$\begin{aligned}\langle s_i \rangle &= \tanh(x_i/T) \\ &= \tanh\left(\frac{1}{T} \sum_j w_{ij} \langle s_j \rangle\right)\end{aligned}\quad (5.113)$$

We get one such nonlinear equation for each unit i . These equations are solved using iterative methods. This, combined with the annealing process, can be used to obtain the average correlation values at thermal equilibrium at the minimum temperature. The average correlation values become the product of the individual average values. That is,

$$\langle s_i s_j \rangle = \langle s_i \rangle \langle s_j \rangle \quad (5.114)$$

The mean-field approximation minimizes the mean-field free energy, given by [Hinton, 1989]

$$G_{mf} = G_0 + \frac{1}{T} [\bar{F}_{mf}^a - F_{mf}] \quad (5.115)$$

where \bar{F}_{mf}^a is the mean-field energy when the visible units are clamped and F_{mf} is the mean-field free energy under **unclamped** conditions. Using gradient descent, the weight update using the mean-field approximation is given by

$$\begin{aligned}\Delta w_{ij} &= -\eta \frac{\partial G_{mf}}{\partial w_{ij}} \\ &= \frac{\eta}{T} [\langle s_i^a \rangle \langle s_j^a \rangle - \langle s_i \rangle \langle s_j \rangle]\end{aligned}\quad (5.116)$$

which is the Boltzmann learning law with the average correlations replaced by the average values for each unit. Table 5.11 gives an algorithm for implementing the mean-field approximation to Boltzmann learning. The mean-field approximation results in **10–30** times speed up of the Boltzmann learning, besides providing somewhat better results [Peterson and Anderson, 1987].

Table 5.11 Algorithm for Mean-field Approximation for Boltzmann Learning

1. Initialize the weights to some random values uniformly distributed in the range ± 1 .
2. Clamp the units with a given pattern. Starting at some high temperature, the network is subjected to an annealing process using each time the mean-field values (\mathbf{s}). The mean-field value is computed using the recursive formula

$$\langle s_i \rangle_{\text{new}} = \tanh \left(\frac{1}{T} \sum_j w_{ij} \langle s_j \rangle_{\text{old}} \right), \quad i = 1, 2, \dots, N$$

3. At the final minimum temperature compute the correlations

$$p_{ij}^+ = \langle s_i \rangle \langle s_j \rangle, \quad i, j = 1, 2, \dots, N$$

4. Likewise compute the correlations p_{ij}^- in the free running case

$$p_{ij}^- = \langle s_i \rangle \langle s_j \rangle, \quad i, j = 1, 2, \dots, N$$

5. Compute the weight update using the Boltzmann learning law:

$$\Delta w_{ij} = \frac{\eta}{T} [p_{ij}^+ - p_{ij}^-]$$

where η is learning rate parameter.

6. Repeat Steps 2 to 5 until convergence of weights.

5.6 Summary

Feedback neural networks are used mainly for pattern storage tasks. In this chapter we have given a detailed analysis of simple feedback networks for storing a set of patterns. Associated with each state of the network is an energy value. The key idea in pattern storage by feedback networks is the formation of basins of attraction in the energy landscape in the activation or output state space. The **Hopfield** conditions for formation of suitable energy landscape are discussed. In order to store a set of patterns in a feedback network with hard-limiting threshold units, a set of inequalities have to be satisfied by the weights connecting the units. Thus there may be several solutions for the weights satisfying the inequalities. The resulting energy landscape may have additional false minima corresponding to patterns not designed for storage. This happens if the storage capacity of the network is higher than the number of patterns required to be stored. The presence of false minima will increase the probability of error in recall of the stored pattern.

The effect of false minima is reduced using stochastic units instead of deterministic units. Analysis of stochastic neural network is based on the concepts of thermal equilibrium and simulated annealing. These concepts are used for traversal along an energy

landscape to reduce the effects of false minima during recall of stored patterns. To reduce the probability of error in recall, the weights of a feedback network are adjusted using the knowledge of the patterns as well the probability distribution of these patterns. Loading of the pattern environment is accomplished in a feedback network with stochastic units using Boltzmann learning law. The learning law, derived based on an information theoretic criterion, involves only local computations, and is implemented using simulated annealing according to a temperature schedule.

Boltzmann learning law is too slow for implementation in any practical situations involving pattern environment storage. For practical implementation, an approximation in the form of mean-field annealing is used. While there is no guarantee for solution, mean-field annealing has been applied in several applications, especially in optimization problems. Some of these applications will be discussed in Chapter 8.

Review Questions

1. Distinguish between autoassociation, pattern storage and pattern environment storage tasks. Give examples for each task.
2. What is the significance of the nonlinear output function of the units in feedback neural networks?
3. Explain the meaning of activation state and energy landscape of a feedback network.
4. What is meant by capacity of a feedback network?
5. What is the **Hopfield** model of a neural network?
6. Explain the differences between discrete and continuous **Hopfield** models in terms of energy landscape and stable states.
7. What is a state transition diagram for a feedback network? Explain how to derive it for a given network.
8. What are hard problems in pattern storage task?
9. How to solve the hard pattern storage problems?
10. Explain with the help of a state transition diagram the meaning of stable states and false minima.
11. **How** to overcome the effects of false minima?
12. What is the significance of hidden units in a feedback network?
13. What is meant by stochastic update of a neuron?
14. Explain the concept of equilibrium in stochastic neural networks.
15. Explain the meaning of stationary probability distribution at thermal equilibrium.
16. What is the significance of Gibb's distribution?

17. What is meant by stability in the case of stochastic neural networks?
18. Show the probability function for update of a neuron for different temperatures. Explain the significance of the temperature parameter.
19. Discuss the behaviour of **trajectories** of the states during the transient portion when temperature is changed.
20. Discuss the behaviour of stationary probability distributions of the states at different temperatures in relation to the energy landscape.
21. Explain the behaviour of a stochastic neural network at thermal equilibrium with reference to Brownian particle motion.
22. Explain how to derive the state transition diagram for a stochastic neural network.
23. What differences will you observe in the state transition diagrams at two different temperatures?
24. Describe a bouncing ball analogy for the dynamics of a stochastic neural network.
25. What is meant by capacity of a stochastic neural network? How does it vary for different temperatures?
26. What is meant by simulated annealing? What is annealing schedule?
27. Describe the Boltzmann machine.
28. What is the basis for Boltzmann learning law?
29. What is the significance of the Boltzmann learning law given by Eq. (5.103)?
30. Distinguish between clamped and free running conditions in a Boltzmann machine during learning.
31. Explain the implementation details of the Boltzmann learning law.
32. Explain the implementation details of recall of patterns in a Boltzmann machine.
33. How to perform the following tasks by a Boltzmann machine?
 - (a) Pattern completion
 - (b) Pattern association
 - (c) Pattern recall from noisy input.
34. What are the limitations of the Boltzmann learning?
35. What is a Cauchy machine?
36. What is the **Markov** property of the simulated annealing process?
37. What is meant by full free energy and clamped free energy in a Boltzmann machine?

38. How do you interpret the Boltzmann learning in terms of full free energy and clamped free energy?
39. What is mean-field approximation to Boltzmann learning?
40. What is meant by deterministic Boltzmann machine?

Problems

1. Derive the **Murakami** result in Eq. (5.12) for autoassociation task.
2. Show the result of **Hopfield** analysis, i.e., $\mathbf{AV} \leq 0$, for a feedback network with binary $\{0, 1\}$ units.
3. Draw a state transition diagram for a 3-unit model with bipolar $\{-1, +1\}$ units.
4. Using the Eqs. (5.65) and (5.69), derive the result in Eq. (5.70).
5. Show that (See Eqs. (5.72), (5.107), (5.108) and (5.111))

$$p_{ij}^- = -\frac{\partial F}{\partial w_{ij}} \quad \text{and} \quad p_{ij}^+ = -\frac{\partial \bar{F}^a}{\partial w_{ij}}$$

where F and \bar{F}^a are the full free energy and the clamped free energy of the Boltzmann machine.

6. Derive the expression for Δw_{ij} for the mean-field approximation of the Boltzmann learning. (See [Hertz et al, 1991, p. 1721].)
7. Show that the information theoretic measure $G \geq 0$. (See Eq. (5.86))
8. Derive the complete state transition diagram for the 3-unit network given in the Figure 5.7(a) for a temperature of $T = 1.0$.
9. For a 5-unit feedback network the weight matrix is given by

$$W = \begin{bmatrix} 0 & 1 & -1 & -1 & -3 \\ 1 & 0 & 1 & 1 & -1 \\ -1 & 1 & 0 & 3 & 1 \\ -1 & 1 & 3 & 0 & 1 \\ -3 & -1 & 1 & 1 & 0 \end{bmatrix}$$

Assuming that the bias and input of each of the units to be zero, compute the energy at the following states.

$$\mathbf{s} = [-1 \ 1 \ 1 \ 1 \ 1]^T \quad \text{and} \quad \mathbf{s} = [-1 \ -1 \ 1 \ -1 \ -1]^T$$

10. A 3-unit feedback network has the weight vector given by

$$W = \begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix}$$

Compute the gradient vector \mathbf{VV} and the Hessian matrix $\nabla^2 V$ for the energy function of the network.

11. Consider the representation of each of the ten digits (0, 1, ..., 9) by a matrix of 10 x 10 elements, where each element is either 1 or -1. Design a **Hopfield** network of 100 units to store these digits. Study the performance of the network for recall of digits if 10% of the elements are randomly switched. (See [Haykin, 1994, p. 297].)
12. Comment on the capacities of the following networks:
- (a) **Feedforward** neural network
 - (i) linear units $C \approx 0.7M$ (dimensionality of the input)
 - (ii) nonlinear units $C = 2M$ for large M (See [Hertz et al, 1991, pp. 111–114].)
 - (b) Feedback neural network with N units (See Hertz et al, 1991, p. 39)
 - $C = 0.138N$ for large N
 - (c) Hamming network (See Hint for Problem 3 in Chapter 8 and [Kung, 1993, p. 611])
 - $C = 2^{pM}$, $p \leq 1$.

Chapter 6

Competitive Learning Neural Networks.

6.1 Introduction

In this chapter we consider pattern recognition **tasks** that a network of the type shown in Figure 6.1 can perform. The network consists of an input layer of linear units. The output of each of these units is given to **all** the units in the second layer (output layer) with adaptive (adjustable) feedforward weights. The output functions of the units in the second layer are either linear or nonlinear depending on the task for which the network is to be designed. The output of each unit in the second layer is fed back to itself in a self-excitatory manner and to the other units in the layer in an excitatory or inhibitory manner depending on the task. Generally the weights on the connections in the feedforward layer are nonadaptive or fixed. Such a combination of both feedforward and feedback connection layers results in some kind of competition among the activations of the units in the output layer, and hence such networks are called *competitive learning neural networks*. Different choices of the output functions

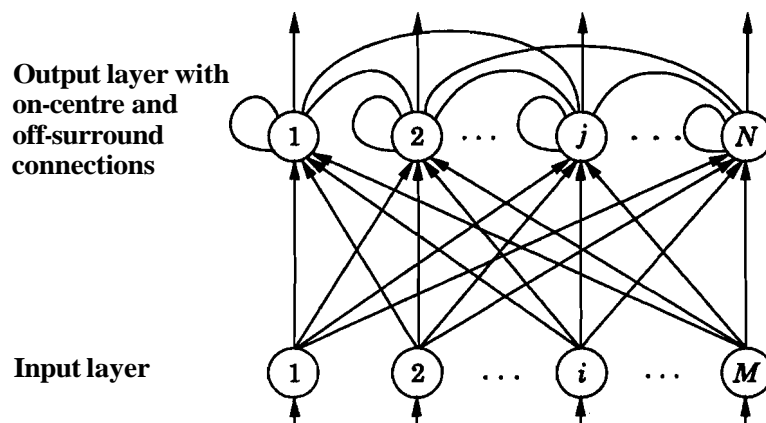


Figure 6.1 A feedforward and feedback structure. The feedforward weights are adaptive and the weights in the feedback layer are fixed.

and interconnections in the feedback layer of the network can be used to perform different pattern recognition tasks. For example, if the output functions are linear, and the feedback connections are made in an on-centre off-surround fashion, the network performs the task of storing an input pattern temporarily. In an on-centre off-surround connection there is an excitatory connection to the same unit and inhibitory connections to the other units in the layer. But such a network is of theoretical interest only, as there are few occasions where one needs to store a pattern temporarily in this manner. On the other hand, if the output functions of the units in the feedback layer are made nonlinear, with fixed weight on-centre off-surround connections, the network can be used for pattern clustering. The objective in pattern clustering is to group the given input patterns in an unsupervised manner, and the group for a pattern is indicated by the output unit that has a nonzero output at equilibrium. The network is called a *pattern clustering* network, and the feedback layer is called a *competitive layer*. The unit that gives the nonzero output at equilibrium is said to be the *winner*. Learning in a pattern clustering network involves adjustment of weights in the feedforward path so as to orient the weights (leading to the winning unit) towards the input pattern.

If the output functions of the units in the feedback layer are nonlinear and the units are connected in such a way that connections to the neighbouring units are all made **excitatory** and to the farther units inhibitory, the network then can perform the task of *feature mapping*. The resulting network is called a *self-organization* network. In the self-organization, at equilibrium the output signals from the nearby units in the feedback layer indicate the proximity of the corresponding input patterns in the feature space. A self-organization network can be used to obtain mapping of features in the input patterns onto a one-dimensional or a two-dimensional feature space.

Table 6.1 shows the organization of the topics to be discussed in this chapter. First a detailed discussion on the components of a competitive learning network is given in Section 6.2. In particular, we will discuss the input layer, a single **instar** network and a group of **instars**. We will also discuss the learning laws for an **instar** and the activation dynamics of the feedback network. We will show that, with some variation of the learning for the **instar** networks, one can obtain the principal component analysis learning networks. In Section 6.3 an analysis of the combination network with linear units in the feedback layer is presented to show the short time memory nature of the pattern recognition task performed by such a network. In this section the significance of different nonlinear output functions of the units in the feedback layer is also discussed. An analysis of the competitive learning network for pattern clustering is given in Section 6.4. Some applications of the pattern clustering networks are

Table 6.1 Pattern Recognition Tasks by **Feedforward (FF)** and Feedback (**FB**) ANN (Competitive Learning Neural Networks)

Pattern storage (STM)

- **Architecture:** Two layers (input and competitive), linear processing units
- **Learning:** No learning in FF stage, **fixed** weights in FB layer
- **Recall:** Not relevant
- **Limitation:** STM, no application, theoretical interest
- **To overcome:** Nonlinear output function in FB stage, learning in FF stage

Pattern clustering (grouping)

- **Architecture:** Two layers (input and competitive), nonlinear processing units in the competitive layer
- **Learning:** Only in FF stage, Competitive learning
- **Recall:** Direct in FF stage, activation dynamics until stable state is reached in FB layer
- **Limitation:** Fixed (rigid) grouping of patterns
- **To overcome:** Train neighbourhood units in competition layer

Feature map

- **Architecture:** Self-organization network, two layers, nonlinear processing units, excitatory neighbourhood units
 - **Learning:** Weights leading to the neighbourhood units in the competitive layer
 - **Recall:** Apply input, determine winner
 - **Limitation:** Only visual features, not quantitative
 - **To overcome:** More complex architecture
-

also discussed briefly in this section. A detailed analysis of the self-organization network is given in Section 6.5. Several examples of feature mapping are given in this section to illustrate the significance of the concept of self-organization.

6.2 Components of a Competitive Learning Network

A competitive learning network consists of an input layer with linear units, a group of **instars forming** a feedforward portion of the network and a feedback layer with linear or nonlinear units. In this section we discuss each of these components in some detail.

6.2.1 The Input Layer

The purpose of this layer is to distribute the given external input pattern vector to the feedforward portion of the network. But in general the input vectors may be of varying magnitude, even though they may contain the same pattern information. Moreover, for any processing by a unit, it is necessary to have the inputs bounded to

some limits. In an on-line situation, the input layer should not feed background noise to the feedforward portion of the competitive learning network. The so called noise-saturation dilemma (discussed in Chapter 2) for input vectors can be handled by feeding the actual inputs from the environment to a layer of input processing units as shown in Figure 6.2. A shunting activation model with on-centre

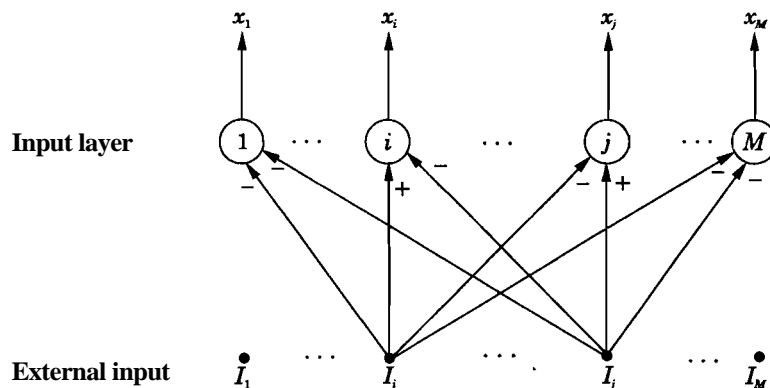


Figure 6.2 Input layer with M processing units, showing a few connections with external inputs.

off-surround configuration takes care of the noise-saturation problem of the input layer, and is given by (see Eq. 2.18)

$$\dot{x}_i = -Ax_i + (B - x_i)I_i - (C + x_i) \sum_{j \neq i} I_j \quad (6.1)$$

As shown in Chapter 2, the steady state activation value of the i th unit is given by

$$x_i^{(\infty)} = \frac{(B + C)I_i - CI}{A + I} = \left[\frac{I_i}{I} - \frac{C}{B + C} \right] \frac{B + C}{1 + A/I} \quad (6.2)$$

where

$$I = \sum_{i=1}^M I_i$$

and all the inputs (I_i) are nonnegative. The above equations show that in the steady state the activation value of the i th unit is confined to the range $[-C, B]$. The output function of these units is assumed to be linear for $x > 0$. That is

$$\begin{aligned} f(x) &= x, \quad \text{for } x \geq 0 \\ &= 0, \quad \text{for } x < 0 \end{aligned} \quad (6.3)$$

The output of the units will be zero as long as the inputs $I_i < CI/(B + C)$. That is, the input should be greater than some minimum

value before it can make the activation of the unit positive. **Thus** the units in the input layers do not respond to noise input, if the noise amplitude is below some threshold value. Therefore the input to the **feedforward** portion is always positive and is limited to a maximum value of B . Thus this input layer not only **normalizes** the input data values, but also takes **care** of the noise-saturation problem with an on-line input data.

6.2.2 The Instar

Each unit in the feedback layer receives inputs from all the input units. A configuration where a unit receives weighted inputs from several units of another layer is called an *instar*, as shown in Figure 6.3. Let $\mathbf{x} = (x_1, x_2, \dots, x_M)^T$ and $\mathbf{w} = (w_1, w_2, \dots, w_M)^T$ be the

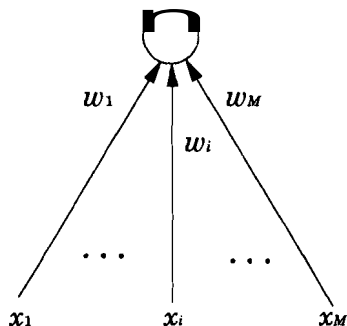


Figure 6.3 An instar configuration.

input and weight vectors, respectively. The net input to the **instar** processing unit is given by $\mathbf{w}^T \mathbf{x}$. The activation **dynamics** of the **instar** processing unit is given by the following additive model with a passive decay term and the net input term:

$$\dot{y}(t) = -y(t) + \mathbf{w}^T \mathbf{x}, \quad (6.4)$$

where we have assumed the decay constant to be 1. The solution of this equation is

$$y(t) = y(0) e^{-t} + \mathbf{w}^T \mathbf{x} (1 - e^{-t}) \quad (6.5)$$

The steady state activation value is given by

$$y(\infty) = \mathbf{w}^T \mathbf{x} \quad (6.6)$$

which will be zero when the external input \mathbf{x} is removed.

6.2.3 Basic Competitive Learning

The steady activation value with an external input depends on the angle between the input and weight vectors as shown in Figure 6.4. For the **instar** to respond maximally for a given input vector \mathbf{x} , the weight vector is moved towards the input vector. The adjustment of

Competitive Learning Neural Networks

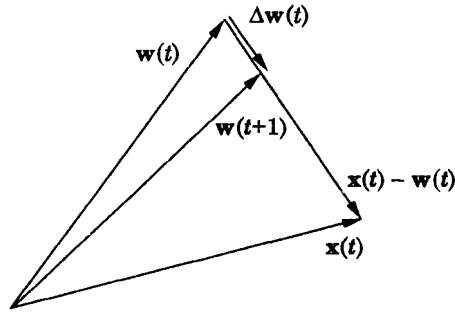


Figure 6.4 Illustration of adjustments of weights in instar

the weight vector is governed by the following synaptic dynamics equation (see Eq. 2.40)

$$\dot{\mathbf{w}} = [\mathbf{x} - \mathbf{w}] f(y) \quad (6.7)$$

where $f(y)$ is the output of the instar processing unit. In discrete implementation the change in the weight $\Delta \mathbf{w}(t)$ is given by

$$\Delta \mathbf{w}(t) = \eta [\mathbf{x} - \mathbf{w}(t)] f(y) \quad (6.8)$$

where η is a learning rate parameter. For binary output function, $f(y) = 1$ or 0 . Therefore the weights in the instar are adjusted only when the output of the instar processing unit is 1. The increment in the weight is given by

$$\Delta \mathbf{w}(t) = \eta [\mathbf{x} - \mathbf{w}(t)] \quad (6.9)$$

The updated weight is given by

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}(t) \quad (6.10)$$

This shows that the weight vector is adjusted in such a way that it is moved towards the input vector to reduce the angle between them as shown in Figure 6.4. This adjustment is repeated several times for a given set of input patterns. When the weight vector reaches an average position, the weight adjustment will be such that the average movement around the weight vector will be zero. That is

$$\mathcal{E}[\Delta \mathbf{w}] = 0 = \eta (\mathcal{E}[\mathbf{x}] - \mathbf{w}) \quad (6.11)$$

Therefore the weight vector \mathbf{w} will be equal to the average of the set of input vectors.

An individual instar responds to a set of input vectors, if it is trained to capture the average behaviour of the set. Generally the input patterns may fall into different categories. More than one instar, i.e., a group of instars (Figure 6.5), can be used to capture the average behaviour of each of the different categories of the input patterns. One instar may be trained to each category of the input patterns, so that the corresponding processing unit responds maxi-

Components of a Competitive Learning Network

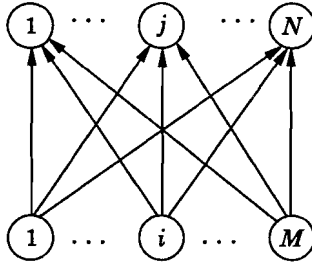


Figure 6.5 A group of N instars.

mally when an input vector belonging to that category is given to the common input layer of the group of **instars**. Typically, the number (N) of **instars** corresponds to the number of different categories of the input patterns. The category of an input vector can be identified by **observing** the **instar** processing unit with maximum response.

Plain Hebbian learning: With *linear* output function for the **instar** processing unit, i.e., $f(y) = y$, one possibility is to make the output y to be a scalar measure of similarity. That is, for the given input vector \mathbf{x} , the weights should be adjusted to give a large output y , on the average. For this, if we use the plain Hebbian learning, we get

$$\Delta w_i = \eta y x_i \quad (6.12)$$

where η is the learning rate parameter. In this case the weights keep growing without bound. That is, the weights never converge, which is equivalent to saying that on the average the weight change will not be zero. This can be proved as follows:

Taking the expectation to compute the average, we get

$$\mathbf{E}[\Delta \mathbf{w}_i] = \mathbf{E} \left[\eta \sum_j w_j x_j x_i \right] = \eta \sum_j w_j \mathbf{E}[x_j x_i], \quad (6.13)$$

where it is assumed that the weight vector is statistically independent of the input vector. Therefore the average change in the weight vector is given by

$$\mathbf{E}[\Delta \mathbf{w}] = \eta \mathbf{R} \mathbf{w} \quad (6.14)$$

where $\mathbf{A} \mathbf{w} = (\Delta w_1, \Delta w_2, \dots, \Delta w_M)^T$, and $\mathbf{R} = \mathbf{E}[\mathbf{x} \mathbf{x}^T]$ is the auto-correlation matrix of the input vectors.

\mathbf{R} is positive semidefinite, since for any vector \mathbf{a} , we have

$$\mathbf{a}^T \mathbf{R} \mathbf{a} = \mathbf{a}^T \mathbf{E}[\mathbf{x} \mathbf{x}^T] \mathbf{a} = \mathbf{E}[\mathbf{a}^T \mathbf{x} \mathbf{x}^T \mathbf{a}] = \mathbf{E}[(\mathbf{x}^T \mathbf{a})^2] \geq 0 \quad (6.15)$$

Therefore \mathbf{R} will have only positive eigenvalues. The weight vector converges if $\mathbf{E}(\Delta \mathbf{w}) = \mathbf{0}$. Then Eq. (6.14) states that for convergence $\mathbf{R} \mathbf{w} = \mathbf{0}$. This means that the resulting weight vector is an eigenvector

with zero eigenvalue, which is not a stable situation, since R has positive eigenvalues. Any fluctuation of the weight vector with a component along an eigenvector of R will result in a weight vector which would grow eventually, and the component along the eigenvector with the largest eigenvalue will dominate.

This can be shown as follows: Taking the **projection** of the average change of the weight vector onto one of the eigenvectors \mathbf{q}_i of R , we get

$$\mathbf{q}_i^T \mathcal{E}[\Delta \mathbf{w}] = \eta \mathbf{q}_i^T R \mathbf{w} \quad (6.16)$$

This will be nonzero if \mathbf{w} is the eigenvector \mathbf{q}_i , since $R \mathbf{q}_i = \lambda_i \mathbf{q}_i$ and the eigenvectors $\{\mathbf{q}_i\}$ are orthonormal. Therefore we have

$$\mathbf{q}_i^T \mathcal{E}[\Delta \mathbf{w}] = \eta \mathbf{q}_i^T R \mathbf{q}_i = \eta \lambda_i.$$

This projection in turn is maximum when \mathbf{q}_i is the eigenvector \mathbf{w}_0 corresponding to the maximum eigenvalue λ_{\max} . Therefore we have

$$\mathcal{E}[\Delta \mathbf{w}] = \eta R \mathbf{w}, = \eta \lambda_{\max} \mathbf{w}_0 \quad (6.17)$$

According to Eq. (6.17) the average change of the weight will be dominated by the component along the eigenvector with maximum eigenvalue. The norm of \mathbf{w} will increase without bound, because

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \mathcal{E}[\Delta \mathbf{w}] = \mathbf{w}(m) + \eta \lambda_{\max} \mathbf{w}_0 \quad (6.18)$$

Starting with an initial value of the weight $\mathbf{w}(0) = 0$, and determining the value of the weight at each time instant, we get from Eq. (6.18)

$$\mathbf{w}(m) = m \eta \lambda_{\max} \mathbf{w}_0 \quad (6.19)$$

Thus, with plain Hebbian learning, there will be only unstable weight values.

Oja's rule: The divergence of the plain Hebbian learning can be prevented by constraining the growth of the weight vector \mathbf{w} . One way of doing this is by normalizing the weight $\|\mathbf{w}\| = 1$ at every stage after adjustment. Another method, called Oja's rule, uses a decay term proportional to y^2 in the **synaptic** dynamics equation. Then the weight update in Eq. (6.12) is modified as

$$\begin{aligned} \Delta w_i &= -\eta y^2 w_i + \eta y x_i \\ &= \eta y (x_i - y w_i) \end{aligned} \quad (6.20)$$

In this case the change in the weight depends on the difference between the actual input and the back propagated output ($y w_i$). The weight vector will eventually align with the eigenvector of R corresponding to the largest eigenvalue [Oja, 1982].

If R is a covariance matrix, i.e., $R = \mathcal{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]$, $\mathbf{u} = \mathcal{E}[\mathbf{x}]$, then the final weight vector will be along the largest

principal component passing through the origin (Figure 6.6a) [Hertz et al, 1991; Haykin, 1994]. See Appendix E for details of principal component analysis. If R is only an autocorrelation matrix, the final weight vector will still be along the largest principal eigenvector, passing through the origin (Figure 6.6b). But in the later case the choice will not be optimal in the sense that the projection of the set of input vectors on the weight vector will not be the least as in the case of the covariance matrix. In both cases the weight vector will settle to the normalized value $\|w\| = 1$.

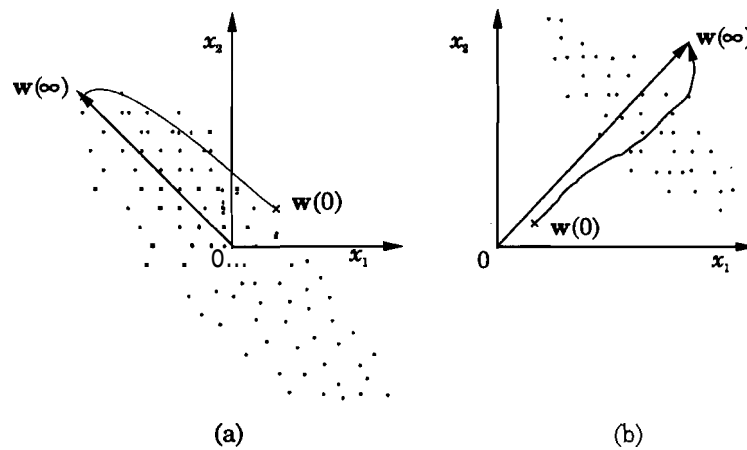


Figure 6.6 Illustration of Oja's rule for training instar. The training data consists of vectors in 2-D space, represented as dots. The line from $w(0)$ to $w(\infty)$ is the trajectory of the weight values. The final weight represents direction of maximum variance of the input data as shown in (a). But in (b) the direction of maximum variance is the average of the input data vectors [Adapted from Hertz et al, 1991, p. 201].

Oja's rule finds a unit weight vector which maximizes the mean squared output $E[y^2]$. For zero-mean input data, this becomes the first principal component. In order to obtain the first p principal components, one can have a network with p instars. Two learning laws for this feedforward network are given by [Sanger, 1989; Oja, 1989]

Sanger's rule:

$$\Delta w_{ij} = \eta y_i \left(x_j - \sum_{k=1}^i w_{kj} y_k \right), \quad i = 1, 2, \dots, p \quad (6.21)$$

Oja's p -unit rule:

$$\Delta w_{ij} = \eta y_i \left(x_j - \sum_{k=1}^p w_{kj} y_k \right), \quad i = 1, 2, \dots, p \quad (6.22)$$

In both the cases the weight vectors w_i converge to orthogonal unit vectors. With Sanger's rule, the weight vectors become exactly the

first p -principal component directions in order, whereas with Oja's p -unit rule, the p weight vectors converge to span the same **subspace** as the first p eigenvectors, but the weight vectors do not correspond to the eigenvectors themselves. Table 6.2 gives a **summary** of the learning algorithms for the **principal** component analysis.

Table 6.2 Summary of Learning Algorithms for Principal Component Analysis Networks

Consider a group of p **instars**, with all the units having linear output function. Let R be the covariance matrix of the input vectors. That is $R = \mathbf{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]$, $\boldsymbol{\mu} = \mathbf{E}[\mathbf{x}]$. Output of each unit

$$y_i(m) = \sum_{j=1}^M w_{ij}(m) x_j(m), \quad i = 1, 2, \dots, p$$

- Plain Hebbian learning for i th **instar**

$$\Delta w_{ij}(m) = \eta y_i(m) x_j(m)$$

Weight vector \mathbf{w}_i converges to the direction of the first principal component but is unbounded.

- Oja's rule for i th **instar**

$$\Delta w_{ij}(m) = \eta y_i(m) [x_j(m) - y_i(m) w_{ij}(m)]$$

Weight vector \mathbf{w}_i converges to the direction of the first principal component of R .

- Oja's p -unit rule for p **instars**

$$\Delta w_{ij}(m) = \eta y_i(m) \left[x_j(m) - \sum_{k=1}^p w_{kj}(m) y_k(m) \right],$$

$$i = 1, 2, \dots, p, \quad j = 1, 2, \dots, M,$$

Weight vectors converge to p orthogonal vectors which span the same space as the **first** p principal components of R .

- Sanger's rule or Generalized Hebbian **rule** for p instars

$$\Delta w_{ij}(m) = \eta y_i(m) \left[x_j(m) - \sum_{k=1}^i w_{kj}(m) y_k(m) \right],$$

$$i = 1, 2, \dots, p, \quad j = 1, 2, \dots, M$$

Weight vectors converge to the first p principal component directions of R .

6.2.4 Feedback Layer

As mentioned earlier, in the arrangement of a group of **instars**, the category of an input vector can be identified by observing the **instar** processing unit with maximum response. The maximum response unit can be determined by the system itself if the outputs of the **instar** processing units are fed back to each other.

If the processing units of a group of **instars** are connected as an on-centre off-surround feedback network, as shown in Figure 6.7, then

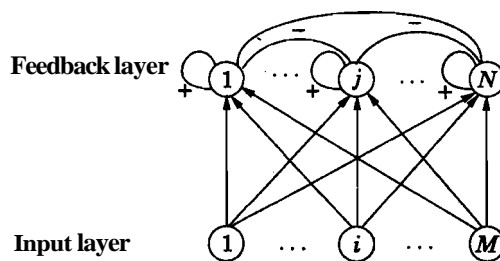


Figure 6.7 Arrangement of group of instars with output units connected in an on-centre off-surround manner.

the feedback layer is called a *competitive layer*. In this layer there is an excitatory self-feedback to each unit, and an inhibitory feedback from a given unit to all other units. The excitatory feedback is indicated by a positive (+) weight, and the inhibitory feedback is indicated by a negative (-) weight. Generally these weights are fixed a priori. The nature of the output function of the **instar** processing units will determine the behaviour expected **from** such a competitive layer. The pattern recognition tasks for different output functions will be discussed in the next section. Figure 6.8 shows the complete competitive learning network we have discussed so far in this section.

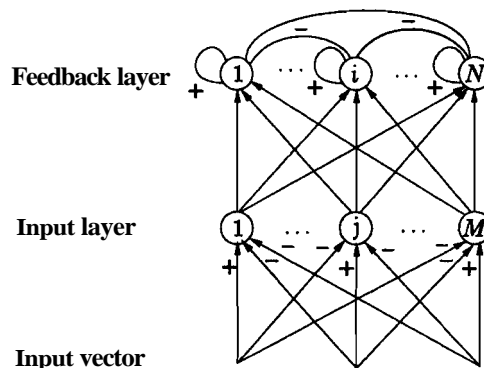


Figure 6.8 The complete competitive learning network structure.

6.3 Analysis of Feedback Layer for Different Output Functions

6.3.1 Pattern Storage (STM) Networks

Earlier in Chapter 5 we have discussed the pattern storage networks

which are fully connected feedback networks with symmetric weights. The pattern information is captured in the weights of the network, and the pattern storage is due to the existence of stable activation states in the feedback network. The pattern storage in such cases is permanent, in the sense that the patterns are available, and they can be recalled from the network as long as the weights are fixed. This type of storage can be viewed as long-term memory (LTM). In contrast, we have seen in Section 6.2.1 that in an on-centre off-surround feedforward network connection (Figure 6.2), the output disappears when the input is removed. The pattern is present in the activation values of the units, and the pattern disappears once the input is removed. Thus the availability of the pattern is purely *temporary* in such a case.

In this section we will study a pattern storage network where the pattern is present even if the input pattern is removed, as long as the network is not disturbed by another external input. Note that in this case the pattern is stored in the **activation values** of a feedback layer, and the activation values remain stable due to feedback. This type of storage is called short-term memory (STM). The existing pattern information disappears if there is an external input due to another pattern, since the new input changes the stable activation values to the ones corresponding to the new pattern. The three types of pattern storage networks are illustrated in Figure 6.9.

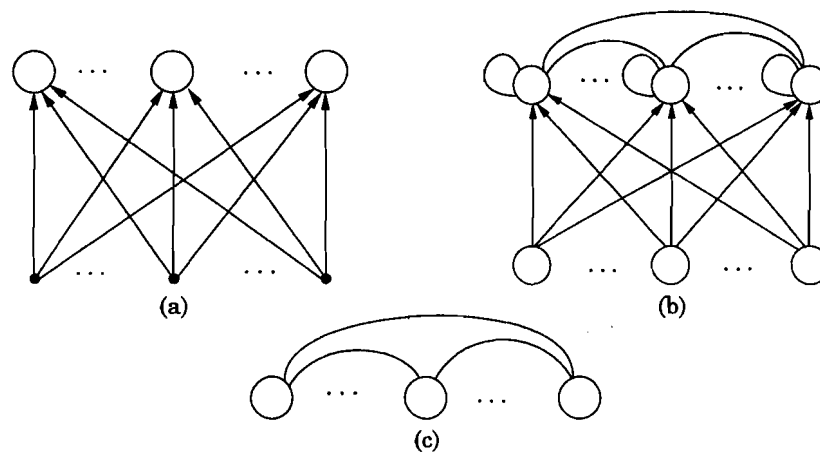


Figure 6.9 Different pattern storage networks: (a) Temporary storage, (b) Short-term memory and (c) Long-term memory.

6.3.2 Analysis with Linear Output Functions

To analyze the pattern storage network corresponding to the short-term memory, we will use the notation shown in Figure 6.10 for the feedback network. The input is fed in an on-centre off-surround type

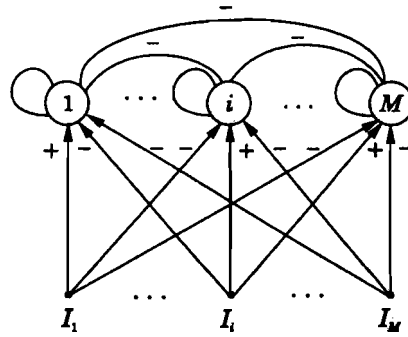


Figure 6.10 Pattern storage network with linear units in the feedback layer.

connection to provide normalized input to the units in the feedback layer. The units in the feedback layers **are** all linear. That is the output functions of the feedback units **are** given by $f(x) = x$.

The **analysis** presented in this section is adapted **from** [Freeman and Skapura, 1991]. The activation dynamics is described by the following shunting model within an operating range $[0, B]$ (See Eq. (2.20)):

$$\begin{aligned} \dot{x}_i &= -Ax_i + (B - x_i) [f(x_i) + I_i] - x_i \left[\sum_{k \neq i} f(x_k) + \sum_{j \neq i} I_j \right] \\ &= -Ax_i + B [f(x_i) + I_i] - x_i \left[\sum_k f(x_k) + \sum_j I_j \right] \end{aligned} \quad (6.23)$$

The passive decay term is given by $-Ax_i$ and the quantity $[f(x_i) + I_i]$ is the total excitatory input to the i th unit, and the quantity $[\sum_{k \neq i} f(x_k) + \sum_{j \neq i} I_j]$ is the total **inhibitory** input to the i th unit. Summing the expression for \dot{x}_i over i and using $x = \sum_{i=1}^M x_i$, we get

$$\dot{x} = -Ax + (B - x) \left[\sum_k f(x_k) + \sum_j I_j \right] \quad (6.24)$$

Let $x_i = x X_i$. Then

$$\dot{x}_i = \dot{x} X_i + x \dot{X}_i \quad (6.25)$$

Using the Eq. (6.23) to (6.25) we get

$$\begin{aligned} x \dot{X}_i &= \dot{x}_i - \dot{x} X_i \\ &= B [f(x_i) + I_i] - x_i \left[\sum_k f(x_k) + \sum_j I_j \right] - (B - x) X_i \left[\sum_k f(x_k) + \sum_j I_j \right] \\ &= B f(x_i) - x_i \sum_k f(x_k) - (B - x) X_i \sum_k f(x_k) + B I_i - x_i \sum_j I_j - (B - x) X_i \sum_j I_j \\ &= B f(x_i) - B X_i \sum_k f(x_k) + B I_i - B X_i \sum_j I_j \end{aligned}$$

Therefore,

$$\dot{x} \mathbf{X}_i = B \left[f(x \mathbf{X}_i) - \mathbf{X}_i \sum_k f(x \mathbf{X}_k) \right] + B I_i - B \mathbf{X}_i \sum_j I_j \quad (6.26)$$

Let $f(y) = y g(y)$. Substituting this for $f(y)$ in **Eq. (6.26)**, we get

$$\begin{aligned} \dot{x} \mathbf{X}_i &= B \left[x \mathbf{X}_i g(x \mathbf{X}_i) - \mathbf{X}_i \sum_k x \mathbf{X}_k g(x \mathbf{X}_k) \right] + B I_i - B \mathbf{X}_i \sum_j I_j \\ &= B x \mathbf{X}_i \sum_k \mathbf{X}_k [g(x \mathbf{X}_i) - g(x \mathbf{X}_k)] + B I_i - B \mathbf{X}_i \sum_j I_j \end{aligned} \quad (6.27)$$

since $\sum_k \mathbf{X}_k g(x \mathbf{X}_i) = g(x \mathbf{X}_i) \sum_k \mathbf{X}_k = g(x \mathbf{X}_i)$.

For a linear output function, i.e., $f(y) = y$, we get $g(y) = 1$. The first term in the **expression** for $\dot{x} \mathbf{X}_i$ will become zero. Therefore

$$\dot{x} \mathbf{X}_i = B I_i - B \mathbf{X}_i \sum_j I_j \quad (6.28)$$

In the steady state $\dot{\mathbf{X}}_i = 0$, and hence

$$\mathbf{X}_i = \frac{I_i}{\sum_j I_j} \quad (6.29)$$

Thus the steady normalized activation values of the units in the feedback layer correspond to the normalized input patterns. Figure 6.11 shows the performance of the feedback layer with linear processing units. Figure **6.11a** shows the linear output function and Figure **6.11b** shows the steady normalized activation values. If the **input** is removed, i.e., $I_i = 0$ for all i , then from **Eq. (6.28)** we get $\dot{x} \mathbf{X}_i = 0$. This shows that for nonzero x , $\mathbf{X}_i = 0$. That is $\mathbf{X}_i = x/x = \text{constant}$, and the activation values are determined by x alone.

When $I_i = 0$, for all i , we also get from **Eq. (6.24)**,

$$\begin{aligned} \dot{x} &= -A x + (B - x) x \\ &= (B - A - x) x \end{aligned} \quad (6.30)$$

In the steady state and for nonzero x , since $\dot{x} = 0$, we get

$$x = B - A = \text{constant} \quad (6.31)$$

Thus both x and \mathbf{X}_i are constant. That means x_i s are constant. That is, the steady state activation values will remain at some fixed, nonzero values even after removal of the input. The steady state values remain until they are disturbed by another set of input values. **Thus** the pattern stored can be viewed as short-term memory. Figure **6.11c** illustrates the performance of the feedback layer with linear processing units, when the input values are set to zero.

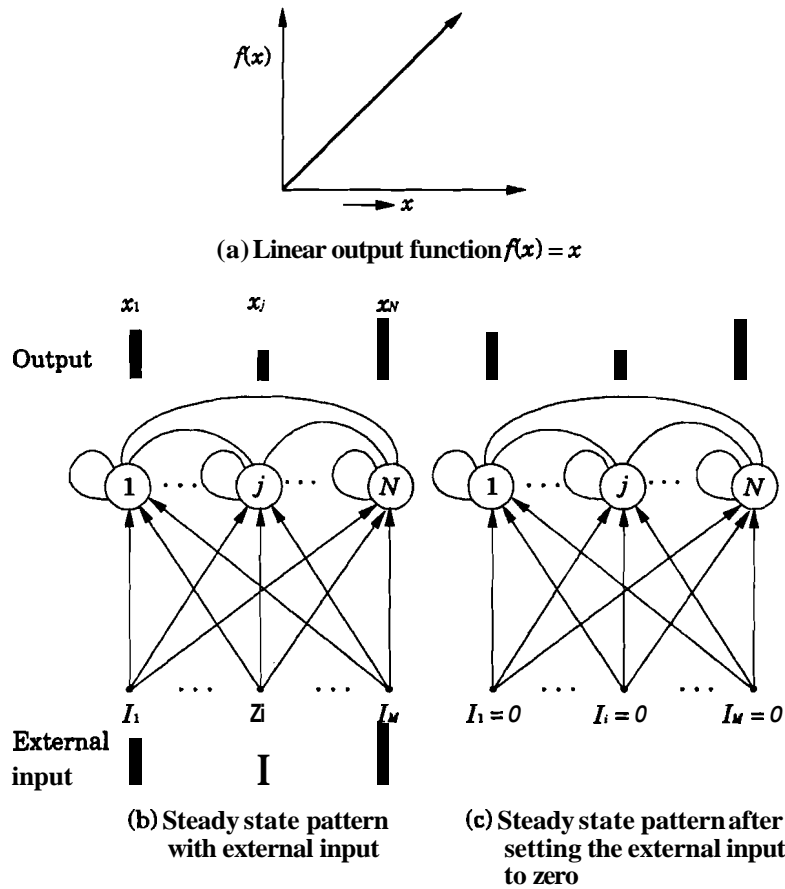


Figure 6.11 Performance of feedback layer with linear processing units [Adapted from Freeman and Skapura, 1991].

6.3.3 Analysis with Quadratic Output Functions

The analysis presented in this section is adapted from [Freeman and Skapura, 1991]. In this section we will discuss the behaviour of the competitive learning network of the type in Figure 6.10 with nonlinear processing units in the feedback layer. In particular, let us consider the quadratic output function $f(y) = y^2$. Then substituting for $g(y) = f(y)/y = y$ in the expression for $x X_i$ in Eq. (6.27), we get

$$\begin{aligned}
 x \dot{X}_i &= BxX_i \sum_k X_k (x X_i - x X_k) + BI_i - BX_i \sum_j I_j \\
 &= BxX_i \sum_k x X_k (X_i - X_k) + BI_i - BX_i \sum_j I_j
 \end{aligned} \tag{6.32}$$

If $X_i > X_k$ for $k \neq i$, then the first term is always positive, and hence

it becomes an excitatory term. In this case the network tries to enhance the activity X_i . If $X_i < X_k$, for $k \neq i$, then the **first** term is negative and hence it becomes an inhibitory term. In this case the network tries to suppress the activity X_i . Thus the largest X_i , say when $i = j$, will get enhanced in the steady state as shown in Figure 6.12b. For the intermediate cases, where $(X_i - X_k)$ is greater than zero in some cases and less than zero in some other cases, the steady state values will be in between the maximum and zero values. When the input is removed, i.e., $I_i = 0$ for all i , the steady state value of X_i is given by setting $\dot{x} X_i = 0$ in Eq. (6.32) and $\dot{x} = 0$ in Eq. (6.24). That is

$$\sum_k x X_k (X_i - X_k) = 0 \quad (6.33)$$

and

$$-Ax + (B - x) \sum_k (x X_k)^2 = 0 \quad (6.34)$$

Eq. (6.33) gives

$$X_i = \sum_k X_k^2 = X_i^2 + \sum_{k \neq i} X_k^2 \quad (6.35)$$

since $\sum_k X_k = 1$. From Eq. (6.35) the only nonzero solution for X_i is $X_i = 1$, and $X_k = 0$, for $k \neq i$.

Likewise from Eq. (6.34) above, we get

$$-Ax + (B - x) \sum_k x_k^2 = 0 \quad (6.36)$$

or,

$$x = \frac{B \sum_k x_k^2}{A + \sum_k x_k^2} \quad (6.37)$$

This shows that the total activation x is bounded by B , since A and B are positive.

The above analysis shows that when the input is zero, in the steady state only one of the units is activated to the maximum value of x_i , which is equal to the total activation value x , as shown in Figure 6.12c. The maximum value is bounded by B . Thus we may say that only one unit in the feedback layer wins the competition for a given input pattern, even after the pattern is removed. Therefore the feedback layer is called a competitive layer in this case. Note that the result is valid for any function of the type $f(x) = x^n$, $n > 1$. In all of these cases only one of the units will have maximum activation and all others will have zero activation values. This may be viewed as noise suppression, in the sense that all activation values lower than the maximum will be reduced to zero in the competition.

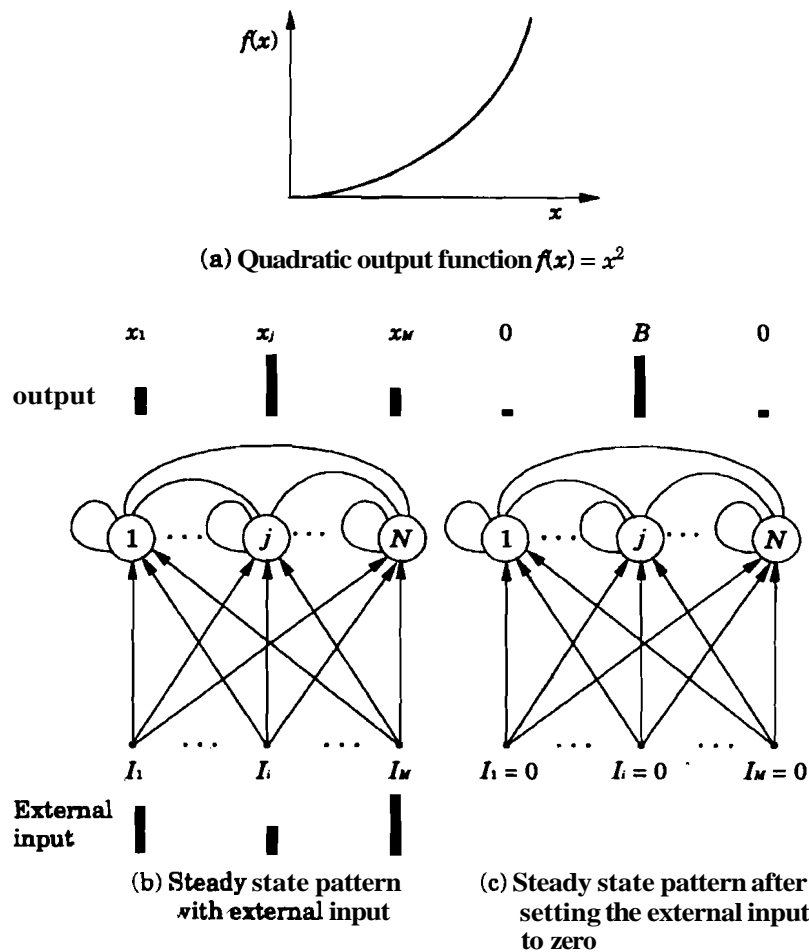


Figure 6.12 Performance of feedback layer with units having quadratic output functions [Adapted from Freeman and Skapura, 1991].

From the analysis in the previous section, we notice that pattern storage is achieved by using feedback units with linear output functions. Analysis in this section shows that noise suppression can be achieved by using quadratic output functions. Thus by using a quadratic output function for low activation and linear output function for high activation, both noise suppression and pattern storage (STM) tasks can be accomplished. In addition, if the output function increases at a rate less than linear for large activation, then the output is bounded all the time. The resulting output function is like a semilinear function as shown in Figure 6.13a. In such a case, in the steady state more than one of the units may have large activation values when the input is present (Figure 6.13b). Likewise, when the input is removed, in the steady state more than one unit

may reach maximum activation and all other units will have zero activation. This is illustrated in Figure 6.13c.

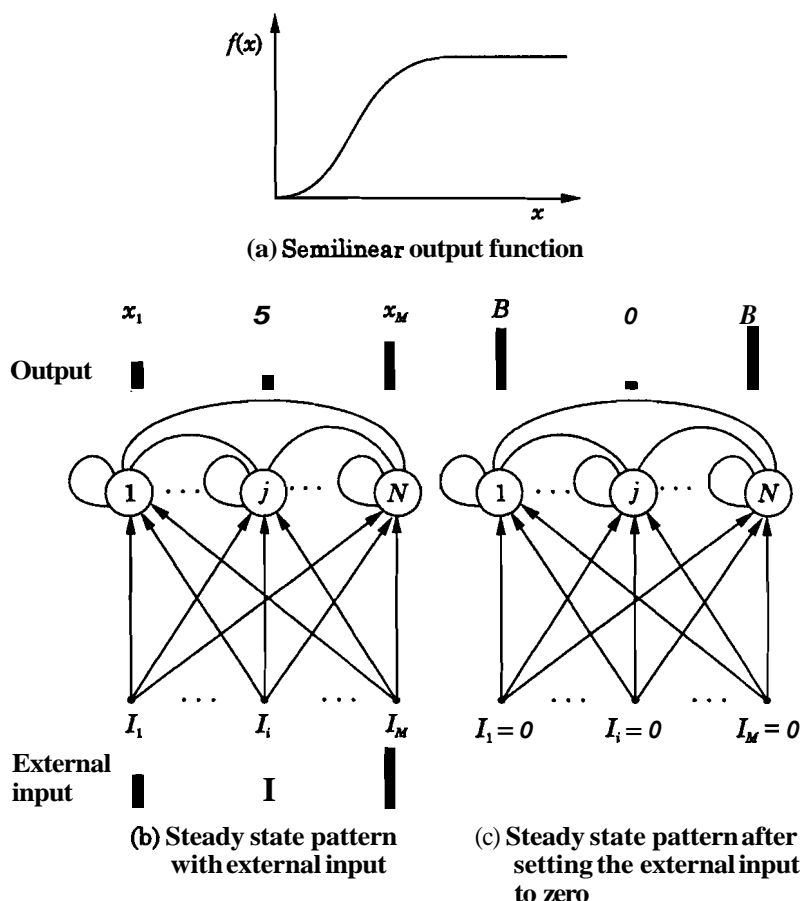


Figure 6.13 Performance of feedback layer with units having semilinear output functions.

6.4 Analysis of Pattern Clustering Networks

In the previous section we have seen that a competitive learning network with nonlinear output functions for units in the feedback layer can produce at equilibrium larger activation on a single unit and small activations on other units. This behaviour leads to a *winner-take all* situation, where, when the input pattern is removed, only one unit in the feedback layer will have nonzero activation. That unit may be designated as the winner for the input pattern. If the feedforward weights are suitably adjusted, each of the units in the feedback layer can be made to win for a group of similar input patterns. The corresponding learning is called *competitive learning*. The

units in the feedback layer have nonlinear $f(x) = x^n, n > 1$ output functions. Other nonlinear output functions such as hard-limiting threshold function or semilinear sigmoid function can also be used. These units are connected among themselves with **fixed** weights in an on-centre off-surround manner. Such networks are called competitive learning networks. Since they are used for clustering or grouping of input patterns, they can also be called pattern clustering networks.

In the pattern clustering task, the pattern classes are formed on unlabelled input data, and hence the corresponding learning is unsupervised. In the competitive learning the weights in the feedforward path are adjusted only **after** the winner unit in the feedback layer is identified for a given input pattern. There are three different methods of implementing the competitive learning as illustrated in Figure 6.14. In the figure, it is assumed that the input

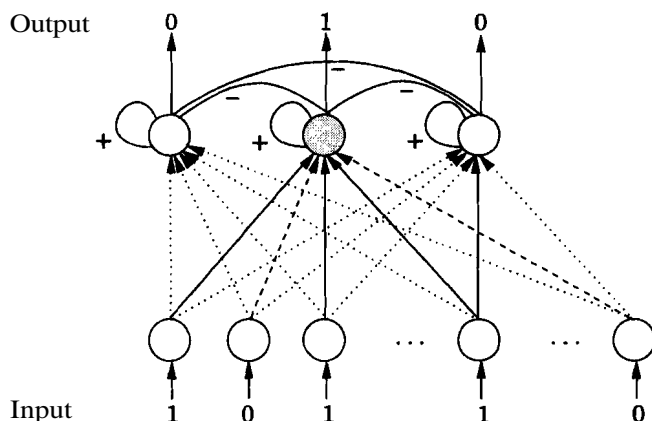


Figure 6.14 Illustration of basic competitive learning laws: (1) Minimal learning—only connections of type A (solid lines) are trained. (2) Malsburg learning—only connections of type A and B (dashed lines) are trained. (3) Leaky learning—connections of all the three types A, B and C (dotted lines) are trained [Adapted from Kung, 1993].

is a binary $\{0, 1\}$ vector. The activation of the i th unit in the feedback layer for an input vector $\mathbf{x} = (x_1, x_2, \dots, x_M)^T$ is given by $y_i = \sum_{j=1}^M w_{ij} x_j$, where w_{ij} is the (i, j) th element of the weight matrix W , connecting the j th input to the i th unit. Let $i = k$ be the unit in the feedback layer such that

$$y_k = \max_i (y_i) \tag{6.38}$$

then

$$\mathbf{w}_k^T \mathbf{x} \geq \mathbf{w}_i^T \mathbf{x}, \text{ for all } i \tag{6.39}$$

Assume that the weight vectors to all the units are normalized, i.e.,

$\|\mathbf{w}_i\| = 1$, for all i . Geometrically, the above result means that the input vector \mathbf{x} is closest to the weight vector \mathbf{w}_k among all \mathbf{w}_i . That is

$$\|\mathbf{x} - \mathbf{w}_k\| \leq \|\mathbf{x} - \mathbf{w}_i\|, \text{ for all } i \quad (6.40)$$

Start with some initial random values for the weights. The given set of input vectors are applied one by one in a random order. For each input the winner unit in the feedback layer is identified, and the weights leading to the unit are adjusted in such a way that the weight vector \mathbf{w}_k moves towards the input vector \mathbf{x} by a small amount, determined by a learning rate parameter η . Note that by doing this we are making that unit to respond more for that input. A straight forward implementation of the weight adjustment is to make

$$\Delta w_{kj} = \eta (x_j - w_{kj}), \quad (6.41)$$

so that

$$\begin{aligned} \mathbf{w}_k(m+1) &= \mathbf{w}_k(m) + \Delta \mathbf{w}_k(m) \\ &= \mathbf{w}_k(m) + \eta (\mathbf{x} - \mathbf{w}_k(m)) \end{aligned} \quad (6.42)$$

This looks more like Hebb's learning with a decay term if the output of the winning unit is assumed to be 1. It works best for prenormalized input vectors. This is called the standard competitive learning. Figure 6.15 shows the performance of the competitive

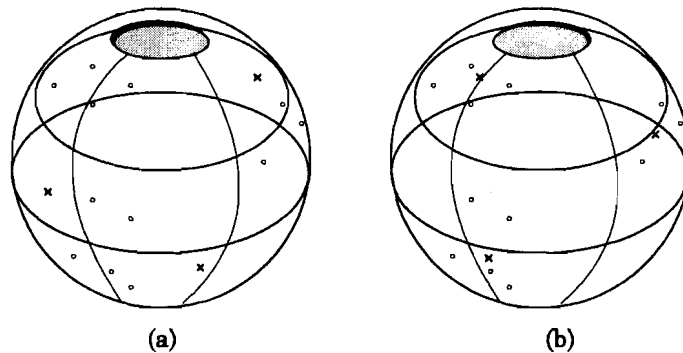


Figure 6.15 Illustration of competitive learning. The circles ('o') represent input vectors and crosses ('x') represent weight vectors: (a) before learning, and (b) after learning.

learning law for **normalized** input and weight **vectors** for a set of 3 dimensional input vector clusters. If the input vectors are not normalized, then they are normalized in the weight adjustment formula as follows:

$$\Delta w_{kj} = \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right], \text{ only for those } j \text{ for which } x_j = 1. \quad (6.43)$$

This can be called *minimal learning* [Kung, 1993]. In the case of binary input vectors, for the winner unit, only the weights which have nonzero input would be adjusted. That is

$$\begin{aligned}\Delta w_{kj} &= \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right], \text{ for } x_j = 1 \\ &= 0, \text{ for } x_j = 0\end{aligned}\quad (6.44)$$

Thus in this case only the connections which have both 1 at either end will be adjusted as shown in Figure 6.14 for the minimal learning case.

In the minimal learning there is no automatic normalization of weights after each adjustment. That is $\sum_{j=1}^M w_{kj} \neq 1$. In order to overcome this problem, Malsburg suggested the following learning law, in which all the weights leading to the winner unit will be adjusted [von der Malsburg, 1973]:

$$\Delta w_{kj} = \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right], \text{ for all } j \quad (6.45)$$

In this law if a unit wins the competition, then each of its input lines gives up some portion of its weights, and that weight is distributed equally among the active connections for which $x_j = 1$.

The unit i with an initial weight vector w_i far from any input vector, may never win the competition. Since a unit will never learn unless it wins the competition, another method called *leaky learning* law is proposed [Rumelhart and Zipser, 1985]. In this case, the weights leading to the units which do not win also are adjusted for each update as follows:

$$\begin{aligned}\Delta w_{ij} &= \eta_w \left[\frac{x_j}{\sum_m x_m} - w_{ij} \right], \text{ for all } j, \\ &\text{if } i \text{ wins the competition, i.e., } i = k \\ &= \eta_l \left[\frac{x_j}{\sum_m x_m} - w_{ij} \right], \text{ for all } j, \\ &\text{if } i \text{ loses the competition, i.e., } i \neq k\end{aligned}\quad (6.46)$$

where η_w and η_l are the learning rate parameters for the winning and losing units, respectively ($\eta_w \gg \eta_l$). In this case the weights of the losing units are also slightly moved for each presentation of an input.

Basic competitive learning and its variations are used for adaptive vector quantization [Nasrabadi and King, 1988], in which the input vectors are grouped based on the Euclidean distance between vectors. Both **unsupervised vector quantization (VQ)** and **supervised** or learning vector quantization (**LVQ**) algorithms are available [Kohonen, 1988; Kohonen, 1989]. The basic learning laws for competitive learning and vector quantization are summarized in Table 6.3.

Table 6.3 Summary of Competitive Learning Methods

Basic competitive learning

Assume prenormalized input vectors and weight vectors normalization.

Let \mathbf{x} be the input vector and \mathbf{w}_i be the weight vector for i th unit in the competitive layer. Thus if $\mathbf{w}_k^T \mathbf{x} \geq \mathbf{w}_i^T \mathbf{x}$, for all i , then k is the winning unit.

$$\Delta w_{kj} = \eta (x_j - w_{kj})$$

Minimal learning (for binary input vectors)

$$\Delta w_{kj} = \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right], \text{ only for those } j \text{ for which } x_j = 1.$$

$$= 0, \text{ for } x_j = 0$$

Malsburg learning (for binary input vectors)

$$\Delta w_{kj} = \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right], \text{ for all } j$$

Leaky learning (for binary input vectors)

$$\Delta w_{ij} = \eta_w \left[\frac{x_j}{\sum_m x_m} - w_{ij} \right], \text{ for all } j$$

if i wins the competition, i.e., $i = k$

$$= \eta_l \left[\frac{x_j}{\sum_m x_m} - w_{ij} \right], \text{ for all } j$$

if i loses the competition, i.e., $i \neq k$

Vector Quantization (unsupervised)

For input vector \mathbf{x} and weight vector for the i th unit \mathbf{w}_i , if $|\mathbf{x} - \mathbf{w}_k| \leq |\mathbf{x} - \mathbf{w}_i|$, for all i , then k is the winning unit. The vector quantization learning law is given by

$$\Delta \mathbf{w}_k = \eta (\mathbf{x} - \mathbf{w}_k)$$

Analysis of Feature Mapping Network

Table 6.3 (Cont.)

Learning Vector Quantization (LVQ1) (supervised)

If the class of the input vector \mathbf{x} is given, then

$$\begin{aligned}\Delta \mathbf{w}_k &= \eta (\mathbf{x} - \mathbf{w}_k), & \text{if the winning class } k \text{ is correct} \\ &= -\eta (\mathbf{x} - \mathbf{w}_k), & \text{if the winning class } k \text{ is incorrect}\end{aligned}$$

Learning Vector Quantization (LVQ2) (supervised)

If the input vector \mathbf{x} is misclassified by the winning unit k , and if the nearest-neighbour unit i has the correct class, then

$$\begin{aligned}\Delta \mathbf{w}_k &= -\eta (\mathbf{x} - \mathbf{w}_k), & \text{for incorrect winning unit} \\ \Delta \mathbf{w}_i &= \eta (\mathbf{x} - \mathbf{w}_i), & \text{for correct neighbouring unit}\end{aligned}$$

If the units in the feedback layer are arranged in a geometrical fashion, like a 1-D or a 2-D array, then the update of weights could be confined to the neighbouring losing units only. This is called *Kohonen's learning* or *feature mapping* which will be discussed in the next section.

6.5 Analysis of Feature Mapping Network

In the pattern clustering network using competitive learning, only one unit in the feedback layer is made to win by appropriate choice of the connections of the units in the feedback layer. The number of units corresponds to the number of possible clusters into which the set of input pattern vectors are likely to form. Each unit is identified with one cluster or a group. The units otherwise have nothing in common among themselves. Even the physical location of a unit relative to the other units in the output layer has no significance.

On the other hand, there are many situations where it is difficult to group the input patterns into distinct groups. The patterns may form a continuum in some feature space, and it is this kind of information that may be needed in some applications. For example, it may be of interest to know how close a given input is to some of the other patterns for which the **feedforward** path has already been trained. In other words, it is of interest to have some order in the activation of a unit in the feedback layer in relation to the activations of its **neighbouring** units. This is called *feature mapping*. The network that achieves this is called *feature mapping network*.

A feature mapping network is also a competitive learning network with nonlinear output **functions** for **units** in the feedback layer, as in the networks used for pattern clustering. But the main **distinction** is in the geometrical arrangement of the output units, and the significance attached to the **neighbouring** units during training.

During recall of information the activations of the **neighbouring** units in the output feedback layer suggest that the input patterns corresponding to these units **are** similar. Thus feature mapping could also be viewed as topology preserving map **from** the space of possible input values to a line or a plane of the output **units** [Kohonen, 1982b; Kohonen, 1989].

The inputs to a feature mapping network could be N-dimensional patterns (See Figure 6.16a for a 3-D input), applied one at a time, and the network is to be trained to map the similarities in the set of input patterns. Another type of input is shown in Figure 6.16b, where

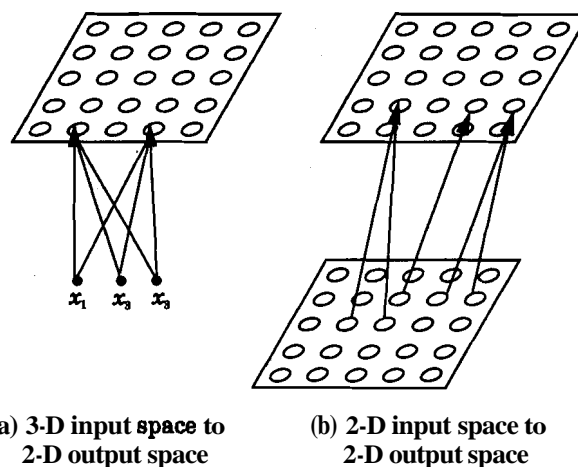


Figure 6.16 Feature mapping networks where the layers are fully connected although only a few connections are shown.

the **inputs** are arranged in a 2-D array so that the array represents the input pattern space **as** in the case of a textured image. At any given time only a few of the input units may be turned on. That is, only the corresponding links are activated. The objective is to capture the features in the space of input patterns, and the connections are like **softwiring** dictated by the unsupervised learning mode in which the network is expected to work. This second type is more common in topological mappings in the brain [Hertz et al, 1991, p. 233].

There are several ways of implementing the feature mapping process. In one method the output layer is organized into predefined receptive fields, and the unsupervised learning should perform the feature mapping by activating appropriate connections. This can also be viewed as orientational selectivity [Hubel and Weisel, 1962; Linsker, 1986; Linsker, 1988]. Another method is to modify the feedback connections in the output layer of Figure 6.16a. Instead of connecting them in an **on-centre** off-surround manner, the connections can be made as indicated by a Mexican hat type function, a **1-D** version of which is shown in Figure 6.17. The function gives the

Analysis of Feature Mapping Network

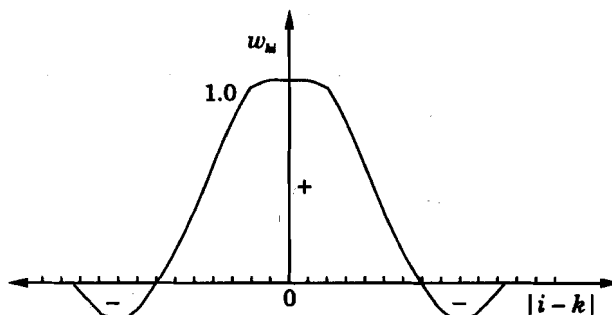


Figure 6.17 A Mexican hat function in 1 dimension.

lateral connection weight value for a **neighbouring** unit k at a distance $|i - k|$ from the current unit i . The unit i and its immediate **neigh-** bours are connected in **an** excitatory (**+ve** weights) manner. The unit i is connected in **an** inhibitory (**-ve** weights) manner to far off units.

A third method of implementing the feature mapping process is to use **an** architecture of a competitive learning network with **on-** centre **off-surround** type of connections among units, but at each stage the weights **are** updated not only for the winning unit, but also for the units in its neighbourhood. The neighbourhood region may be progressively reduced during learning. This is called self-organization network with Kohonen's learning [Kohonen, 1982a]. Figure 6.18 shows an example of a feature mapping using a self-organization network. It shows a 2-D input vector and a feedback layer with units **arranged** as a 2-D grid. The input and the output layers **are** fully connected.

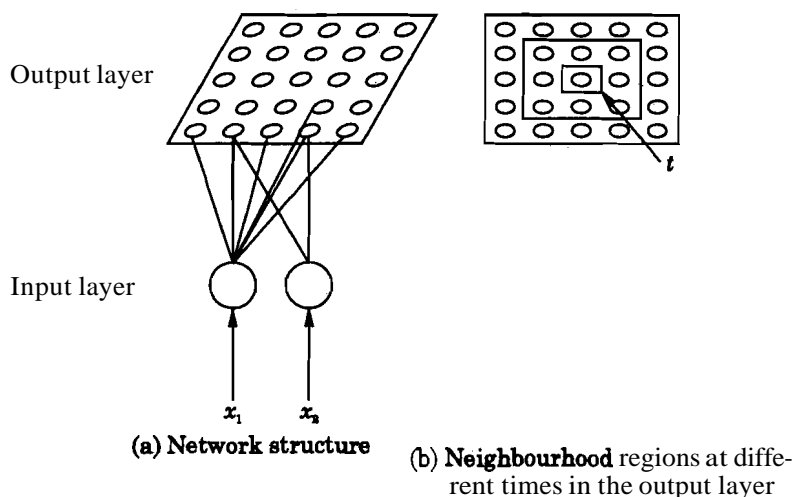


Figure 6.18 Illustration of a self-organizing network. In the network structure in (a) the input units are **connected** to all the units in the output layer, although only a few connections are shown. In the output layer all the units are connected to each other, although the connections are not shown in the **figure**.

The self-organization network is trained as follows: The weights are set to random initial values. When an input vector \mathbf{x} is applied, the winning unit k in the output layer is identified such that

$$\|\mathbf{x} - \mathbf{w}_k\| \leq \|\mathbf{x} - \mathbf{w}_i\|, \text{ for all } i \quad (6.47)$$

The weights associated with the winning unit k and its **neighbouring** units m , identified by a neighbourhood function $\lambda(k, m)$, are updated using the expression

$$\Delta \mathbf{w}_m = \eta \lambda(k, m) (\mathbf{x} - \mathbf{w}_m) \quad (6.48)$$

The neighbourhood function $\lambda(k, m)$ is maximum for $m = k$. A suitable choice for $\lambda(k, m)$ is a Gaussian function of the type

$$\lambda(k, m) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\|\mathbf{r}_k - \mathbf{r}_m\|^2/2\sigma^2) \quad (6.49)$$

where \mathbf{r}_k refers to **the** position of the k th unit in the 2-D plane of the output units. The width of the Gaussian function, described by σ , is gradually decreased to reduce the neighbourhood region in successive iterations of the training process. Even the learning rate parameter η can be changed **as** a function of time during the training phase. The weights are **renormalized** each time **after** the update. Table 6.4 gives an algorithm for **implementing** the self-organizing feature map learning.

Table 6.4 An Algorithm for Self-organizing Feature Map Learning

-
1. Initialize the weights **from** M inputs to the N output units to small random values. Initialize the size of the neighbourhood region $\mathcal{R}(0)$.
 2. Present a new input \mathbf{a} .
 3. Compute the distance d_i between the input and the weight on each output unit i :

$$d_i = \sum_{j=1}^M [a_j(t) - w_{ij}(t)]^2, \text{ for } i = 1, 2, \dots, N$$

where $a_j(t)$ is the input to the j th input unit at time t and $w_{ij}(t)$ is the weight from the j th input unit to the i th output unit.

4. Select the output unit k with minimum distance

$$k = \text{index of } \left[\min_i (d_i) \right]$$

5. Update weight to node k and its **neighbours**

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) (a_j(t) - w_{ij}(t))$$

for $i \in \mathcal{R}_k(t)$ and $j = 1, 2, \dots, M$, where $\eta(t)$ is the learning rate parameter ($0 < \eta(t) < 1$) that decreases with time. $\mathcal{R}_k(t)$ gives the neighbourhood region around the node k at time t .

6. Repeat Steps 2 through 5 for all inputs several times.
-

Figure 6.19 illustrates the result of Kohonen's learning for feature mapping. In the figure the mapping is from a 2-D input to a 2-D

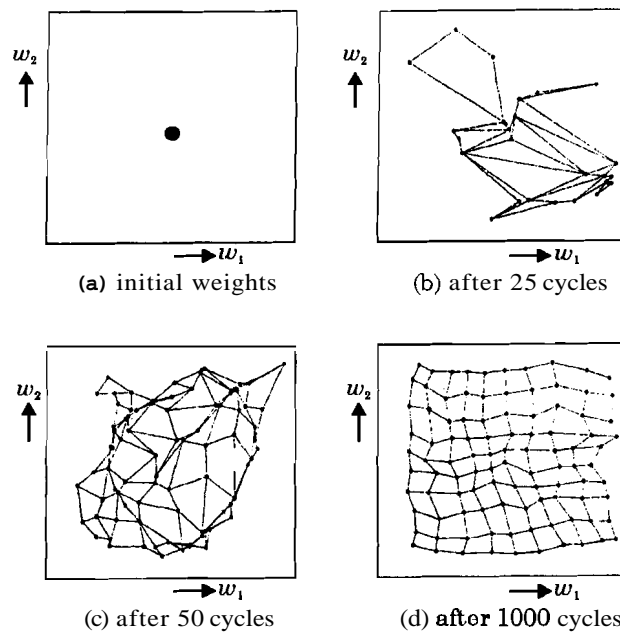


Figure 6.19 An illustration of Kohonen's learning for self-organizing networks.

feature space. The input consists of a point chosen at random from the input space defined by the region of interest, a square in this case. The feature map is displayed by the weight values of the connections leading to each unit in the output layer **after** training. The weights for each unit is a point in the (w_1, w_2) space, and the points corresponding to adjacent units in the output layer are joined in the figure. Initially the weights are set to some random values around the mean. As training proceeds, the weights are **modified** to span the space in the (w_1, w_2) plane. The shape of the spanned space is dictated by the shape of the region (a unit square in this case) from which the input values are selected at random. Figure 6.20 illustrates the feature mapping from a 2-D input space to a 1-D layer of output units. The space filling characteristic of the feature mapping can be seen from the figure.

Finally, Figure 6.21 illustrates the feature mapping from a 1-D input space to a 1-D layer of output units. Here the input data is uniformly distributed random numbers in the interval $[0, 1]$. The initial weights for all the units in the 1-D output layer are shown in Figure 6.21a. The feature mapping in Figure 6.21c shows that the weights are organized along a line. The line could have a positive or a negative slope depending on how the feature mapping evolves.

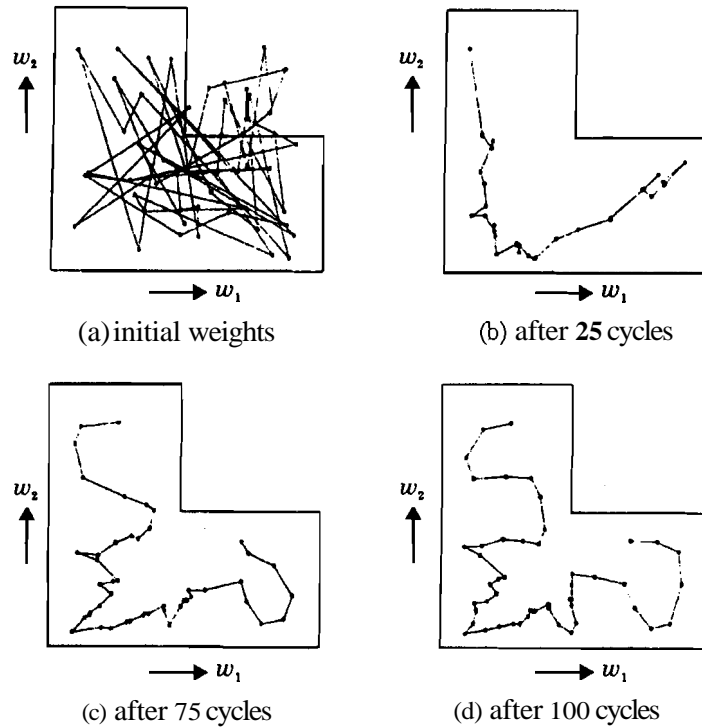


Figure 6.20 An illustration of Kohonen's learning for feature mapping from 2-D input to 1-D feature mapping.

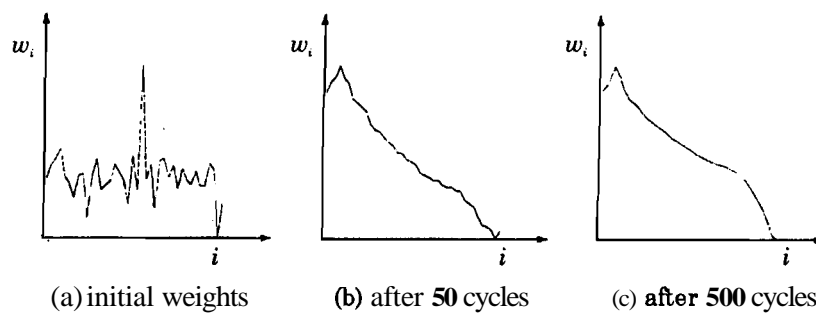


Figure 6.21 An illustration of Kohonen's feature mapping from 1-D input to 1-D feature mapping.

6.6 Summary

In this chapter simple networks to perform the task of **pattern** clustering have been analyzed in detail. The components of a competitive **learning** network were discussed individually. In **particular**, the principal component analysis feature of the **instar** layer with linear output units is useful for feature extraction and data compression.

A competitive learning network consists of a feedforward part and a feedback part. Use of nonlinear units in the feedback layer of the competitive learning network leads to the concept of pattern clustering. Different choices of competitive learning are available for adjusting the weights of the feedforward part of the network. Basic competitive learning and its variations are useful for adaptive vector quantization. Supervised form of vector quantization, called Learning Vector Quantization, was also described briefly.

The competitive learning networks can be modified to perform the task of feature mapping. **Kohonen's** self-organization learning law enables us to interpret the outputs of the units in the second layer as features in the input data. Illustrations of feature mapping from 2-D data to 2-D space, 2-D data to 1-D space and 1-D data to 1-D space were given. These illustrations demonstrate the significance of SOM feature of the competitive learning networks. Combination of SOM and classification networks have been used effectively for pattern classification applications in speech and images [Huang et al, 1992; Raghur et al, 1995].

Review Questions

1. What **are** the components of a competitive learning network?
2. Describe the operation of an input layer when it is directly connected to the environment.
3. What is an **instar** network?
4. Describe the basic learning feature of an **instar** and discuss its application.
5. What is the main difference between an **instar** network with competitive learning and an **instar** network with Hebbian learning?
6. Explain how pattern clustering can be achieved by a **group** of **instars** with binary output functions.
7. What are principal components of an autocorrelation matrix?
8. Explain the distinction between eigenvectors of autocorrelation and covariance matrices.
9. What is the **Oja's** learning law for a single **instar**? How is it different from plain Hebbian learning?
10. Explain the difference between **Sanger's** rule and **Oja's** p-unit rule.
11. What is meant by on-centre **off-surround** feedback network?
12. Give a diagram of the complete competitive learning network.
13. Distinguish among temporary storage, short-time memory and long-term memory.
14. Explain how a competitive learning network with linear units performs a short-term memory task.

15. Explain the noise suppression property of the quadratic output function in the feedback layer of a competitive learning network.
16. Discuss the performance of a competitive learning network for **semilinear** output function in the feedback layer.
17. What is a pattern clustering network?
18. What are the three basic competitive learning laws?
19. What is adaptive vector quantization? What is learning vector quantization?
20. Explain the difference between pattern clustering and feature mapping.
21. Explain the three different methods of implementing the feature mapping process.
22. What is a self-organization network?
23. What are the salient features of the Kohonen's self-organizing learning algorithm?
24. Illustrate the concept of feature mapping with the help of an example of mapping 2-D input onto a 2-D feature space.
25. Explain the feature mapping of 2-D input onto **1-D** feature space.
26. Explain the build up of the **1-D** feature map of **1-D** input values selected at random from an interval **0** to **1**.

Problems

1. Show how a single unit **instar** with Hebbian learning results in a weight vector that will increase without bound. (See [Hertz et al, 1991, p. 200; Hassoun, 1995, p. 901])
2. Show that for the competitive network of Figure 6.12, in the steady state when the inputs are removed, only one of the units will have maximum output value for any output function $f(x) = x^n$, $n \geq 1$. Also show that the activation value is bounded. (See [Freeman and Skapura, 1991])
3. Show that the weight vector in the **Oja's** rule aligns with the eigenvector of the autocorrelation matrix corresponding to the largest eigenvalue. (See [Hertz et al, 1991, pp. 202-204; Haykin, 1994, p. 374])
4. Find the **eigenvector** corresponding to the first principal component of the following correlation matrix of a stochastic process (See Appendix E and [Kung, 1993, p. 305]):

$$R = \begin{bmatrix} 5 & 3 & 0 \\ 3 & 5 & 3 \\ 0 & 3 & 5 \end{bmatrix}$$

5. The following is the cross-correlation matrix of two stochastic processes (See [Kung, 1993, p. 3061]):

$$R_{xy} = \begin{bmatrix} 5 & 3 & 0 & 0 \\ 2 & 4 & 2 & 0 \\ 0 & 2 & 5 & 1 \end{bmatrix}$$

Find the **left** and the right **eigenvectors** corresponding to the **first** asymmetric principal component.

6. For feature extraction Linsker used a modified Hebbian rule for a single unit given by (See [Hertz et al, 1991, p. 211; Hassoun, 1995, pp. 95-97])

$$\Delta w_i = \eta (a_i x + \alpha a_i + \beta x + \gamma)$$

where

$$x = \sum_{j=1}^M w_j a_j + \theta$$

and α , β and γ are the parameters that can be tuned to produce the desired **behaviour**. Assuming that all the input components a_i have the same mean \bar{a} , so that $a_i = \bar{a} + \epsilon_i$, Show that

$$\langle \Delta w_i \rangle = \eta \left[\sum_j C_{ij} w_j + \lambda \left(\mu - \sum_j w_j \right) \right]$$

where λ and μ are some combination of the constants θ , α , β , γ and \bar{a} , and $C_{ij} = \langle \epsilon_i \epsilon_j \rangle$.

7. Show that the **Linsker's** rule can be defined by computing average of the gradient descent learning $\Delta w_i = -\eta \partial E / \partial w_i$ on the cost function

$$E = -\frac{1}{2} \mathbf{w}^T \mathbf{C} \mathbf{w} + \frac{\lambda}{2} \left(\mu - \sum_j w_j \right)^2$$

where $\mathbf{C} = [C_{ij}]$. **Linsker's** rule **tries** to maximize the output variance **subjected** to the constraint that $\sum_j w_j = \mu$.

8. Show that Linsker rule is unstable, i.e., $\langle \Delta w_i \rangle$ does not tend to zero unless the weights are **subjected** to the boundary constraint $w_- \leq w_i \leq w_+$. Note that in contrast, the **Oja's** rule maximizes the output variance subjected to $\sum_j w_j^2 = 1$ and hence does not need a boundary constraint on the weights. (See [Hertz et al, 1991, p.213; Hassoun, 1995, p. 97; Haykin, 1994, pp.357-362])

9. Substituting $y_i = \sum_{j=1}^M w_{ij} a_j$ in the **Sanger's** rule

$$\Delta w_{ij} = \eta y_i \left[a_j - \sum_{k=1}^i y_k w_{kj} \right]$$

and taking the average, show that

$$\langle \Delta \mathbf{w}_i \rangle / \eta = \mathbf{R} \mathbf{w}_i - \sum_{k=1}^{i-1} [\mathbf{w}_k^T \mathbf{R} \mathbf{w}_i] \mathbf{w}_k - [\mathbf{w}_i^T \mathbf{R} \mathbf{w}_i] \mathbf{w}_i$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \dots, w_{iM}]^T$, and $R = [\langle \mathbf{a}_i \mathbf{a}_i \rangle]$. Also show by induction that it leads to i th **eigenvalue** of R (see [Hertz et al, 1991, p. 2081]).

10. Nonlinear PCA neural networks can capture higher order statistics of the input data. One way of introducing nonlinearity into a **PCANN** is by using higher order units whose output is given by [Hassoun, 1995, p.101]

$$y = \sum_{i=1}^M \left(w_i + \sum_{j=1}^M w_{ij} x_j \right) x_i$$

Show that this can be interpreted as the output of a linear unit **with** the input vector as

$$\mathbf{z} = [x_1, x_2, \dots, x_M, x_1^2, x_1x_2, \dots, x_1x_M, x_2^2, x_2x_3, \dots, x_M^2]^T$$

and the weight vector as

$$\mathbf{w} = [w_1, w_2, \dots, w_M, w_{11}, w_{12}, \dots, w_{22}, w_{23}, \dots, w_{MM}]^T.$$

That is, $y = \mathbf{w}^T \mathbf{z}$

Also show that the principal component resulting from this network corresponds to the principal eigenvector of the matrix $\langle \mathbf{z} \mathbf{z}^T \rangle$.

11. Show that the Malsburg learning Eq. (6.45) for competition layer results in $\sum_{j=1}^M w_{ij} = 1$, if initially the weights are normalized.

12. Generate a set of random points in 2-dimensional space using four Gaussian distributions with the following means and variances.

Mean:	$[-5 \ -5]^T$	$[-5 \ 5]^T$	$[5 \ -2]^T$	$[5 \ 5]^T$
Variance:	4	3	2	2

Starting with four random initial weight vectors, determine the cluster centres formed by using the competitive learning law given in Eq. (6.42). Study the effect of different values of the learning rate parameter.

13. Determine the cluster centres for the data in the Problem 12 above using the **LVQ1** algorithm given in Table 6.3.

14. Determine the self-organizing map generated by points selected at random from an annular ring formed by two concentric circles. Consider the following two cases:

- (a) The units in the SOM are arranged in a 2-dimensional plane.
 (b) The units in the SOM are **arranged** in a 1-D layer.

Chapter 7

Architectures for Complex Pattern Recognition Tasks

7.1 Introduction

In the previous chapters the principles of artificial neural networks and some basic **functional** units were presented. These **functional** units are basic structures of **neural** networks capable of performing simple pattern recognition tasks. In practice the pattern recognition tasks are much more complex, and each task may require evolving a new structure based on the principles discussed in the previous chapters. In fact, designing an architecture for a given task involves developing a suitable structure of the **neural** network and defining appropriate activation and synaptic dynamics.

The pattern **recognition** tasks **performed** by **human** beings are several orders of magnitude more complex than the simple tasks like pattern association, classification, storage and clustering discussed earlier. For example, the associative memory function of the biological neural network is highly sophisticated in **terms of its** ability to perform the learning, storing and recall operations. Likewise, the abilities of the biological network in dealing with pattern variability as in the hand-written characters, or **with** temporal pattern recognition as in speech and image sequence are at present impossible to realize by an **artificial** system. However, these features of the biological system motivate people to develop new architectures of artificial neural networks.

While the urge is to develop an architecture to solve a real world problem, such as involving pattern variability, the structure of the network is still based on the well understood principles (which **are** very few) of models of neurons, connections and the network dynamics. In **all** these cases the real world problems **are** simplified or tailored to satisfy the constraints of the architecture, rather than developing suitable **architectures** for the problems. Thus an architecture is restricted to a class of simplified problems or to a specific problem, but not universal.

One way to organize the networks at architectural level is as proposed by **Simpson** [1990]. They are organized along the broad

categories of learning (supervised and unsupervised) and along the broad categories of **structures** (feedforward and feedback). In supervised learning the weight adjustment at each step is based on the given input and the desired output. The adjustment may be of correlation type, perceptron learning, delta learning, reinforcement learning, etc. In supervised learning, the weights of the network are determined either by learning or by computation from the given input patterns. In feedforward structures the pattern recall is a straightforward application of the computations involved for each unit once. But in feedback structures the pattern recall involves several cycles of computations, each cycle consisting of computations by all the processing units on the average. The cycles are repeated until an equilibrium state is reached. The architectures in each category are described in a common format consisting of description of the task, description of the topology of the network, the encoding scheme (i.e., determination of weights), the decoding scheme (i.e., recall of pattern information), stability, performance of the network in terms of capacity and some applications of the architecture [Simpson, 1990].

We adopt a different approach in this chapter. We consider a few issues in pattern recognition tasks and discuss evolution of architectures for addressing these issues. This chapter presents five different classes of architecture, to address five different classes of complex pattern recognition tasks. While these architectures may not solve the real world problems completely, their descriptions do help in understanding the issues better and also in developing new architectures, once the issues for new classes of problems are clear.

Table 7.1 gives the organization of topics for this chapter. We consider associative memories in Section 7.2, where we discuss bidirectional associative memory in some detail. Pattern mapping architectures are considered in Section 7.3. In particular, we discuss the radial basis function networks for pattern classification and function approximation problems. We also consider the **counter-propagation** network which can capture both forward mapping as well as inverse mapping (if it exists) between a pair of patterns. In Section 7.4 the issue of stability-plasticity dilemma is addressed using the class of Adaptive Resonance Theory (ART) models. Architectures for temporal pattern recognition and generation are described in Section 7.5. In particular, we discuss the Avalanche architecture and Time Delay Neural Networks (TDNN) for recognition of sequences of patterns. The issue of pattern variability is discussed in Section 7.5 through the neocognitron architecture. While the pattern recognition issues of memory, mapping, stability-plasticity, temporal patterns and pattern variability are easily handled by human beings, the developments of architectures in this chapter clearly bring out the advantages and limitations of ANN models to deal with these issues.

Table 7.1 Organization of Neural Network Architectures based on Pattern Recognition Tasks

Associative memories

- Linear associative memories (Hetero and Autoassociative)
- Autoassociative memories (Hopfield network and Boltzmann machine)
- Bidirectional associative memories
- Multidirectional associative **memories**
- Temporal associative memories

Pattern mapping networks

- Multilayer **feedforward networks**
- Radial basis function networks for
 - (a) Classification
 - (b) Mapping or function approximation
- Generalized regression neural networks
- Probabilistic neural networks
- Counterpropagation network

Pattern classification: Stability-plasticity dilemma

- Adaptive Resonance Theory (ART)
 - ART1, ART2 and ART3
- **ARTMAP**
- Fuzzy **ARTMAP**

Temporal patterns

- Avalanche
- **Kohonen's** phonetic typewriter
- Associative memory based network
- Partially recurrent network
- Fully recurrent network
- Backpropagation through time
- Real-time recurrent learning network

Pattern variability

- Neocognitron
-

7.2 Associative Memory

Pattern storage is an obvious pattern recognition task that one would like to realize using an artificial neural network. This is a memory function, where the network is expected to store the pattern information (not data) for later recall. The patterns to be stored may be of spatial type or spatio-temporal (pattern sequence) type. Typically, an artificial **neural** network behaves like an associative memory, in which a pattern is associated with another pattern, or with itself. This is in contrast with the random access memory which maps an

address to a data. An artificial neural network can also function as a content addressable memory where data is mapped onto an address.

The pattern information is stored in the weight matrix of a feedback neural network. The stable states of the network represent the stored patterns, which can be recalled by providing an external stimulus in the form of partial input. If the weight matrix stores the given patterns, then the network becomes an autoassociative memory. If the weight matrix stores the association between a pair of patterns, the network becomes a bidirectional associative memory. This is called heteroassociation between the two patterns. If the weight matrix stores multiple associations among several (> 2) patterns, then the network becomes a multidirectional associative memory. If the weights store the associations between adjacent pairs of patterns in a sequence of patterns, then the network is called a temporal associative memory.

Some desirable characteristics of associative memories are: (a) The network should have a large capacity, *i.e.*, ability to store a large number of patterns or pattern associations. (b) The network should be fault tolerant in the sense that damage to a few units or connections should not **affect** the performance in recall significantly. (c) The network should be able to recall the stored pattern or the desired associated pattern even if the input pattern is distorted or noisy. (d) The network performance as an associative memory should degrade only gracefully due to damage to some units or connections, or due to noise or distortion in the input. (e) Finally, the network should be flexible to accommodate new patterns or associations (within the limits of its capacity) and to be able to eliminate unnecessary patterns or associations.

Linear associative memory and autoassociative memory were discussed in detail in Chapter 5. In this section the discrete Bidirectional Associative Memory (**BAM**) is discussed in some detail. Extensions of the BAM concepts to multidirectional and temporal associative memories are discussed briefly.

7.2.1 Bidirectional Associative Memory (BAM)

The objective is to store a set of pattern pairs in such a way that any stored pattern pair can be recalled by giving either of the patterns as input. The network is a two-layer heteroassociative neural network (Figure 7.1) that **encodes** binary or bipolar pattern pairs (\mathbf{a}_l , \mathbf{b}_l) using the Hebbian learning. It can learn on-line and it operates in discrete time steps. The **BAM** weight matrix **from** the first layer to the second layer is given by

$$\mathbf{W} = \sum_{l=1}^L \mathbf{a}_l \mathbf{b}_l^T \quad (7.1)$$

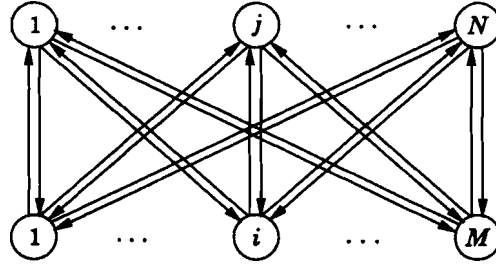


Figure 7.1 Bidirectional associative memory.

where $\mathbf{a}_i \in \{-1, +1\}^M$ and $\mathbf{b}_i \in \{-1, +1\}^N$ for bipolar patterns, and L is the number of training patterns. For binary patterns $\mathbf{p}_l \in \{0, 1\}^M$ and $\mathbf{q}_l \in \{0, 1\}^N$, the bipolar values $a_{li} = 2p_{li} - 1$ and $b_{li} = 2q_{li} - 1$ corresponding to the binary elements p_{li} and q_{li} , respectively, are used in the computation of the weight matrix. The weight matrix from the second layer to the first layer is given by

$$\mathbf{W}^T = \sum_{l=1}^L \mathbf{b}_l \mathbf{a}_l^T \quad (7.2)$$

The activation equations for the bipolar case are as follows:

$$\mathbf{b}_j(m+1) = \begin{cases} 1, & \text{if } y_j > 0 \\ \mathbf{b}_j(m), & \text{if } y_j = 0 \\ -1, & \text{if } y_j < 0 \end{cases} \quad (7.3)$$

where $y_j = \sum_{i=1}^M w_{ji} a_i(m)$, and

$$\mathbf{a}_i(m+1) = \begin{cases} 1, & \text{if } x_i > 0 \\ \mathbf{a}_i(m), & \text{if } x_i = 0 \\ -1, & \text{if } x_i < 0 \end{cases} \quad (7.4)$$

where $x_i = \sum_{j=1}^N w_{ij} b_j(m)$. In the above equations $\mathbf{a}(m) = [a_1(m), a_2(m), \dots, a_M(m)]^T$ is the output of the first layer at the m th iteration, and $\mathbf{b}(m) = [b_1(m), b_2(m), \dots, b_N(m)]^T$ is the output of the second layer at the m th iteration.

For recall, the given input $\mathbf{a}_i(0)$, $i = 1, 2, \dots, M$, is applied to the first layer and the activation equations are used in the forward and backward passes several times until equilibrium is reached. The stable values $\mathbf{b}_j(\infty)$, $j = 1, 2, \dots, N$ are read out as the pattern associated with the given input. Likewise the pattern at the first layer can be recalled given the pattern at the second layer.

The updates in the **BAM** are synchronous in the sense that the units in each layer are updated simultaneously. **BAM** can be shown

to be unconditionally stable using the Lyapunov energy function given by [Kosko, 1992]

$$V(\mathbf{a}, \mathbf{b}) = -\frac{1}{2} \mathbf{b}^T \mathbf{W}^T \mathbf{a} - \frac{1}{2} \mathbf{a}^T \mathbf{W} \mathbf{b} = -\mathbf{a}^T \mathbf{W} \mathbf{b} = -\mathbf{b}^T \mathbf{W}^T \mathbf{a} \quad (7.5)$$

Therefore,

$$V(\mathbf{a}, \mathbf{b}) = -\sum_{j=1}^N \sum_{i=1}^M w_{ji} a_i b_j = -\sum_{i=1}^M \sum_{j=1}^N w_{ij} b_j a_i \quad (7.6)$$

The change in energy due to change Δa_i in a_i is given by

$$\Delta V_{a_i} = -\left(\sum_{j=1}^N w_{ij} b_j \right) \Delta a_i, \quad i = 1, 2, \dots, M \quad (7.7)$$

Likewise the change in energy due to change Δb_j in b_j is given by

$$\Delta V_{b_j} = -\left(\sum_{i=1}^M w_{ji} a_i \right) \Delta b_j, \quad j = 1, 2, \dots, N \quad (7.8)$$

For bipolar units,

$$\Delta a_i = \begin{cases} 2 \text{ or } 0, & \text{if } x_i > 0 \\ 0, & \text{if } x_i = 0 \\ -2 \text{ or } 0, & \text{if } x_i < 0 \end{cases} \quad (7.9)$$

where $x_i = \sum_{j=1}^N w_{ij} b_j$. Similarly,

$$\Delta b_j = \begin{cases} 2 \text{ or } 0, & \text{if } y_j > 0 \\ 0, & \text{if } y_j = 0 \\ -2 \text{ or } 0, & \text{if } y_j < 0 \end{cases} \quad (7.10)$$

where $y_j = \sum_{i=1}^M w_{ji} a_i$. From these relations, it is obvious that

$$\Delta V_{a_i} \leq 0, \quad \text{for } i = 1, 2, \dots, M, \quad (7.11)$$

and

$$\Delta V_{b_j} \leq 0, \quad \text{for } j = 1, 2, \dots, N, \quad (7.12)$$

which means that the energy either decreases or remains the same in each iteration. Therefore the BAM reaches a stable state for any weight matrix derived from the given pattern pairs.

The BAM is limited to binary or bipolar valued pattern pairs. The upper limit on the number (L) of pattern pairs that can be stored is $\min(M, N)$ [Kosko, 1988]. The performance of BAM depends on the nature of the pattern pairs and their number. As the number of pattern pairs increases, the probability of **error** in recall will also

increase. The error in the recall will be large if the memory is filled to its capacity. Improved methods of encoding (determination of weights) were proposed through the use of matrix transformations [Wang et al, 1990a; 1990b; 1991].

Extensions of the discrete BAM have been proposed to deal with analog pattern pairs in continuous time. The resulting network is called Adaptive BAM (ABAM) [Kosko, 1987]. In this case the pattern pairs are encoded using Hebbian learning with a passive decay term in learning. For recall of the patterns, the additive model of the activation dynamics is used for units in each layer separately. According to the ABAM theorem the memory is globally stable.

7.2.2 Multidirectional Associative Memory

The bidirectional associative memory concept can be generalized to store associations among more than two patterns. The multiple association memory is also called multidirectional associative memory (MAM) [Hagiwara, 1990]. As an illustration, the architecture of MAM is shown in Figure 7.2 for the case of associations among three

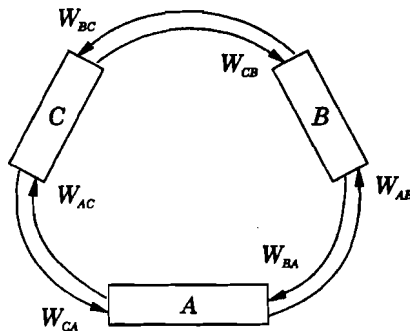


Figure 7.2 Illustration of multidirectional associative memory for three layers of units.

patterns ($\mathbf{a}_l, \mathbf{b}_l, \mathbf{c}_l$). The three layers of units are denoted as A, B, C in the figure. The dimensions of the three vectors $\mathbf{a}_l, \mathbf{b}_l$ and \mathbf{c}_l are N_1, N_2 and N_3 , respectively. The weight matrices for the pairs of layers are given by

$$W_{AB} = \sum_{l=1}^L \mathbf{a}_l \mathbf{b}_l^T, \quad W_{BC} = \sum_{l=1}^L \mathbf{b}_l \mathbf{c}_l^T, \quad W_{CA} = \sum_{l=1}^L \mathbf{c}_l \mathbf{a}_l^T \quad (7.13)$$

and

$$W_{BA} = W_{AB}^T, \quad W_{CB} = W_{BC}^T, \quad W_{AC} = W_{CA}^T \quad (7.14)$$

For recall, the activation equations for the bipolar case are computed as shown below for the layer B

$$b_j(m+1) = \begin{cases} 1, & \text{if } y_j > 0 \\ b_j(m), & \text{if } y_j = 0 \\ -1, & \text{if } y_j < 0 \end{cases} \quad (7.15)$$

for $j = 1, 2, \dots, N_2$, where

$$y_j = \sum_{i=1}^{N_1} w_{ABji} a_i(m) + \sum_{i=1}^{N_3} w_{CBji} c_i(m) \quad (7.16)$$

where w_{ABji} is the j th element of the weight matrix \mathbf{W}_{AB} , and w_{CBji} is the j th element of the weight matrix \mathbf{W}_{CB} .

The outputs $c_j(m+1)$ and $a_j(m+1)$ are likewise computed. Each unit in a layer is updated independently and **synchronously** based on the net input from units in the other two layers. The updating is performed **until** a **multidirectionally** stable state is reached.

The BAM is only a special case of MAM. Due to associations among several layers to be satisfied simultaneously, the information recovery for the partial input is better in MAM than in BAM [Hagiwara, 1990].

7.2.3 Temporal Associative Memory (TAM)

The **BAM** can be used to store a sequence of temporal pattern vectors, and recall the sequence of patterns [Zurada, 1992, Sec. 6.61]. The basic idea is that the adjacent overlapping pattern pairs are to be stored in a BAM. Let $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_L$ be a sequence of L patterns, each with a dimensionality of M . Then $(\mathbf{a}_1, \mathbf{a}_2), (\mathbf{a}_2, \mathbf{a}_3), \dots, (\mathbf{a}_i, \mathbf{a}_{i+1}), \dots, (\mathbf{a}_{L-1}, \mathbf{a}_L)$ and $(\mathbf{a}_L, \mathbf{a}_1)$ form the pattern pairs to be stored in the BAM. Note that the last pattern in the sequence is paired with the first pattern. The weight matrix in the forward direction is given by

$$\mathbf{W} = \sum_{i=1}^{L-1} \mathbf{a}_i \mathbf{a}_{i+1}^T + \mathbf{a}_L \mathbf{a}_1^T \quad (7.17)$$

The weight matrix for the reverse direction is given by the transpose of the forward weight matrix, **i.e.**, by \mathbf{W}^T .

The recall steps are exactly the same as for BAM. When stable conditions are reached, then it is possible to recall the entire sequence of patterns from any one pattern in the sequence. The TAM has the same kind of limitations as those of BAM in its error performance in recall and also in its capacity for storing a given length (L) of a sequence of patterns.

7.3 Pattern Mapping

7.3.1 Background for Pattern Mapping Networks

The multilayer feedforward neural network with **error backpropaga-**

tion learning was primarily developed to overcome the limitation of a single layer perceptron for classification of hard problems (non-linearly separable classes) and to overcome the problem of training a **multilayer** perceptron (due to hard-limiting output function) for these hard problems. In this so called backpropagation network the objective is to capture (in the weights) the complex nonlinear hypersurfaces separating the classes. The complexity of the surface is determined by the number of hidden units in the network. Strictly speaking, any classification problem specified by the training set of examples can be solved using a network with sufficient number of hidden units. In such a case, the problem is more of a pattern association type than of a classification type, with no restrictions on the associated patterns as in the case of a linear associative network.

In a classification **problem** the input patterns belonging to a class are expected to have some common features which are different for patterns belonging to another class. The idea is that, for a pattern belonging to any of the trained classes, the network is supposed to give the correct classification. In other words, for a classification problem, the trained neural network is expected to perform some kind of generalization, which is possible only if there are some features common among the input patterns belonging to each class, and these features are captured by the network during training. Generalization has no meaning for arbitrary association of one pattern to another as in the case of arbitrary Boolean **functions**. Generalization also has no meaning if the training set consists of all possible input patterns as in the **XOR** problem.

Some special association tasks may have common features hidden too deep in the input, like in the parity problem [Minsky and Papert, 1990]. In this case the feature characterizing the similarity of patterns belonging to the same class is not **reflected** directly in the bit pattern of the input vector. For example, **00110000** and **00001111** both belong to the same class of even parity, and **01110000** and **00000111** both belong to the class of odd parity, although the odd parity pattern **01110000** is closer to the even parity pattern **00110000** in the Hamming distance sense. In these cases the representation of the input patterns is crucial to achieve generalization by a network. Preprocessing of the input vectors is to be performed separately to extract the desired features for feeding the features to a neural network. A neural network by itself may not be able to extract automatically the desired features due to limitations of operations it can perform on the input data, and more importantly, due to masking of the desired features by the other undesirable, but dominating features in the input.

Therefore a trained neural network is expected to exhibit the generalization property for the classification problems in which groups of the input patterns belonging to a class possess some

common features within each group. Another way of looking at this problem is that there should be some redundancy among the patterns in each group in order to develop a system for classification. Note that the class label that is assigned to a group or a collection of groups could be quite arbitrary. In other words, the mapping of a group of input patterns to an output or a class label need not have any **restrictions**.

Another class of problems deals with capturing the mapping function implied in the given training set of input-output pattern pairs. Here the mapping function represents the system that produced the output for a given input for each pair of patterns. A trained neural network is expected to capture the system characteristics in their weights. The network is supposed to have generalized from the training data, if for a new input the network produces the same output which the system would have produced. The generalization of this mapping function in the network can be tested by a set of test input-output pattern pairs. Note that in this case the mapping function must exhibit some smoothness or redundancy, as the given training data is usually not adequate to sample the function at all points. Note also that the set of input patterns themselves could be quite arbitrary. So generalization in these cases is possible only if the mapping function satisfies certain constraints. Otherwise, the problem of capturing the mapping function from the training data set will be an ill-posed problem [Tikhonov and Arsenin, 1977]. Assuming that the constraints are satisfied by the mapping function, they are forced on the approximating function using regularization methods, so that the ill-posed problem becomes well-posed [Poggio et al, 1985]. Finally, generalization by a network is also possible in situations where both the input patterns in a group and the mapping function have redundancies displayed in the **form** of common features among the patterns in a group and smoothness in the function, respectively.

A multilayer perceptron (**MLP**) architecture is suggested to address arbitrary pattern association tasks which could not be solved by either a linear associative network due to restriction on the type and number of input patterns or by a single layer perceptron due to linear separability constraint on the classification task specified by the input-output mapping.

A multilayer **feedforward** neural network (**MLFFNN**) can be used to realize an approximation to a multilayer perceptron (**MLP**) for complex (arbitrary) pattern association tasks. It is not intended specifically to solve a pattern classification or pattern mapping problem, as both require generalization based on 'closeness' property in classification and 'smoothness' property in mapping, respectively. In other words, a **MLFFNN** trained with backpropagation learning is neither designed to exploit the property of 'closeness' for generalizing a classification task, nor is it designed to exploit the

property of 'smoothness' to generalize a function approximation task. It is designed mainly to provide discrimination between patterns belonging to different classes.

The distinction between what is being achieved by a MLFFNN and what is needed to be achieved for classification and function approximation tasks is illustrated in Figure 7.3 [Lowe, 1995]. Mere

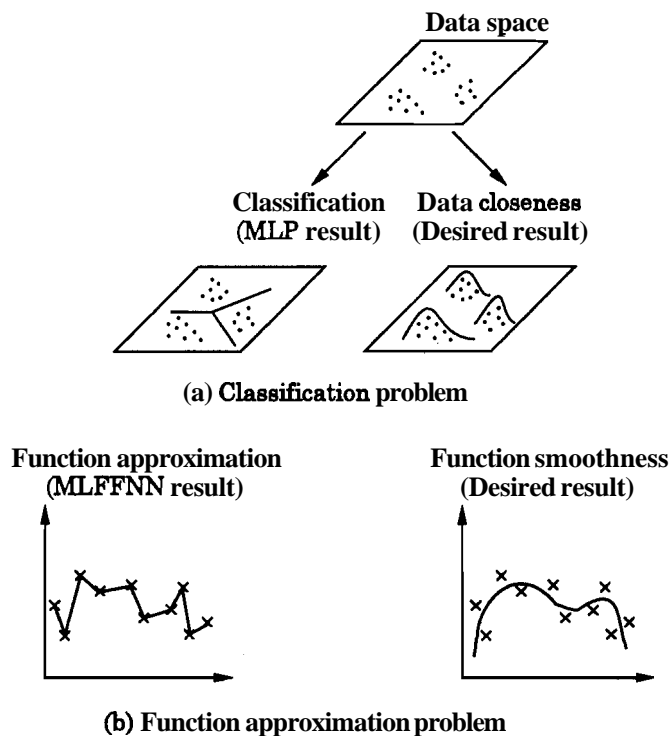


Figure 7.3 Distinction between two pattern recognition tasks as realized by a trained MLFFNN and the desired results: (a) Classification problem and (b) Function approximation problem.

manipulation of the **structure** of a neural network and learning of a MLFFNN **are** not likely to achieve the generalization required for a given problem. Even if the generalization behaviour of a trained MLFFNN is confirmed by cross-validation, it is only an ad **hoc** solution. There is no guarantee of obtaining the desired result. This is because, the network is not designed specifically to address the generalization problem. Moreover, it is not generally possible to analyze a MLFFNN to understand the task each layer is performing.

In fact, if the given problem is known to be a classification problem based on the closeness of data in the input vectors, then specific architectures can be evolved to achieve generalization. Such architectures tend to be much simpler than a general MLFFNN, **and**

training also is likely to be simpler than the backpropagation learning. It is also possible to improve the **generalization** capability by incorporating a priori knowledge about the patterns in the classification task. However, developing architectures will be much more difficult if the classification is based on deep features present in the data, and if preprocessing needed to extract these features is not explicitly included as part of the network.

Likewise, if the given problem is a function approximation based on the smoothness of the mapping function of the given input-output data, then specific architectures can be evolved to achieve generalization in such cases. Here again these architectures tend to be much simpler than the MLFFNN, and the training involved also will be trivial in most cases. It is possible to improve the generalization capability using regularization which involves imposing some smoothness constraints explicitly on the mapping function. The smoothness constraint is intended to reflect the a priori knowledge of the function. However, developing architectures for proper generalization is much more difficult if the mapping function is not smooth at the given data level. Even if smoothness of the mapping function is present at some deep feature level, it is not possible for the network to generalize, unless preprocessing of the data to obtain the features is explicitly known and implemented in the network.

For discussion, we assume that the training set data consists of pairs of input-output vectors represented by $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \dots, L$. For a classification task, \mathbf{b}_l is an N -dimensional vector of zeros and ones, with a 1 in the j th position if the input vector \mathbf{a}_l belongs to the j th class. This is called 'hard' classification. There may be several input vectors, which are close to each other, and hence may have the same \mathbf{b}_l associated with them. In many situations, it may be desirable to have the N -dimensional output vector to represent an estimate of the probability distribution of the classes for the given input. That is, the j th component of \mathbf{b}_l corresponds to the probability that the input vector belongs to the class j . In this case the sum of all the components in \mathbf{b}_l will add upto 1. The input vector \mathbf{a}_l could be an M -dimensional vector of ones and zeros or a vector of real numbers.

In the function estimation or pattern mapping the output vector \mathbf{b}_l is an N -dimensional vector of real values. The function estimation can also be viewed as a **nonparametric** regression problem, as we are trying to determine a network that realizes the best fit **function** for the given input-output pairs of data.

In this section we will consider the tasks of pattern classification and multivariate function approximation (or pattern mapping). In both cases the learned network should generalize well, which means that the network should give the correct classification for a new (test) data input in the case of a classification task and a reasonable approximation to the true function value for a new (test) data input

in the case of a function approximation task. We will discuss architectures of the Radial Basis Function (RBF) network suitable for these tasks. We assume that the 'closeness' property for the classification tasks and the 'smoothness' property for the **function** approximation tasks **are** satisfied in the given training set data.

7.3.2 Architecture of Radial Basis Function (RBF) Networks

The architecture of a radial basis function network is shown in Figure 7.4. It consists of a single hidden layer with nonlinear units, followed by an output layer with linear units.

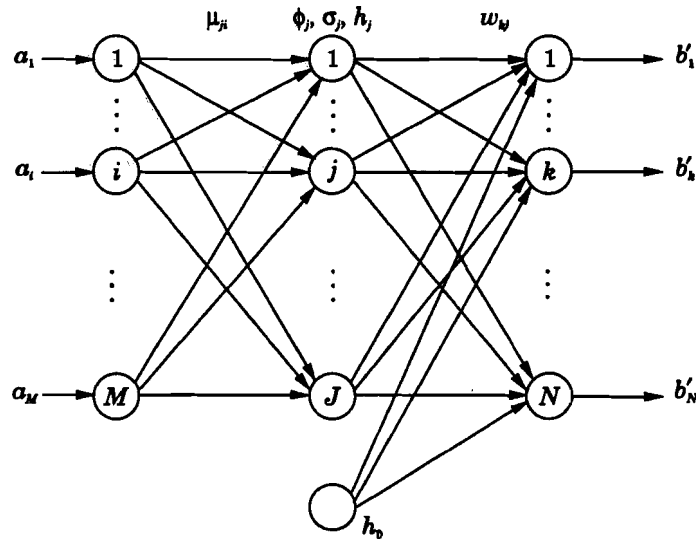


Figure 7.4 General form of a radial basis function network. The nonlinear basis function of the \$j\$th hidden unit is a function of the normalized radial distance (\$\|a - \mu_j\|/\sigma_j\$) between the input vector \$a = [a_1, a_2, \dots, a_M]^T\$ and the weight vector \$\mu_j = [\mu_{j1}, \mu_{j2}, \dots, \mu_{jM}]^T\$ associated with the unit. Normalization factor \$\sigma_j\$ decides the range of influence of the \$j\$th unit around its centre \$\mu_j\$.

The output of the \$k\$th unit in the output layer of the network is given by

$$b'_k = \sum_{j=0}^J w_{kj} h_j \quad (7.18)$$

where \$h_j = \phi_j(\|a - \mu_j\|/\sigma_j)\$, \$j = 1, 2, \dots, J\$ and \$h_0 (= -1)\$ is the output of the bias unit, so that \$w_{k0}\$ corresponds to the bias on the \$k\$th output unit. The nonlinear basis function \$\phi_j(\cdot)\$ of the \$j\$th hidden unit is a function of the normalized radial distance between the input vector \$a = (a_1, a_2, \dots, a_M)^T\$ and the weight vector \$\mu_j = (\mu_{j1}, \mu_{j2}, \dots, \mu_{jM})^T\$

associated with the unit. The normalizing factor σ_j decides the range of influence of the j th unit. If the basis function is a Gaussian function of the type $\phi(x) = \exp(-x^2/2)$, then the weight vector μ_j corresponds to the mean value of the function and σ_j corresponds to the standard deviation. For a multidimensional vector \mathbf{x} with zero mean, the Gaussian function is given by $\phi(\mathbf{x}) = \exp(-\mathbf{x}^T \mathbf{R}^{-1} \mathbf{x})$, where \mathbf{R} is the covariance matrix of the input vectors, namely, expectation of $\mathbf{x}\mathbf{x}^T$. The significance of the radial basis function is that the output of the unit is a function of the radial distance, i.e., $h_j = \phi_j(\|\mathbf{a} - \mu_j\|/\sigma_j)$. On the other hand, in a multilayer feedforward neural network, $h_j = \phi_j(\mathbf{a}^T \mathbf{w}_j)$, i.e., the output is a nonlinear function of the scalar product of the input vector and the weight vector.

7.3.3 Theorems for Function Approximation

Before we discuss RBF networks for function approximation, it is worthwhile noting the following two theorems for function approximation, one based on the linear basis function and the other based on the radial basis function [Kung, 1993].

Theorem 1: Function approximation by linear basis function. [Cybenko, 1989; Funahashi, 1989]. Let A be a compact subset of \mathcal{R}^M and $F(\mathbf{x})$ be a continuous function on A . Then for any $\varepsilon > 0$, there exist an integer N and real constants c_i, w_{ij} and θ_i such that $|\hat{F}(\mathbf{x}) - F(\mathbf{x})| < \varepsilon$ for all $\mathbf{x} \in A$, where

$$\hat{F}(\mathbf{x}) = \hat{F}(x_1, x_2, \dots, x_M) = \sum_{i=1}^N c_i f\left(\sum_{j=1}^M w_{ij} x_j + \theta_i\right) \quad (7.19)$$

and $f(\cdot)$ is any nonconstant, bounded and monotonically increasing continuous function. The argument of the function $f(\cdot)$ is a linear weighted sum of the component values, i.e., $\sum_j w_{ij} x_j + \theta_i$. Therefore $f(\cdot)$ is called linear basis function. In this approximation the function $f(\cdot)$ could be like the semilinear output function of a MLFFNN. Thus this function approximation, in principle, can be realized by a MLFFNN with a single layer of hidden units and an output layer of linear units.

Theorem 2: Function approximation by radial basis function. Let A be a compact subset of \mathcal{R}^M and $F(\mathbf{x})$ be a continuous function on A . Then for any $\varepsilon > 0$, there exist an integer N and parameters \mathbf{w}_i and c_i such that $|\hat{F}(\mathbf{x}) - F(\mathbf{x})| < \varepsilon$ for all $\mathbf{x} \in A$, where \mathbf{w}_i s are M -dimensional parameter vectors corresponding to the centroids of clusters, so that

$$\hat{F}(\mathbf{x}) = \hat{F}(x_1, x_2, \dots, x_M) = \sum_{i=1}^N c_i g(\|\mathbf{x} - \mathbf{w}_i\|) \quad (7.20)$$

where $g(\cdot)$ is a nonlinear function with unique **maximum** centered at \mathbf{w}_i [Powell, 1988; Broomhead and Lowe, 1988; Moody and Darken, 1989]. The argument of the function $g(\cdot)$ **forms** the basis of the function. Since the argument is the radial distance between the variable vector \mathbf{x} from the centroid vector \mathbf{w}_i , the function $g(\cdot)$ is called the radial basis function. The function approximation itself is called the radial basis function approximation. Gaussian function is one of the commonly used nonlinear functions for $g(\cdot)$.

The above theorems are called *universal approximation theorems*. They show the existence of the parameters to approximate a given function. In the context of neural networks, both the theorems suggest that a feedforward network with a single hidden layer with nonlinear units can approximate any arbitrary function. But the theorems do not suggest any method of determining the parameters, such as the number of hidden **units** and weights in order to achieve a given accuracy for the approximation of the function.

7.3.4 RBF Networks for Function Approximation

In the **RBF** networks the weights (μ_j, σ_j) of the hidden layer units are determined directly from the data. No learning is involved. The weights \mathbf{w}_k of the output layer are determined by supervised learning [Broomhead and Lowe, 1988; Moody and Darken, 1989]. For function approximation task the error to be minimized in the supervised learning is given by

$$E(F) = E_D(F) + \lambda E_R(F) \quad (7.21)$$

where E_D is the error due to the given data in the form of **input-output** pair $(\mathbf{a}_l, \mathbf{b}_l)$, E_R is the contribution due to regularization, and λ is the regularization parameter. The error term E_D is given by

$$E_D(F) = \frac{1}{2} \sum_{l=1}^L \sum_{k=1}^N (b_{lk} - b'_{lk})^2 \quad (7.22)$$

Note that b'_{lk} is a function of the input data and the parameters of the network, i.e., $b'_{lk} = F(\mathbf{a}_l, \mathbf{w}_k)$. The regularization term E_R depends on the prior knowledge of the function or some global knowledge derived **from** the given data. This knowledge is normally represented in the form of a smoothness constraint on the mapping function between the input and the output. In one form of smoothing, the weights of the network are constrained using the following expression for the regularization term [Hergert et al, 1992].

$$E_R(F) = \frac{1}{2} \sum_j \sum_k w_{kj}^2 \quad (7.23)$$

Inclusion of this term favours small values of the weights. It can also

be viewed as a 'weight decay' term, since the weight change is proportional to the negative gradient of the error (See Eq. (4.77)), and the negative gradient of $E_R(F)$ gives a decay term in the learning equation (See Eq. (2.22)).

Smoothing constraint is also expressed in the form of the square of the derivatives of the mapping function, namely,

$$E_R(F) = \frac{1}{2} \|PF\|^2 \quad (7.24)$$

where P is a linear differential operator, and $\|\cdot\|$ is the L_2 norm. For example, P could be a simple second derivative of F with respect to w . Then minimization of the square of the derivative restricts the discontinuous jumps in the function [Poggio and Girosi, 1990].

In the expression for $E(F)$ in (7.21), if $\lambda = 0$, then the error is dictated by the given training data. For a sufficiently large (in terms of number of hidden units) network, it is possible to determine the weights of the network so that we can get an approximate interpolating function as shown by the solid line in Figure 7.5a for a 1-D case, whereas what is desired is the function shown by the dashed line [Wahba, 1995]. Thus the network fails to generalize from the data due to overfitting of the data. This is not desirable, as the given training data can be usually noisy. In order to improve the generalization capability, the λ parameter is made nonzero. For a suitable choice of λ , we get a reasonable estimation of the function as shown by the solid line in Figure 7.5b, where the dashed line shows the desired function. For large h , if the smoothing function is restricted to small value ϵ , i.e., $\|PF\|^2 < \epsilon$, then the resulting function is a poor estimate of the desired function as shown in Figure 7.5c, because the error term due to data does not play a significant role. Thus the parameter λ controls the performance of the pattern mapping network. The value of the parameter λ can be inferred from the given data using probability theory and Bayes theorem for conditional probability [Mackay, 1995].

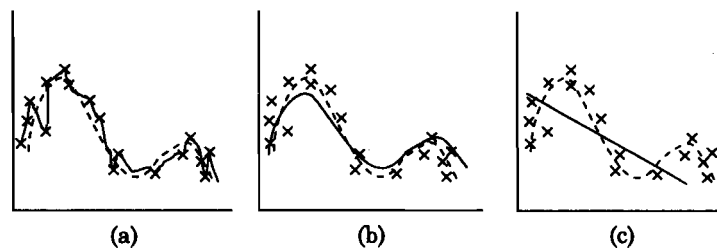


Figure 7.5 Function approximation for different values of regularization parameter λ : (a) λ too small, (b) λ near optimal and (c) λ too large. 'x' indicates actual points. Dashed line is the desired function. Solid line is the realized function.

The regularization problem is to find the function $F(\cdot)$ that minimizes the error given by Eq. (7.21), where the second term due to regularization is a constraint on the desired level of smoothness of the function. The smoothness constraint is usually in the form of a differential operator, as noted before.

Let us consider one dimensional case, i.e., the function $F(\cdot)$ is a scalar function and hence there is only one output unit for the network. Then

$$E_D(F) = \frac{1}{2} \sum_{l=1}^L [d_l - F(\mathbf{a}_l)]^2 \quad (7.25)$$

We have $E_R(F) = \frac{1}{2} \|PF\|^2$, where P is a linear differential operator. The minimization problem reduces to solving the following differential equation [Poggio and Girosi, 1990]:

$$P^*PF = \frac{1}{\lambda} \sum_{l=1}^L [d_l - F(\mathbf{a}_l)] \delta(\mathbf{a} - \mathbf{a}_l). \quad (7.26)$$

where P^* is the **adjoint** of the differential operator P [Haykin, 1994] and $\delta(\cdot)$ is a delta function.

Let G denote the Green's function for the operator P^*P , so that

$$P^*PG(\mathbf{a} : \mathbf{a}_l) = \delta(\mathbf{a} - \mathbf{a}_l) \quad (7.27)$$

Then the solution of the regularization problem is given by [Haykin, 1994]

$$F(\mathbf{a}) = \sum_{l=1}^L w_l G(\mathbf{a} : \mathbf{a}_l) \quad (7.28)$$

where

$$w_l = \frac{1}{\lambda} [d_l - F(\mathbf{a}_l)] \quad (7.29)$$

G is the Green's function for the l th input pattern. The Green's function is a result of the smoothness constraint expressed in the form of a differential operator. If the differential operator P^*P is invariant to rotation and translation, then the Green's function is a function of the magnitude of the difference of its arguments, i.e., $G(\mathbf{a} : \mathbf{a}_l) = G(\|\mathbf{a} - \mathbf{a}_l\|)$. In such a case G is called a radial basis function. For some special cases of the differential operator P, the radial basis function becomes a multivariate Gaussian function [Haykin, 1994].

Thus the solution of the regularization problem leads to a radial basis function as shown in Figure 7.6 for the 1-D case, where the Green's function is shown by $\phi(\cdot)$. The weights are determined by minimizing the error E in Eq. (7.21) which consists of the squared error between the desired and actual output values, and the regularization term, the extent of which is **determined** by the

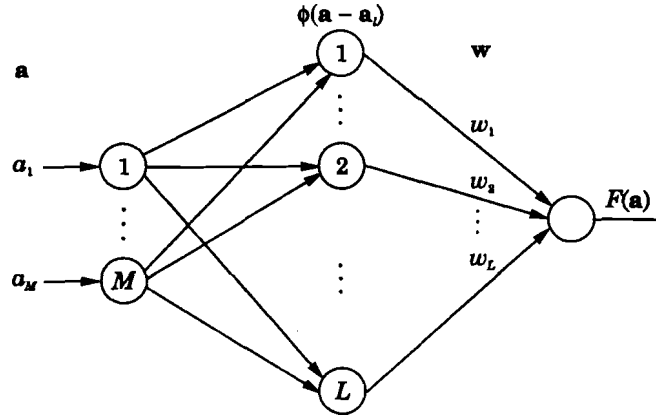


Figure 7.6 Radial basis function network for function approximation, with one output unit and L hidden units.

parameter h . In the above formulation the number of hidden units is equal to the number of training samples (L) and the centres of the basis function are located at the sample values \mathbf{a}_i .

A suboptimal solution to the function approximation is obtained using fewer basis functions. That is, using the radial basis functions, the function F is given by

$$F(\mathbf{a}) = \sum_{i=1}^H w_i \phi(\|\mathbf{a} - \boldsymbol{\mu}_i\|) \quad (7.30)$$

where $H < L$, and $\boldsymbol{\mu}_i$ are the centres of the basis functions to be determined from the given data. The weights of the output units and the centres of the radial basis function can be determined by computation using all the training set data [Haykin, 1994].

For a $\mathbf{1-D}$ output function, the desired output d_i is a scalar for the given input vector \mathbf{a}_i . That is,

$$d_i = F(\mathbf{a}_i). \quad (7.31)$$

Minimizing the error in Eq. (7.21) without regularization ($h = 0$), the optimum weight vector $\mathbf{w} = [w_1, w_2, \dots, w_H]^T$ is given by

$$\mathbf{w} = \Phi^+ \mathbf{d} \quad (7.32)$$

where $\mathbf{d} = [d_1, d_2, \dots, d_L]^T$, and Φ^+ is the pseudoinverse of the matrix

$$\Phi = \begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1H} \\ \phi_{21} & \phi_{22} & \dots & \phi_{2H} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{L1} & \phi_{L2} & \dots & \phi_{LH} \end{bmatrix} \quad (7.33)$$

The elements of the matrix are given by

$$\phi_{ji} = \phi(\|\mathbf{a}_j - \boldsymbol{\mu}_i\|), \quad i = 1, 2, \dots, H, \quad j = 1, 2, \dots, L \quad (7.34)$$

The pseudoinverse of Φ^+ is given by

$$\mathbf{a}^+ = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{d} \quad (7.35)$$

If regularization ($h \neq 0$) is used in the error function, then the corresponding expression for the optimal weight vector is given by [Haykin, 1994]

$$\mathbf{w} = (\Phi^T \Phi + h \Phi_0)^{-1} \Phi^T \mathbf{d} \quad (7.36)$$

where Φ_0 is an $H \times H$ matrix whose j th element is $\phi(\|\boldsymbol{\mu}_j - \boldsymbol{\mu}_i\|)$.

The weights can also be determined by using supervised learning methods like LMS algorithm, in which the weight adjustment at each stage is given by (See Chapter 1)

$$\Delta w_i = -\eta [d_i - f(\mathbf{a}_i)] \quad (7.37)$$

where η is the learning rate parameter.

7.3.5 RBF Networks for Pattern Classification

Given a set of **training** samples in the form of input pattern vectors \mathbf{a}_l , $l = 1, 2, \dots, L$ and the associated class labels, the objective in pattern classification is to design a system which can classify any new input vector \mathbf{a} correctly by assigning the correct class label to it. Note that in a classification problem there will be fewer classes than the number of **training** patterns, and hence all the class labels are not distinct. In the training set there may be many pattern vectors associated with each of the distinct classes in the problem.

The pattern classification problem can be posed as follows [Lowe, 1995]: Given an input pattern \mathbf{a} , determine the class label C_i such that the a posteriori probability $P(C_i | \mathbf{a})$ of the class C_i is maximum among all classes. This probability can be computed using the probabilities $p(\mathbf{a} | C_i)$ and $p(\mathbf{a})$, since

$$P(C_i | \mathbf{a}) = \frac{p(\mathbf{a} | C_i) P(C_i)}{p(\mathbf{a})} \quad (7.38)$$

where $p(\mathbf{a} | C_i)$ gives the probability distribution of the data generated for the class C_i and $p(\mathbf{a})$ is the probability distribution of the data vector irrespective of the class. **These** probability distributions can be expressed in terms of a linear combination of some standard distribution $\psi(\mathbf{a})$, say Gaussian, in order to reflect the possible **multimodal** nature of the distribution of the data \mathbf{a} belonging to any class. These mixture distributions can be written as [Lowe, 1995]

$$p(\mathbf{a}) = \sum_k \alpha_k \psi_k(\mathbf{a}), \quad (7.39)$$

and

$$p(\mathbf{a}|C_i) = \sum_j \beta_j^i \psi_j(\mathbf{a}). \quad (7.39)$$

where α_k and β_j^i are coefficients of the standard distribution functions. Therefore,

$$\begin{aligned} P(C_i|\mathbf{a}) &= \sum_j P(C_i) \frac{\beta_j^i \psi_j(\mathbf{a})}{\sum_k \alpha_k \psi_k(\mathbf{a})} \\ &= \sum_j \frac{P(C_i) \beta_j^i}{\alpha_j} \frac{\alpha_j \psi_j(\mathbf{a})}{\sum_k \alpha_k \psi_k(\mathbf{a})} \\ &= \sum_j w_{ij} \phi_j(\mathbf{a}) \end{aligned}$$

where $\phi_j(\mathbf{a})$ is the normalized basis function and $w_{ij} = P(C_i) \beta_j^i / \alpha_j$ is the contribution of the output of the basis function to the output unit corresponding to the i th class C_i . Thus the expression for $P(C_i|\mathbf{a})$ can be viewed as a basis function formulation of the classification problem, and the corresponding radial basis function network is shown in Figure 7.7.

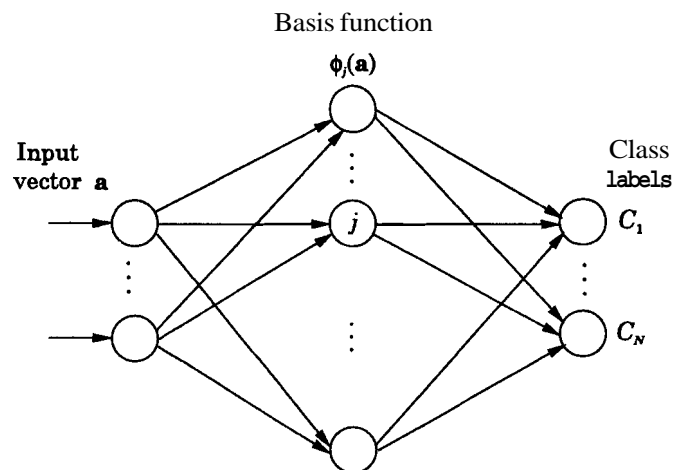


Figure 7.7 Radial basis function network for pattern classification. The number of input nodes depends on the dimensionality of the input vector. The number of output nodes is equal to the number of distinct classes. The number of hidden nodes is equal to the number of basis functions used in the network. The number and shapes of the basis functions depend on the closeness of the input data in the training set.

The basis functions for classification tasks **are** determined from the input data in the training set. The number and shapes of the basis functions depend on the 'closeness' property of the input data in the training set. This can be determined using an unsupervised clustering of the input data. The representation of the clusters is somewhat simplified if the basis functions are assumed to be of Gaussian type, so that the parameters of each cluster can be determined by computing the first (mean) and second (covariance matrix) order statistics of the input data for each cluster. In other words, the probability distribution of each cluster is assumed to be elliptical. Note that in the classification task the basis **functions are** solely determined by the distribution of points in the training set in the input space. It involves determination of clusters first and then fitting a distribution to each cluster. The number of clusters and the parameters of the basis functions can be either computed using the entire training set data or can be learned using learning techniques [Haykin, 1994].

Once the basis functions are determined, then the weights in the output layer can be obtained from the training data either by computation using matrix inversion or by supervised learning using gradient descent methods.

To illustrate the steps involved in a pattern classification task, let us consider the **2-D cluster points** given in Figure 7.8, where each class is indicated by a separate symbol like 'x', 'O', etc. The first step is to determine the clusters using any standard clustering algorithm [Haykin, 1994] or by any of the **unsupervised** learning methods

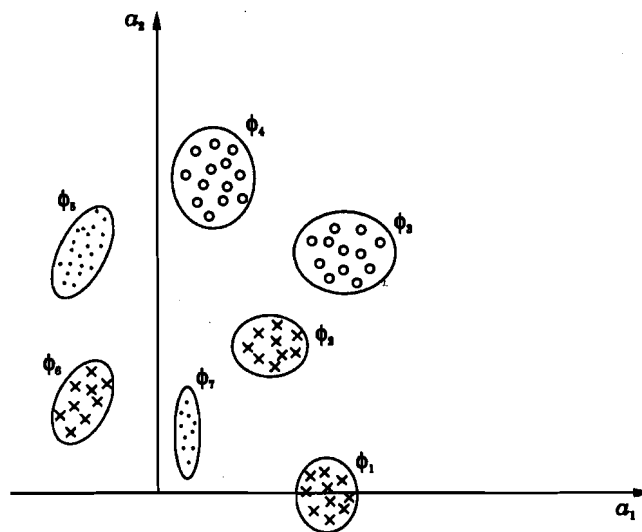


Figure 7.8 2-D data points belonging to three classes.

described in Chapter 6. The **number** of clusters can be fixed a priori, or a criterion may be used to determine the optimum number of clusters [Dubes, 1987]. **Then** a basis function is derived for each cluster. If a 2-D Gaussian function is assumed, then the mean and covariance matrix are **derived** for each cluster to represent the corresponding Gaussian **basis** function $\phi(\mathbf{a})$. That is

$$\phi(\mathbf{a}) = \frac{1}{\sqrt{(2\pi)^M |\mathbf{R}|^{1/2}}} e^{-(\mathbf{a} - \boldsymbol{\mu})^T \mathbf{R}^{-1} (\mathbf{a} - \boldsymbol{\mu})} \quad (7.41)$$

where $\boldsymbol{\mu}$ is the mean vector of the cluster points and \mathbf{R} is the covariance matrix of the cluster. The *ij*th element of the matrix \mathbf{R} is given by

$$\mathbf{R}_{ij} = \sum_I (\mathbf{a}_{iI} - \boldsymbol{\mu}_i) (\mathbf{a}_{jI} - \boldsymbol{\mu}_j), \quad i, j = 1, 2 \quad (7.42)$$

where I is the index for the sample pattern in the cluster, and $\boldsymbol{\mu}_i = \bar{\mathbf{a}}_i$ and $\boldsymbol{\mu}_j = \bar{\mathbf{a}}_j$ are the mean values of the *i*th and *j*th components of the input vectors. They are given by $\bar{\mathbf{a}}_i = \sum \mathbf{a}_{iI}$, for **all** *i*. For the *k*th cluster, $\phi = \phi_k$ and the mean vector and covariance matrix can be indicated by $\boldsymbol{\mu} = \boldsymbol{\mu}^k$ and $\mathbf{R} = \mathbf{R}^k$, respectively.

The basis functions specified by the mean vector and the covariance matrix for each cluster determine the computations to be performed at the hidden units. The number (H) of hidden **units** is equal to the total number of clusters:

The number (N) of the output units is equal to the number of distinct classes. The desired response for the classification task is represented by an N -dimensional vector, with a **1** at the output of the unit corresponding to the correct class, and a **0** at the output of all other units. That is, the desired output vector is given by $\mathbf{d} = [0 \ 0 \ 1 \ 0, \dots, 0]^T$ for an input pattern \mathbf{a} belonging to the class 3.

Using the training set data, which consists of a set of input vectors and the desired class labels, the output weights w_{ij} can be determined by any of the following methods:

1. Determination of weights by matrix inversion:

For the *I*th pattern pair, the error between the desired and actual outputs is given by

$$\mathbf{E}_I = \sum_{i=1}^N \left[d_{iI} - \sum_{j=1}^H w_{ij} \phi_j(\mathbf{a}_I) \right]^2 \quad (7.43)$$

The total error $\mathbf{E} = \sum \mathbf{E}_I$ is minimized to determine the optimum weight matrix. This requires computation of pseudoinverse of a matrix and uses all the training data in the computation of the matrix.

2. Determination of weights by learning using LMS algorithm: Since the output units are assumed linear, the instantaneous error can be used to adjust the weights as in the LMS algorithm. That is,

$$\Delta w_{ij} = -\eta \left[d_i - \sum_{j=1}^H w_{ij} \phi_j(\mathbf{a}_i) \right] \quad (7.44)$$

where η is the learning rate parameter.

For the optimum weight matrix, the network output $F_i(\mathbf{a})$ from the i th unit is an approximation to the conditional expectation $\mathbf{E}[d_i | \mathbf{a}]$ in the mean squared error minimizing sense. The conditional expectation $\mathbf{E}[d_i | \mathbf{a}]$ in turn is equal to the probability $P(C_i | \mathbf{a})$ [White, 1989; Richard and Lippmann, 1991].

Thus a trained radial basis function for classification gives as output the a posteriori probabilities $P(C_i | \mathbf{a})$, $i = 1, 2, \dots, N$, for a given input vector. The class C_k for which $P(C_i | \mathbf{a})$ is maximum for all i , is the class to which the input vector \mathbf{a} belongs.

We have noted that the basis function networks provide several advantages over the **multilayer** feedforward neural networks. The main advantage is that the training of the basis function networks is much faster than the MLFFNN. This is because the basis function networks are developed specifically for the tasks such as function approximation or pattern classification, instead of arbitrary mapping that is sought to be achieved by the MLFFNN. The first layer of the basis function network involves computation of the nonlinear basis function values for each new input vector in order to determine the outputs of the hidden units. These computations generally take much more time than for the linear basis function (inner product) computations in a MLFFNN. Thus the pattern recall takes more time for the basis function networks.

There are other types of networks where the training is completely avoided. They are called Generalized Regression Neural Networks (**GRNN**) for function approximation tasks and Probabilistic Neural Network (**PNN**) for pattern classification tasks [Specht, 1991; Specht, 1988; Specht, 1990; Wasserman, 1993]. Both of them typically use as many hidden units as there are training input patterns. These networks are similar to the basis function networks, except that there is no **training involved**. GRNN is based on nonlinear regression theory and can be designed to approximate any continuous function [Specht, 1991]. On the other hand, PNN is based on Bayesian classification theory and uses Parzen windows to approximate the probability distribution of the input pattern [Parzen, 1962]. GRNN finds the regression estimate, i.e., the expected value of the output of the network given the input vector. This can be shown to be an optimal estimate in the mean squared sense. Any estimate that is optimal in

the mean squared sense also approximates a Bayesian classifier [Geman et al, 1992; Richard and Lippmann, 1991]. Thus GRNN and the PNN can be related.

The effectiveness of all the basis function networks depends on the choice of suitable windows or basis functions with appropriate values for the spreads. This is because of the dependence of the networks on the local nature of the input space. In contrast, the MLFFNN captures the global information. Any attempt to make the choices of the windows optimal increases the training time due to the optimization process involved in determining the number of clusters, cluster centres and their spreads.

7.3.6 Counterpropagation Network

In a multilayer feedforward neural network the training process is slow, and its ability to generalize a pattern mapping task depends on the learning rate and the number of units in the hidden layer. In the use of radial basis functions the unsupervised part of the learning involves determination of the local receptive field centres and the spread in the input data corresponding to each hidden unit. The centres are determined using a vector quantization approach. This could be done either by computation or by learning from the input data. On the other hand, a different pattern mapping strategy, namely counterpropagation, uses winner-take-all instar learning for the weights from the units in the input layer to the units in the hidden layer. The counterpropagation network (CPN) provides a practical approach for implementing a pattern mapping task, since learning is fast in this network [Hecht-Nielsen, 1987; Hecht-Nielsen, 1988]. The network (Figure 7.9) consists of two feedforward networks with a common hidden layer. The feedforward network formed by

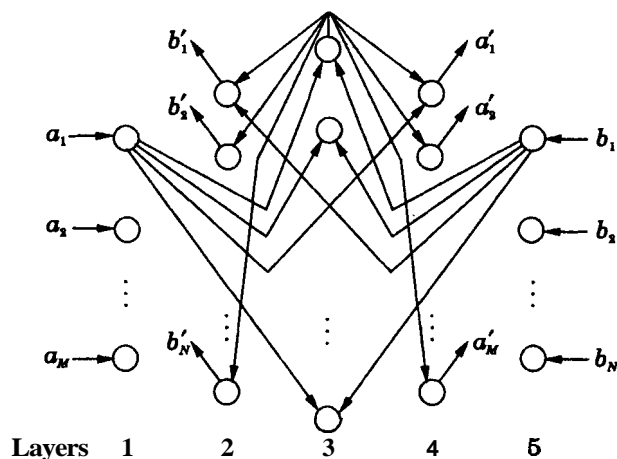


Figure 7.9 Counterpropagation network.

layers 1, 3 and 2 is used for forward mapping and the network formed by layers 5, 3 and 4 is used for inverse mapping (if it exists) between the given input-output pattern pairs. Each **feedforward network** uses a combination of **instar** and **outstar** topologies. The first and second (hidden) layers of a **feedforward network** form a competitive learning system and the second (hidden) and third layers form an **outstar** structure. Learning takes place in the **instar** structure of the competitive learning system to code the input patterns \mathbf{a}_l and in the **outstar** structure to represent the output patterns \mathbf{b}_l . The training of the **instar** and **outstar** structures are as follows:

Training Instars of CPN:

1. Select an input vector \mathbf{a}_l from the given training set $(\mathbf{a}_l, \mathbf{b}_l), l = 1, 2, \dots, L$.
2. Normalize the input vector and apply it to the CPN competitive layer.
3. Determine the unit that wins the competition by determining the unit k whose vector \mathbf{w}_k is closest to the given input.
4. Update the winning unit's weight vector as

$$\mathbf{w}_k(m + 1) = \mathbf{w}_k(m) + \eta (\mathbf{a}_l - \mathbf{w}_k(m)).$$

5. Repeat Steps 1 through 4 until all input vectors are grouped properly by applying the training vectors several times.

After successful training the weight vector **leading** to each hidden unit represents the average of the input vectors corresponding to the group represented by the unit.

Training outstars of CPN:

1. After training the **instars** apply a normalized input vector \mathbf{a} , to the input layer and the corresponding desired output vector \mathbf{b} , to the output layer.
2. Determine the winning unit k in the competitive layer.
3. Update the weights on the connections from the winning competitive unit to the output units

$$\mathbf{v}_k(m + 1) = \mathbf{v}_k(m) + \eta (\mathbf{b}_l - \mathbf{v}_k(m)).$$

4. Repeat Steps 1 through 3 until all the vector pairs in the training data are mapped satisfactorily.

After successful training the **outstar** weight vector for each unit in the hidden competitive layer represents the average of the subset of the output vectors corresponding to the input vectors belonging to that unit.

Depending on the number of units in the hidden layer, the

network *can* perform any desired **mapping** function. In the extreme case, if a unit is provided in the hidden layer for each input pattern, then any arbitrary mapping $(\mathbf{a}_i, \mathbf{b}_i)$ can be realized. But in such a case the network fails to generalize. It merely stores the pattern pair. By using a small number of units in the hidden layer, the network can accomplish data compression. Note also that the network can be trained to capture the inverse mapping as well, i.e., $\mathbf{a}_i = \phi^{-1}(\mathbf{b}_i)$, provided such a mapping exists and it is unique. The name *counterpropagation* is given to this architecture due to the **network's** ability to learn both forward and inverse mapping functions.

7.4 Stability-Plasticity Dilemma: ART

Many pattern mapping networks can be transformed to perform **pattern** classification or category learning tasks. However these networks have the disadvantage that during learning the weight **vectors** tend to encode the **presently** active pattern, thus weakening the traces of patterns it had already learnt. Moreover, any new pattern that does not belong to the categories already learnt is still forced into one of them using the best match strategy, without taking into account how good even the best match is. The lack of stability of weights as well the inability to accommodate patterns belonging to new categories, led to the proposal of new **architectures** for pattern classification. These architectures are based on adaptive resonance theory (ART) and **are** specially designed to take care of the so called *stability-plasticity dilemma* in pattern classification [Carpenter and Grossberg, 1988].

ART also uses a combination of instar-outstar networks as in the CPN, but with the output layer merged with the input layer, thus forming a two-layer network with feedback as shown in Figure 7.10. The minimal ART network includes a bottom-up competitive learning

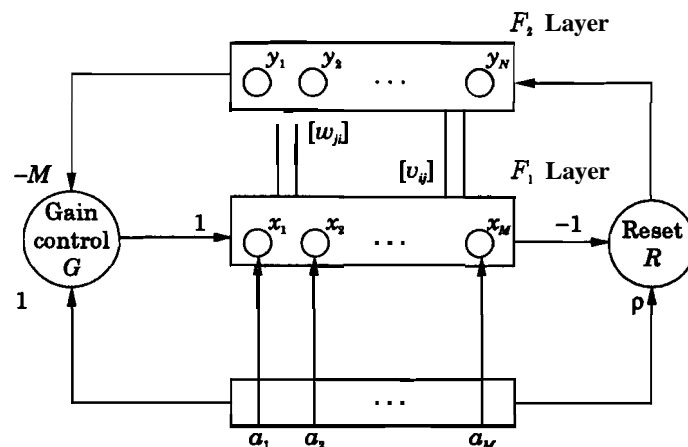


Figure 7.10 ART network.

system (F_1 to F_2) combined with a top-down (F_2 to F_1) **outstar** pattern learning system. The number of units in the F_2 layer determines the number of possible categories of the input patterns. When an input pattern \mathbf{a}_i is presented to the F_1 layer, the system dynamics initially follows the course of competitive learning, leading to a winning unit in the competitive F_2 layer depending on the past learning of the adaptive weights of the bottom-up connections from F_1 to F_2 . The signals sent back **from** the winning unit in the F_2 layer down to F_1 via a **top-down outstar** network correspond to a prototype vector. This prototype vector is compared to the input pattern vector at the F_1 layer. If the two vectors match well, then the **winning** unit **in** the F_2 layer determines the category of the input pattern. If the match is poor, **as** determined by a vigilance parameter, then the winning unit in the F_2 layer does not represent the proper class for the input pattern \mathbf{a} . That unit is removed from the set of allowable winners in the F_2 layer. The other units in the F_2 layer are likewise skipped until a suitable match is obtained at the F_1 layer between the top-down prototype vector and the input **vector**. When a match is obtained, then both the bottom-up and top-down network weights are adjusted to reinforce the input pattern. If no match is obtained, then an uncommitted unit (whose category is not identified during training) in the F_2 layer is committed to this input pattern, and the corresponding weights are adjusted to reinforce the input. The above sequence of events conducts a search through the encoded patterns associated with each category, trying to find a **sufficiently** close match with the input pattern. If no category exists, a new category is made. The search process is controlled by two subsystems, namely the orienting subsystem and the attentional subsystem. The orienting subsystem uses the dimensionless vigilance parameter that establishes the criterion for deciding whether the match is good enough to accept the input pattern **as** an exemplar of the chosen category. The gain control process in the attentional subsystem allows the units **in** the F_1 layer to be engaged only when an input pattern is present, and it also actively regulates the learning [Freeman and Skapura, 1991].

Stability is achieved in the ART network due to dynamic matching and control in learning. Plasticity is achieved in the ART due to its ability to commit an uncommitted unit in the F_2 layer for an input pattern belonging to a category different **from** the categories already learnt.

In ART information from units reverberates back and forth between two layers. Once the proper patterns develop, the neural network can be said to be in resonance. **During** this resonance period the adaptive weights are adjusted. No learning takes place before the network reaches a resonant state.

ART1 network was proposed to deal with binary input patterns [Carpenter and Grossberg, 1988]. The algorithm for binary valued **ART1** is as follows [Bose and Liang, 1996]:

w_{ji} is the weight from the i th unit in the F_1 layer to the j th unit in the F_2 layer.

\mathbf{w}_j is the weight vector leading to the j th unit in the F_2 layer from all units in the F_1 layer.

v_{ij} is the weight from the j th unit in the F_2 layer to the i th unit in the F_1 layer.

\mathbf{v}_j is the weight vector (prototype vector) emanating from the j th unit in the F_2 layer to all the units in the F_1 layer.

Initially set all the components of all the prototype vectors to 1. That is $v_{ij} = 1$, for all i and j . This will enable the uncommitted units in the F_2 layer also to compete in the same way as the learned units.

Initialize all w_{ij} s to random values in the range 0 to 1.

1. Enable all the units in the F_2 layer.
2. For an input binary pattern \mathbf{a} to the F_1 layer, determine the winner unit k in the F_2 layer by computing

$$k = \arg [\max_j \mathbf{w}_j^T \mathbf{a}] .$$

3. A similarity measure **between** the winning prototype \mathbf{v}_k and the input \mathbf{a} is computed and compared with a vigilance parameter ($0 < \rho < 1$). The similarity measure gives the **fraction** of bits of \mathbf{a} that are also present in \mathbf{v}_k . That is

$$\frac{\mathbf{v}_k^T \mathbf{a}}{\sum_{i=1}^M a_i} \geq \rho$$

Once the prototype associated with the **winner** unit k passes the vigilance test, then go to Step 4 to adjust the weight vectors associated with the k th unit both in the forward and backward directions.

If the vigilance test fails, then the output unit k is disabled and another winner is selected by repeating Steps 2 and 3.

If none of the committed units in the F_2 layer passes the vigilance test, then an uncommitted unit is committed to the input pattern and the corresponding prototype vector \mathbf{v}_k is set equal to the input pattern \mathbf{a} . That is $\mathbf{v}_k = \mathbf{a}$.

4. The weights **are** adjusted as follows:

$$v_{ik}(m+1) = v_{ik}(m) \wedge a_i, \quad i = 1, 2, \dots, M$$

where \wedge is the logical AND operation and

$$w_{ki}(m+1) = \frac{v_{ik}(m+1)}{0.5 + \sum_{i=1}^M v_{ik}(m+1)}, \quad i = 1, 2, \dots, M$$

$\mathbf{w}_k(m+1)$ can be viewed as normalized version of $\mathbf{v}_k(m+1)$,

normalized with the number of **1**'s in the vector. The factor 0.5 in the denominator is used to avoid division by zero. With this choice of \mathbf{w}_k the inner product $\mathbf{w}_k^T \mathbf{a}$ computed in Step 2 can be interpreted as the fraction of bits of the prototype vector \mathbf{v}_j that are in the input vector \mathbf{a} also. Thus the **winner-take-all decision** selects the winner unit that corresponds to a prototype vector that has maximum number of bits matching with the bits in the input vector \mathbf{a} .

In implementation the **ART1** network operates automatically through the use of the gain parameter (\mathbf{G}) and the reset parameter (\mathbf{R}).

The gain control unit operates as follows: If all the units in the \mathbf{F}_2 layer are OFF, then $G = 1$. If one of the units in the \mathbf{F}_2 layer is ON, then $G = 0$. The gain parameter G is generated using the function

$$G = f \left(\sum_{i=1}^M a_i - M \sum_j y_j - 0.5 \right) \quad (7.45)$$

where $f(x) = 1$, if $x > 0$, and $f(x) = 0$, if $x \leq 0$. The quantity y_j is the output of the j th unit in the \mathbf{F}_2 layer and is either **1** or **0** depending on whether the unit j is a winner or not. If all units in the \mathbf{F}_2 layer are OFF, then $y_j = 0$, for $j = 1, 2, \dots, N$. Assuming that there is at most one nonzero component in the input vector \mathbf{a} , the argument of $f(\cdot)$ is greater than **0**. Hence $G = 1$.

For any winning unit in the \mathbf{F}_2 layer, one of the y_j s will be **1**. Then the argument of $f(\cdot)$ is less than **0**. Hence $G = 0$.

The output of the i th unit in the \mathbf{F}_1 layer is given by

$$x_i = f \left(a_i + \sum_{j=1}^N v_{ij} y_j + G - 1.5 \right) \quad (7.46)$$

This computes the output of the i th unit in the \mathbf{F}_1 layer as $x_i = a_i \wedge v_{ik}$ if the unit k is the winner, since $y_k = 1$ and $y_j = 0$, for $j \neq k$, and also $G = 0$. If none of the units in the \mathbf{F}_2 layer are ON, then $y_j = 0$, for all j and $G = 1$, and hence $x_i = a_i$, for all i . Thus equation (7.46) represents a **2/3 rule** since $x_i = 1$ if any two out of the three variables in the argument are **1**.

The reset value R is computed as follows:

$$R = f \left(\rho \sum_{i=1}^M a_i - \sum_{i=1}^M x_i \right) \quad (7.47)$$

If the vigilance test

$$\frac{\sum_i x_i}{\sum_i a_i} = \frac{\sum_i a_i \wedge v_{ik}}{\sum_i a_i} = \frac{\mathbf{v}_k^T \mathbf{a}}{\sum_i a_i} > \rho \quad (7.48)$$

Architectures for Complex Pattern Recognition Tasks

succeeds, then the argument of $f(\cdot)$ will be negative, and hence $R = 0$. That is there is no reset. On the other hand, if the vigilance test fails, then the argument of $f(\cdot)$ is positive and hence $R = 1$. Then the current winning unit is disabled and all the units in the F_2 layer are reset to OFF. Hence $G = 1$. Therefore another winning unit will be selected.

Figure 7.11 illustrates the clustering performance of **ART1** for 24 binary patterns each having 4×4 pixels. It can be seen that lower vigilance $\rho = 0.5$ case produces fewer clusters than the larger vigilance $\rho = 0.7$ case. ART2 network was developed to self-organize recognition categories for analog as well as binary input patterns [Carpenter and Grossberg, 1987; Carpenter et al, 1991b]. Figure 7.12 illustrates the clustering of analog signals by an ART2 network. Here 50 patterns, each of 25-dimensional vector of analog values are clustered for two values of the vigilance parameter ρ . As expected smaller vigilance value produces fewer clusters.

A minimal ART network can be embedded in a larger system to realize an associate memory. A system like CPN or multilayer feedforward network directly maps pairs of patterns $(\mathbf{a}_i, \mathbf{b}_i)$ during learning. If an ART system replaces the CPN, the resulting system becomes self-stabilizing. Two ART systems can be used to pair sequences of the categories self-organized by the input sequences. The pattern recall can occur in either direction during performance as in BAM. This scheme brings to the associate memory paradigm the code compression capabilities, as well as the stability properties of ART [Carpenter, 1989].

ART3 network was developed for parallel search of distributed recognition codes in a multilevel network hierarchy [Carpenter and Grossberg, 1990]. All these three ART models are based on unsupervised learning for adaptive clustering. On the other hand, **ARTMAP** architecture performs supervised learning by mapping categories of one input space onto categories of another input space, and both the sets of categories are determined by two separate ART systems [Carpenter et al, 1991a]. Fuzzy **ARTMAP** extends the ideas of **ARTMAP** to include additional knowledge in the form of production rules and fuzzy logic [Carpenter et al, 1991c; Carpenter and Grossberg, 1996].

Note that ART models belong to the class of match-based learning as opposed to error-based learning of the backpropagation networks. In match-based learning the weights are adjusted only when the external input matches one of the stored prototypes, whereas in error-based learning the weights are adjusted only if there is an error between the actual output and the desired output. Thus match-based learning tends to group similar patterns whereas error-based learning tends to discriminate dissimilar patterns.

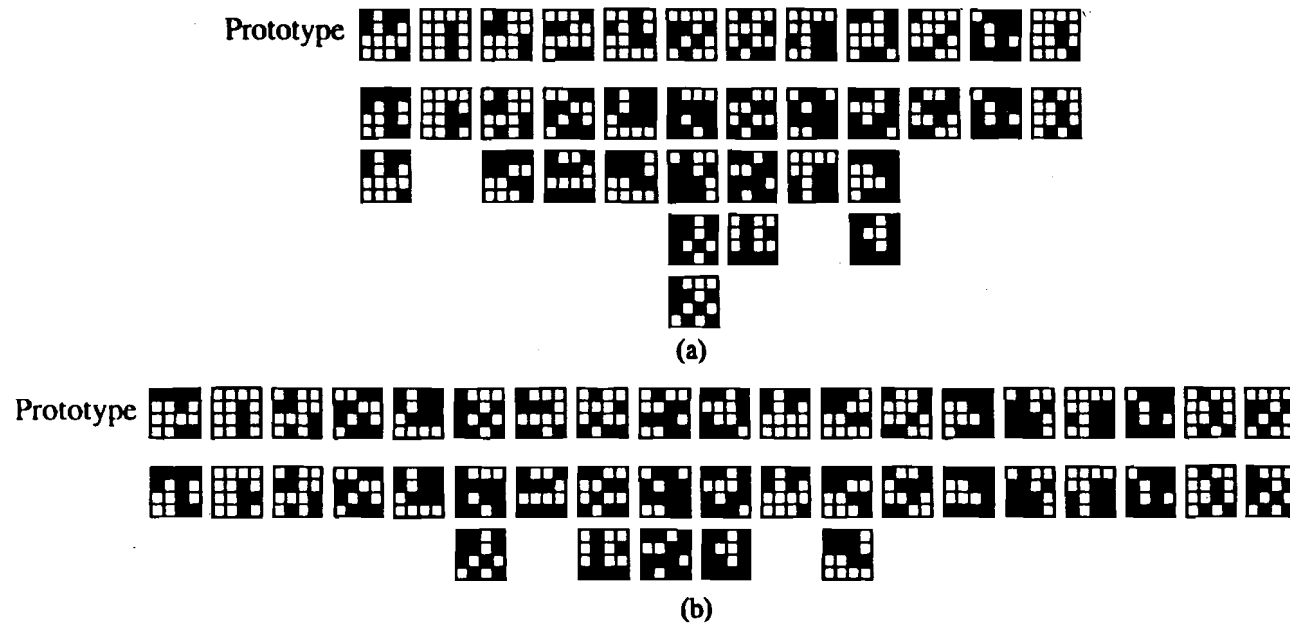


Figure 7.11 Clustering of random binary patterns by ART1 network for two different values of the vigilance parameter. (a) $\rho = 0.5$ and (b) $\rho = 0.7$. The top row in each case shows the prototype patterns extracted by the ART1 network [Adapted from Hassoun, 1995].

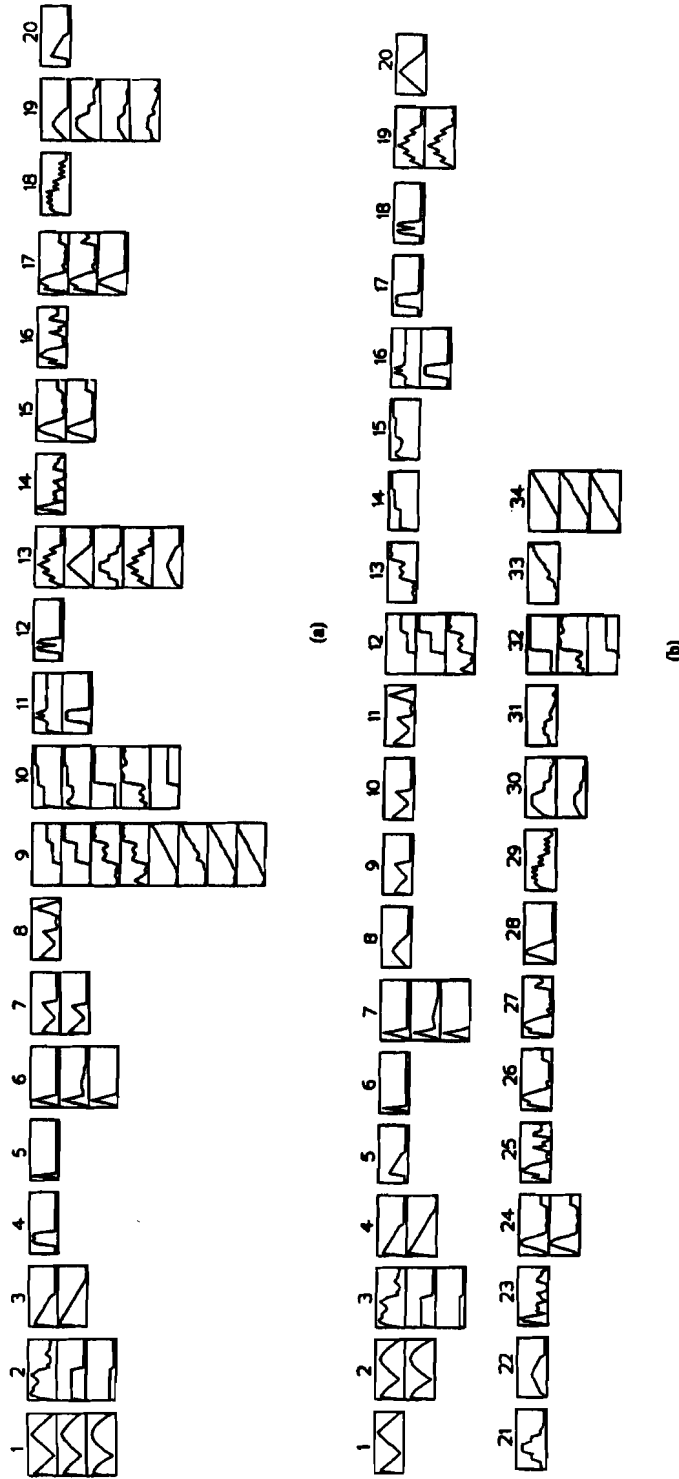


Figure 7.12 Clustering of analog signals by ART2 network for two different values of the vigilance parameter. The vigilance parameter value for (a) is smaller than the value for (b). [Adapted from Hassoun, 1995].

7.5 Temporal Patterns

The ANN architectures described so far are applicable for recognition of patterns on the basis of information contained within the pattern itself. Even if a sequence of patterns with temporal correlations is presented, the previous or subsequent patterns have no effect on the classification of the current input pattern. But there are many applications (for example, speech recognition) where it is necessary to encode the information relating to the time correlation of spatial patterns, as well as the spatial pattern information itself.

In a temporal pattern the ordering among the components in the sequence is important. The components themselves may be **fixed/rigid** like printed text symbols, or they may be varying naturally due to production and context as in the **case** of sound units in speech or symbols in a cursive script. Temporal patterns could be very complex depending on the extent of influence of the context and the inherent variability of each component. In this section we consider simple **temporal** pattern sequences in which each component is of fixed duration, and it depends only on the components adjacent to it.

There are three types of problems involving temporal sequences [Hertz et al, 1991]:

(a) **Sequence recognition** in which the objective is to determine the class label of a given temporal pattern. This is like the standard pattern classification task performed by a multilayer **feedforward** neural network.

(b) **Sequence reproduction** in which the desired temporal pattern is generated from a partial input of the pattern. This is like an autoassociation task in the feedback neural networks. This can also be viewed as a pattern completion task. One can also interpret prediction of time-series data as a sequence reproduction task.

(c) **Temporal association** in which the desired sequence is generated as an output in response to a given input sequence. **This** can be viewed as generalization of the **heteroassociation** task for temporal sequences.

Architectures for temporal pattern recognition tasks have evolved from the well-understood principles of multilayer feedforward and feedback neural networks. In order to use models based on these known architectures, it is necessary to represent the temporal pattern as a static spatial pattern. For this representation, delay units **are** used to store a fixed number of components belonging to the preceding instants. Thus a temporal pattern is represented using a tapped delay line as shown in the input layer in Figure 7.13. The figure illustrates an architecture for temporal pattern recognition using a **multilayer** feedforward neural network. The disadvantage of this approach is that the length of the sequence has to be fixed **a priori**. Also, a large number of training sample sequences **are** required for learning and

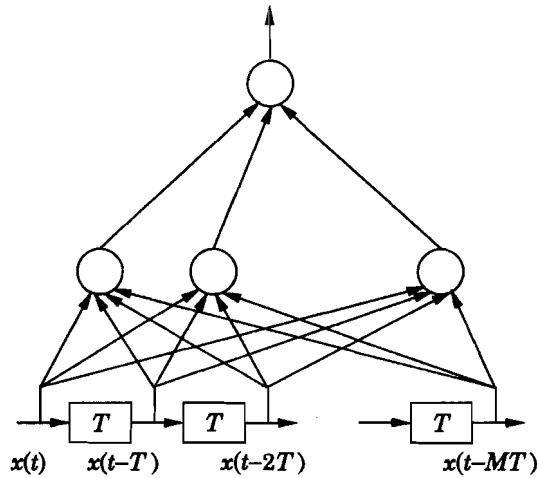


Figure 7.13 A tapped delay neural network with one input and M delays in the input layer, one hidden layer and a single unit output layer.

generalization. Moreover, the input signal must have precise time registration. Many natural signals like speech do not conform to these restrictions.

One of the early architectures proposed for classification of **spatio-temporal patterns (STP)** is based on the Grossberg formal avalanche structure [Grossberg, 1969]. The structure of the network shown in Figure 7.14 resembles the top two layers of the CPN, and both use multiple **outstars** [Freeman and Skapura, 1991]. The avalanche architecture shows how a complex spatio-temporal pattern can be learned and recalled. Let $\mathbf{a}(t) = (a_1(t), a_2(t), \dots, a_M(t))$ be the spatial pattern required at time t . The sequence of $\mathbf{a}(t)$ at time intervals of Δt in the range $t_0 \leq t \leq t_1$ correspond to the desired spatio-temporal pattern. The unit labelled t_0 is activated and $\mathbf{a}(t_0)$ is applied, which

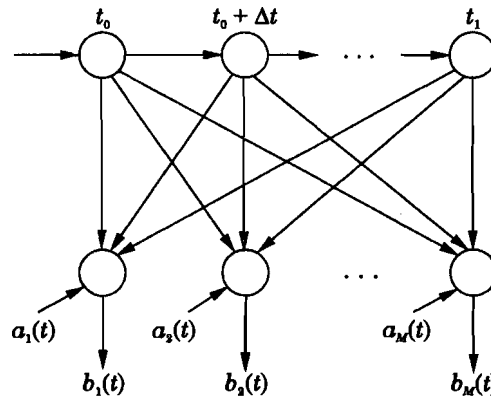


Figure 7.14 Avalanche architecture.

is to be learned by the outstar's output units. The second pattern $\mathbf{a}(t_0 + \Delta t)$ is applied while activating the second **outstar**, labelled $t_0 + \Delta t$. This process is continued by activating successive **outstars** until all the patterns in the sequence have been learned. Replay of the learned sequence can be initialized by stimulating the t_0 unit, while a zero vector is applied to the \mathbf{a} inputs. The output sequence $\mathbf{b}(t) \approx \mathbf{a}(t)$, for $t_0 \leq t \leq t_n$, is the learned sequence.

More sophisticated time delay neural network architectures were proposed for recognition of speech patterns [Waibel, 1989]. These will be discussed in Chapter 8. Once the temporal pattern is represented as a static pattern, a recognition system can be developed by template matching using principles of competitive learning or **self-organization**. Kohonen's phonetic typewriter is an example of such an architecture, which will be described in Chapter 8 [Kohonen, 1988].

Tank and **Hopfield** [1987a; 1987b] proposed an associative memory based approach for temporal pattern recognition using the exponential kernels representation of temporal patterns. This representation replaces the **fixed** delays with filters that broaden the signal duration in time as well as delaying it. Figure 7.15 shows four

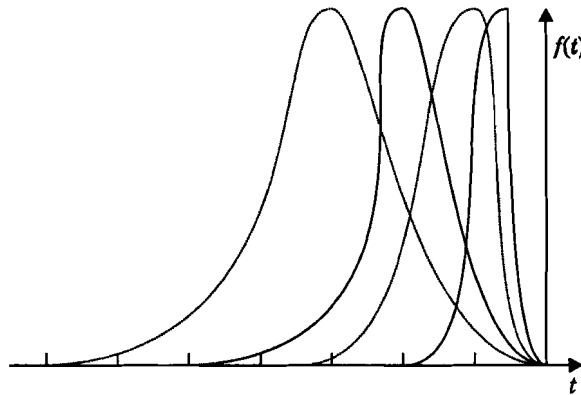


Figure 7.15 Four time reversed exponential kernel functions which are used to window the time signal $x(t)$. The network input at time t for a four delay network are averages of the past signal weighted by these functions.

typical exponential kernels for four delays. The network inputs at time t for a four delay network are averages of the past signal weighted by these functions. This representation is more robust and can handle speech-like signals.

Recurrent network models are more natural models to deal with temporal patterns. But training will be a problem with these models. Several partially recurrent models were proposed in the literature [Elman, 1990; Jordan, 1986; Stornetta et al, 1988; Mozer, 1989]. The connections are mostly feedforward with a few selected fixed feedback

connections so as to keep the training within manageable complexity. Thus the recurrent part is realized with context units in different configurations. The context units receive the feedback signals as shown in Figure 7.16 for the configuration proposed by Jordan [1986].

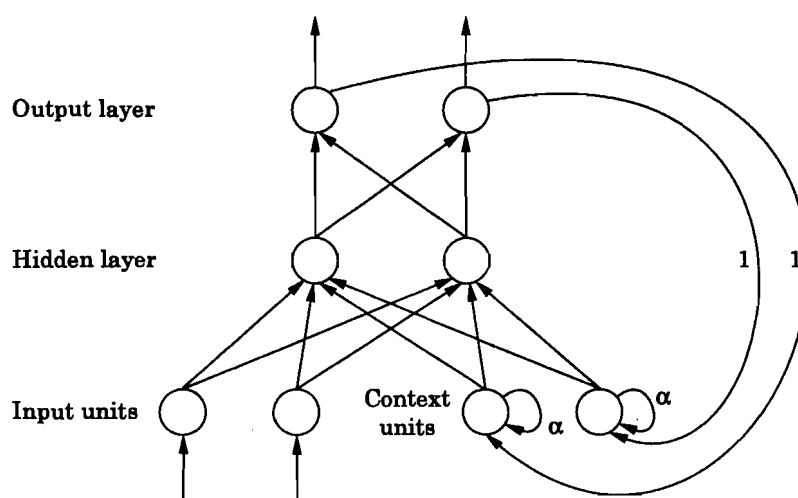


Figure 7.16 Architecture with context units to receive feedback signals. Only two units are considered for each of the input, feedback, hidden and output layers for illustration.

The input layer consists of two parts. One part (input units) receives external inputs and the other part (context units) receives feedback from output units with unit gain. There is also a self-feedback with gain $\alpha < 1$ on the context units so that the inputs to the hidden layer units from the context units have exponentially decaying memory of the past. Therefore the output of the i th context unit $C_i(t)$ is given by

$$\begin{aligned} C_i(t) &= o_i(t-1) + \alpha o_i(t-2) + \alpha^2 o_i(t-3) \dots \\ &= \sum_{\tau=0}^{t-1} \alpha^{t-1-\tau} o_i(\tau) \end{aligned} \quad (7.49)$$

where $o_i(t)$ is the output of the i th unit in the output layer at time t . Thus the context units accumulate the weighted average of the past **output** values. With a fixed input pattern the network can be trained using backpropagation learning to generate a desired output sequence. Thus different fixed input patterns can be associated with different output sequences [Jordan, 1986; Jordan, 1989]. By applying a sequence of patterns at the input, one at a time, and a fixed output for each sequence of the inputs, the network can be trained to distinguish different input sequences. Thus temporal pattern

recognition can be achieved. Anderson et al, [1989] have studied the problem of recognizing a class of English syllables using this network.

Partially recurrent networks have been proposed for time-series prediction which involves prediction of the future patterns based on the patterns learnt from the past data. In these cases the network is designed to capture the pattern behaviour embedded in the past data. These ideas have been applied in several forecasting situations such as in the case of financial markets [Weigend and Gershenfeld, 1993; Lapedes and Farber, 1988; Weigend et al, 1991].

Ideas based on time-series prediction have also been exploited for identification of nonlinear dynamical systems using partially recurrent networks [Narendra and Parthasarathy, 1990]. The nonlinear plant dynamics is given by

$$\mathbf{x}(t+1) = \mathbf{g}[\mathbf{x}(t), \mathbf{x}(t-1), \dots, \mathbf{x}(t-n); \mathbf{u}(t), \mathbf{u}(t-1), \dots, \mathbf{u}(t-m)]$$

where $m \leq n$, and $\mathbf{u}(t)$ and $\mathbf{x}(t)$ are the input and output signals of the plant at t , respectively. The function $\mathbf{g}(\cdot)$ is a nonlinear function representing the dynamics of the plant. The network shown in Figure 7.17 is trained with backpropagation learning using the actual

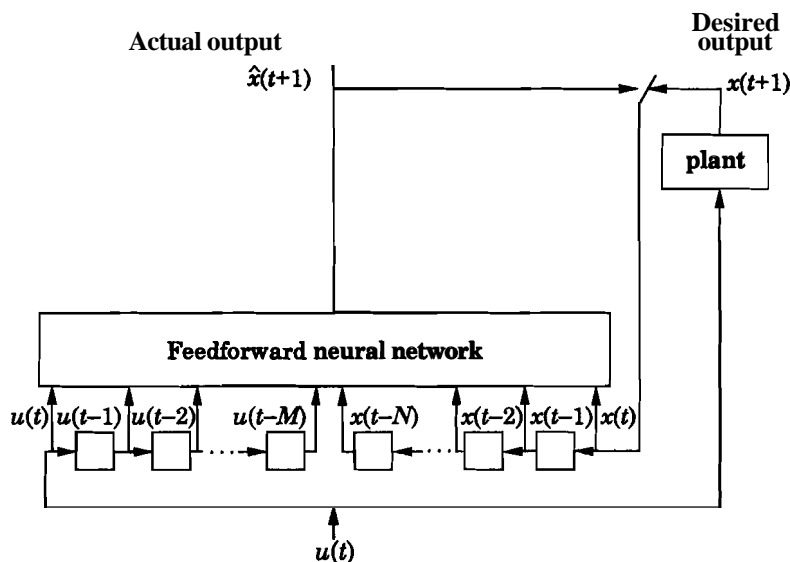


Figure 7.17 Partially recurrent neural network for identification of nonlinear dynamical system.

output from the plant. During training the same input is given to the plant as well as to the network. If the network has generalized from the training data, then for an input $\mathbf{u}(t)$ it produces an output $\hat{\mathbf{x}}(t+1)$ which is almost close to the actual output, thus predicting the plant's output.

Fully recurrent networks are more efficient in terms of number

of units, in order to realize temporal association tasks. Here the individual units may represent input units, output units or both. The desired outputs are specified on some units at some predetermined time instants. A two-unit fully recurrent network is shown in Figure 7.18 with the unit 1 as input unit and the unit 2 as the **output** unit. The desired output is specified on the unit 2.

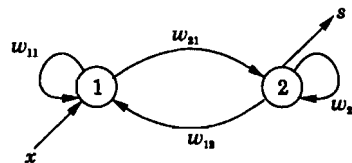


Figure 7.18 A two unit recurrent network.

If sequences of small lengths (P) (measured in time units) are involved, then the recurrent network **may** be unfolded into a feedforward network with P layers as shown in Figure 7.19 for

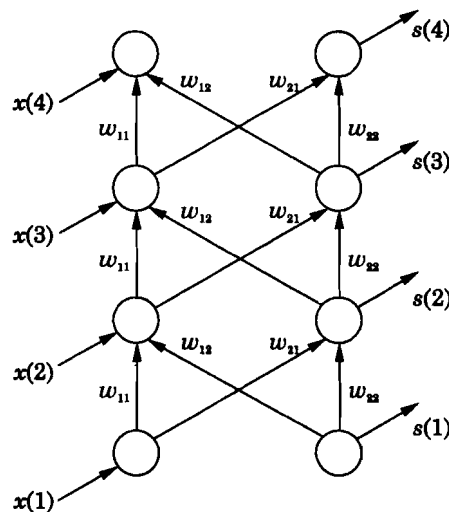


Figure 7.19 Feedforward network generated by unfolding a recurrent network in time by four time units.

$P = 4$. In this case the desired outputs are specified for units in the hidden layers also. Moreover, the **errors** are propagated not only from the outputs of the final layer but also from the outputs of the hidden layers as well. It should also be noted that the weights are copied for different layers. The average increment of all the corresponding weights is used for updating. This is called backpropagation-through-time learning method [Rumelhart et al, 1986]. This is not a very efficient method for long sequences. One interesting application of backpropagation-through-time is the truck backer-upper problem

described in [Rumelhart et al, 1986], in which the goal is to design a controller that successfully backs up a truck so that the back of the **trailer** designated by the (x, y) coordinates ends at $(0, 0)$ with the trailer perpendicular to the dock, when only backward movements of the truck are allowed.

Williams and Zipser [1989] proposed a real time recurrent learning method for on-line learning of the time sequences. It **can thus** deal with sequences of arbitrary length. It was shown that the real time recurrent network can be **trained** to be a **flip-flop** or even a **finite** state machine. **Finally**, Pearlmutter [1989] developed an algorithm for training a continuous time recurrent network. It can be viewed as a continuous time extension of backpropagation-through-time learning.

7.6 Pattern Variability: Neocognitron

Visual pattern recognition, such as recognition of handwritten characters or hand-drawn figures, is done effortlessly by human beings despite variability of features in different realizations of the pattern of the same character or figure. The patterns considered in the architectures described so far assume that the objects in the training and test patterns are identical in size, shape and position, except that in some cases there may be some noise added or some portions of the pattern missing. Models of associative memory can recover complete patterns **from** such imperfections, but normally cannot work if there is variability or deformation in the patterns of the test input.

Neural network models based on our understanding of human visual pattern recognition tend to perform well even for shifted and deformed patterns. In the visual area of the cerebrum, neurons respond selectively to local features of a visual pattern such as lines and edges. In areas higher than the visual context, cells exist that respond selectively to certain figures like circles, triangles, squares, human faces, etc [Fukushima, 1975]. Thus the human visual system seems to have a hierarchical structure in which simple features are first extracted from the stimulus pattern, then integrated into more complicated ones. A cell at a higher stage generally receives signals from a wider area of the retina and is less sensitive to the position of the stimulus. Within the hierarchical structure of the visual system are forward (afferent or bottom-up) and backward (efferent or top-down) propagation of signals. This kind of physiological evidence suggests a neural network structure for modelling the phenomenon of visual pattern recognition.

The objective is to synthesize a 'neural network model for pattern recognition for shifted and deformed patterns. The network model learns with a teacher (supervised learning) for reinforcement of the adaptive weights. The network model is called **neocognitron**. It is a hierarchical network (Figure 7.20) consisting of many layers of cells,

Architectures for *Complex* Pattern Recognition Tasks

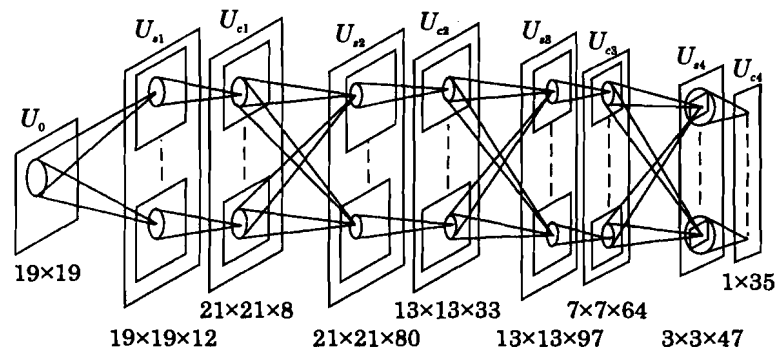


Figure 7.20 Neocognitron architecture. A hierarchical structure of neocognitron for recognition of alphanumeric characters. The first stage of the network consists of a 2-dimensional array of receptor cells. Each succeeding stage has layers consisting of S cells and C cells alternatively. Each layer is organized into groups of these cells, each group responding to a particular geometrical position. The numbers show the total numbers of S and C cells in individual layers of the network S cells are feature extracting cells. The C cells are inserted to allow for positional errors in the feature. [Adapted from Fukushima et al, 1991].

and has variable connections between cells in adjoining layers. It can be trained to recognize any set of patterns. **After** training, pattern recognition is performed on the basis of similarity in shape between patterns, and the recognition is not **affected** by deformation, or changes in size, or shifts in the positions of the input patterns [Fukushima, 1988].

In the hierarchical network of the neocognitron, local features of the input pattern are extracted by the cells of the lower stage, and they are gradually integrated into more global features. Finally, each cell of the highest stage integrates all the information of the input pattern, and responds only to one specific pattern. During the process of extracting and integrating features, errors in the relative positions of the local features are gradually tolerated. The operation of tolerating positional error a little at a time at each stage, rather than **all** in one step, plays an important role in endowing the network with the ability to recognize even distorted patterns [Fukushima et al, 1991].

Neocognitron also provides backward connections which will enable it to realize the selective attention feature of the visual pattern recognition system. The selective attention feature relates to two or more patterns simultaneously present in the data, and our ability to focus on the desired one.

Neocognitron was developed for recognition of handwritten characters, although the ideas used in the architecture may be extended to other situations of pattern variability [Fukushima et al, 1991].

Summary

7.7 Summary

The objective of this chapter is to highlight the need for evolving architectures specific to particular tasks. In this context we have discussed neural network architectures for five classes of pattern recognition tasks, namely, associative memory, pattern mapping, stability-plasticity dilemma, temporal patterns and pattern variability. These architectures use the well understood principles of models of neurons, their interconnections and network dynamics. The bidirectional associative memory is similar in principle to the **Hopfield** model. The extension of these principles to multidirectional and temporal associative memories is straightforward. Pattern mapping task is one of the well studied pattern recognition tasks in neural network studies. We have also highlighted the fact that all pattern mapping problems are not generalizable by a neural network architecture. The specific characteristics of generalizable problems are exploited for developing suitable architectures as in the radial basis function networks. It is interesting to note that pattern classification and function approximation tasks automatically lead to radial basis function network architectures.

The adaptive resonance theory networks for stability-plasticity dilemma have evolved over a long period of nearly 20 years, with different networks addressing different situations, such as discrete, analog and fuzzy data situations. It is one of the most sophisticated architectures developed for a variety of problems [Carpenter and Grossberg, 1996; Grossberg, 1996]. We have considered a few simple neural network architectures for temporal pattern recognition as well as generation. More sophisticated architectures **are** needed to exploit the temporal pattern behaviour directly without processing individual frames of data. Finally, the neocognitron architecture for pattern variability task has been discussed briefly. Development of neocognitron structure clearly demonstrates how issues **specific** to a given task need to be addressed.

Review Questions

1. Explain the following with reference to memory in artificial neural networks:
(a) Transient memory, (b) Temporary memory, (c) Short-time memory, and (d) Long-term memory.
2. Distinguish between content-addressable and address-addressable memories.
3. What is an associative memory?
4. What are the requirements of an associate memory?
5. Distinguish between static and dynamic memories.

6. Distinguish between heteroassociative and autoassociative memories.
7. What is a linear association? What are its limitations as an associative memory?
8. What is a recurrent autoassociative memory?
9. How is noise suppression achieved in a recurrent autoassociative memory?
10. What is a Bidirectional Associative Memory? What is meant by 'BAM is unconditionally stable'?
11. Explain the following terms with reference to an autoassociative memory:
 - (a) Storage, (b) Encoding, (c) Retrieval, (d) Stability, and (e) Performance
12. What is meant by synchronous and asynchronous update in BAM?
13. What is an adaptive BAM?
14. What is a MAM? Explain why MAM will have superior performance over BAM for pattern retrieval.
15. What is a temporal associative memory? What are its limitations in recalling a sequence of temporal patterns?
16. Explain the distinction between
 - (a) pattern association and pattern classification tasks.
 - (b) pattern classification and function approximation tasks.
17. What is meant by generalization in the context of (a) pattern classification and (b) function approximation tasks? Illustrate with examples.
18. Why is it that any arbitrary pattern association task does not fall under the category of generalizable problems?
19. What is meant by (a) surface features and (b) deep features?
20. Why a general MLFFNN is not likely to generalize a problem always?
21. Explain the concept of 'closeness' of data and 'smoothness' of a mapping function.
22. Explain why is it that an MLFFNN does not take closeness of data into account.
23. Give the **architecture** of a basis function network.
24. What is the significance of the regularization term in the cost function for a function approximation problem?

25. Explain the significance of regularization using constraints on the weights.
26. Explain how the constraint of smoothness is realized by the square integrable derivatives of the mapping function.
27. What is the significance of Green's function?
28. Explain the behaviour of a radial basis function method for function approximation for different values of the regularization parameter.
29. Explain how a pattern classification problem leads to a radial basis function network.
30. What decides the basis functions in a pattern classification problem?
31. Explain the basis for the statement:
A trained radial basis function for classification gives as output the a posteriori probabilities $P(C_i|x)$ of each class for a given input **vector** x .
32. How do you determine the basis **functions** for a given pattern classification task?
33. How do you determine the weights of the output layer of a radial basis **function** network for a given pattern classification problem?
34. Discuss the significance of the number and distribution of clusters on the performance of a pattern classification task.
36. What is a probabilistic neural network? In what way it is different from a basis **function** network?
36. What is a generalized regression neural network? In what way it is **different** from a basis function network for function approximation?
37. What is a counterpropagation network?
38. Explain **the** differences in the performance of multilayer feedforward neural network and counterpropagation network for a pattern mapping task.
39. What is the significance of 'resonance' in ART network?
40. Explain briefly the operation of an ART for binary patterns?
41. Explain the 'gain control' mechanism in ART.
42. Explain how the orienting subsystem works in ART network.
43. What are some extensions of the ART concept?
44. What is a temporal pattern, and in what way it is different from a static pattern?
45. Explain the three categories of problems involving temporal patterns.

46. What is an 'avalanche' architecture?
47. What is the disadvantage of fixed delay neural networks for temporal pattern classification and how is it overcome in an associative memory based approach?
48. What are partially recurrent neural networks?
49. What is meant by backpropagation-through-time?
50. Explain the principle of neocognitron for pattern variability task.

Problems

1. Prove that BAM is unconditionally stable for any binary units.
2. Prove that BAM for binary or bipolar units is stable for asynchronous update of units. (Hint: Convert **BAM** into a feedback network of **Hopfield** type.)
3. ~~Construct an example~~ to show that pattern recall is superior in a **tridirectional** associative memory compared to a bidirectional associative **memory**. (See [Zurada, 1992, p. 3681])
4. Show that the weight vector for a radial basis function network for function approximation task is given by (Eq. (7.36)). (See [Haykin, 1994, p. 258])

$$\mathbf{w} = (\Phi^T \Phi + \lambda \Phi_0)^{-1} \Phi^T \mathbf{d}$$

5. Generate training data for the following two **functions**: $f(x) = \log x$, $1 \leq x \leq 10$ and $f(x) = \exp(-x)$, $1 \leq x \leq 10$. Design a suitable **MLFFNN** with one hidden layer to capture the mapping function **from** the training data. Explain the complexity (in terms of number of hidden **units** of the network) to the function being mapped. (See Haykin, 1994, p.231])
6. Cluster all the 5-bit binary vectors except the all zero vector, $\{00 \dots 0\}^T$, using **ART1** algorithm. Study the effect of vigilance parameter on the resulting clusters.
7. Study the classification performance of a **RBFNN** for the 2-class problem given in the Problem 4.12. Study the performance for two sets of clusters with $H = 20$ and $H = 10$. Choose the centres (\mathbf{t}_i) arbitrarily and the variance of the Gaussian distribution for each cluster as σ^2/M , where σ is the maximum distance between the chosen cluster centres. That is

$$G(\|\mathbf{x} - \mathbf{t}_i\|^2) = \exp\left(-\frac{M}{\sigma^2}\|\mathbf{x} - \mathbf{t}_i\|^2\right)$$

for $i = 1, 2, \dots, N$.

The weight matrix of the output layer of the network is given by

$$\mathbf{W} = (\mathbf{G}^T \mathbf{G} + \lambda \mathbf{G}_0)^{-1} \mathbf{G}^T \mathbf{d}$$

where G is an $M \times H$ matrix of the basis function and G_0 is an $H \times H$ matrix whose elements are given by $g_{ij} = G(\|t_j - t_i\|^2)$. Examine the performance of the network for different values of the regularization parameter $X = 0.01$, $\lambda = 1.0$ and $\lambda = 100$.

8. For binary $\{0, 1\}$ patterns, the weight vectors are obtained by using $(2a_{ij} - 1)$ in place of a_{ij} , and the threshold of a unit is given by $\theta_i = -\frac{1}{2} \sum_{j=1}^M w_{ij}$, where w_{ij} is the weight leading to the i th unit from the j th input and a_{ij} is the j th component of the M -dimensional input pattern.

Determine the weights and discuss retrieval of patterns for the following pattern recognition tasks with binary patterns

- (a) Feedforward *neural* network (pattern association)

$$\text{input } \mathbf{a}_1 = [11110111]^T \quad \mathbf{a}_2 = [10101010]^T$$

$$\text{output } \mathbf{b}_1 = [1010]^T \quad \mathbf{b}_2 = [1111]^T$$

$$\text{Test pattern } \mathbf{t} = [11011001]^T$$

- (b) Feedback neural network (pattern storage)

$$\mathbf{a}_1 = [11110111]^T \text{ and } \mathbf{a}_2 = [10101010]^T$$

$$\text{Test pattern } \mathbf{t} = [11011001]^T$$

- (c) Hamming network (pattern storage) **using** the data given in (b) (see Section 8.2.1)

- (d) Bidirectional associative *memory* using the input-output pattern given in (a).

9. **Train** a MLFFNN to capture the nonlinear **dynamical** system given by [Narendra and Parthasarathy, 1990]

$$x(t+1) = \frac{x(t)x(t-1)x(t-2)u(t-1)[x(t-2)-1] + u(t)}{1 + x^2(t-2) + x^2(t-1)}$$

using **inputs** $u(t)$ generated using samples **uniformly** distributed in the range $[-1, +1]$. Consider a network with two hidden layers of 20 and 10 units with bipolar output functions. The inputs to the network during training are $x(t)$, $x(t-1)$, $x(t-2)$, $u(t)$ and $u(t-1)$. Study the **performance** of the **system** for two learning rates $\eta = 0.1$ and $\eta = 0.3$. Compare the outputs of the model and the system by plotting the outputs for the following input:

$$u(t) = \begin{cases} \sin(2\pi t/250) & \text{for } 0 \leq t \leq 500 \\ 0.8 \sin(2\pi t/250) + 0.2 \sin(2\pi t/25) & \text{for } t > 500 \end{cases}$$

Chapter 8

Applications of ANN

8.1 Introduction

This chapter is devoted to applications of artificial neural network models and some research issues that are being currently addressed in this field. In the applications two different situations exist: (a) where the known neural networks concepts and models are directly applicable, and (b) where there appears to be potential for using the neural networks ideas, but it is not yet clear how to formulate the real world problems to evolve a suitable neural network architecture. Apart from the attempts to apply some existing models for real world problems, several fundamental issues are also being addressed to understand the basic operations and dynamics of the biological neural network in order to derive suitable models of artificial neural networks.

In problems such as pattern classification, associative memories, optimization, vector quantization and control applications, the principles of neural networks are directly applicable. Many real world problems are first formulated as one of these problems, identifying the relation between the parameters from the physical data with the **input/output** data and other parameters describing a neural network. Note that in these cases the ingenuity of the problem solver lies in the formulation part, and several compromises may have to be made in arriving at the formulation. These direct applications are discussed in Section 8.2.

While neural network concepts and models appear to have great potential for solving problems arising in practice, for many such problems the solution by neural networks is not obvious. This is because the problems **cannot** be mapped directly onto an existing (known) neural network architecture. In fact there are no principles guiding us to this mapping. As human beings we seem to perform effortlessly many pattern recognition tasks in speech, vision, natural language processing and decision making, although we do not understand how we do it. For example, in speech our auditory mechanism processes the signal directly in a manner suitable for later

neural processing. To prepare input to an artificial neural network, the speech signal is normally processed in fixed frames of 10–20 msec duration to extract a fixed number of spectral or related parameters. In this process the temporal and spectral features with proper resolution needed for recognition may not have been captured. There is as yet no neural network architecture which could perform the speech pattern recognition task with the same effectiveness as human beings do. Similar comments apply to problems in the visual pattern recognition also. Some of the other areas where human performance could not be matched by the existing neural network architectures are in motor control and decision making. Despite realization of these issues, there are several situations where neural principles have been used successfully. Some of these applications are discussed in Section 8.3.

The most important issue for solving practical problems using the principles of artificial neural networks is still in evolving a suitable architecture to solve a given problem. Neural network research is expanding in its scope to take into account the fuzzy nature of the real world data and reasoning, and the complex (and largely unknown) processing performed by the human perceptual mechanism through the biological neural networks. Some of the current research issues are discussed in Section 8.4. Table 8.1 gives an organization of the topics to be discussed in this chapter.

Table 8.1 Organization of Topics on Applications of Artificial Neural Networks

Direct applications

Pattern classification

- Recognition of Olympic symbols
- Recognition of printed characters
- Making an opening bid in Contract Bridge game

Associative memories

- Image pattern recall
- Content addressable memory
- Information retrieval

Optimization

- Graph bipartition problem
- Linear programming problem
- **Travelling** salesman problem
- Smoothing images with discontinuities

Vector quantization

Control applications

Table 8.1 Organization of Topics on Applications of Artificial Neural Networks (Cont.)

Application areas

Applications in Speech

- NETtalk
- Phonetic typewriter
- Vowel classification
- Recognition of consonant-vowel (CV) segments
- Recognition of stop CV utterances in Indian languages

Applications in Image Processing

- Recognition of handwritten digits
- Image segmentation
- Texture classification and segmentation

Applications in decision making

8.2 Direct Applications

8.2.1 Pattern Classification

Pattern classification is the most direct among all applications of neural networks. In fact, neural networks became very popular because of the ability of a multilayer feedforward neural network to form complex decision regions in the pattern space for classification. Many pattern recognition problems, **especially** character or other symbol recognition and vowel recognition, have been implemented using a multilayer neural network. Note, however, that these networks are not directly applicable for situations where the patterns are deformed or modified due to transformations such as translation, rotation and scale change, although some of them may work well even with large additive **uncorrelated** noise in the data.

Direct applications are successful, if the data is directly presentable to the classification network. Three such cases are considered for detailed discussion in this section. They are: (a) Recognition of Olympic games symbols, (b) Recognition of characters, and (c) Making an opening bid from a dealt hand in the card game of Contract Bridge. As can be seen below, in **each** of these cases there is no difficulty in presenting the input data to a multilayer neural network. Limits of classification performance will be reached if the symbols are degraded due to 'deformations' in the case of Olympic symbols, or if the input corresponds to casually 'handwritten' characters in the case of character recognition, or if the **'knowledge'** of the bidding sequence and the 'reasoning' power of human players have to be used in the bridge bidding problem.

Recognition of Olympic Games Symbols: We consider a set of 20 Olympic games symbols shown in Figure 8.1 for illustrating a pattern

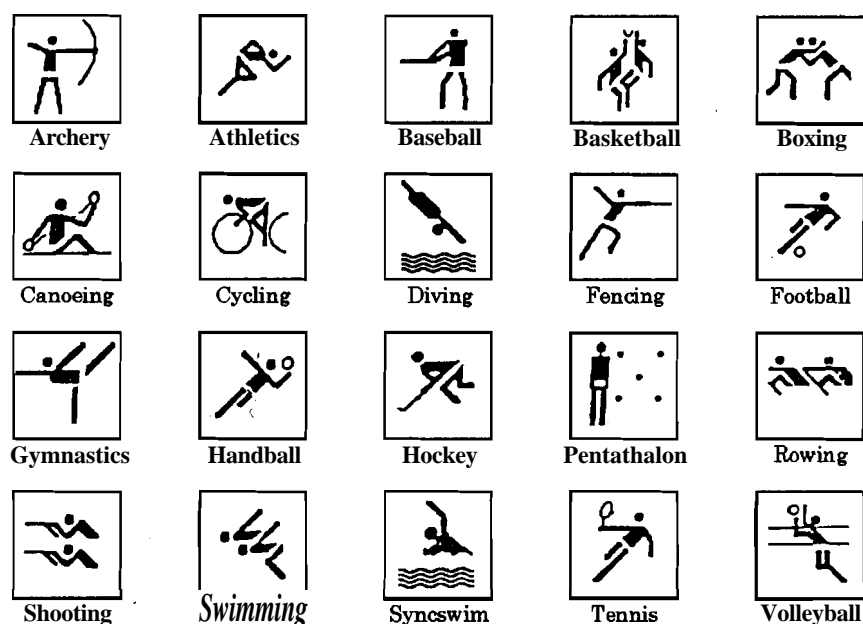


Figure 8.1 Olympic games symbols (20) used for studies on recognition of objects from degraded images.

classification task by a neural network [Ravichandran and Yegnanarayana, 1995]. The symbols are all represented as black and white pixels on a 128 x 128 points grid. Although the symbols appear complex in terms of detail, each symbol represents a rigid-object-like behaviour. This behaviour ensures that the relative pixel positions of a symbol do not change even under severe degradation, such as translation, rotation and scaling. For such objects the performance of a neural network classifier is always satisfactory. We discuss the results of classification studies for various types of degradation. The type of degradation studied in this case corresponds to the poor resolution of the image obtained when the image is reconstructed using a sparse set of elements as in a sensor array imaging situation [Yegnanarayana et al, 1990]. For example, the reconstructed (128 x 128 pt) images from a 16 x 16 element array are shown in Figure 8.2. In this case a Hamming network which performs template matching using neural principles is suitable for classification. The Hamming network is a maximum likelihood classifier for binary inputs, and it performs correlation matching between the input and the stored templates [Lippmann, 1987]. It consists of two subnets as shown in Figure 8.3. The lower subnet consists of $M(128 \times 128)$ input

Applications of ANN

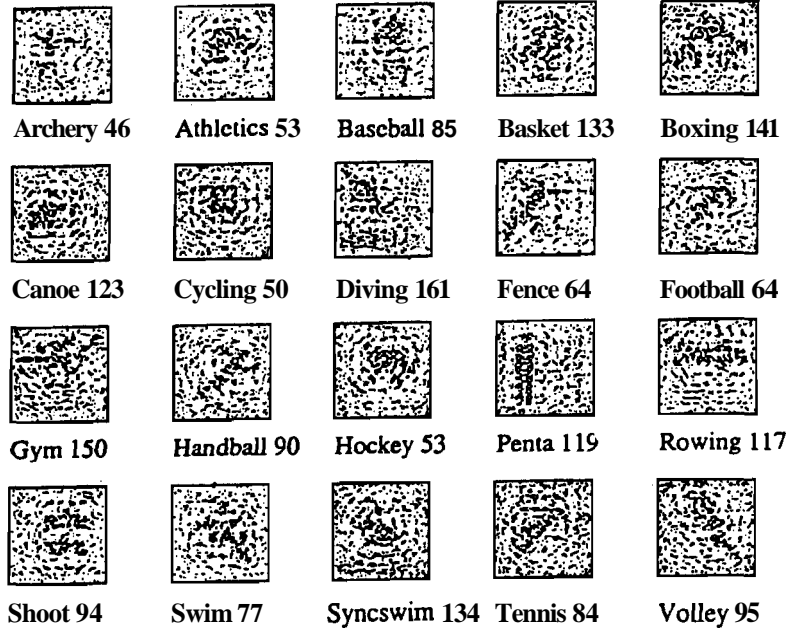


Figure 8.2 Recognition results for images reconstructed from data collected from a 16×16 element sensor array. The class decision of the network is given along with the activation value of the winning pattern. In this case, all of the 20 images were correctly identified.

units, each corresponding to a pixel in the given pattern, and N output units corresponding to the N pattern classes, which in this case is **20**.

In the lower **subnet** the connection weights between the input and output units are fixed in such a way that the network calculates the distance **from** the input pattern to each of the N stored pattern classes. The weights are given by [Lippmann, 1987]

$$w_{ij} = a_{ij} / 2, \quad \theta_i = M / 2, \quad 1 < j \leq M, \quad 1 < i \leq N \quad (8.1)$$

where w_{ij} is the **connection** weight from the input unit j to the output unit i in the lower **subnet**, θ_i is the threshold for the i th output unit and a_{ij} is the element j of the pattern for the i th symbol. The values of a_{ij} are **-1** or **1**.

In the upper **subnet**, the weights are fixed in such a way that the output units inhibit each other. That is

$$\begin{aligned} v_{kl} &= 1, & \text{for } k=l \\ &= -\epsilon, & \text{for } k \neq l \end{aligned} \quad (8.2)$$

where v_{kl} is the connection weight between the units k and l in the upper net, and ϵ is a small positive number, say $\epsilon = 0.1$.

When a bipolar pattern is presented for classification, the lower **subnet** calculates its matching score (s_i) with the stored pattern for

the i th class as follows:

$$s_i = s_i(0) = f\left(\sum_j w_{ij} x_j - \theta_i\right) \quad (8.3)$$

where $f(\cdot)$ is the output function, x_j is the j th element of the input pattern and θ_i is the threshold value for the i th unit.

The output of the lower **subnet** is presented to the upper **subnet**, where a competitive interaction takes place among the units. The dynamics of the upper **subnet** is given by

$$s_i(t+1) = f\left(s_i(t) - \varepsilon \sum_{k \neq i} s_k(t)\right), \quad i = 1, 2, \dots, N \quad (8.4)$$

The competition continues **until** the output of only one unit **remains** positive, and the outputs of all other units become negative. The positive unit corresponds to the class of the input pattern.

For the set of degraded symbols given in **Figure 8.2**, the correct classification performance is 100%. The performance is impressive,

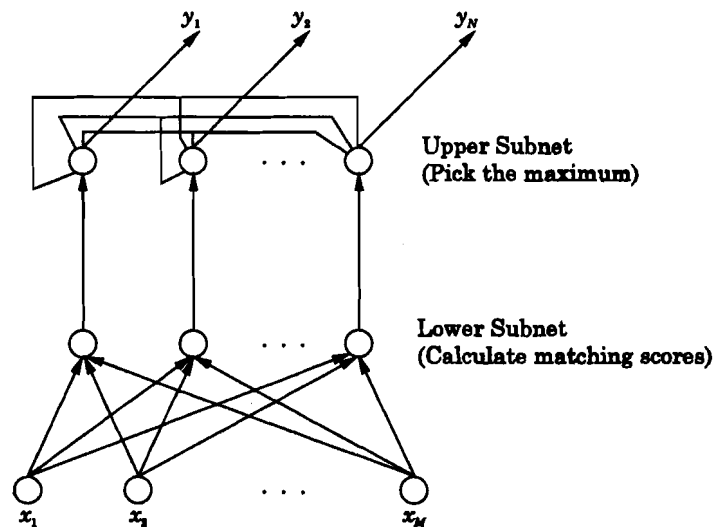


Figure 8.3 The Hamming network. The input and output units are represented by x and y vectors, respectively.

since it is **difficult** even for us to identify visually the **discriminating** features in many of these images. When the degradation is increased by reducing the resolution **using** an array of 8×8 elements, the recognition performance is only 13 out of 20. **Figure 8.4** gives a summary of the recognition performance with different sparse arrays [Ravichandran, 1993]. The figure shows the number of patterns correctly classified out of 20. The **above** experiment illustrates the following points: (a) It is interesting to note that many images for

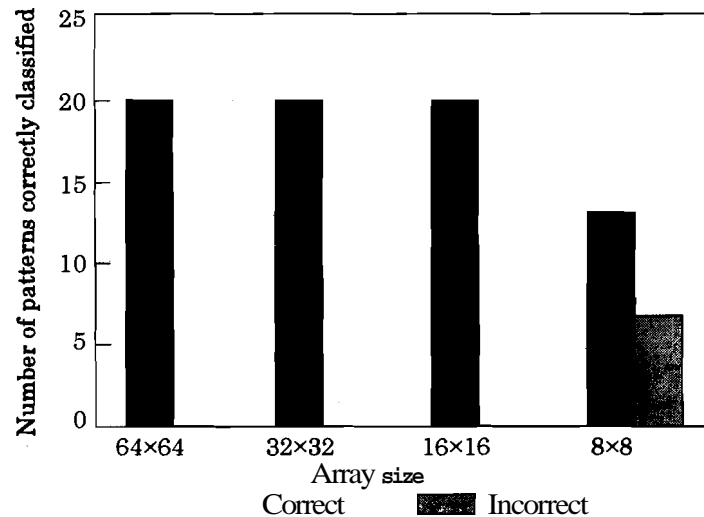


Figure 8.4 Summary of recognition performance with different sparse arrays (64×64 , 32×32 , 16×16 and 8×8 sensors). Graph shows the number of patterns correctly classified out of twenty patterns in each case.

the 16×16 sensor array size case seem to have very few visual clues (Figure 8.2) for us to recognize, but were recognized correctly by the network. (b) The performance of the classifier degrades gradually with increasing image degradation due to sparsity and noise. (c) The activation values of the winner units are indicative of the level of the image degradation, i.e., greater the degradation the lower is the activation of the winning units. (d) The matching scores obtained at the first layer are measures of similarity of the input pattern with each of the patterns stored in the network. But the activation level of each unit in the second layer is **affected** by the activation values of all other units. Hence when an output unit becomes positive, its activation level not only reflects how close the input image is to the identified pattern, but also gives an idea of the degree of confidence given to this decision relative to other patterns stored in the network. Thus the activation value also reflects the complexity of the symbol set in terms of how close in shape the symbols are.

Thus this study indicates that if the set of expected objects is known, then it is possible to design a neural network for object recognition, where the network performs a simple correlation matching only. In this study only direct pixel-wise description of the object was used, and hence the network may not function well if the objects are deformed due to transformation and scaling.

If the images are clean and noise-free, then there exist methods to overcome the effects of metric transformations of the objects. Several neural network models have been proposed for **invariant** pattern

recognition [Barnard and Casasent, 1991]. In the case of networks which achieve invariance by structure, the structure of the network is designed such that the output is invariant to the transformations of interest. In the case of invariance by training, representative samples of various transformations are presented during training so that the network learns equivalent transformations.

Invariance by **structure** or by training assumes the existence of a fixed set of weights which provide invariance over the continuum of transformations. It also assumes that a network can be trained to estimate this set of weights from examples. But invariances cannot be built as static functions in the structure. They have to be dynamically estimated from the data. Alternatively, one can first address the transformation invariance by feature extraction, and then use these features as input to a classifier [Ravichandran and Yegnanarayana, 1991].

Methods based on the theory of geometric moments have been used for normalization and invariant feature extraction [Hu, 1962]. If the object is compact and has only a few details, these invariant measures, which are stable over a wide range of spatial transformations can be designed. In this study the six moment values proposed by Hu [1962] are used as features invariant with respect to scale, position and orientation. Since these features values vary over a wide range, logarithm of the absolute values of these moments are used as features **representing** an image.

For classification, a **feedforward** neural network with **6** units in the input layer, corresponding to the input features, and **20** units in the output layer corresponding to the number of **different** symbols, are used. The network has one hidden layer with **8** units. The number of units in the hidden layer in this case appears to be not very critical as long as it is above a certain minimum value, which in this case is **8**. The network was trained with eight different transformed images for each of the twenty symbols. Some samples of the transformed images used in the training are shown in Figure 8.5 for **six** different symbols. Since reduction in the size of the image causes loss of detail, **100%** classification accuracies were obtained only for images reduced **upto 1/3** of the linear dimension of its original, **i.e.**, when the **128 x 128** pt image was reduced to a **40 x 40** pt image. Further reduction in scale decreases the classification accuracy as shown in Figure 8.6.

When sparse data reconstruction is used on the transformed images, the reconstructed images are not only transformed, but also noisy as shown in Figure 8.7 for images reconstructed **from** a **32 x 32** element sensor array. In this case the computation of moment features had to be done after preprocessing the noisy image. A neural network-based method proposed in [Ravichandran, 1993] for preprocessing the noisy images is used to derive the preprocessed



Figure 8.5 Some rotated, scaled and translated images of Olympic games symbols used to study transformation invariant recognition of objects.

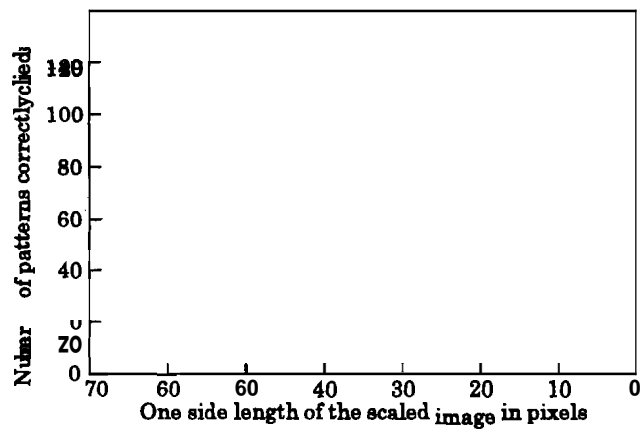


Figure 8.6 Transformation invariant recognition performance for Olympic games symbols. Graph shows the number (out of a set of 120 test patterns) of objects (maximum size 128 x 128) correctly classified as the size of the image is reduced.

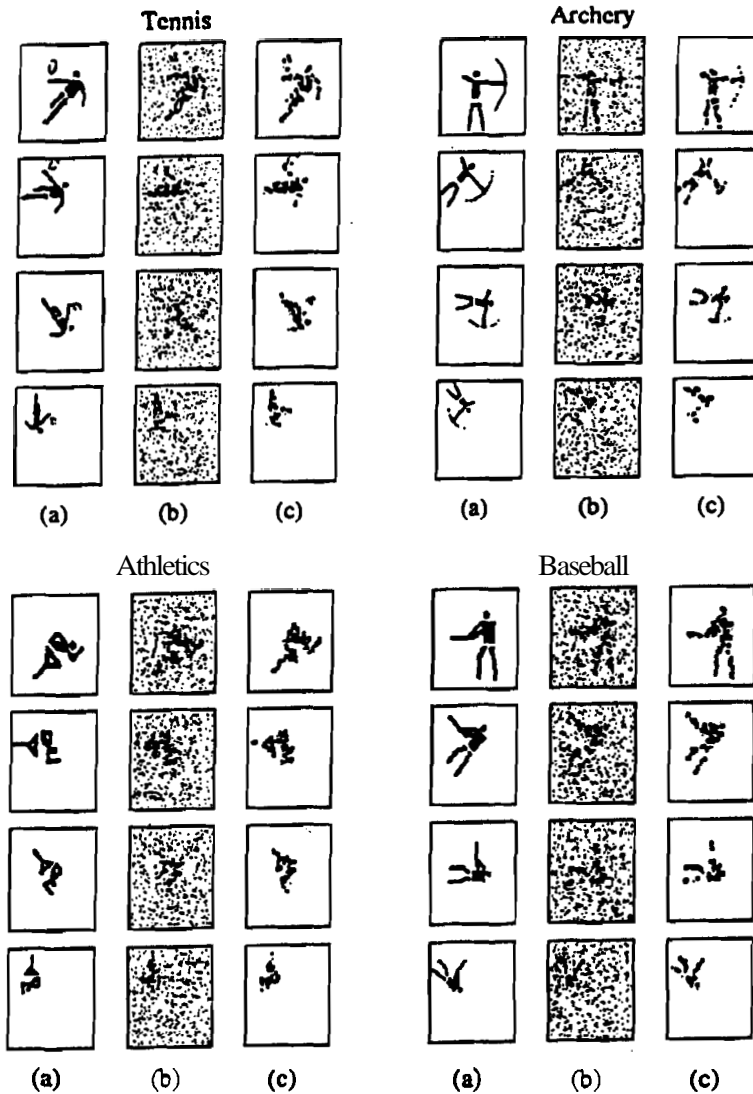


Figure 8.7 Some examples of transformed images. (a) Transformed images of four Olympic games symbols, tennis, archer, athletics and baseball. (b) Corresponding images obtained by reconstruction from data collected by a sparse 32 x 32 sensor array. (c) Images in (b) after noise suppression.

images of the objects as shown in Figure 8.7(c) for a few cases of transformation. In this case some features are lost even when there is a scale change of less than 1/2 along linear dimensions, and hence there is loss of recognition accuracy as shown in Figure 8.8.

Recognition of printed characters: Similar results were obtained

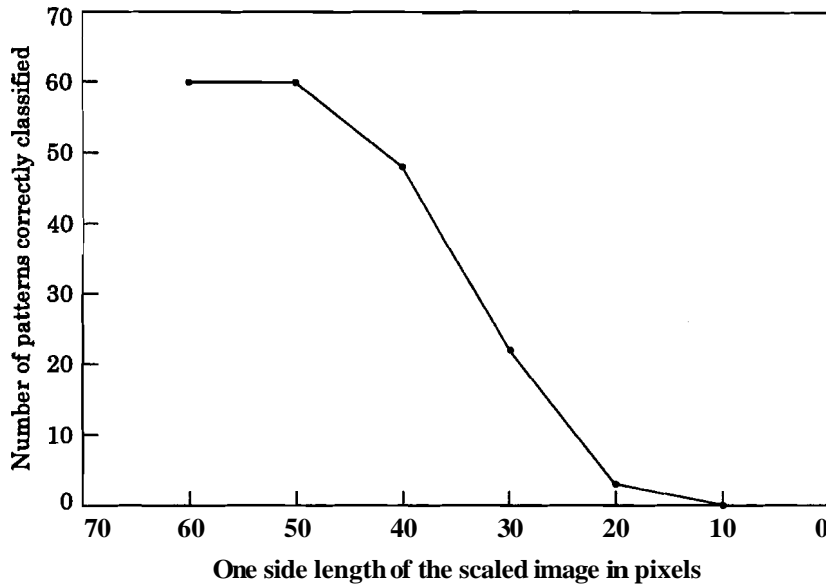


Figure 8.8 Transformation invariant recognition of olympic game symbols from degraded images obtained by reconstruction from data collected by a 32×32 array. Graph shows the number (out of a set of 60 test patterns) of objects (maximum size 128×128) correctly classified as the size of the image is reduced.

when images (128×128 pts) of ten characters of alphabet were used in the study of transformation invariant object recognition. The ten characters and some transformed versions of these characters used in the study are shown in Figure 8.9. In this case 100% classification accuracies could be obtained for all the test data upto a scale reduction of $1/12$ of the linear dimension of the original, i.e., the reduced image is about 10×10 pts. Thus the moment feature approach gives better transformation invariant recognition in this case than in the case of the Olympic games symbols, since the objects are simpler in detail in the case of the printed characters of the alphabet.

The above studies illustrate that pattern classifications can be accomplished using neural network models for objects whose images are severely degraded by transformations and noise. But in all these cases the objects were assumed to be rigid, in the sense that there was no relative displacement in different parts of the object. Note that the above illustrations differ from the handwritten characters in which different parts of a character are deformed differently in each sample. Thus the classification methods based on correlation matching or training a feedforward network using moment features are not useful for problems such as recognition of handwritten characters.

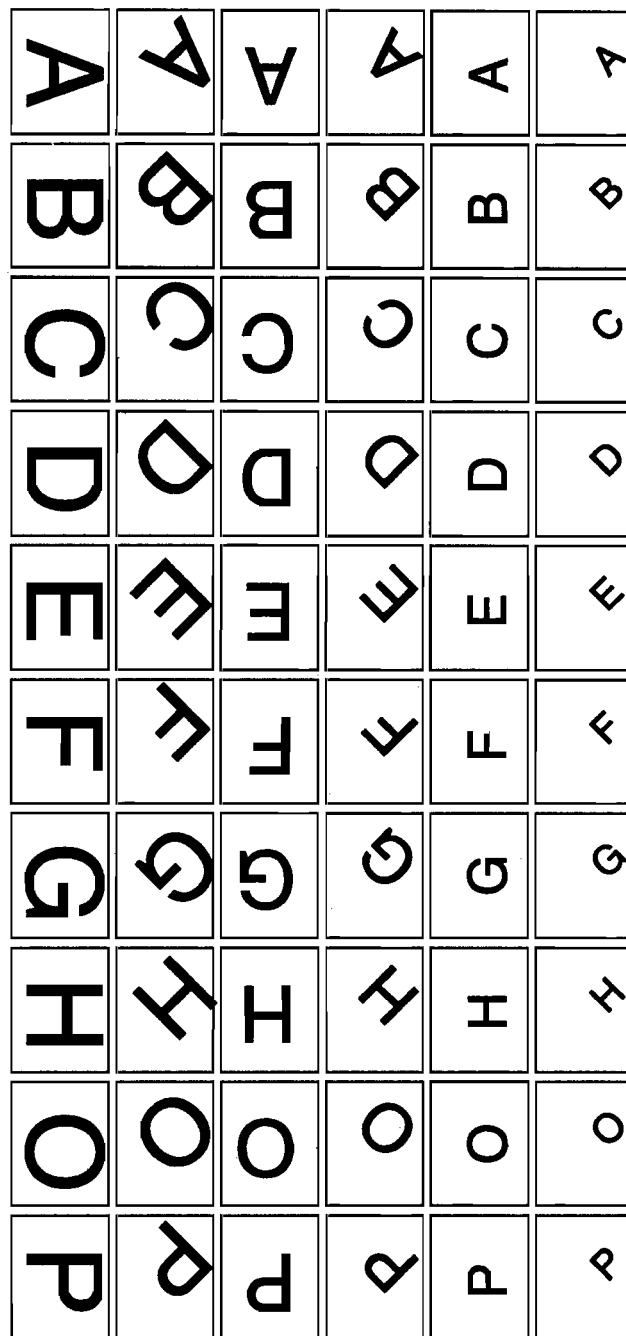


Figure 8.9 Some rotated, scaled and translated images of characters generated to test translation invariant recognition.

player. Errors in the 1-level network occur mainly because there are many borderline hands which may fall into either 1-level or into OTHER categories. Table 8.2 shows the expert bids and the bids made by a 1-level network for some sample hands. In the table strong unbalanced hands are labelled *unknown* (U) for training purpose. Sometimes the system did well by opening 1C as in Example 2. Also, in Example 3 the network's bid seems to be better than the expert. The discrepancy in the last example is also typical of human players.

Table 8.2 Bids made by the 1-Level Network. (Strong Unbalanced Hands were Labelled 'Unknown' (U) for Training Purposes.)

No.	(S)-(H)-(D)-(C)	Points	Expert bids	1-level network
1	KJ6-Q62-K94-A864	13	1C	1C
2	-4-AT942-AKJ8974	12	U	1C
3	J8-KQT643-KQ92-T	11	P	1H
4	K64-AQ874-AT953	13	1C	1C
5	AKQT9-J765432-9-	10	U	P
6	AT63-KQJT6-AT73	14	1D	1D
7	AQT87-K4-AKT4-Q7	18	1S	1S
8	AKJ9542-QJ852-6	11	U	U
9	62-AT3-J942-AKT7	12	1C	1C
10	Q653-AKJ8-AJ-974	15	1H	1S

The above study clearly shows that it is possible to capture the implicit reasoning process by a pattern classification network. Unlike in the case of Olympic games symbols example, here the input representation is accurate and noise-free, whereas the output class level is fuzzy. The fuzzy nature of output is due to the possibility of making different bids on the same hand by different players or by the same player at different times.

8.2.2 Associative Memories

As discussed earlier, the objective of an associative memory is to store a pattern or data for later recall with partial or noisy version of the pattern as input, or to store association between two patterns for later recall of one of the patterns given the other. Both feedback and feedforward topologies of neural networks are directly useful for these applications. Associative memory, if used in a feedback structure of the Hopfield type, can function as a content addressable memory as well. The stable states of the network, which represent the energy

minima or basins of attraction, are used to store the pattern information. In a feedforward network the associations corresponding to the input-output pattern pairs are stored in the weights of the network.

Applications of these networks for associative memory are direct, if the patterns are available in the form of one or two-dimensional arrays of values. Associative memories as content addressable memories are quite powerful. For example, if information about individuals are stored in a network, then it is possible to retrieve the complete data by providing partial or even noisy clues [Haken, 1991; Haken, 1995; Mhaswade, 1997]. Other useful applications for an associative memory are recognition of images, and retrieval of bibliography information from partial references such as from incomplete title of a paper [Kohonen, 1997].

Three applications of the associative memory function of neural networks are discussed in this section. The first application refers to the image pattern recall from partial or noisy clues. The second application refers to the content addressability feature of the associative memories. The third application deals with information retrieval.

Image pattern recall: Suppose a set of L distinct images of size $N \times N$ pixels is given, each described by a set of gray level values for all the pixel positions. Then the images can be stored in a feedback network consisting of $N \times N$ units, with weights determined by the correlation between pixel values. That is,

$$w_{ij} = \sum_{l=1}^L a_{li} a_{lj} \quad (8.5)$$

where a_{li} is the i th pixel value in the l th image. Then any image can be recalled by giving a partial input to the network and letting the network relax to an equilibrium state corresponding to the image. Reliability of the image recall is governed by the uncorrelatedness between images, and the correlation among pixels within an image. The representation of the Olympic symbols in the previous section can also be viewed as an illustration of image pattern storage and recall. It is obvious that such a method of storage will not work if there are deformations in the image. Better ways of storage and recall of image patterns are possible, if the images are represented by suitable models like Markov random fields [Rangarajan and Chellappa, 1995] or by Gabor features [Daugman, 1988]. But storage and recall performance of any such representation models depend critically on the effectiveness of the models in capturing the significant image features.

Content addressable memory: Information stored in a neural network model can be accessed using partial contents of the required

information. This task can be accomplished by a suitably designed feedback network. The illustration of data storage using the IAC model in Appendix A is an example of information access using the contents [McClelland and Rumelhart, 1988]. For the example of *Jets and Sharks* data in the IAC model, the name of the individual can be obtained by specifying his other characteristics such as 'age' group, 'education', etc.

Information retrieval: Representation of the *Jets and Sharks* data through the IAC model also illustrates how any data can be stored in a network for retrieval with partial inputs. For more complex situations of accessing information with noisy or degraded input, suitable representation of the input and its matching procedure with the stored data would be needed. For example, to access items in a data base with voice inputs, it is necessary to process the input speech signal to derive parameters or features suitable for matching with the stored information. This is true even for accessing bibliographic information with a string of characters containing some random errors. In all these cases the information access mechanism depends critically on the representation of the input information and the algorithms for matching the input features with the stored data [Cherkassky and Vassilas, 1990; Mhaswade, 1997; Kohonen, 1997] .

8.2.3 Optimization

One of the most successful applications of neural network principles is in solving optimization problems [Peterson and Soderberg, 1995; Hertz et al, 1991; Muller and Reinhardt, 1991; Yuille, 1995]. There are many situations where a problem can be formulated as minimization or maximization of some cost function or objective function subject to certain constraints. It is possible to map such a problem onto a feedback network, where the units and connection strengths are identified by comparing the cost function of the problem with the energy function of the network expressed in terms of the state values of the units and the connections strengths. The solution to the problem lies in determining the state of the network at the global minimum of the energy function. In this process, it is necessary to overcome the local minima of the energy function. This is accomplished by adopting a simulated annealing schedule for implementing the search for global minimum [Kirkpatrick et al, 1983].

The solution to an optimization problem by neural networks consists of the following steps:

(a) Express the objective function or cost function and the constraints of the given problem in terms of the variables of the problem:

$$\text{Objective function } (E) = \text{cost} + \text{global constraints} \quad (8.6)$$

(b) Compare the objective function in Eq. (8.6) with the energy function (Eq. (8.7)) of a feedback neural network of Hopfield type to identify the states and the weights of the network in terms of the variables and parameters appearing in the objective function.

$$\text{Energy function: } E = -\frac{1}{2} \sum_{i \neq j} w_{ij} s_i s_j. \quad (8.7)$$

(c) The solution to the optimization problem consists of determining the state corresponding to the global minimum of the energy function of the network. Assuming bipolar states for each unit, the dynamics of the network can be expressed as

$$s_i(t+1) = \text{sgn} \left(\sum_{j \neq i} w_{ij} s_j(t) \right) \quad (8.8)$$

(d) Direct application of the above dynamics in search of a stable state may lead to a state corresponding to a local minimum of the energy function. In order to reach the global minimum, bypassing the local minima, the concept of stochastic unit is used in the activation dynamics of the network. As discussed in Chapter 5, for a stochastic unit the state of the unit is updated using a probability law, which is controlled by a temperature parameter (T). At low temperatures, the stochastic update approaches the deterministic update, which is dictated by the output function of the unit.

(e) The state of a neural network with stochastic units is described in terms of probability distribution. The probability distributions of the states at thermal equilibrium follow the Boltzmann-Gibb's law (see Sec. 5.4.2), namely

$$P(\mathbf{s}_\alpha) = \frac{1}{Z} e^{-E_\alpha/T} \quad (8.9)$$

where E_α is the energy of the network in the state \mathbf{s}_α and Z is the partition function given by

$$Z = \sum_{\alpha} e^{-E_\alpha/T} \quad (8.10)$$

The network is allowed to relax to thermal equilibrium at a given temperature (T). Due to stochastic update the state of the network does not remain constant at thermal equilibrium. But the average value of the state of the network remains constant due to stationarity of the probabilities $P(\mathbf{s}_\alpha)$ of the states of the network at thermal equilibrium. The average value of the state vector is given by

$$\langle \mathbf{s} \rangle = \sum_{\alpha} \mathbf{s}_\alpha P(\mathbf{s}_\alpha). \quad (8.11)$$

(f) At higher temperatures many states are likely to be visited, irrespective of the energies of those states. Thus the local minima of

the energy function can be escaped. As the temperature is gradually reduced, the states having lower energies will be visited more frequently. Finally, at $T = 0$, the state with the lowest energy will have the highest probability. Thus the state corresponding to the global minimum of the energy function can be reached, escaping the local minima. This method of search for the global minimum of the energy function is called simulated annealing.

Implementation of simulated annealing requires computation of stationary probabilities at thermal equilibrium for each temperature in the annealing schedule. Moreover, the convergence to the global minimum is guaranteed only if the temperature parameter is reduced slowly starting from a high value initially [Geman and Geman, 1984]. The state probabilities are computed by collecting the distribution of the states for a large number of cycles of updates of the states of the network at a given temperature. The cycles are repeated until the probabilities of states do not change substantially for different sets of cycles. Once the thermal equilibrium is reached, the temperature is changed to the next lower value. Thus the process of implementation of simulated annealing is very slow. The Metropolis algorithm gives a simpler method for implementing the simulated annealing [Metropolis et al, 1953]. However, in this case the convergence is not guaranteed.

(g) In order to speed up the process of simulated annealing, the mean-field annealing approximation is used [Peterson and Anderson, 1987], in which the stochastic update of the binary/bipolar units is replaced by deterministic analog states [Glauber, 1963]. The basic idea of mean-field approximation is to replace the fluctuating activation values of each unit by its average value. That is x_i is replaced by $\langle x_i \rangle$.

$$\langle x_i \rangle = \left\langle \sum_j w_{ij} s_j \right\rangle = \sum_j w_{ij} \langle s_j \rangle \quad (8.12)$$

where $\langle \cdot \rangle$ represents the expectation or average of the random quantities. Likewise, in the average of the state of the i th unit given by (see Eq. (5.78))

$$\langle s_i \rangle = \tanh(x_i/T), \quad (8.13)$$

If x_i is replaced by $\langle x_i \rangle$, we get from Eqs. (8.12) and (8.13)

$$\langle s_i \rangle = \tanh \left(\frac{1}{T} \sum_j w_{ij} \langle s_j \rangle \right) \quad (8.14)$$

The mean-field approximation involves solving the following recursive equations involving the average values of the states of the units.

$$\langle s_i(t+1) \rangle = \tanh \left(\frac{1}{T} \sum_{j=1}^N w_{ij} \langle s_j(t) \rangle \right), \quad i = 1, 2, \dots, N. \quad (8.15)$$

These are a set of coupled nonlinear deterministic equations. The equations are solved iteratively starting with some arbitrary values $\langle s_i(0) \rangle$ initially. Once the steady equilibrium values of $\langle s_i \rangle$ have been obtained, then the temperature is lowered. The next set of average states at thermal equilibrium are determined using the average state values at the previous thermal equilibrium condition as the initial values $\langle s_i(0) \rangle$ in the equations above for iterative solution. Note that, due to deterministic set of equations involved in this computation, the computation will be much faster than in the case of simulated annealing. While the convergence to the global minima is not guaranteed in the mean-field approximation, it seems to yield good results [Haykin, 1994, Sec. 8.14].

The set of equations of the mean-field approximation is a result of minimization of an effective energy defined as a function of the temperature. This results in an alternative expression for Eq. (8.14) and is given by [Haykin, 1994, p. 338].

$$\langle s_i \rangle = \tanh \left[-\frac{1}{T} \frac{\partial E(\langle \mathbf{s} \rangle)}{\partial \langle s_i \rangle} \right] \quad (8.16)$$

where the effective energy $E(\langle \mathbf{s} \rangle)$ is the expression for energy of the Hopfield model using averages for the state variables.

Probably, the most studied problem in the context of optimization using the principles of neural networks is the travelling salesman problem, where the objective is to find the shortest route connecting all cities to be visited by a salesman [Peterson, 1990]. Other optimization problems include the weighted matching problem, where a number of points must be pairwise connected such that the sum of lengths of all connections is as short as possible, and the stereo vision matching in optical image processing. The method of simulated annealing has also been successfully employed to find the optimal arrangement of integrated electronic circuits on semiconductor chips [Hertz et al, 1991; Muller and Reinhardt, 1991].

In this section four optimization problems are discussed, showing in each case the formulation of the optimization problem, mapping the problem onto a neural network, and a solution using neural networks principles. The problems to be discussed are: Graph bipartition problem, Linear programming problem, Travelling salesman problem and Image processing.

Graph bipartition problem: The problem is to partition a graph of N nodes equally as shown in Figure 8.10 such that the connectivity (measured in terms of number of links) between the two partitioned graphs is minimum. The problem can be mapped onto a Hopfield network, in which each bipolar unit corresponds to a node in the graph, with the state $s_i = +1$ representing the node in one half and

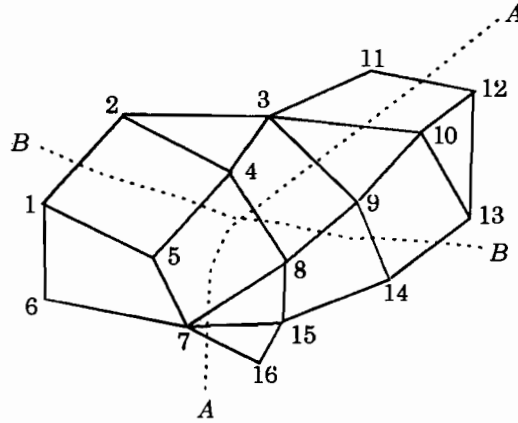


Figure 8.10 Graph bipartition problem. The connectivity for A-A partition is 7, whereas the connectivity for B-B partition is 6.

$s_i = -1$ representing the nodes in the other half. Let $c_{ij} = 1$ if the nodes i and j are connected and $c_{ij} = 0$ if the nodes are not connected. Thus the cost term $c_{ij}s_i s_j$ contributes a nonzero value only if the nodes are connected. We have $c_{ij}s_i s_j = +1$ if the nodes are in the same partition, and $c_{ij}s_i s_j = -1$ if they are in different partitions. For equal division of nodes $\sum_i s_i = 0$. Therefore the cost term with equality constraint is given by

$$E = -\frac{1}{2} \sum_{i,j} c_{ij} s_i s_j + \frac{\alpha}{2} \left(\sum_i s_i \right)^2 \quad (8.17)$$

where the positive constant α is used to indicate the relative strengths of the two terms in the energy function. Due to conflicting requirements of the two terms, there will be several local minima in the energy function. The cost function E can be written in the Hopfield energy form as

$$E = -\frac{1}{2} \sum_{ij} w_{ij} s_i s_j + \frac{N\alpha}{2} \quad (8.18)$$

where $w_{ij} = c_{ij} - \alpha$. The term $N\alpha/2$ is to take care of the term corresponding to $i = j$, since $w_{ii} = 0$ for a Hopfield model. A solution by the mean-field approximation is derived using the values of w_{ij} in Eq. (8.15). For a given temperature, the iterative solution of the set of N nonlinear equations is obtained. Using the resulting average values $\langle s_i \rangle$ as initial values, the temperature is lowered and the set of $\langle s_i \rangle$ is again obtained iteratively at the new temperature. The final solution is obtained using this mean-field annealing according to a preset annealing schedule. Note that at the final (near zero) temperature, the average value $\langle s_i \rangle = s_i$, as the network reaches the

stable state. The final states (s_i) of the units partition the nodes of the graph into two with minimal cost, with the units having $s_i = +1$ into one partition and the units having $s_i = -1$ into the other partition. Good results were obtained for the graph bipartition problem for a wide range of problem sizes. The solutions obtained by mean-field annealing are comparable to those obtained by simulated annealing [Peterson and Soderberg, 1989].

Linear programming problem: In a linear programming problem the aim is to maximize an objective function $\sum_{i=1}^N c_i s_i$ subjected to a set of constraints given by the inequalities of the type

$$\sum_{i=1}^N w_{ki} s_i \leq b_k, \quad k = 1, 2, \dots, M. \quad (8.19)$$

If the unit values s_i are confined to integers, then the problem becomes an integer programming problem. In particular, for $s_i \in \{0, 1\}$ the following cost function can be defined

$$E = -\sum_{i=1}^N c_i s_i + \alpha \sum_{k=1}^M \Phi \left(\sum_{i=1}^N w_{ki} s_i - b_k \right) \quad (8.20)$$

where Φ is a function defined to ensure that the constraints given in Eq. (8.19) are satisfied. For example, $\Phi(x) = x U(x)$, where $U(x) = 1$, for $x > 0$, and $U(x) = 0$, for $x \leq 0$. Because of the non-polynomial form of the constraints in (8.19), the derivative $\partial E / \partial s_j$ is replaced by a difference as follows [Peterson and Soderberg, 1995]

$$\frac{\partial E}{\partial s_j} = -c_j + \alpha \sum_{k=1}^M \left[\Phi \left(\sum_{i=1}^N w_{ki} s_i - b_k \right) \Big|_{s_j=1} - \Phi \left(\sum_{i=1}^N w_{ki} s_i - b_k \right) \Big|_{s_j=0} \right] \quad (8.21)$$

Minimizing the energy function using the mean-field approximation requires iterative solution of Eq. (8.16), after substituting the expression for $\partial E / \partial \langle s_i \rangle$ in terms of the average values $\langle s_i \rangle$ and using a suitable annealing schedule as discussed before. The solutions to these class of problems with constraints in the form of inequalities are described in [Peterson and Soderberg, 1989].

Travelling salesman problem: For a given number of cities (N) and their intercity distances, the objective is to determine a closed loop of the tour of the cities such that the total distance is minimized subjected to the constraints that each city is visited only once, and all cities are covered in the tour. Denoting the state of a binary unit

of a Hopfield network as s_{ia} where $s_{ia} = 1$ indicates that the city a is to be visited at the i th stage of the tour, we can write the following cost function [Hopfield and Tank, 1985]:

$$E = \sum_i \sum_{a,b}^{a \neq b} d_{ab} s_{ia} (s_{(i-1)b} + s_{(i+1)b}) + \frac{\alpha}{2} \sum_i \sum_{a,b}^{a \neq b} s_{ia} s_{ib} + \frac{\beta}{2} \sum_a \sum_{i,j}^{i \neq j} s_{ia} s_{ja} + \frac{\gamma}{2} \left(\sum_{i,a} s_{ia} - N \right)^2 \quad (8.22)$$

where the first term gives the total distance in the tour, with d_{ab} representing the distance between the cities a and b . The second term vanishes if no more than one city is visited at each stage. The third term vanishes if each city is visited not more than once. The last term vanishes when each city is visited exactly once and all cities are covered in the tour. The positive constants α , β and γ denote the relative importance given to the constraints. Note that these constraints can also be viewed as soft constraints or weak constraints which ought to be satisfied in the overall energy minimization process, as in the constraint satisfaction model (see Appendix A).

The Hopfield neural network consists of N^2 units, with each unit assuming a binary value, $s_{ia} \in \{0, 1\}$, where $s_{ia} = 1$ in the solution indicates that the city a is to be visited at the i th stage in the tour.

A suitable choice of the energy function for this network is

$$E = \frac{1}{2} \sum_{i,j}^{i \neq j} \sum_{a,b}^{a \neq b} w_{iajb} s_{ia} s_{jb} - \sum_{i,a} \theta_{ia} s_{ia} \quad (8.23)$$

The expression for the weights can be obtained by comparing the expressions for the energy with the cost function. The weights are given by

$$w_{iajb} = d_{ab}(1 - \delta_{ab})(\delta_{(i-1)j} + \delta_{(i+1)j}) + \alpha(1 - \delta_{ab})\delta_{ij} + \beta(1 - \delta_{ij})\delta_{ab} + \gamma \quad (8.24)$$

The threshold θ_{ia} for each unit is given by γN .

A solution to the travelling salesman problem is obtained by determining the stable state of the network using either a deterministic relaxation procedure or a stochastic relaxation procedure with an annealing schedule. Studies have shown that this neural network approach does not yield minimum cost function solution most of the time [Wilson and Pawley, 1988]. The performance does not improve with the adjustment of the scaling parameters α , β and γ for changing the relative importance of different constraints.

In most of the cases involving large number of cities, the optimum solution for the travelling salesman problem depends critically on the

choice of the parameters used for the constraint terms and for implementing the annealing process. A suboptimal solution is possible for a large size (30 cities or more) of the travelling salesman problem using the elastic ring method and self-organization network [Durbin and Willshaw, 1987] (see Figure 8.11).

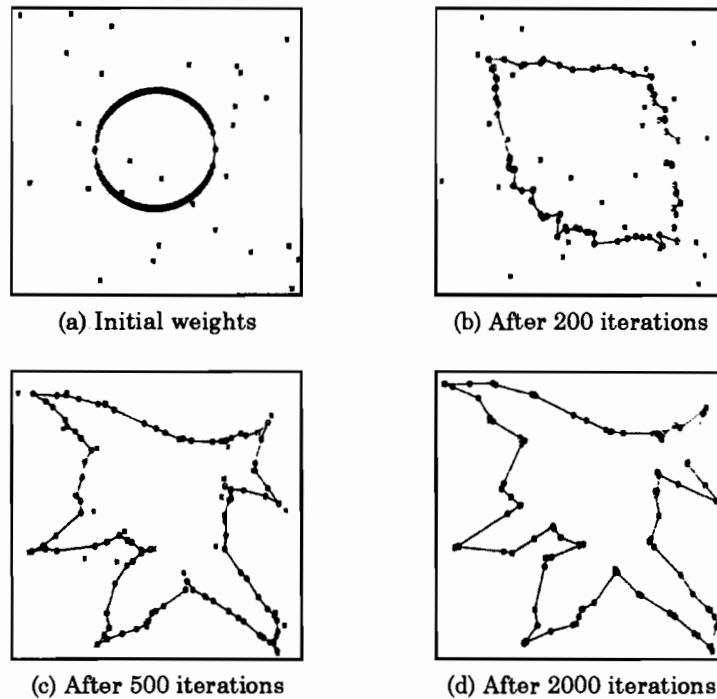


Figure 8.11 Kohonen's self-organization feature map for travelling salesman problem for 30 cities and 100 units in the output layer. The cities are shown as 'x' and the units as 'o'.

The elastic ring method is based on Kohonen's feature mapping approach. The location of the N cities are shown as 'x' in the two-dimensional plane in Figure 8.11a. A tour can be represented as a line passing through these points. Therefore the travelling salesman problem can be viewed as mapping from the plane to a line (see Figure 6.20). Consider an N -unit output layer and a 2-unit input layer of Kohonen's self-organization map network. The units in the output layer are arranged along a closed curve, with initial weights corresponding to the points on the curve as shown in Figure 8.11a. Note that the axes represent the two weights for each unit, and the weight vectors for adjacent units are joined to form the closed curve. The network is trained by presenting the coordinate values of each city as input. The resulting self-organized feature maps after three different number of iterations are shown in Figures 8.11b, 8.11c and

8.11d. Since more than one unit can be attracted to the same city, normally the number of units in the output layer are made much larger than the number of cities. The approximate tour shown in Figure 8.11 is for 30 cities and 100 units. Figure 8.12 shows the

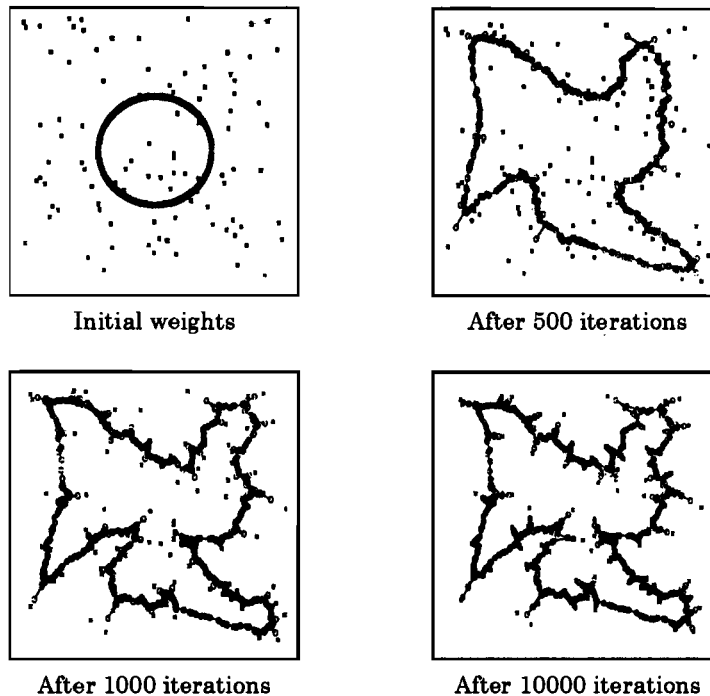


Figure 8.12 Kohonen's self-organization feature map for travelling salesman problem for 100 cities and 1000 units in the output layer. The cities are shown as 'x' and the units as 'o'.

approximate tour developed when the number of cities is 100 and the number of units is 1000. Note that, while an approximate tour can be obtained quickly, there is no optimization criterion used in arriving at the tour using this method.

Smoothing images with discontinuities: Suppose we have a noisy or blurred image with discontinuities due to edges. We can apply optimization using simulated annealing by formulating the problem as an energy minimization problem [Hertz et al, 1991, p. 85]. Let s_i be the smoothed i th pixel value that we want to reconstruct from the given noisy data d_i . If there are no discontinuities, the cost function to be minimized for deriving a smoothed surface is given by

$$E = \frac{1}{2} \alpha \sum_i (s_i - s_{i+1})^2 + \frac{1}{2} \beta \sum_i (s_i - d_i)^2 \quad (8.25)$$

where the first term ensures that the difference between smoothed values (s_i) for adjacent pixels is small, and the second term ensures that the smoothed pixel value does not deviate much from the observed noisy data. The constants α and β will determine the relative importance to the costs of these two terms. Here s_i can represent the output of a processing unit of a neural network with a continuous output function $f(x)$.

If the image has discontinuities (see Figure 8.13), we can introduce an additional unit v_i between two adjacent pixel units s_i

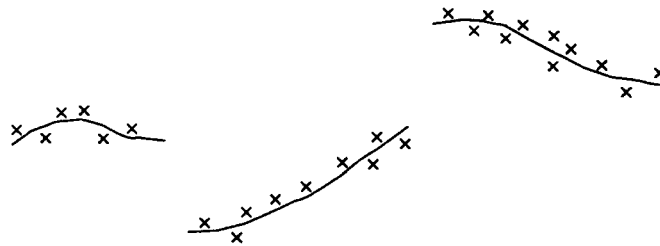


Figure 8.13 Illustration of image discontinuity problem in 1-D. This shows fitting piecewise smooth curve to noisy data to allow breaks in the fitted curve.

and s_{i+1} . These additional units should have bipolar values ± 1 , so that $v_i = +1$ indicates the presence of a discontinuity and $v_i = -1$ the absence of a discontinuity. Therefore the first term in the cost function in Eq. (8.26) below has the additional $(1 - v_i)$ to make this cost zero if there is a discontinuity.

$$E = \frac{1}{2} \alpha \sum_i \frac{1}{2} (1 - v_i) (s_i - s_{i+1})^2 + \frac{1}{2} \beta \sum_i (s_i - d_i)^2 + \gamma \sum_i v_i \quad (8.26)$$

To prevent the network from putting too many discontinuities, the cost function includes an additional term $\sum_i v_i$. The constants α , β and γ determine the relative importance to smoothing, data and discontinuities, respectively.

The s_i units of the network should be analog in order to reflect the continuous nature of the pixel intensity, and the v_i units should be discrete to indicate the presence or absence of discontinuity. Both these requirements are met using stochastic binary or bipolar units. The units must be operating at temperatures (T) high enough compared to the constants α and β so that the average state values operate more or less in the linear operating range of the unit's activation value. At the same time the temperature T should be small compared to the constant γ in order to obtain values of v_i close to saturation values of $+1$ or -1 . One can also achieve this by gradually

varying the gain parameter λ for a continuous output function of v_i units (see Figure 5.4a) [Cortes and Hertz, 1989].

The cost function of Eq. (8.26) for a one-dimensional case can be generalized to two-dimension representing an image by using the following cost function [Koch et al, 1986].

$$E = \frac{1}{2} \alpha \sum_{(ij)} \frac{1}{2} (1 - v_{ij}) (s_i - s_j)^2 + \frac{1}{2} \beta \sum_i (s_i - d_i)^2 + \gamma \sum_{(ij)} v_{ij} \quad (8.27)$$

where the pair (ij) indicates that i and j are the adjacent pixels in the image lattice. A discontinuity between two adjacent pixels is indicated by $v_{ij} = +1$. Figure 8.14 shows the geometry of the image

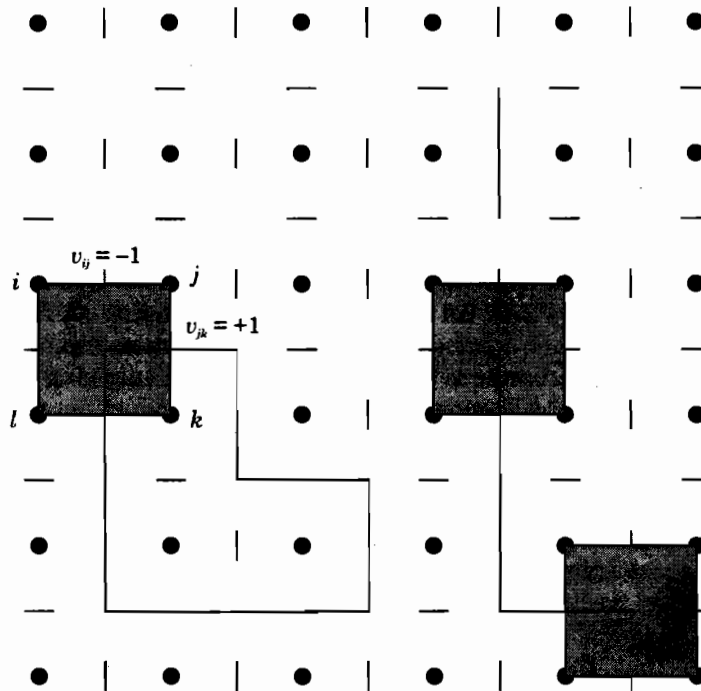


Figure 8.14 An image lattice with dots (.) indicating pixel positions. Between two adjacent pixel units (i, j) is a v_{ij} unit indicated by a short horizontal or vertical bars. A discontinuity between two adjacent pixels (k, l) is obtained if $v_{kl} = +1$ and is indicated by a long bar.

lattice, where dots (•) indicate the pixel units (i, j, k, l) , and the horizontal and vertical short bars indicate the units (v_{ij}) in between two adjacent pixels (ij) . The state of the pixel unit (i) is indicated by s_i and the state of the in-between units as v_{ij} , with $v_{ij} = +1$ indicating discontinuity and $v_{ij} = -1$ indicating continuity.

It is possible to reduce the presence of several isolated discontinuities due to noise by adding an additional term E_{loop} to the cost function, which gives importance to the closed contours formed by regions in an image [Hertz et al, 1989]. The term is given by

$$E_{\text{loop}} = -\delta \sum_{(ijkl)} v_{ij} v_{jk} v_{kl} v_{li} \quad (8.28)$$

where $(ijkl)$ refer to the set of four adjacent pixels that form a closed loop. For a closed contour case formed by the discontinuities, the E_{loop} contributes a negative term $-\delta$ (see shaded area *A* in Figure 8.14). On the other hand, isolated discontinuities due to open contour contribute $+\delta$ for the E_{loop} term (see shaded areas *B* and *C* in Figure 8.14).

8.2.4 Vector Quantization

Vector quantization (VQ) typically encodes a large set of training data vectors into a small set of representative points, thus achieving a significant compression in the representation of data. Vector quantization has been shown to be useful in compressing data that arises in speech processing, image processing, facsimile transmission and weather satellites [Gray, 1984; Nasrabadi and King, 1988].

Formally, vector quantization maps data vectors onto a binary representation or a symbol. The mapping is from an N -dimensional vector space to a finite set of symbols corresponding to K classes. Associated with each symbol $k \in K$ is a reproduction vector $\hat{\mathbf{x}}_k$. The encoding of a data vector \mathbf{x} to the symbol k is the mapping in vector quantization. The collection of all possible reproduction vectors is called the codebook.

The design of a codebook is called training, and it can be implemented using neural network models. The learning vector quantization is one such network model. Several other models have been proposed, for example, Kohonen's self-organizing feature map, to construct vector quantization codebooks for storage and transmission of speech and image data [Kohonen, 1995; Kohonen, 1992].

Note that the process of vector quantization involves merely grouping the given vectors into different classes based on similarity of the data vectors. The question of how well the given data represents the physical situation is not relevant. In this sense the vector quantization is a direct application of neural network principles for data compression.

The simple competitive learning algorithm (see Eq. (6.42)) provides a method for vector quantization of a given data. The algorithm is for unsupervised learning, and is given by first finding the minimum distance weight vector $\mathbf{w}(m)$ corresponding to the input vector $\mathbf{a}(m)$, and then adjusting the weight vector of the competitive learning network according to the following equation:

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \eta (\mathbf{a}(m) - \mathbf{w}(m)) \quad (8.29)$$

where η is the learning rate parameter.

Kohonen suggested a supervised version of the vector quantization called Learning Vector Quantization (LVQ) Kohonen, 19891. This learning law is applicable when labelled sets of input data are given. The algorithm is given by

$$\begin{aligned} \mathbf{w}(m+1) &= \mathbf{w}(m) + \eta (\mathbf{a}(m) - \mathbf{w}(m)), && \text{if the input is classified} \\ & && \text{correctly.} \\ \mathbf{w}(m+1) &= \mathbf{w}(m) - \eta (\mathbf{a}(m) - \mathbf{w}(m)), && \text{if the input is classified} \\ & && \text{incorrectly.} \\ \mathbf{w}(m+1) &= \mathbf{w}(m), && \text{if the input does not} \\ & && \text{belong to the class} \\ & && \text{corresponding to } \mathbf{w}(m). \end{aligned}$$

8.2.5 Control Applications

There are several situations in control applications where the principles of neural networks can be directly applied. The applications include process control, robotics, industrial manufacturing, aerospace and several others [Zurada, 1992]. The main task in a control situation is to generate an appropriate input signal to the physical process (plant) to obtain the **desired** response **from** the plant [Narendra and Parthasarathy, 1990; Nguyen and Widrow, 1990].

The controller generates the actuating signal when the external input is given. The design of a controller depends on the nature of the plant and the way the input is derived for the controller in the operation of the plant. The plant may be static or dynamic. For a static plant, the transfer function is given by a constant. For a **dynamical** plant, the transfer function is given by the ratio of the **Laplace** transform of the plant's output to the **Laplace** transform of the plant's input [Zurada, 1992].

There are two ways of controlling a plant: open-loop control and feedback control. In an open-loop control the controller consists of cascade of a system and the inverse of the plant. The system is used to achieve the desired response for the input. The controller thus generates an actuating signal to the plant to obtain the desired response at the output of the plant. This needs inverse transfer function of the plant, and the plant should not change its characteristics during its operation. Both these problems are overcome in a feedback control mechanism where the controller is designed in such a way that the output becomes independent of the plant transfer function.

Multilayer feedforward networks can be used to capture the characteristics of the plant transfer function or the plant's inverse

transfer function. The neural network is then used to design a controller. A detailed discussion on the use of neural networks for control applications can be found in [Hunt et al, 1992; Narendra and Mukhopadhyay, 1992].

8.3 Application Areas

The excitement in neural networks started mainly due to difficulties in dealing with problems in the field of speech, image, natural language and decision making using known methods of pattern recognition and artificial intelligence. Several of these problems have been attempted using the principles of neural networks, and some of these attempts will be discussed in this section.

The main issue in all these problems is the representation of the real world problem in a system. The power of a neural network can be exploited provided the problem can be well represented in the network as discussed in **Sec. 8.2** on direct applications. But in the application areas to be discussed in this section, the poor and fragile performance of the neural network based system may be attributed to the weakness in the input processing and the mapping of the problem onto the neural network model. Since problems in speech, image, natural language and decision making seem to be solved effortlessly by human beings, our expectations from an artificial system are also high [Reddy, 1996; Dreyfus, 1992]. In this context, it is worth remembering that the human pattern recognition processing is an integrated system of data acquisition, input preprocessing, feature extraction, recognition and understanding. It is not feasible to assess the performance of each of these processes in isolation.

In this section some problems in the application areas of speech and image processing are discussed. A brief discussion on the use of neural networks for expert decision making is also given.

8.3.1 Applications in Speech

Speech signal is the output of a time-varying vocal tract system excited by a time-varying excitation signal. The vocal tract system, including the coupling of the nasal tract, can be accurately described in terms of the positions of the articulators such as tongue, lips, jaw, velum, etc. Generally the vocal tract system is approximately described in terms of the acoustic features such as the frequency response or the resonances (formants) and anti-resonances (anti-formants) of the system. These features are easier to extract from the signal than the articulatory parameters. The excitation of the vocal tract system consists of broadly three categories: (a) Voiced source (the quasiperiodic excitation due to the vibrating vocal folds),

(b) Unvoiced source (the turbulent flow of air at a narrow constriction created in the vocal tract during production), and (c) Plosive source (the abrupt release of the pressure built up behind a closure in the vocal tract system). The voiced source is characterized by the periodicity (pitch period), and the change of the pitch period with time (intonation). In **general** the short-time characteristics of the speech signal are represented by the short-time (10–20 ms) spectral features of the vocal tract system as well as the nature of excitation in the short-time segment. These are **called** segmental features. **Supra**-segmental features of speech are represented by the variation of the pitch period (intonation), the durations of different sound units, and the coarticulation reflecting the dependence of characteristics of one sound unit on the **neighbouring** sound units during speech production.

Speech is a sequence of sound units corresponding to a linguistic message. Important applications in speech area are:

(a) **Speech Recognition**: The objective is to determine the sequence of sound **units** from the speech signal so that the linguistic message in the form of text can be decoded from the speech signal.

(b) **Speech Synthesis**: The objective is to determine the sequence of sound units corresponding to a text so that the given text message can be encoded into a speech signal.

(c) **Speaker Identification**: The objective is to determine the identity of the speaker from the speech signal.

The main problem in these speech applications is processing of the speech signal in a manner similar to human auditory processing mechanism, so that features relevant to a particular task can be extracted. The speech problem is further complicated by the fact that the message is conveyed not only through the segmental features but also by the suprasegmental features. It is our lack of understanding of these segmental and suprasegmental features and their extraction mechanism that makes the speech tasks extremely difficult for implementation by machines [Flanagan, 1972; Rabiner and Juang, 1993]. In this section we will briefly discuss neural network models for some speech tasks. We will discuss in detail the development of neural network architectures for recognition of consonant-vowel (CV) segments. Other interesting applications in speech can be found in [Lippmann, 1989; Narendranath et al, 1995; Cole et al, 1992; Pal and Mitra, 1992].

NETtalk: The **NETtalk** is a multilayer **feedforward** neural network (Figure 8.15) developed to generate pronunciation units or phoneme code **from** an input text [Sejnowsky and Rosenberg, 1987]. The phoneme code is presented as input to a speech synthesizer to produce speech corresponding to the text. The network consists of an input

Applications of ANN

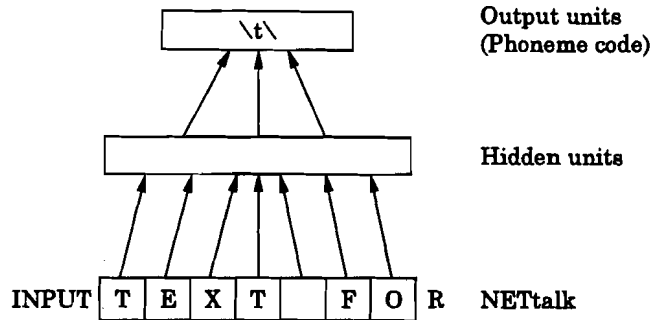


Figure 8.15 NETtalk: A feedforward neural network to convert English text to speech.

layer of 7×29 binary input units corresponding to 7 input text characters including punctuation, one hidden layer with 80 units and one output layer with 26 units corresponding to 26 different phonemes or phonetic units. The network takes as input 7 consecutive characters at a time and produces the phoneme corresponding to the pronunciation of the letter at the centre of the input string. Each character of the input is represented as a 29-bit string. The network was trained on 1024 words from a set of English phoneme exemplars giving the pronunciation of words, and was tested using different text sentences. The system produces a string of phonemes for a given input text. The speech produced by a synthesizer using these strings of phonemes as input was intelligible, although the quality was poor. The network was capable of distinguishing between vowels and consonants. On a new text the network achieved a generalization accuracy of 78% in phonetic transcription. The system merely demonstrated the ability of a neural network to capture the letter to sound rules from input-output data. The quality will be obviously poor, as it cannot capture the significant suprasegmental features which are essential for producing natural sounding synthetic speech.

NETtalk is a good example to illustrate the generalization feature of neural networks. The network is able to learn from examples and produce intelligible speech for new text input, even though the quality of speech is poor. Rule-based text-to-speech systems produce significantly better quality speech since knowledge is explicitly incorporated into the system [Klatt, 1980; DECTalk, 1983; Yegnanarayana et al, 1994]. But development of rule-based systems takes several years of effort, whereas neural networks can be trained to learn within a few-hours.

Phonetic typewriter: The objective in speech recognition is to transform a given utterance into a sequence of phoneme-like units and convert this sequence into the text corresponding to the spoken

utterance. A block diagram of the phonetic typewriter developed by Kohonen [Kohonen, 1988; **Torkkola** et al, 1991] is given in Figure 8.16. The input speech signal to the phonetic typewriter is

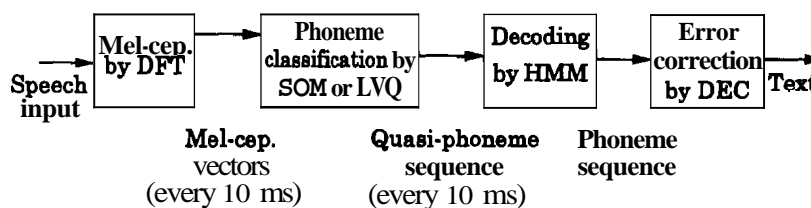


Figure 8.16 Block diagram of a phonetic typewriter.

processed to obtain a spectral representation using 20 mel-scale **cepstral** coefficients for every 10 ms segments of data [Davis and Mermelstein, 1980]. The sequence of coefficient vectors is given as input to a phoneme classifier, one vector at a time, to obtain the quasi-phoneme sequence as output. The phoneme classification is achieved by using either **LVQ** or **SOM** learning [Kohonen, 1990a; Kohonen, 1990b]. The sequence of phoneme-like units is converted to the phonetic transcription using a multiple **codebook** Hidden **Markov** Model (**HMM**) technique [Rabiner, 1989]. The errors in the phonetic decoding by the **HMM** are corrected using the Dynamically Expanding Context (**DEC**) algorithm [Kohonen, 1986], and then converted into the text corresponding to the input utterance. The phonetic typewriter was able to produce letter accuracies of 95% for the Finnish language. The approach was reported to have worked well for languages whose orthography and phonemic transcriptions have simple **correspondence**.

Vowel classification: The classic Peterson and Barney [1952] formant data for 10 vowels is a good test case for a real-world classification problem. It consists of the first two **formants** collected from spectrographic analysis of the vowel data. The vowel data was collected for a total of 67 men, women and children. The data was **collected** in the constant-vowel-consonant context of *hVd*. **Figure** 8.17 shows the distribution of the first two formants data for the vowels in the following ten words: heed, head, *had*, hud, *hod*, *hawed*, *who'd*, hood, heard and hid. The available vowel data was split into two sets, one set was used for training the classification network and the other set for testing the performance. Using a radial basis **function** network for classification, Nowlan obtained a recognition accuracy of 87% [Nowlan, 1990]. Considering the fact that there is significant overlap among the classes, the classification performance of the network is indeed significant.

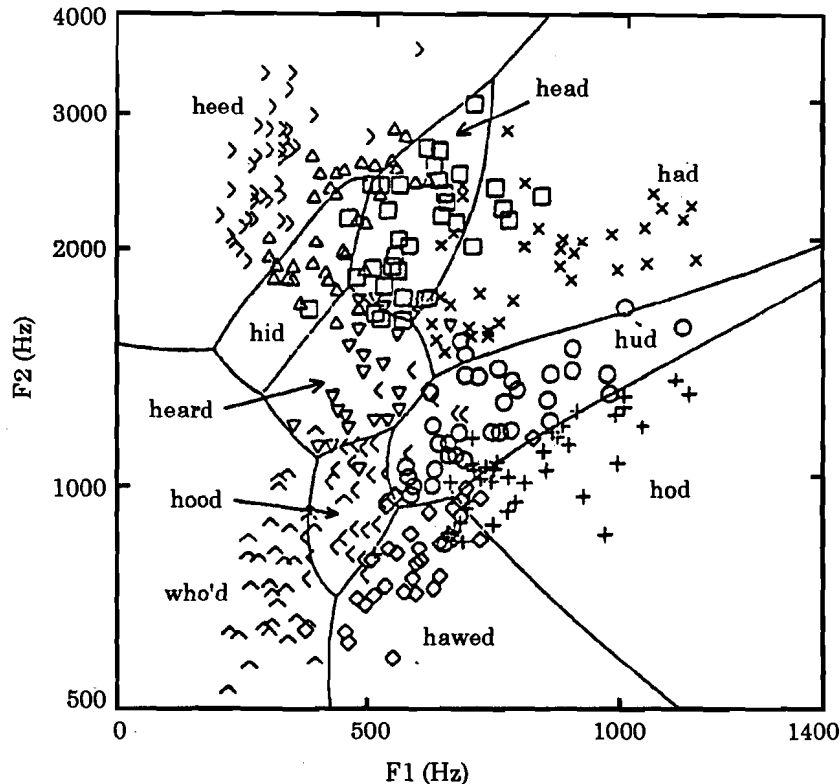


Figure 8.17 Vowel data from Peterson and Barney [1952]. The lines are the **class** boundaries obtained by a two-layer feedforward network. [Adapted from Huang and Lippmann, 1988].

Recognition of Consonant-Vowel (CV) segments: Consonant-Vowel (CV) utterance typically forms a production unit (syllable) in speech, and hence several attempts have been reported for recognition of CV utterances [Harrington, 1988]. Since these are dynamic sounds, the spectral patterns change with time. Each utterance of a CV unit is represented as a temporal sequence of spectral vectors. Each spectral vector corresponding to a fixed 10 ms segment may be represented using 16 log spectral coefficients on a mel-frequency scale or using the corresponding mel-scale cepstral coefficients [Davis and Mermelstein, 1980]. The number of spectral vectors per CV utterance generally varies. But usually a fixed duration (50–200 ms) segment of CV enclosing the vowel onset, the transition to vowel and some steady part of the vowel, is used to represent a CV unit. The CV units are thus temporal sequence patterns and hence static pattern recognition networks like multilayer feedforward neural network (MLFFNN) are not suitable for recognition of these units. Moreover, **discriminability** among these CV units is low due to domination of the vowel context.

An obvious method to perform sequence recognition is to view the temporal sequence of the spectral vectors as a two-dimensional spatial input pattern for a MLFFNN. The conventional backpropagation learning can then be used to train the network. A better approach for CV recognition is through timedelay neural networks (TDNN) [Waibel, 1989; Waibel et al, 1989]. TDNN is a MLFFNN with its input consisting of time-delayed input frames of data. The input to the intermediate hidden layers also consists of time-delayed outputs of the preceding layer. Figure 8.18 illustrates the idea of a time-delay

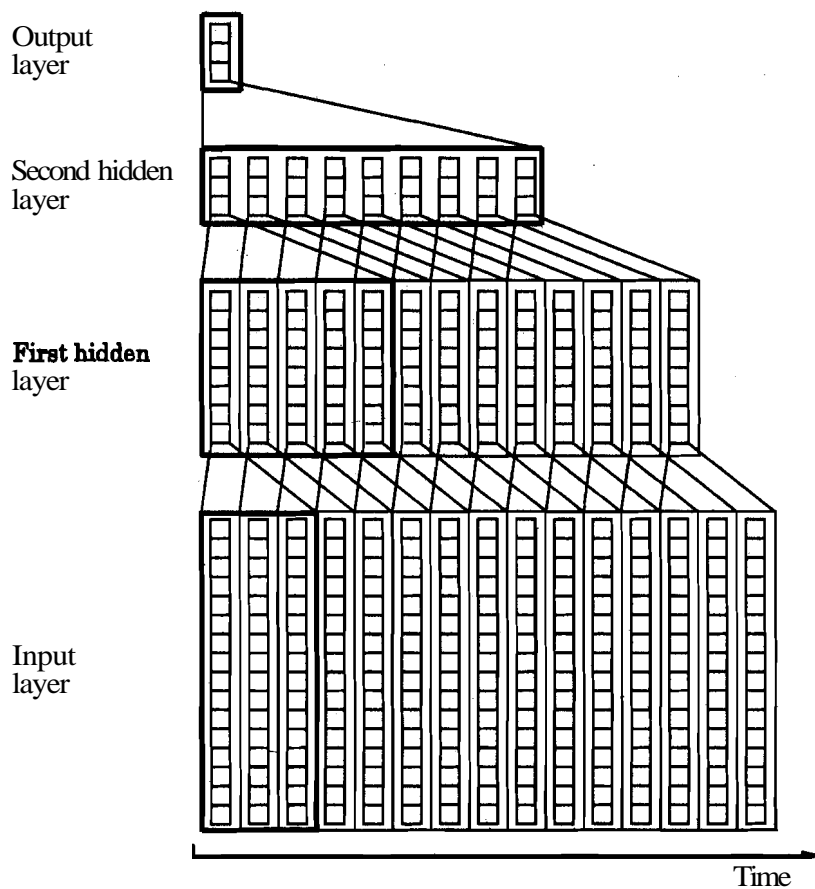


Figure 8.18 Architecture of a Time Delay Neural Network (TDNN) for classification of three CV units. Multiple copies of the TDNN are aligned with adjacent spectral vectors. The first TDNN is shown in boxes marked by thick lines.

neural network applied for classification of three CV units /b/, /d/, and /g/. The time sequence is reflected in the computation of block of three frames of data at a time, with two frames overlapping for

successive blocks. In other words, **Figure 8.18** shows multiple copies of the TDNN aligned with adjacent input spectral vectors. The first TDNN is shown in the boxes marked by thick lines. In this case each utterance has **15** frames of data, each frame consisting of **16** spectral components. For this, **13** different copies of TDNN are created. These include **13** copies of the input layer, nine copies of the first hidden layer, and only one second hidden layer and one output layer. The output layer has three units corresponding to the three classes */b/*, */d/*, and */g/*. For each TDNN, each unit in a layer is connected to all the units in the layer below it. For each TDNN there are **16 x 3 units** in the input layer, **8 x 5 units** in the first hidden layer and **9 x 3 units** in the second hidden layer and **3 units** in the output layer. Multiple copies of the TDNN as shown in the **Figure 8.18** enable the entire history of the network activity to be present at the same time. This allows the use of the backpropagation learning algorithm to train the network.

The **16** log spectral coefficients for each **10 ms** frame are normalized using the average of each coefficient for **all** the **15** frames in the utterance, and the coefficients are mapped into the range **[-1, 1]**. The normalized values are given as input to the TDNN network. The speech data for the three classes was excised from continuous speech in Japanese, and a database of about **800 CV** units was collected for a given speaker. The TDNN was able to discriminate the three classes with an accuracy of **98.5%**. Considering the fact that the data for each class has significant variation due to contextual effects, this result is impressive.

Extending this network model for large number of CV classes requires modular approach, where it is necessary to distinguish the groups of CV classes first before the individual classes can be identified [**Waibel, 1989**]. Moreover, because of the large size of the network, the training of the network **will** be slow. It will also be difficult to **collect** sufficient data to obtain good generalization performance from such a large network.

Recognition of stopconsonant vowel utterances in indian languages:

For the development of a recognition system for large number of CV classes, recognition of Stop-Consonant-Vowel (SCV) utterances in Indian languages is considered [**Chandrasekhar, 1996**]. In particular, we consider the SCV classes of the Indian language, Hindi. The **80** SCV classes considered in this study are given in **Table 8.3**, where the classes are organized according to the 4 manners of articulation, namely, **UnVoiced-UnAspirated (UVUA)**, **Unvoiced-Aspirated (UVA)**, **Voiced-UnAspirated (VUA)** and **Voiced-Aspirated (VA)**. These are highly confusable set of sound classes. A modular network approach followed by a Constraint Satisfaction Model (CSM) approach is proposed for recognition of isolated utterances of the **80** SCV classes.

Table 8.3 Arrangement of SCV Classes into Subgroups using Manner of Articulation for Grouping

Subgroup	SCV Classes				
UVUA	ka	ki	ku	ke	ko
	ṭa	tṭi	ṭu	ṭe	ṭo
	ta	ti	tu	te	to
	pa	pi	pu	pe	po
UVA	kha	khi	khu	khe	kho
	ṭha	ṭhi	ṭhu	the	tho
	tha	thi	thu	the	tho
	pha	phi	phu	phe	pho
VUA	ga	gi	gu	ge	go
	ḍa	ḍi	ḍu	ḍe	ḍo
	ḍa	ḍi	ḍu	ḍe	ḍo
	ba	bi	bu	be	bo
VA	gha	ghi	ghu	ghe	gho
	ḍha	ḍhi	ḍhu	ḍhe	ḍho
	dha	dhi	dhu	dhe	dho
	bha	bhi	bhu	bhe	bho

When the number of classes is large and the similarity amongst the classes is high, it is difficult to train a monolithic neural network classifier based on the All-Class-One-Network (ACON) architecture to form the necessary decision surfaces in the input pattern space [Kung, 1993]. An attempt has been made to train a **multilayer** feedforward neural network for all the 80 SCV classes. It was observed that even after a large number of epochs, the sum of the squared error remained high and it did not change significantly from one epoch to another. It shows that a single network could not be trained for these large number of classes. It is possible to develop a classifier based on the **One-Class-One-Network** (OCON) architecture in which a separate network is trained for each class [Kung, 1993]. But the discriminatory capability of the OCON classifiers was found to be poor [Chandrasekhar and Yegnanarayana, 1996].

Modular approaches [Haykin, 1994] can be used to overcome the limitations of the ACON and OCON architectures. In modular approaches large number of classes **are** grouped into smaller subgroups, and a separate neural network (**subnet**) is trained for each subgroup. A post-processor can be used to combine the outputs of the **subnets**.

Criteria guided by the phonetic descriptions of the SCV classes can be used to form subgroups. Such criteria are useful in analyzing the performance of the classifiers and determining the sources of errors in classification. A unique phonetic description can be given for each of the 80 SCV classes in terms of three features, namely, (a) the manner of articulation (**MOA**) of the stop consonant, (b) the

place of articulation (POA) of the stop consonant, and (c) the identity of the vowel in the SCV. For example, the class /ka/ is described as *unvoiced unaspirated velar stop consonant followed by the vowel /a/*. The phonetic descriptions of the SCV classes suggest that grouping can be done in such a way that one of the three features is common to the classes in a subgroup. This gives 3 different criteria for grouping.

Grouping based on MOA leads to 4 subgroups given in Table 8.3. Grouping based on POA leads to 4 subgroups: Velar (eg. /ka/, /kha/, /ga/, /gha/), Alveolar (eg. /t̪ a/, /t̪ ha/, /d̪ a/, /d̪ ha/), Dental (eg. /ta/, /tha/, /da/, /dha/), and Bilabial (eg. /pa/, /pha/, /ba/, /bha/). Each POA subgroup consists of 20 classes, and the stop consonants in these classes have the same place of articulation. Grouping based on the vowel leads to five subgroups with one subgroup for each of the five vowels: /a/, /i/, /u/, /e/ and /o/. Each vowel subgroup consists of 16 classes, and these classes have the same vowel.

We consider each of the three grouping criteria to develop a modular network for all the SCV classes. The classification performance of the modular network depends on the performance of its **subnets** and on the way the outputs of the **subnets** are combined. A simple way of combining the outputs of the **subnets** is to assign to the test input the class corresponding to the largest value among the outputs of all the **subnets**. Better performance can be obtained by combining the evidence from the output values of each **subnet** in an effective way [Chandrasekhar, 1996].

Data from isolated utterances of all the 80 SCV classes was collected from three male speakers. For each class, 12 tokens were collected from each speaker. The training data for a class consists of 4 tokens from each speaker. The remaining 8 tokens from each speaker are used as the test data.

A fixed duration portion of the signal around the Vowel Onset Point (VOP) of an SCV utterance is processed to derive a pattern vector. A 200 ms portion of the signal with 60 ms before and 140 ms after the VOP is considered for analysis. This fixed duration signal is processed (using a frame size of 20 ms and a frame shift of 5 ms) to obtain 40 frames of parameter data consisting of 12 weighted cepstral coefficients in each frame. The size of the pattern vector is reduced using the average of the coefficients for every two adjacent frames. Thus a $20 \times 12 = 240$ dimensional pattern vector is used to represent an SCV utterance. A **multilayer feedforward** neural network (MLFFNN) is used to build the **subnets**. The network has 70 units in the first hidden layer and 50 units in the second hidden layer.

The training and test data sets for each **subnet** consists of pattern vectors belonging to the classes in that subgroup only. Performance of the **subnets** for different subgroups is given in Table 8.4. The performance is given as percentage of the total number of pattern vectors in the data set that are correctly classified by the **subnet**.

Table 8.4 Classification Performance of Subnets for Different Subgroups of SCV Classes

(e) Performance of subnets for MOA subgroups

Subgroup	Training data	Test data
UVUA	98.1	77.1
W Λ	98.1	70.0
W Λ	94.2	68.9
VA	91.0	66.1
Average	95.4	66.6

(b) Performance of subnets for POA subgroups

Subgroup	Training data	Test data
Velar	96.6	54.6
Alveolar	96.0	84.2
Dental	93.3	77.9
Bilabial	91.7	80.0
Average	93.9	74.2

(e) Performance of subnets for Vowel subgroups

Subgroup	Training data	Test data
/a/	92.7	61.5
/i/	93.2	66.6
/u/	92.2	68.8
/e/	94.0	62.0
/o/	91.7	67.7
Average	92.8	65.1

The average performance on the test data for all the 80 SCV classes for the modular networks *bund* on different grouping criteria is given in Table 8.5. The performance is measured *bund* on the outputs of the subnets. In the Table 8.5 Case_*k* indicates that the correct class is in the first *k* largest outputs of the subnets. For example, for the POA grouping criterion, the correct class is within the first four classes for 76.6% of the test patterns. This performance

Table 8.5 Average Classification Performance on Test Data for the Modular Networks based on Different Grouping Criteria

Grouping criterion	Classification performance			
	Case_1	Case_2	Case_3	Case_4
MOA	29.2	50.2	59.0	65.3
POA	35.1	56.9	69.5	76.6
Vowel	30.1	47.5	58.8	63.6

is significant considering the fact that there are 80 different classes, and that they are confusable. The modular network for the POA grouping criterion gives a better performance compared to the other two grouping criteria. It is important to develop techniques to reduce the errors in classification at the level of **subnets** in order to improve the overall performance of the modular networks.

It is also possible to improve the **classification** performance by properly combining the evidence available at the outputs of the **subnets**. Confusability among the classes can be resolved to some extent by using the acoustic-phonetics knowledge of the classes. This knowledge **can** be incorporated **as** constraints to be met by the classes. A constraint satisfaction model [McClelland and Rumelhart, 1986] that tries to satisfy as many of these constraints **as** possible can be used to process the outputs of the **subnets** (see Appendix A). The advantage is that it may work even if some of the constraints derived from the acoustic-phonetic knowledge are weak, conflicting and erroneous.

For the constraint satisfaction model, a feedback neural network is used with one unit for each of the 80 **SCV** classes. The weight on the connection between a pair of units is determined based on the similarity between the classes represented by the units. The similarity between two **SCV** classes is determined from the knowledge of speech production features and also from the confusability between them indicated by the outputs of the **subnets**.

There are 3 different feedback networks, one for each of the 3 grouping criteria. Since the **SCV** classes within a subgroup are designed to compete among themselves during training, excitatory connections are provided between the units in the subgroup. The connections across the subgroups are made inhibitory. The weights for the excitatory and inhibitory connections are derived from the similarity matrices derived from the classification performance of **subnets** on test data. The similarity matrix for different manners of articulation is given in Table 8.6(a). The matrices for different places of articulation and for different vowels in **SCVs** are given in Tables 8.6(b) and 8.6(c), respectively. An excitatory connection is provided between units of two **SCV** classes within a subgroup if they differ in only MOA or POA or vowel. The weight of an excitatory connection is equal to the similarity measure between the production features of the two classes. An inhibitory connection is provided between classes in different subgroups only if the two classes differ in MOA or POA or vowel. The weight for the inhibitory connection is inversely proportional to the similarity measure between the production features of the two classes. If the similarity measure is C (in the range 0.0 to 1.0), then the inhibitory weight w is assigned as

$$w = -\frac{1}{100 \times C} \quad (8.30)$$

Table 8.6 Similarity Matrices for (a) Different Manners of Articulation of Stop Consonants, (b) Different Places of Articulation of Stop Consonants and (c) Different Vowels in SCVs**(a) Similarity matrix for manners of articulation**

MOA	UVUA	UVA	W A	VA
UVUA	0.87	0.03	0.06	0.02
UVA	0.03	0.84	0.04	0.08
W A	0.06	0.04	0.78	0.12
VA	0.02	0.08	0.12	0.82

(b) Similarity matrix for places of articulation

POA	Velar	Alveolar	Dental	Bilabial
Velar	0.72	0.08	0.08	0.08
Alveolar	0.08	0.75	0.10	0.09
Dental	0.08	0.10	0.68	0.10
Bilabial	0.08	0.09	0.10	0.80

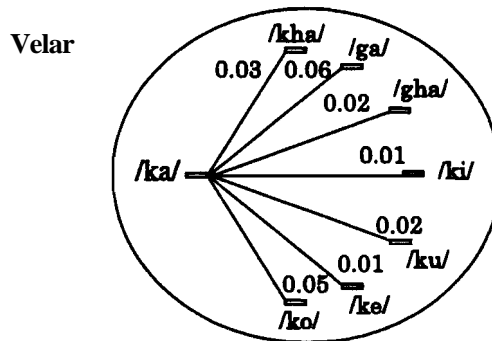
(c) Similarity matrix for vowels

Vowel	/a/	/i/	/u/	/e/	/o/
/a/	0.89	0.01	0.02	0.01	0.05
/i/	0.01	0.86	0.02	0.08	0.00
/u/	0.02	0.02	0.82	0.01	0.16
/e/	0.01	0.08	0.01	0.90	0.00
/o/	0.05	0.00	0.16	0.00	0.77

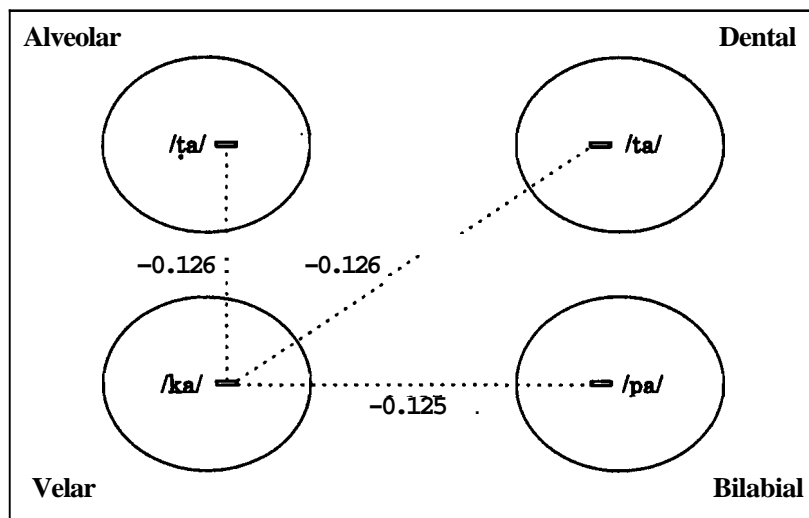
If the similarity measure C is less than 0.01, then the corresponding inhibitory weight is assigned as -1.0 .

The connections in the feedback network for the **POA** grouping criterion are illustrated in Figure 8.19. The excitatory connections for the class **/ka/** in the 'Velar' subgroup are shown in Figure 8.19a and the inhibitory connections for the same class across the subgroups are shown in Figure 8.19b.

The feedback networks for different grouping criteria **interact** with each other through a pool of units, called instance pool [McClelland and Rumelhart, 1988] (see Appendix A). There are as many (80) units in the instance pool as the number of SCV classes. Each unit in the instance pool (for example, the unit corresponding to the class **/ka/**) has a bidirectional excitatory connection with the corresponding units in the feedback networks (for example, units corresponding to **/ka/** in the MOA group, **/ka/** in the **POA** group and **/ka/** in the Vowel group). Units within the instance pool compete with one another and hence are **connected** by a fixed negative weight (-0.2). The 3 feedback networks along with the instance pool



(a) Excitatory connections for the class /ka/ in the POA feedback network



(b) Inhibitory connections for the class /ka/ in the POA feedback network

Figure 8.19 Connections for the class /ka/ in the POA feedback network. The excitatory connections for the class /ka/ in the 'Velar' subgroup are shown in (a). The inhibitory connections for the class /ka/ are shown in (b).

constitute the constraint satisfaction model reflecting the known speech production **knowledge** of the **SCVs** as well as the knowledge derived **from** the trained **subnets**. The complete constraint satisfaction model developed for **classification** of SCVs is shown in Figure 8.20.

A **Multilayer Feedforward** Neural Network (MLFFNN) trained for a subgroup of classes can be considered as a set of nonlinear filters designed to provide **discrimination** among the classes. There are 16 or **20 filters** in each subgroup and a total of **80 filters** for each grouping criterion. It may be noted that each **SCV** class occurs in a different subgroup for each of the three grouping criteria. The **outputs**

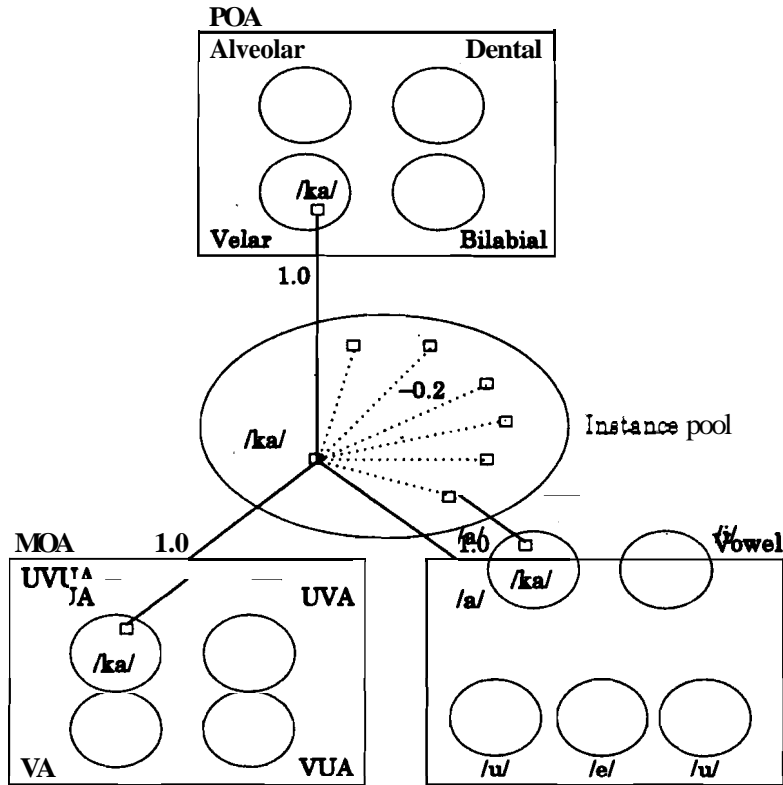


Figure 8.20 A constraint satisfaction model for classification of SCV utterances. Connections for /ka/ are shown for illustration.

of the filters for each subgroup for a given training sample is considered as a feature vector. The distribution of the feature vectors for each class is obtained from a second set of training data. The distribution is represented in terms of a mean vector μ and a covariance matrix R derived from the feature vectors for the class.

The operation of the constraint satisfaction model is as follows: Each unit j in the constraint satisfaction model computes the weighted sum of the inputs from the other units (s_j) in the model. An external input for each of the units in the feedback networks is provided as bias. The bias is derived from the 16- or 20-dimensional feature vector \mathbf{x} of the subgroup to which the unit belongs. The bias for the unit j is given by

$$B_j = \frac{1}{\sqrt{(2\pi)^M |R_j|}} e^{-\frac{(\mathbf{x} - \mu_j)^T R_j^{-1} (\mathbf{x} - \mu_j)}{2}} \quad (8.31)$$

where M is the dimension of the feature vector, μ_j is the mean feature vector of the class associated with the j th unit and R_j is the covariance matrix associated with the class of the j th unit.

The net input to the unit j is given by

$$C_j = \alpha B_j + \beta s_j + \gamma \quad (8.32)$$

where α , β and γ are constants in the range 0.0 to 1.0, chosen empirically by trial and **error**. The output function for each unit is a sigmoid function.

The constraint satisfaction model is initialized as follows: For a new input pattern, the feature vectors (\mathbf{x}) are obtained **from** all the **MLFFNNs**. The outputs of the **units** in the feedback networks for which the corresponding feature vector component value is above a threshold δ ($= 0.3$) are initialized to **+1.0**, and the outputs of all other units in the feedback networks are initialized to 0.0. The bias for a unit in the instance pool is computed from the net input to the unit **after** the feedback networks are initialized. The output of a unit in the instance pool is initialized to **+1.0**, if the net input to the unit is greater than 0.0. The constraint satisfaction model is then allowed to relax until a stable state is reached for a given input using a deterministic relaxation method. In this method a unit in the model is chosen at random and **its** output is computed. This is continued until there is no significant change in the outputs of all the units. When a stable state is reached, then the outputs of the instance pool units can be interpreted to determine the class of the input pattern.

The class of the instance pool unit with the largest output value is assigned as the class of the input utterance. Because of similarity among the SCV classes, we consider the cases (**Case_ k**) in which the correct class is among the classes corresponding to the k largest output values. The classification performance of the constraint satisfaction model (**CSM**) for different cases is given in Table 8.7. The performance of the modular network is also given in the table for comparison. Here the performance of the modular network based on POA grouping is given, **as** it gave the best classification performance among the three grouping criterion.

Table 8.7 Classification Performance of the Constraint Satisfaction Model using Test Data for all the 80 SCV Classes

Model	Case-1	Case_2	Case-3	Case-4
CSM	65.6	75.0	80.8	82.6
Modular Network	35.1	56.9	69.5	76.6

It can be seen that the performance of the CSM is significantly better than the performance of the modular networks. The performance of the CSM for Case-1 is as high as **65%** indicating that the instance pool unit with the largest output value gives the class of the input utterance correctly for **65%** of the total number of test utterances. This result is significant considering the fact that the classification is

performed by the CSM by discriminating among 80 SCV classes and that many of these classes are similar. The performance of the CSM increases to 82% for the Case-4 of the decision criterion.

The ability of the CSM to combine evidence from multiple sources may be useful for performing **even** speaker independent classification. The **subnets** are trained using the data collected from multiple speakers. Since the operation of the CSM is speaker independent, it is expected that the CSM would show a distinct improvement for speaker independent classification over the modular networks [Chandrasekhar, 1996].

The description of the recognition system for the SCV units given above clearly demonstrates the need to evolve an architecture suitable for a given problem. For example, the acoustic-phonetic knowledge has been used effectively in the form of constraints to improve the performance of the 80 class network for the confusable set of **SCV** units. It is obvious that ultimately the recognition performance is limited primarily by the features derived from the signal.

8.3.2 Applications In Image Processing

An image is represented as a two-dimensional array of pixels, with some gray value or colour associated with each pixel. Characteristics of an image are: (a) the local structure, dictated by the spatial correlations among nearby pixels, and (b) the global structure, conveying the semantics of the image. These local and global features are used in interpreting an image for recognition. Standard neural network models accept the input data in an unstructured manner, in the sense that the input to each unit in the input layer is considered independent. Thus when an image is fed as an input to a neural network the gray value of each pixel is provided as input, and the input units have no spatial structure reflecting the spatial correlations among the pixel values. Before feeding an image to a network, the image is **size-normalized**, since the dimensionality of the input to the network is fixed. In some cases like handwriting, the normalization may be carried out at word level, in which case the size, slant and position variations of the individual characters will cause difficulty for recognition by the neural network. **Thus** the main difficulty of the unstructured **nature** of the input units of neural network architectures is that there is no built-in invariance to translation, rotation, scaling and distortion at local or global features levels [LeCun and Bengio, 1995a and 1995b].

In this section we describe the development of some neural network models for three applications in image processing, namely, handwritten character recognition, image segmentation and texture classification.

Recognition of handwritten digits: The objective is to develop a recognition system for binary images consisting of handwritten **characters** or digits. Even after an overall size-normalization of each character before inputting, the system still has to take care of the variations due to **shifts** and local distortions. Convolutional network architectures are proposed for such images [LeCun and Bengio, 1995a; Säckinger et al, 1992; LeCun et al, 1990]. These architectures use three key ideas, namely, local receptive field, weight sharing and spatial subsampling. The images are applied directly to the network after some size-normalization and centering. The network architecture for an optical character recognition for a digits task is shown in **Figure 8.21** [Säckinger et al, 1992]. It is a **multilayer feed-**

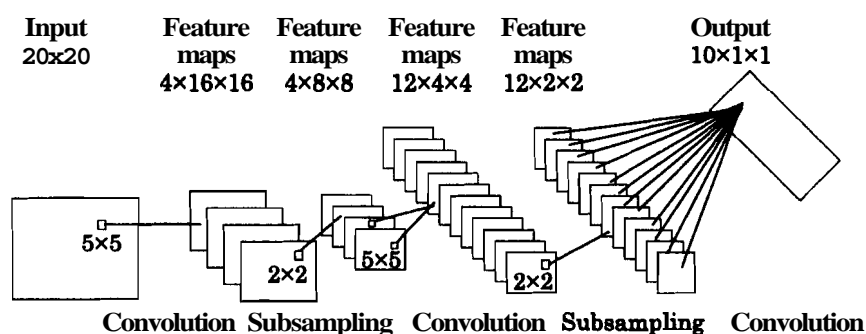


Figure 8.21 A convolutional network for optical character recognition.

forward neural network with one input layer, four hidden layers and one output layer. The input layer **consists** of **400** units corresponding to a **20 x 20** normalized binary image of each digit. The first hidden layer consists of 4 feature maps, each unit in the feature map is connected to a local receptive field of **5 x 5** pixel neighbourhood in the input layer. Two adjacent units in a feature map have their local receptive fields displaced by one unit shift in the input layer.

Weight sharing is achieved by assuming the same set of weights to each unit in the feature map. Thus there are **25** free parameters only for each of the 4 feature maps in the first hidden layer. From each local receptive field, each unit extracts some elementary features such as oriented edges, corners, **etc.** Each of the feature maps extracts different features from the same **receptive** field. The first hidden layer thus performs four separate, **2-D** nonlinear convolutions of the feature map with the input image.

The features extracted in the first hidden layer are combined in the second hidden layer to take care of shifts in the features. A shift of the input of a convolutional layer will shift only the features but the output remains the same. Therefore once the feature has been detected, its location is not very critical, as long as its position relative

to other features does not change. Therefore the second hidden layer is used to reduce the resolution by subsampling. Each unit in a feature map in the second hidden layer computes the average of the outputs of four (2×2) neighbouring units in the corresponding feature map in the **first** layer. The local receptive fields for adjacent units in this case do not overlap.

The third and fourth hidden layers perform feature extraction and subsampling **as** in the case of the **first** and second hidden layers, respectively. Each unit in each feature map of the third hidden layer extracts features from 5×5 units receptive field in the output of the second hidden layer. But each unit this time is connected to the outputs of more than one feature map in the second hidden layer. For example, 5×5 units **from** each of the two feature maps in the second hidden layer may be combined by connecting outputs of **all** the 50 units to each unit in the third layer. The weight sharing is done for each unit as in the case of the **first** hidden layer. The number of feature maps in the third layer are 12 corresponding to $4C_2$ combinations of features **from** the lower layer. The fourth hidden layer performs averaging and subsampling as in the case of the second hidden layer.

The outputs of **all** the units in the **fourth** hidden layer are connected to each of the units in the output layer. The output layer **has** 10 units corresponding to the 10 digits to be recognized. The classification is indicated by the maximum output among the 10 units in the output layer.

The network is trained using samples of handwritten digits collected **from** real life situation such **as** the digits in the postal addresses on the envelopes. Note that the weights **from** the hidden layers 1 to 2 and **from** the hidden layers 3 to 4 **are** fixed, as they perform only averaging. **All** other weights are adjusted using the standard **backpropagation** learning algorithm. The results reported in [Säckinger et al, 1992] are impressive as the network achieved an error rate of 4.9% compared to human performance of 2.5%.

The architecture implementing the convolution and subsampling operations in these networks is similar to the neocognitron developed by Fukushima [1975, 1981] (see Sec. 7.6). But **neocognitron** uses self-organization learning, thus it is unsupervised, whereas the **backpropagation** learning in convolutional networks is supervised.

Image segmentation: LeCun has described an architecture for recognition of handwritten **characters** [LeCun et al, 1990]. But when the characters form a portion of a cursive script, then it is necessary to segment the word or sentence in cursive writing into individual characters, **size-normalize**, and then feed to the convolutional network for recognition. Automatic segmentation of cursive writing into characters is not normally possible. But convolutional networks can

be used to scan over large, variable size input fields. Convolutional networks can also be replicated to scan a cursive writing. This is because the output of a network will be high when the input encloses most part of a character, and then the adjacent replicated network produces a low output. The outputs of such replicated convolutional networks have to be interpreted further. Two-dimensional replicated convolutional networks are called Space Displacement Neural Networks [SDNN] Wolf and Platt, 1994; Burges, 1995. These SDNNs together with Hidden Markov Models (HMM) have been proposed for segmentation of handwritten word recognition [Keeler et al, 1991].

Texture classification and segmentation: Intensity-based methods for segmentation of an image into different regions depend on the distributions of gray values of the pixels in each region. But images of many natural scenes appear as distribution of spatial patterns consisting of repetition or a quasi-repetition of some fundamental image features. Such image regions may be hypothesized as texture-like. Intensity-based methods do not perform well on such images. One has to adopt a texture-based scheme which involves identification and extraction of some local features which can efficiently characterize different textures in the image.

Neural networks have been successfully applied to texture classification and segmentation [Visa, 1990; Chellappa et al, 1992; Schurnacher and Zhang, 1994; Hwang, 1995; **Raghu et al, 1993, 1997a**]. In some methods the neural network itself was used to sort out the unknown and complicated neighbouring-pixel interaction, whereas in some other methods these interactions were explicitly captured by extracting features using deterministic or stochastic modelling of textures. Among the stochastic models, the Markov random field models have been studied extensively [**Rangarajan and Chellappa, 1995**]. Among the deterministic models, feature extraction by multiresolution filters like **Gabor** filters and wavelets have been explored [Greenspan et al, 1991; Schumacher and Zhang, 1994].

Image segmentation can be posed as an optimization problem, the optimality criterion involving **maximizing** a posterior probability (MAP) distribution of the intensity field given the label field [Richard and Lippmann, 1991]. The a posteriori distribution is derived using Markov random fields. An approximate solution to the MAP estimate can be realized by mapping the optimization problem as relaxation of a **Hopfield** neural network model. The relaxation could be implemented either in a deterministic manner or in a stochastic manner. In these cases the neural network can be viewed as a constraint satisfaction model, where each unit represents a hypothesis and the connection between two units as a constraint [Rangarajan et al, 1991; **McClelland and Rumelhart, 1986**].

Constraint satisfaction models for texture classification and segmentation based on Markov random fields assume a fixed resolution of the **neighbourhood**, whereas real images contain textures with large variations in the texture sizes. Moreover, the texture properties in real images **are** not stationary.

A texture classification scheme using a constraint satisfaction model was **proposed** in [Raghu, 1995; Raghu and Yegnanarayana, 1996]. The textual features **are** extracted using a set of **Gabor filters** [Bovik et al, 1990]. Image-specific constraints are used to represent domain-specific knowledge. The classifier uses two kinds of image-specific constraints, namely, feature-label interaction and label-label interaction. These constraints are defined using three random processes: feature formation, partition and label competition. The *a posteriori* probability of the label of each pixel is modelled based on these constraints in the form of Gibb's distribution. The corresponding posterior energy is used to derive a **Hopfield** network. The details of the texture classification scheme are described below.

A 2-D **Gabor** filter is an oriented complex sinusoidal grating modulated by a 2-D **Gaussian** function, and is given by

$$f(x, y, w, \theta, \sigma) = e^{-(1/2\sigma^2)(x^2 + y^2) + jw(x \cos \theta + y \sin \theta)} \quad (8.33)$$

where σ is the spatialwidth, θ is the orientation and w is the radial frequency of the **Gabor** filter [Daugman, 1985].

Gabor-filtered output of the image is obtained by convolution of the image with the **Gabor** function. The power spectrum of the filtered image at each pixel position is used as a feature to characterize that pixel. If T is the textured image, the feature vector at each spatial location (i, j) is specified as,

$$\mathbf{g}_{ij} = \{\mathbf{g}_{ij}(w, \theta, \sigma)\}_{w, \theta, \sigma} \quad (8.34)$$

where

$$\begin{aligned} \mathbf{g}_{ij}(w, \theta, \sigma) &= \left\| \iint T(x, y) f(i-x, j-y, w, \theta, \sigma) dx dy \right\|^2 \\ &= \| T(i, j) * f(i, j, w, \theta, \sigma) \|^2 \end{aligned} \quad (8.35)$$

where $*$ denotes 2-D convolution operation.

Let us assume that M **Gabor** filters (defined by different sets of w , θ and σ) are used for feature extraction. The M -dimensional vector \mathbf{g}_{ij} constitutes the feature vector to characterize the pixel (i, j) in the image.

Consider the domain $\Omega = ((i, j), 0 \leq i < I, 0 \leq j < J)$ designating the pixel positions of the given textured image T_Ω . Each pixel $s \in \Omega$ in the image T_Ω is characterized by an M -dimensional feature vector \mathbf{g}_s which is generated by **Gabor** filtering the image. Assume each \mathbf{g}_s as the realization of an M -dimensional random process G_s , called the

feature process. Let us assume that the image T_Ω consists of K different textures, so that each pixel \mathbf{s} can **take** any texture label 0 to $K-1$. The corresponding texture classes are denoted by C_0, \dots, C_{K-1} . Also, let Ω_k , a subset of Ω , be the training site for the class C_k . The **Gabor** features of the training site of a given class are used to estimate the model parameters for that class. We use the notation L_s to denote the random variable describing the texture label of the pixel \mathbf{s} .

The feature formation process is defined by the probability of assigning a value $\mathbf{g}_s \in \mathbb{R}^M$ to the feature process G_s of the pixel \mathbf{s} , given the model parameters of each texture label k . It is given by the conditional probability of $G_s = \mathbf{g}_s$, given the label of the pixel \mathbf{s} as k . Writing the feature formation process as a **Gibb's** distribution, we get

$$P(G_s = \mathbf{g}_s | L_s = k) = \frac{e^{-E_f(G_s = \mathbf{g}_s | L_s = k)}}{Z_f(L_s = k)} \quad (8.36)$$

Here $Z_f(L_s = k)$ is the normalization **factor**. One may use any of the three standard statistical distributions (namely, Gaussian, multivariate Gaussian and Gaussian mixture models) to model $P(G_s = \mathbf{g}_s | L_s = k)$ [Raghu, 1995].

Assuming a Gaussian model to describe the feature formation process, the energy function $E_f(\cdot)$ is given by

$$E_f(G_s = \mathbf{g}_s | L_s = k) = \frac{\|\mathbf{g}_s - \boldsymbol{\mu}_k\|^2}{2\sigma_k^2} \quad (8.37)$$

The normalization factor $Z_f(L_s = k)$ for this Gaussian process is given by

$$Z_f(L_s = k) = \sqrt{(2\pi)^M \sigma_k^2} \quad (8.38)$$

The model parameters, mean $\boldsymbol{\mu}_k$ and variance σ_k , are defined for each class C_k as,

$$\boldsymbol{\mu}_k = \frac{1}{S_k} \sum_{\mathbf{s} \in \Omega_k} \mathbf{g}_s \quad (8.39)$$

$$\sigma_k^2 = \frac{1}{S_k} \sum_{\mathbf{s} \in \Omega_k} \|\mathbf{g}_s - \boldsymbol{\mu}_k\|^2 \quad (8.40)$$

where $S_k = \text{card}(\Omega_k)$ is the cardinality of the subset.

We can write the conditional probability distribution of the feature formation process as,

$$P(G_s = \mathbf{g}_s | L_s = k) = e^{-\left(\frac{\|\mathbf{g}_s - \boldsymbol{\mu}_k\|^2}{2\sigma_k^2}\right) - (1/2) \log((2\pi)^M \sigma_k^2)} \quad (8.41)$$

The label-label interaction constraint is defined using two random processes: partition and label competition. The partition

process describes **the** probability of the label of each pixel s given the labels of pixels in a uniform p th order neighbourhood N_s^p of s . The neighbourhood of any pixel s is defined as the set of pixels $s + r$, $\forall r \in N_L^p$. The operator $+$ is defined as follows: For **any** pixel $s = (i, j) \in \Omega$ and for any displacement $r = (k, l) \in N_L^p$, $s + r = (i + k, j + l)$.

The partition process is expressed as a p th order **Markov** random field model defined by [Raghu, 1995, Appendix C]

$$P(L_s = k | L_{s+r}, \forall r \in N_L^p) = \frac{e^{-E_p(L_s = k | L_{s+r}, \forall r \in N_L^p)}}{Z_p} \quad (8.42)$$

This relation is also a **Gibb's** distribution.

We define the energy function E_p as follows:

$$E_p(L_s = k | L_{s+r}, \forall r \in N_L^p) = - \sum_{\forall r \in N_L^p} \beta \delta(L_s - L_{s+r}) \quad (8.43)$$

where β is a positive constant, and $\delta(\cdot)$ is the Kronecker delta function, defined by

$$\delta(l) = \begin{cases} 1, & \text{if } l = 0 \\ 0, & \text{otherwise} \end{cases} \quad (8.44)$$

The normalization factor Z_p for the partition process is given as,

$$Z_p = \sum_{\forall L_s} e^{-E_p(L_s = k | L_{s+r}, \forall r \in N_L^p)} \quad (8.45)$$

which is independent of s and k . The importance of partition process is that it acts as a mechanism for partitioning an image into **its** texture boundaries. It also smoothens the classification output at each step of relaxation.

The label competition process is based on the fact that any pixel in an image can belong to only one class, and hence a pixel can have only one label. It is expressed as the conditional probability of assigning a new label to an already labelled pixel.

Assuming that the current label of a pixel s is l , let us define the probability of assigning a new label k to that pixel as,

$$P(L_s = k | L_s = l) = \frac{e^{-\alpha \bar{\delta}(k-l)}}{Z_{c,l}} \quad (8.46)$$

where α is a positive constant. The function $\bar{\delta}$ is the inverse of **Kronecker** delta function given by,

$$\bar{\delta}(l) = \begin{cases} 0, & \text{if } l = 0 \\ 1, & \text{otherwise} \end{cases} \quad (8.47)$$

$Z_{c,l}$ is a local **normalization** factor, where c denotes that it is for the competition process.

We define the label competition process using a conditional probability $P(L_s = k | L_s = l)_{\forall l}$, where k is the new label for pixel s and l stands for any label already assigned to s . This probability can be expressed in the following form,

$$\begin{aligned} P(L_s = k | L_s = l)_{\forall l} &= \prod_{\forall l} P(L_s = k | L_s = l) \\ &= \frac{e^{-E_c(L_s = k | L_s = l)_{\forall l}}}{Z_c} \end{aligned} \quad (8.48)$$

where the energy function $E_c(\cdot)$ is,

$$E_c(L_s = k | L_s = l)_{\forall l} = \alpha \sum_{\forall l} \bar{\delta}(k - l) \quad (8.49)$$

and $Z_c = \prod_l Z_{c,l}$, independent of s and k . The energy function E_c is such that it reduces the probability of having another label when the pixel is already labelled. The competition process controls the labelling of each pixel by shutting off other possible labels for that pixel.

The objective is to find the label of each pixel in the image such that the constraints defined above are satisfied to a maximum extent. We can model the *a posteriori* probability $P(L_s = k | G_s, L_{s+r}, \forall r \in N_L^p, L_s = l)_{\forall l}$ of the label of each pixel s based on these constraints.

Using Bayes theorem, this probability can be written as,

$$\begin{aligned} P(L_s = k | G_s, L_{s+r}, \forall r \in N_L^p, L_s = l)_{\forall l} \\ = \frac{P(G_s | L_s = k) P(L_s = k | L_{s+r}, \forall r \in N_L^p) P(L_s = k | L_s = l)_{\forall l}}{P(G_s) P(L_s = k)} \end{aligned} \quad (8.50)$$

where the processes described by $P(G_s)$ and $P(L_s = k)$ are assumed independent of each other. See [Raghu, 1995, Appendix A] for details of the derivation leading to Eq. (8.50).

The *a posteriori* probability can be expressed as a **Gibbs** distribution,

$$\begin{aligned} P(L_s = k | G_s, L_{s+r}, \forall r \in N_L^p, L_s = l)_{\forall l} \\ = \frac{e^{-E(L_s = k | G_s, L_{s+r}, \forall r \in N_L^p, L_s = l)_{\forall l}}}{Z} \end{aligned} \quad (8.51)$$

where the energy function $E(\cdot)$ is given by

$$\begin{aligned} E(L_s = k | G_s, L_{s+r}, \forall r \in N_L^p, L_s = l)_{\forall l} &= E_p(G_s | L_s = k) \\ &+ \frac{1}{2} \log((2\pi)^M \sigma_k^2) + E_p(L_s = k | L_{s+r}, \forall r \in N_L^p) \\ &+ E_c(L_s = k | L_s = l)_{\forall l} \end{aligned} \quad (8.52)$$

and $Z = Z_p Z_c P(L_s = k) P(G_s)$ is a normalization constant. The total energy of the system is given by

$$E^{\text{total}} = \sum_{s, k} E(L_s = k | G_s, L_{s+r}, \forall r \in N_L^p, L_s = l)_{\forall l} \quad (8.53)$$

Substituting Eqs. (8.37), (8.43) and (8.49) in Eq. (8.52), the total Gibbs energy in Eq. (8.53) can be written as

$$E^{\text{total}} = \sum_{s, k} \left[\frac{\| \mathbf{g}_s - \boldsymbol{\mu}_k \|^2}{2\sigma_k^2} + \frac{1}{2} \log((2\pi)^M \sigma_k^2) - \sum_{\forall r \in N_L^p} \beta \delta(k - L_{s+r}) + \sum_{l=0}^{K-1} \alpha \bar{\delta}(k - l) \right] \quad (8.54)$$

This energy function can be considered as a set of feature and label constraints acting on each pixel and the corresponding possible labels. Estimation of a state configuration L , for all pixels s which minimizes the energy will yield an optimal classification of the textured image. This is the maximum *a posteriori* (MAP) estimate of the pixel labels since minimization of this energy maximizes the *a posteriori* probability given in Eq. (8.50). To evolve such an optimal state one can use a **Hopfield** neural network model whose energy function is matched with the energy expression in Eq. (8.54). This can be interpreted as a constraint satisfaction neural network with a suitable relaxation method.

The neural network consists of a 3-dimensional lattice of units. For an image T_Ω of size $I \times J$ with K possible labels for each pixel, the size of the network is $I \times J \times K$ units. Each unit in the network is designated as (i, j, k) , where $(i, j) = s$ is the pixel position and k is the label index of the pixel. The network can also be interpreted as having K layers of units. Each layer is a *label layer*. For a given pixel (i, j) , the corresponding units in different label layers constitute a column of units, which we can call a *label column*.

Each unit (i, j, k) in the network represents a hypothesis giving the label status of the pixel (i, j) . The *a priori* knowledge about the truth value of each hypothesis is represented by providing a bias $B_{i,j,k}$ to the unit. Constraints among the hypotheses are indicated by the symmetric weights $W_{i,j,k;i_1,j_1,k_1}$ between the units (i, j, k) and (i_1, j_1, k_1) .

Let $\Lambda_{i,j,k} \in \{0, 1\}$ denote the output state of the unit (i, j, k) . Let the state of the unit at the n th iteration is $\Lambda_{i,j,k}(n)$. The energy function of the network is given by

$$E^{\text{Hopfield}} = - \frac{1}{2} \sum_{i,j,k} \sum_{i_1,j_1,k_1} W_{i,j,k;i_1,j_1,k_1} \Lambda_{i,j,k} \Lambda_{i_1,j_1,k_1} - \sum_{i,j,k} B_{i,j,k} \Lambda_{i,j,k} \quad (8.55)$$

The bias values and the weights of the neural network are determined by comparing the energy expression in Eq. (8.55) with the expression in Eq. (8.54). The feature formation term in Eq. (8.54) is active only if $L_s = k$, i.e., $\Lambda_{i,j,k} = 1$, where $s = (i, j)$. The instantiation k for the label random variable L_s of the pixel $s = (i, j)$ denotes the truth value $\Lambda_{i,j,k} = 1$ of the hypothesis corresponding to the unit (i, j, k) . Similarly, the instantiation $L_{s+r} = k$, $s+r = (i_1, j_1)$ indicates that $\Lambda_{i_1, j_1, k} = 1$ for the hypothesis corresponding to the unit (i_1, j_1, k) . So the term $\delta(k - L_{s+r})$ in Eq. (8.54) is equivalent to the product $\Lambda_{i,j,k} \Lambda_{i_1, j_1, k}$, and it is active only if (i_1, j_1) is in the p^{th} order neighbourhood of (i, j) . The label competition term $\bar{\delta}(k - l)$ is 1 only if $k \neq l$. Therefore

$$\bar{\delta}(k - l) = \begin{cases} \Lambda_{i,j,k} \Lambda_{i,j,l} & \text{if } k \neq l \\ 0, & \end{cases} \quad (8.56)$$

So the energy function in Eq. (8.54) can be written as,

$$E^{\text{total}} = \sum_{(i,j),k} \left[\left(\frac{\| \mathbf{g}_{(i,j)} - \boldsymbol{\mu}_k \|^2}{2\sigma_k^2} + \frac{1}{2} \log((2\pi)^M \sigma_k^2) \right) \Lambda_{i,j,k} \right. \\ \left. + \sum_{\forall (i-i_1, j-j_1) \in N_L^p} \beta \Lambda_{i,j,k} \Lambda_{i_1, j_1, k} + \sum_{\forall l \neq k} \alpha \Lambda_{i,j,k} \Lambda_{i,j,l} \right] \quad (8.57)$$

Comparing Eqs. (8.55) and (8.57), the bias $B_{i,j,k}$ and the weight $W_{i,j,k; i_1, j_1, k_1}$ can be written as

$$B_{i,j,k} = -\frac{\| \mathbf{g}_{(i,j)} - \boldsymbol{\mu}_k \|^2}{2\sigma_k^2} - \frac{1}{2} \log((2\pi)^M \sigma_k^2) \quad (8.58)$$

and

$$W_{i,j,k; i_1, j_1, k_1} = \begin{cases} 2\beta, & \text{if } (i-i_1, j-j_1) \in N_L^p \text{ and } k = k_1 \\ -2\alpha, & \text{if } (i_1, j_1) = (i, j) \text{ and } k \neq k_1 \\ 0, & \text{otherwise} \end{cases} \quad (8.59)$$

The expression for the weight shows the connections in the network. Any unit (i, j, k) at a pixel position (i, j) has excitatory connections with strength 2β from all nodes in a neighbourhood defined by the displacement vector set N_L^p in the same label layer k . The -2α term denotes inhibitory connections from all nodes in the label column for each pixel (i, j) in the image. The structure of the 3-D Hopfield neural network is given in Figure 8.22.

In order to obtain a minimum energy state of the network, one can use either deterministic or stochastic relaxation strategies [Raghu, 1995]. Figure 8.23a shows a textured image of 256×256 pixels. The image has four different types of textures, two of them are nearly deterministic (dots and diamonds: upper left and lower right tiles)

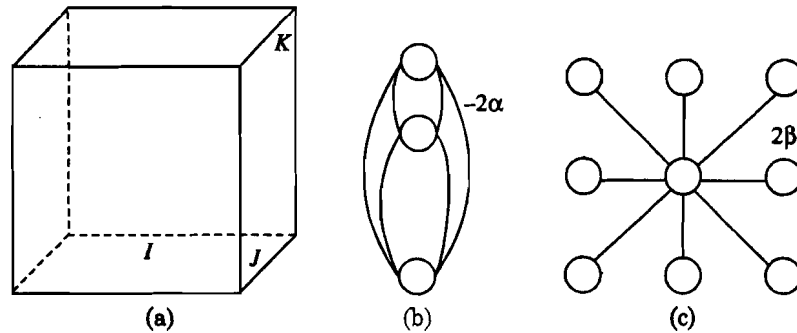


Figure 8.22 Structure of 3-D riopfield network: (a) 3-D lattice of size $I \times J \times K$. (b) Connections among nodes in the label column of each pixel. Each connection is of strength -2α . (c) Connections from a set of neighbouring nodes to each node in a label layer. Each connection has a strength 2β .

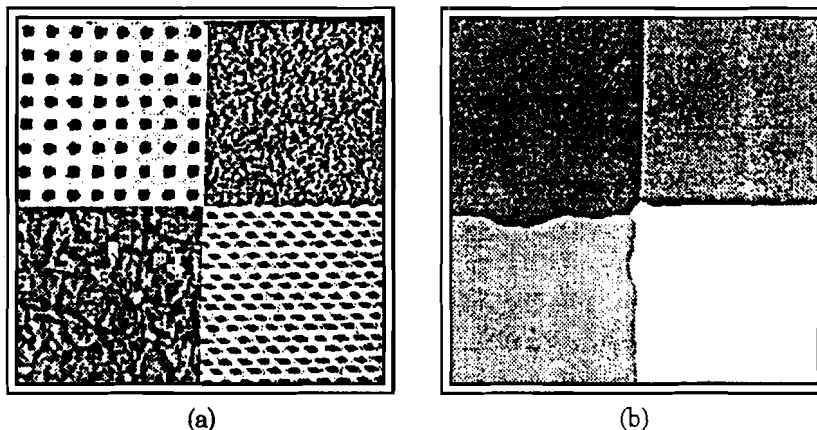


Figure 8.23 Image classification using the constraint satisfaction network for an image with four texture tiles. (a) Original image containing four textures. (b) Final segmentation result.

and the other two are stochastic (sand and pebbles: upper right and lower left tiles). A 16-dimensional feature vector is used to represent each pixel in the image. The 16 dimensions correspond to 16 different Gabor filters with 2 spatialwidths, 2 frequencies and 4 orientations.

A training site consisting of 1000 pixels per class is used for parameter estimation for each class. For this image, a simple Gaussian model is used for feature formation process. A deterministic relaxation strategy is applied for obtaining the maximum *a posteriori* probability state. Figure 8.23b shows the result of classification.

The approach was also proved to be successful for classification of multispectral band imagery from remote sensing. For illustration, an image obtained by the NASA Jet Propulsion Laboratory using

their Spaceborne Imaging Radar-C and X-Synthetic Aperture Radar (SIR-C/X-SAR) setup [Jordan et al, 1991] is considered. The SIR-C/X-SAR setup acquires images in three microwave bands with four linear polarization states. Figure 8.24a is a gray level representation of one such image with three spectral components. The texture-like image is known to have four classes. The image is represented using 8 Gabor filters for each of the three spectral bands, giving a 24-dimension feature vector for each pixel. The feature formation process is described in terms of a multivariate Gaussian distribution in which the covariance matrix characterizes the interband correlation as well as the intraband correlation of the Gabor-filtered multispectral imagery. Stochastic relaxation using simulated anneal-

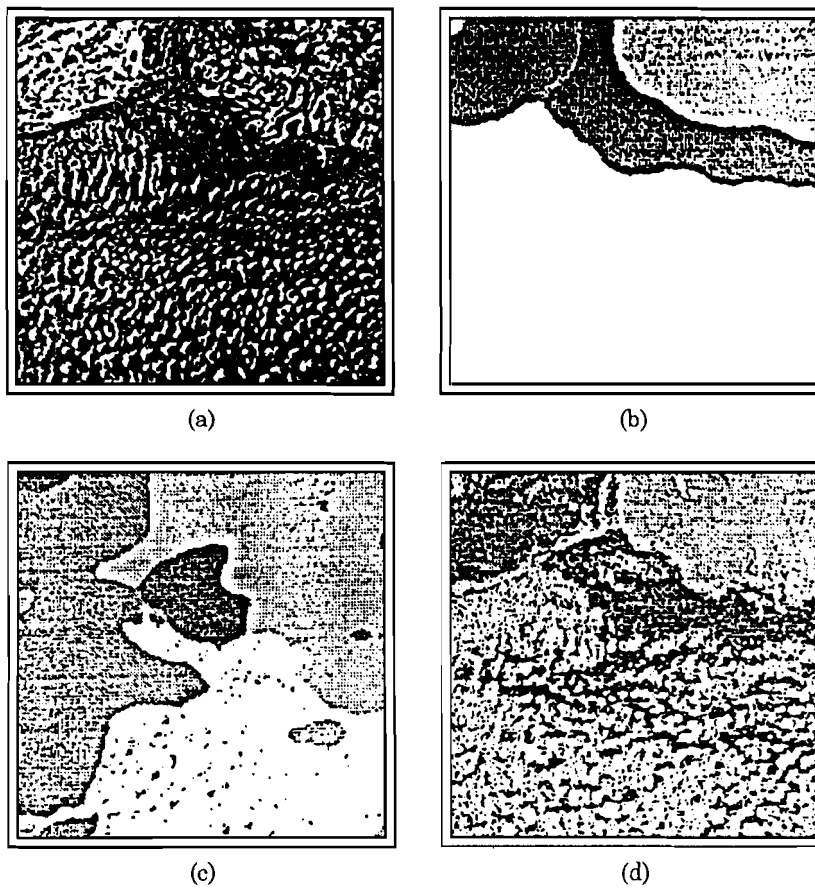


Figure 8.24 Multispectral classification of SAR imagery using constraint satisfaction network with multivariate Gaussian model for feature formation. (a) Gray level representation of the image. (b) Classification based on multispectral textural information. (c) Classification using textural information in one band. (d) Classification based on gray level information in multispectral data.

ing can be used to obtain the minimum energy state of the **Hopfield** model [Raghu and Yegnanarayana, 1997]. Figure 8.24b shows the classification based on multispectral-textural information. For comparison, Figure 8.24c gives the result of classification using textural information in one band only. Figure 8.24d shows the result of classification using only the pixel intensities of the multispectral data in all the three bands. That means the textural features of the image are not used. It can be seen from these results that incorporation of multispectral and textural knowledge in the classification process improves the performance much better than a scheme based only on the textural information or only on the multispectral information.

8.3.3 Application in Decision Making

Like speech and image processing, decision making is also a purely human attribute. Just as the feature extraction process is in-built into our biological system, decision making also seems to be in-built into our system. We **arrive** at an intelligent decision based on partial, noisy and sometimes inconsistent data mainly because we seem to invoke our acquired knowledge over a period of time. It is difficult to articulate the acquired knowledge, although expert systems have been developed to represent the articulated knowledge by a domain expert to arrive at a decision [Duda and Shortliffe, 1983; Gallant, 1993]. The knowledge in an expert system is represented in the form of *if-then* rules, and the rules are fired in a desired manner. Since expert systems require this knowledge explicitly, there is a basic limitation on the utility of expert systems in many decision **making** situations.

Due to the inherent nonlinearity and also due to the learning ability, neural networks appear to be promising in some decision **making** applications [Baxt, 1990; Tan et al, 1996]. The nonlinear regression capability of **feedforward** neural networks and the constraint satisfaction property of feedback neural networks are exploited in medical diagnosis, investment management and automated inspection of industrial parts [Chen, 1996]. The threshold output function of the nonlinear processing unit in a neural network can be effectively used for decision making [Gallant, 1995]. For example, the symptoms and parameters of a patient can be given as input to a MLFFNN, and the corresponding diseases as output of the network. The network can be trained using the known data from medical records. The trained network can now be used as an automated medical diagnosis tool, so that any new test input will give an idea of the disease. Such a network is called *connectionist expert system* [Gallant, 1988]. The network can be continuously updated with additional knowledge acquired in the form of input and output parameters. The main disadvantage of these systems is that

it does not provide an explanation or justification for the decision arrived at by the system as in the case of a rule-based expert system. Neural network-based systems were developed, for example, for **skin** diseases diagnosis and for low back pain diagnosis [Yoon et al, 1989; Bounds et al, 1988]. The scope of these neural network expert systems could be enhanced significantly using fuzzy logic to represent the linguistic form of input and output variables [Chen, 1996, Ch. 7]. Many other useful applications of neural networks-based decision systems have been implemented, such as investment management, credit scoring, fraud detection and fault detection [Burke, 1992; McGough, 1992].

A MLFFNN for pattern mapping can be viewed as a **nonparametric** nonlinear regression analysis. This property has been exploited for forecasting applications, especially in exchange rate forecasting and stock prices [Ungar, 1995; Zapranis and Refenes, 1995].

8.4 Summary

The field of artificial neural networks came into prominence mainly because of our inability to deal with natural tasks such as in speech, image processing, decision **making** and natural language processing. Our discussion on some tasks in these areas suggests that we still have not succeeded in realizing the natural human-like preprocessing of speech and images for feature extraction and in modelling the higher levels of cognition for decision **making** and for natural language processing [Morgan and Scofield, 1991]. It is likely that new models may evolve to deal with issues such as invariant pattern recognition, interaction of local and global knowledge, stability-plasticity, feature extraction from temporal sequences like image sequences and matching patterns at semantic levels.

Some recent trends in *ANN* research are briefly discussed in this section. Appendix F gives a more detailed discussion on some of the trends.

The learning laws in *ANN* basically optimize certain objective functions which reflect the constraints associated with the given task. Most of the learning laws utilize gradient based approach for this optimization purpose. However, due to its deterministic nature, gradient based methods frequently get stuck at local optima or saddle points. This is because the step size and step direction of the optimization process are dictated by the local information supplied by the gradient. We try to overcome this drawback by choosing the step size and step direction stochastically in a controlled manner. The efficiency of the search for global optimum can be enhanced further if it is carried out in parallel. Evolutionary Computation is one such (biologically inspired) method, where a population of solutions are explored over a sequence of generations to reach globally **optimal**

solution. The integration of evolutionary computational technique **into** *ANN* models is called Neuro-Evolutionary technique, which can be used to enhance the learning capability of the model. The technique is also useful to determine suitable topology of the network and to select proper learning rule [Fogel, 1994].

In a classification task an *ANN* is used to find the decision regions in the input pattern space. But if the **patterns** from **different** classes are overlapping, then it is difficult for an *ANN* to find the class boundaries. In pattern mapping also similar problems may arise when the inputs or target outputs are ill-defined or *fuzzy*. These situations are common in many pattern recognition tasks because of inherent fuzziness associated with human reasoning. The pattern recognition capability of an *ANN* can be made powerful, if fuzzy logic is incorporated into the conventional *ANN* models. The resulting systems are called *Neuro-Fuzzy systems* [Lin and Lee, 1996].

In some cases an *ANN* training may end up finding a **class** boundary when the same input pattern belongs to one class in **some** examples, and to another class in some other examples. This scenario is due to the presence of *rough* uncertainty, which arises from *indiscernibility* of the objects based on input features. The classification ability of an *ANN* can be **significantly** improved if the input data set is processed to reduce the rough uncertainty. Motivated by this idea, a new promising area based on *Neuro-Rough* synergism is emerging. It has already been employed to reduce the size of the input data set, to determine the number of input units needed and to accelerate networks training [Pawlak, 1991; Pawlak et al, 1995; Sarkar and Yegnanarayana, 1997c].

The ability of a feedback network to store patterns can be improved, if we can exploit the *chaotic* nature of the networks dynamics. This observation has resulted in proposing hybrid neurons, known as *chaotic neurons*. Different models of chaotic neurons are studied, and initial results are quite promising [Andreyev et al, 1996].

There are several other attractive paradigms which can be fused with the current *ANN* techniques. For example, *Artificial Ant System* [Dorigo et al, 1996], *Cultural Evolution* [Belew, 1989], *DNA Computing* [Adleman, 1994], and *Immunity Net* [Hunt and Cooke, 1996], seem to be attractive and viable approaches that can be amalgamated with *ANN*.

One key advantage of *ANN* is that it is adaptive. Many existing paradigms can be fused into it easily. Although, as of now there are no guidelines for developing hybrid paradigms, the urge to develop models to perform human cognition tasks will continue to motivate researchers to explore new directions in this field.

The most important issue for solving practical problems using the principles of *ANN* is still in developing a suitable architecture to solve a problem. This continues to dominate this research area. *ANN*

research has to expand its scope to take into account the fuzzy nature of real data and reasoning, and the complex (unknown) processing performed by the human perceptual mechanism.

It is possible to view research in ANN along the following directions:

Problem level: This involves issues in mapping the real world problems as pattern processors. This may require good understanding of human information processing both from the psychological and the biological angles.

Basics level: It is necessary to evolve better models for neurons as processing units, their interconnections, dynamics (activation and synaptic), learning laws and recall procedures.

Functional level: This involves development of basic structures which can solve a class of pattern recognition problems. These form building blocks for development of new architectures.

Architectural level: This requires ideas to evolve new architectures from known principles, components and structures to solve complex pattern recognition problems. It is possible that the problems may be tailored somewhat to suit the architectural principles.

Application level: The objective is to solve a given practical problem using generally the principles of ANN, but with ideas from other areas such as physics and signal processing.

Review Questions

1. What are some direct applications of the principles of neural networks? Why are they called 'direct' applications?
2. Under what conditions mere correlation matching can be successfully applied for pattern classification?
3. What is meant by complexity of a set of **objects** for classification? Explain this in relation to Olympic symbols and printed characters.
4. Why is it that mere correlation matching cannot be used for classification of deformed patterns?
5. Discuss the significance of backpropagation learning in situations like 'Contract Bridge Bidding'.
6. Explain why information retrieval can be viewed as a direct application of neural network principles.
7. How is an optimization problem formulated for solution using a neural network model?
8. Explain the steps in the solution of a general optimization problem by a neural network.

9. What is a local minima problem in optimization?
10. How is mean-field annealing applied in the solution of optimization problems?
11. Explain the formulation of the graph bipartition problem as an optimization problem.
12. Explain the difficulties in the solution of travelling salesman problem by a feedback neural network.
13. Explain how an image smoothing problem can be solved by principles of neural networks.
14. What is the problem in vector quantization?
15. Explain how neural network principles **are** useful in control applications.
16. Explain why a speech recognition problem is not a direct application of neural network principles.
17. What is the significance of neural networks in the **NETtalk** application?
18. What neural network ideas are used in the development of phonetic typewriter?
19. Why is the problem of vowel classification based on Peterson and Barney formant data a difficult one for neural networks?
20. Discuss some neural network methods for classification of consonant-vowel (**CV**) segments of speech.
21. What is a time-delay neural network architecture? How is it suitable for classification of CV segments?
22. What is a modular architecture in neural networks?
23. What is the problem in the classification of large (20 or more) number of CV segments?
24. How is a modular architecture useful for classification of large number of CV segments?
25. Discuss the significance of a constraint satisfaction model for combining multiple evidences for large number of classes.
26. Explain how a constraint satisfaction model can be exploited for improving the recognition accuracy for CV units.
27. What is the problem in the recognition of handwritten digits?
28. What is a convolutional network architecture and how is it useful for the problem of handwritten digit recognition?
29. What is image segmentation problem?
30. How are convolutional networks used for segmentation of handwritten characters in a cursive script?

31. Explain how neural network principles are useful for a texture classification problem.
32. Explain the constraints in a texture classification problem that can be used as a priori knowledge for a network formulation.
33. Discuss the feature formation process in texture analysis as Gibbs distribution.
34. Discuss the partition process in texture analysis as Gibbs distribution.
35. Discuss the label competition process in texture analysis as Gibbs distribution.
36. Show the formulation of maximum a posteriori probability estimation for texture analysis as an energy minimization problem.
37. Discuss a neural network model for energy minimization in a texture classification problem.
38. Discuss the relaxation strategies (deterministic and stochastic) for texture classification.
39. What are the issues in decision **making** problems?
40. Discuss the application of neural network principles for decision **making**.
41. What are some recent trends in neural networks?

Problems

1. For the Hamming net given in Figure 8.3, the input to the unit i in the upper **subnet** is given by

$$x_i = \sum_{j=1}^M w_{ij} a_{ij} + M/2$$

where a_{ij} is the j th component of the input vector \mathbf{a}_i . Show that

$$x_i = M - HD(\mathbf{a}_i, \mathbf{a}_i)$$

where $HD(\mathbf{a}_i, \mathbf{a}_i)$ refers to the Hamming distance between the input vector \mathbf{a}_i and the vector \mathbf{a}_i corresponding to the i th unit. (See [Zurada, 1992, p. 3921])

2. Compute the weight matrix of a Hamming network for the following three prototype vectors

$$\mathbf{a}_1 = [1 \ -1 \ 1 \ -1 \ 1 \ -1]^T$$

$$\mathbf{a}_2 = [-1 \ -1 \ -1 \ -1 \ -1 \ 1]^T$$

$$\mathbf{a}_3 = [1 \ 1 \ 1 \ -1 \ -1 \ 1]^T$$

Find the output of each unit for the input vector $\mathbf{a}_i = [1 \ 1 \ 1 \ 1 \ 1 \ 1]^T$. Verify that the Hamming distance computed by the

- network agrees with the **actual** Hamming distance. Find the steady state output of the upper **subset** of the Hamming network.
3. Explain why the capacity of the Hamming network to store patterns is higher than the **Hopfield** network. (See **Kung**, 1993, p. 59)
(Hint: The number of connections increase linearly for the Hamming network as the number of bits in the input patterns increase. **On** the other hand, the number of connections increase quadratically with the number of bits for the **Hopfield** network.)
 4. Study the performance of the graph bipartition problem for the example shown in Figure 8.10.
 5. Study the performance of the Travelling Salesman problem for ten cities, using the formulation given in Eq. (8.22). Assume the distances between the cities. Examine the effect of the parameters α, β, γ on the **performance**.
 6. Implement the image smoothing algorithm for an image with discontinuities in a **1-D** case as shown in Figure 8.13.
 7. Weighted matching problem [**Hertz** et al, 1991, p.72] : Suppose there are N (even) points in d -dimensional space with known distances between each pair of points. The problem is to link the points together in pairs, with each point linked to exactly one other point, so as to **maximize** the total length of the links. Let d_{ij} be the distance between the points i and j . Let n_{ij} be a unit in a **Hopfield** network, such that the state $n_{ij} = 1$ indicates that the points are linked and the state $n_{ij} = 0$ indicates that the points are not linked. The optimization problem involves minimizing the total length of links $L = \sum_{i < j} d_{ij} n_{ij}$ subjected to the constraint that $\sum n_{ij} = 1$ for all i .
Assuming $n_{ij} = n_{ji}$ and $n_{ii} = 0$, solve the problem for $N = 4$. Assume **four** random points in a unit square in the 2-dimensional space.
Discuss the solutions of the problem obtained by using
 - (a) Deterministic relaxation algorithm.
 - (b) Stochastic relaxation using simulated annealing (use suitable stochastic update and annealing schedule).
 - (c) Mean-field annealing.
 8. Study the solution of the Travelling Salesman problem using SOM for the following different cases:
 - (a) 30 cities, 30 units
 - (b) 30 cities, 100 units
 - (c) 30 cities, 500 units
 - (d) 100 cities, 100 units
 - (e) 100 cities, 500 **units**

Appendix A

Features of Biological Neural Networks through Parallel and Distributed Processing Models

Some of the features of the biological neural networks were demonstrated using parallel and distributed processing (PDP) models in [Rumelhart and McClelland, 1986; McClelland and Rumelhart, 1986; McClelland and Rumelhart, 1988]. We will consider two of those models for illustration, namely, the Interactive Activation and Competition (IAC) model and the Constraint Satisfaction (CS) model.

A.1 Interactive Activation and Competition Model

The objective of the IAC model is to illustrate the process of retrieving general and specific knowledge **from** stored knowledge of specifics [McClelland and Rumelhart, 1988, p. 391. Some of the features of human memory that are illustrated through this model are: Retrieval by key (name) and by context, retrieval with noisy clues, assignment of default values and spontaneous generalization. The model is illustrated through the example of Jets and Sharks database described in [McClelland, 1981] and given in Table A.1. Information in such a data, if stored in a computer memory, can be **accessed** by name or by any other set of items, provided the method of access is preprogrammed into the system. Moreover, certain characteristics of the data like the distribution of persons in different age groups, or the nearly 'common' characteristics among some persons, etc., can be obtained only by explicitly programming to derive the information **embedded** in the data. In other words, any information in the data has to **be** sought explicitly. Whereas human memory stores the data in terms of the patterns implicit in the data automatically, and these patterns can be recalled even with partial clues. These features of human memory can be demonstrated through a parallel and **distri-**

Table A.1 The Jets and the Sharks Data for IAC Model [Adapted from McClelland and Rumelhart, 1988, Ch. 2; with permission from MIT Press].

Name	Gang	Age	Edu	Mar	Occupation
Art	Jets	40's	J.H.	Sing.	Pusher
Al	Jets	30's	J.H.	Mar.	Burglar
Sam	Jets	20's	Col.	Sing.	Bookie
Clyde	Jets	40's	J.H.	Sing.	Bookie
Mike	Jets	30's	J.H.	Sing.	Bookie
Jim	Jets	20's	J.H.	Div.	Burglar
Greg	Jets	20's	H.S.	Mar.	Pusher
John	Jets	20's	J.H.	Mar.	Burglar
Doug	Jets	30's	H.S.	Sing.	Bookie
Lance	Jets	20's	J.H.	Mar.	Burglar
George	Jets	20's	J.H.	Div.	Burglar
Pete	Jets	20's	H.S.	Sing.	Bookie
Fred	Jets	20's	H.S.	Sing.	Pusher
Gene	Jets	20's	Col.	Sing.	Pusher
Ralph	Jets	30's	J.H.	Sing.	Pusher
Phil	Sharks	30's	Col.	Mar.	Pusher
Ike	Sharks	30's	J.H.	Sing.	Bookie
Nick	Sharks	30's	H.S	Sing.	Pusher
Don	Sharks	30's	Col.	Mar.	Burglar
Ned	Sharks	30's	Col.	Mar.	Bookie
Karl	Sharks	40's	H.S.	Mar.	Bookie
Ken	Sharks	20's	H.S.	Sing.	Burglar
Earl	Sharks	40's	H.S.	Mar.	Burglar
Rick	Sharks	30's	H.S.	Div.	Burglar
01	Sharks	30's	Col.	Mar.	Pusher
Neal	Sharks	30's	H.S.	Sing.	Bookie
Dave	Sharks	30's	H.S	Div.	Pusher

buted processing model shown in Figure A.1. In the figure the units are organized in different pools, such as 'names' pool, 'age' pool, etc. The number of units in each pool corresponds to different possibilities in that category, as for example, 27 units in 'names' pool and 3 units in 'age' pool, etc. There are as many pools as there are categories (6 in this case), plus one additional pool called 'instance' pool. Units within each pool are **connected** in an 'inhibitory' manner, *i.e.*, the output of each unit is fed with a negative weight to all other units in the same pool. The units in each pool are **connected** to the corresponding units in the instance pool in an 'excitatory' manner, *i.e.*, the connection weight is positive. For example, the 'Ralph' unit in the 'names' pool, 'Jets' units in the 'gang' pool, '30' unit in the 'age'

Interactive Activation and Competition Model

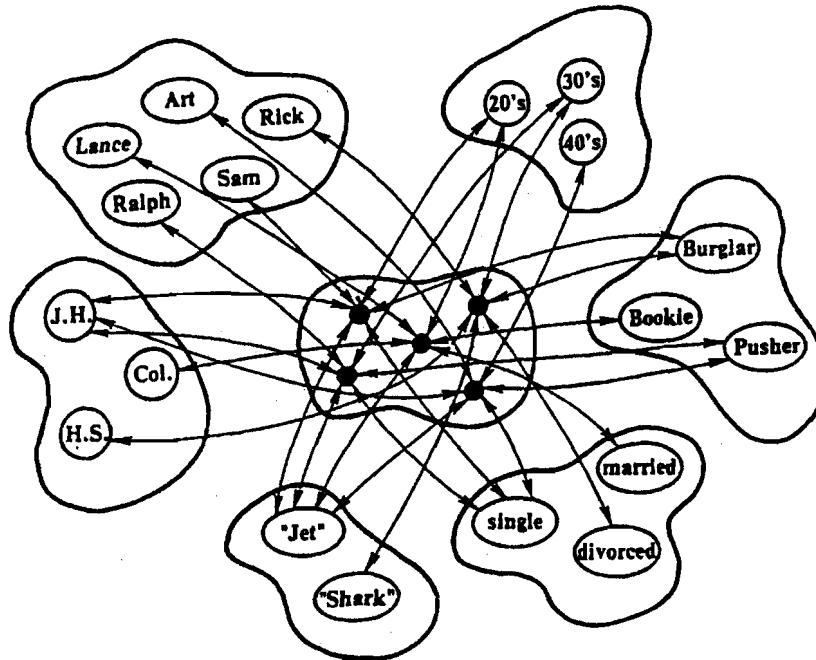


Figure A.1 The units and connections for some of the individuals in Table A.1 [Adapted from McClelland and Rumelhart, 1988, Ch. 2; with permission from MIT Press].

pool, 'JH' unit in the 'education' and the 'pusher' unit in the 'occupation' pool are **all connected** to the 'Ralph' unit in the 'instance' pool with positive weights. The units in the 'instance' pool are called 'hidden' units, since by design they are not accessible for external input or output. The units in all other pools are called 'visible' units. Only 5 of the 27 **units** are shown in the 'names' pool and 'instance' pool for illustration. Also the inhibitory connections within the pools are not shown in the figure.

There are a total of 68 units in the model. Each unit computes a weighted sum of the input values to the unit, fed from other units as well as **from** external inputs, if any. The weighted sum or the activation value is passed through a nonlinear output function, which gives as output the activation value itself, if the sum lies between some prespecified minimum and maximum values. Otherwise, the function gives either the minimum value at the lower limit or maximum value at the upper limit. The state of the model is described as the outputs of all the 68 units at any given instant of time. Starting from any state, the next state can be computed by selecting a unit at random and computing the weighted sum of its inputs first and then the output of the unit. Due to change in the value of the output of this unit, the model goes to a different state. Then another unit is selected at random, and the new state for that unit is determined.

All the units are updated by selecting the units in a random sequence, to compute one 'cycle' of activation dynamics. **After** several cycles, the model is guaranteed to reach a stable equilibrium state, when there will not be any further change in the state of the model.

For each set of external inputs, the model reaches a stable state eventually. **From** the stable state the stored data can be read out. For example, if we want the data about 'Ralph', an external input is given to the input of the 'Ralph' unit in the 'names' pool. Starting with some initial state, the network activations are computed for several cycles, until an equilibrium state is reached. At equilibrium, there will be one unit in each pool having large positive value. Those units correspond to the data that belongs to 'Ralph'. The details of implementation are discussed in [McClelland and Rumelhart, 1988].

The model demonstrates several features of the functioning of the biological neural network in human memory. Some of the features are: (a) Retrieving an individual's data from his name, (b) Retrieval from a partial description, (c) Graceful degradation, (d) Default assignment, and (e) Spontaneous generalization for novel inputs. The most important point is that the model stores the patterns embedded in the given data. Therefore one could get from the model even such **information** for which the model was not explicitly designed. For example, in the feature of default assignment, the model gives possible good guess about missing information, even though we do not know certain things about an individual. It evaluates the relative strengths of the attributes from the given data in a complex manner, which is difficult to describe explicitly. Thus this model clearly brings out the distinction between computer memory and human memory for storage and retrieval of information. The model also brings out the features of content-addressable memories and associative memories for information retrieval. Note the distributed nature of the memory in this model in the sense that the information is distributed in the weights throughout the network. Also note the parallel and distributed nature of the **activation** dynamics when the model is realized in hardware, and when it is allowed to relax naturally changing from one state to another until an equilibrium state is reached from the given initial state and external input. There will be several equilibrium states, some of them correspond to the desired information about the data in the Table A.1 [McClelland and Rumelhart, 1988, p. 64].

A.2 Constraint Satisfaction Model

We consider another PDP model, namely the constraint satisfaction model, to illustrate how we attempt to build concepts or arrive at conclusions based on some limited, partial, and sometimes partially erroneous knowledge. The key idea in this model is that a large

number of weak constraints together will evolve into a definitive conclusion. For example, even in an apparently simple task of recognition of handwritten characters, it is difficult to articulate precisely what we capture as features in the patterns of several samples of each character. But when we are presented with a new sample of a handwritten character, most of the time we have no difficulty in recognizing it correctly. It is likely that from the samples of a handwritten character we may have captured a large number of weak evidence of features in our memory, so that with a new sample as input, the memory relaxes to a state that satisfies as many constraints as possible to the maximum extent.

The above idea of constraint satisfaction can be captured in a PDP model consisting of several units and connections among the units. In this model the units represent *hypotheses* and the connections represent the *knowledge* in the form of constraints between any two hypotheses. It is obvious that the knowledge cannot be precise and hence the representation of the knowledge in the form of constraints may not also be precise. So the solution being sought is to satisfy simultaneously as many constraints as possible. Note that the constraints usually are weak constraints and hence all of them need not be satisfied fully as in the normal constrained optimization problems. The degree of satisfaction is evaluated using a *goodness-of-fit* function, defined in terms of the output values of the units as well as the weights on the connections between units.

The constraint satisfaction PDP model is illustrated here with an example of how our concepts of various types of rooms can be captured by the model from samples of description of these rooms [McClelland and Rumelhart, 1986, Ch. 14]. Let us assume that we collect data from subjects about their understanding of the following five types of rooms: Living room, kitchen, bedroom, office and **bathroom**. In order to elicit information from the subjects, 40 descriptors are provided to them, in terms of which each subject can be asked to give his/her view of the above room types. The descriptors are shown in Table A.2.

Table A.2 The 40 Room Descriptors used in the Constraint Satisfaction Model [From McClelland and Rumelhart, 1988, p. 64; with permission from MIT Press].

ceiling	walls	door	windows	very-large
large	medium	small	very-small	desk
telephone	bed	typewriter	bookshelf	carpet
books	desk-chair	clock	picture	floor-lamp
sofa	easy-chair	coffee-cup	ashtray	fireplace
drapes	stove	coffeepot	refrigerator	toaster
cupboard	sink	dresser	television	bathtub
toilet	scale	wen	computer	clothes-hanger

Each subject can be asked to mark which of these descriptors are valid for each room type. **From** the data collected from a number of subjects, the weights between units are captured. Here the units represent the descriptors. The output of a unit is binary indicating whether the description is present or not. The connection weights between units are derived **from** the co-occurrence patterns of the descriptors in the responses of the subjects for all the room types. One method of describing the weights is as follows:

$$w_{ij} = -\log \frac{P(x_i=0 \text{ and } x_j=1)P(x_i=1 \text{ and } x_j=0)}{P(x_i=0 \text{ and } x_j=0)P(x_i=1 \text{ and } x_j=1)} \quad (\text{A.1})$$

where w_{ij} represents the symmetric weight connecting the unit j and unit i . The numerator represents the product of probabilities that the hypotheses of the units i and j are competing with each other, i.e., one unit has the value $x_i = 1$ and the other has the value $x_j = 0$. The denominator represents the product of probabilities that the hypotheses of units i and j support each other. **Thus** if the evidence is greater for **supporting** the hypotheses, then the weight w_{ij} will be positive, otherwise it is negative. Note that the probabilities can be replaced by the relative frequencies in the data. In addition, each unit can have a bias reflecting the prior information about the hypothesis the unit represents. In this case the bias is given by

$$b_i = -\log \frac{P(x_i=0)}{P(x_i=1)} \quad (\text{A.2})$$

There can be direct evidence for a hypothesis through an external input. The corresponding input unit could be clamped indicating that the hypothesis is either always 'on' or always 'off'. Other types of external input could be a graded one indicating a weak constraint.

A constraint satisfaction model can be displayed by means of a **Hinton** diagram as shown in **Figure A.2** [McClelland and Rumelhart, 1988, p. 661]. Each larger square in the figure represents a unit. There are 40 units corresponding to 40 descriptors. Within the square of each unit a replica of all the 40 units are displayed as dots, each dot representing the unit in its relative position in the diagram. Around each dot, a white square indicates a positive weight connecting the unit representing the dot and the unit enclosing the dot. Thus for example, the white square on the second dot in the unit for 'ceiling' indicates that the 'walls' unit is connected to the ceiling unit with a positive weight. The size of the small white square indicates the strength of the positive connection. Likewise in the last unit corresponding to 'oven', the small dark square around the last but one dot indicates that the units 'oven' and 'computer' are connected with a negative weight. There are many units which have no **connections** at all.

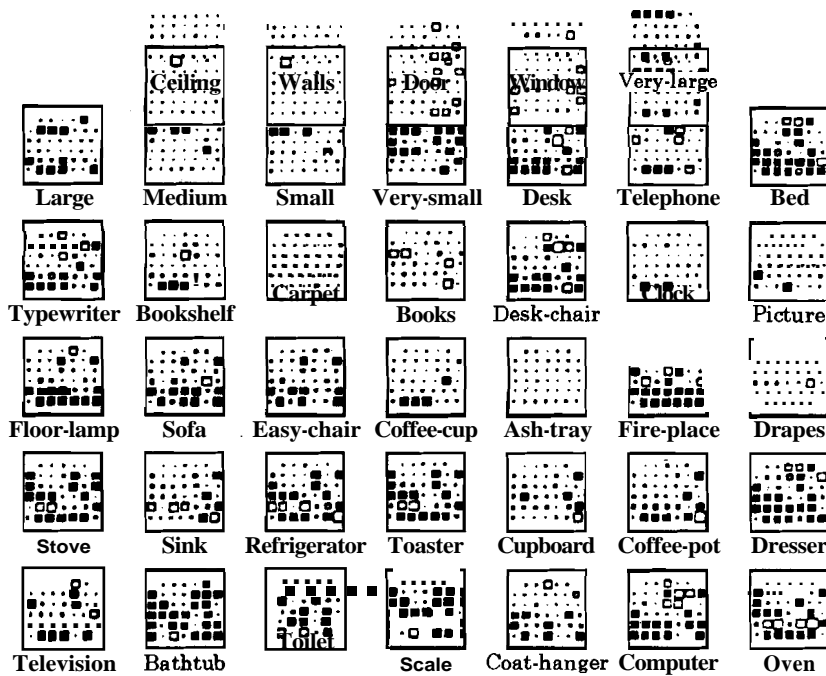


Figure A.2 The figure uses the method of Hinton and Sejnowski [1986] to display the weights. Each unit is represented by a square. The name below the square is the descriptor represented by each square. Within each unit, the small black and white squares represent the negative or positive weights, respectively, from that unit to each of the other units in the system. The relative position of the small squares within each unit indicates the unit with which the unit is connected [Adapted from McClelland and Rumelhart, 1988, Ch. 3; with permission from MIT Press].

The model is allowed to relax by computing the next state for each unit selected at random, computing sum of its weighted inputs and thresholding the weighted sum using a hard-limiting output function. For a given external evidence, say like 'oven' and 'ceiling' in Figure A.3, the state of the network after each cycle is shown in the figure. After 17 cycles the model settles down to an equilibrium state closest to the given external evidence, and the state description gives a description of the concept of the room satisfying the external evidence, namely 'kitchen', in this case. Thus the PDP model clearly demonstrates the concepts of rooms captured by the weak constraints derived from the data given by the subjects. The model captures the concepts of the five room types at the equilibrium states corresponding to the description that best fits each room type. A goodness-of-fit function (g) is defined for each state (x_1, x_2, \dots, x_N) , where $x_i = 1$ or 0, as

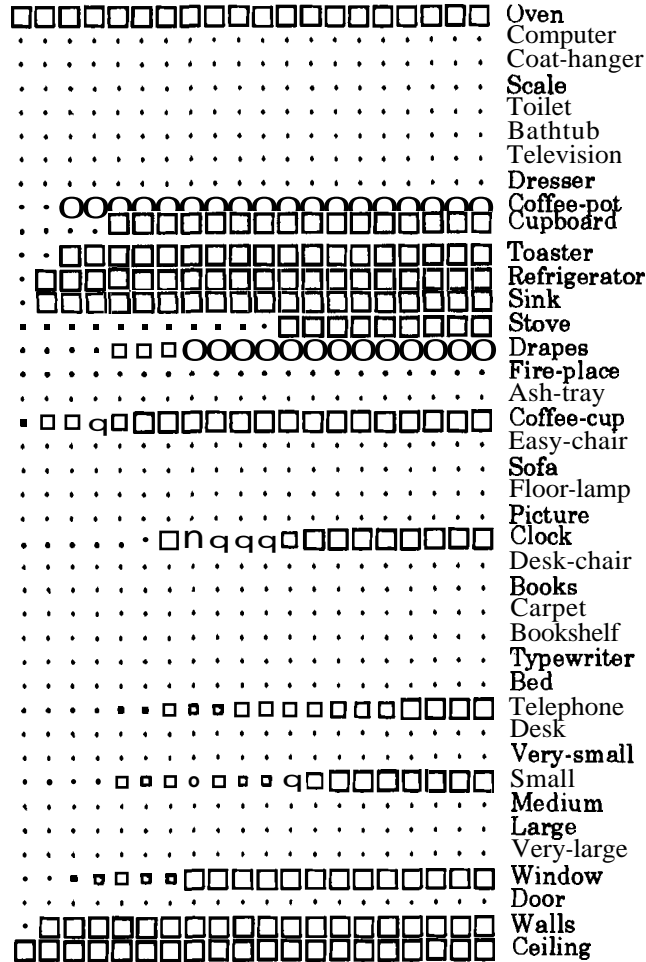


Figure A.3 The state of the CS model after each cycle, starting with an initial state where the units 'ceiling' and 'oven' are clamped. The system settles down to a prototype for the type of room most closely related to the clamped units, which in this case is 'kitchen' [Adapted from Rumelhart et al., 1986b; with permission from MIT Press].

$$g = \sum_{i,j=1}^N w_{ij} x_i x_j + \sum_{i=1}^N e_i x_i + \sum_{i=1}^N b_i x_i \tag{A.3}$$

where e_i is the **external** input to the i th unit and b_i is the bias of the unit i . At each of the equilibrium states the goodness-of-fit function is maximum.

The model not only captures the concepts of the room types, but it also gives an idea of their relative separation in the 40 dimensional space. In order to visualize the shape of the goodness-of-fit surface

as a function of the state of the model, the goodness-of-fit surface for states lying on a plane passing through the three equilibrium states corresponding to 'kitchen', 'office' and bedroom is shown in Figure A.4 [McClelland and Rumelhart, 1986]. Note that the x - y plane

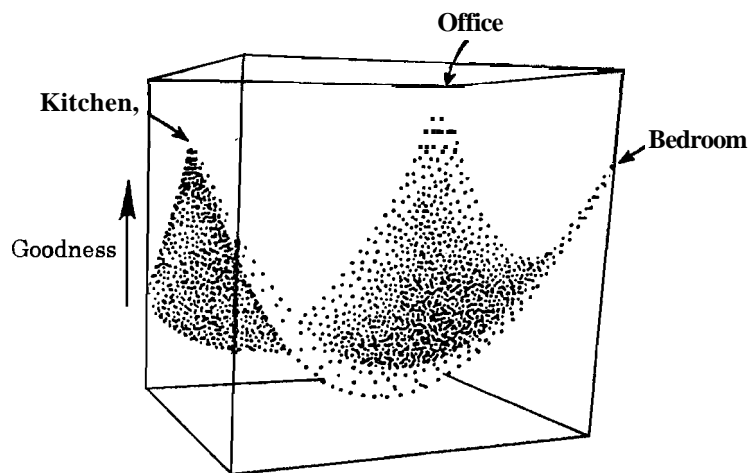


Figure A.4 The value of the goodness-of-fit function for the states on the plane passing through the three goodness maxima corresponding to the prototypes for 'kitchen', 'bedroom' and 'office'. [Adapted from Rumelhart et al., 1986b; with permission from MIT Press].

corresponds to the plane passing through the selected equilibrium states. The shaded curve gives values of goodness-of-fit function for all states on this plane. Obviously, the function is maximum at each of the three equilibrium states. The figure shows that the concepts of 'office' and bedroom have more in common than that of kitchen, since the peak for the kitchen is far off from the other two.

The model will have several other equilibrium states corresponding to some local peaks of the goodness-of-fit function. These peaks do not correspond to the room types intended to be captured by the model from the data. In general the model is allowed to relax in order to reach some global peak overcoming the insignificant local peaks. This is generally accomplished using the method of simulated annealing.

Note that this constraint satisfaction model is a good illustration of learning or capturing the concepts from examples. The derived constraints will be more representative of **the knowledge** if the examples are large in number. It is also interesting to note that there is no uniqueness about the model in terms of descriptors, weights, etc. The objective is to represent somehow the **knowledge** of the problem domain in the form of weights. Obviously the model will function better if the number of hypotheses or units and the

constraints are large, and the number of concepts is much smaller compared to the dimensionality of the state of the model. The model clearly illustrates the pattern behaviour in human cognition, which we cannot articulate precisely, but we use effectively in classifying correctly when a new sample is presented.

Appendix B

Mathematical Preliminaries

B.1 N-Dimensional Euclidean Geometry

To appreciate the problems in representation and recall of information in artificial neural networks, concepts of higher dimensional geometry are useful. In this section we will describe some basic principles of N-dimensional Euclidean geometry that will help **visualise** the behaviour of the state of a neural network [Hecht-Nielsen, 1990]. Let us consider a network with N processing units. Let the output signal of each unit be one of the following four types:

- Binary: $\{0, 1\}$
- Bipolar: $\{-1, 1\}$
- Continuous in the range $[0, 1]$
- Continuous in the range $[-1, 1]$

The output signal vector of the network can be represented as $\mathbf{s} = (s_1, s_2, \dots, s_N)$.

The domains of the signal vectors for the four types of output in the N-dimensional space \mathcal{R}^N are hypercubes defined as follows:

$$\{0, 1\}^N \equiv \{\mathbf{s} = (s_1, s_2, \dots, s_N) \in \mathcal{R}^N \mid s_i \in \{0, 1\}, \quad 1 \leq i \leq N\}$$

$$\{-1, 1\}^N \equiv \{\mathbf{s} = (s_1, s_2, \dots, s_N) \in \mathcal{R}^N \mid s_i \in \{-1, 1\}, \quad 1 \leq i \leq N\}$$

$$[0, 1]^N \equiv \{\mathbf{s} = (s_1, s_2, \dots, s_N) \in \mathcal{R}^N \mid 0 \leq s_i \leq 1, \quad 1 \leq i \leq N\}$$

$$[-1, 1]^N \equiv \{\mathbf{s} = (s_1, s_2, \dots, s_N) \in \mathcal{R}^N \mid -1 \leq s_i \leq 1, \quad 1 \leq i \leq N\}$$

The points in the discrete binary cube $\{0, 1\}^N$ are located at varying distances **from** the origin in the range $(0 \text{ to } \sqrt{N})$. On the other hand, the points in the discrete bipolar cube $\{-1, 1\}^N$ are all of length \sqrt{N} . Therefore, the hypercube $\{-1, 1\}^N$ is enclosed in the sphere of radius \sqrt{N} in \mathcal{R}^N .

The volumes and areas of the hypersphere (radius r) and hypercube (side length l) in \mathcal{R}^N are given by the following expressions:

Mathematical Preliminaries

Volume of hypersphere:

$$V_s(N, r) = \begin{cases} \frac{\pi^{N/2}}{(N/2)!} r^N & \text{if } N \text{ is even} \\ \frac{2^N (\pi)^{(N-1)/2} ([N-1/2]!)}{N!} r^N & \text{if } N \text{ is odd} \end{cases}$$

Area of hypersphere:

$$A_s(N, r) = \begin{cases} \frac{(N\pi)^{N/2}}{(N/2)!} r^{N-1} & \text{if } N \text{ is even} \\ \frac{2^N (\pi)^{(N-1)/2} ([N-1/2]!)}{(N-1)!} r^{N-1} & \text{if } N \text{ is odd} \end{cases}$$

Volume of hypercube: $V_c(N, l) = l^N$

Area of hypercube: $A_c(N, l) = 2Nl^{N-1}$

Therefore we notice the following interesting properties:

Volume of a unit side length cube is a constant, since $V_c(N, 1) = 1$

Volume of a sphere inscribed in a cube is $V_s(N, 0.5) \rightarrow 0$, as $N \rightarrow \infty$

The distance from the centre to the vertices of a unit cube in \mathcal{R}^N is $\sqrt{N}/2$, which is a function of N .

The distance from the centre of the sphere inscribed in the unit cube to any point on its surface is $1/2$.

B.2 Linear Algebra

Some Definitions

Linear dependence: A set of M -dimensional vectors $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ is said to be *linearly dependent* if there exist numbers $\{c_1, c_2, \dots, c_N\}$ not all zero such that

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \dots + c_N\mathbf{x}_N = 0 \quad (\text{B.1})$$

If the vectors are not linearly dependent, they are said to be *linearly independent*.

Rank: The *rank* of a matrix $A \in \mathcal{R}^{N \times M}$ is defined as the number of linearly independent rows or columns of A . If A is full rank, then its rank is N or M , whichever is lower.

Inner product: The *inner product* of two vectors $\mathbf{x}, \mathbf{y} \in \mathcal{R}^M$, $\mathbf{x} = [x_1, x_2, \dots, x_M]^T$ and $\mathbf{y} = [y_1, y_2, \dots, y_M]^T$, is defined as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^M x_i y_i = \mathbf{x}^T \mathbf{y} \quad (\text{B.2})$$

When the inner product of the two **vectors** (x, y) is zero, the vectors are said to be *orthogonal*.

Outer product: The *outer product* of two vectors x, y is defined as

$$\mathbf{xy}^T = \begin{bmatrix} x_1y_1 & x_1y_2 & \dots & x_1y_M \\ x_2y_1 & x_2y_2 & \dots & x_2y_M \\ \vdots & \vdots & \ddots & \vdots \\ x_My_1 & x_My_2 & \dots & x_My_M \end{bmatrix} \quad (\text{B.3})$$

The rank of the outer product is one. The outer product is a **symmetric** matrix when $y = x$.

Norm: The \mathcal{L}_p *norm* of the vector x is **defined as**

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^M |x_i|^p \right)^{1/p} \quad (\text{B.4})$$

The \mathcal{L}_2 norm is called *Euclidean norm* and is given by $(\mathbf{x}^T \mathbf{x})^{1/2}$

Gradient: The *gradient* of a **multivariable** function $\phi(\mathbf{x})$ w.r.t. x is defined as

$$\nabla\phi(\mathbf{x}) = \frac{\partial\phi(\mathbf{x})}{\partial\mathbf{x}} = \left[\frac{\partial\phi(\mathbf{x})}{\partial x_1}, \frac{\partial\phi(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial\phi(\mathbf{x})}{\partial x_M} \right]^T \quad (\text{B.5})$$

If x is a time varying **vector** (denoted as $\mathbf{x}(t)$), then the derivative of $\phi(\mathbf{x}(t))$ w.r.t. time t is

$$\begin{aligned} \frac{d\phi(\mathbf{x}(t))}{dt} &= \frac{\partial\phi(\mathbf{x})}{\partial x_1} \frac{dx_1(t)}{dt} + \dots + \frac{\partial\phi(\mathbf{x})}{\partial x_M} \frac{dx_M(t)}{dt} \\ &= [\nabla\phi(\mathbf{x})]^T \frac{d\mathbf{x}(t)}{dt} \end{aligned} \quad (\text{B.6})$$

Jacobian: For a vector function of a vector given by

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}) f_2(\mathbf{x}) \dots f_L(\mathbf{x})]^T$$

the *Jacobian matrix* is defined as

$$J(\mathbf{x}) = \frac{\partial\mathbf{f}(\mathbf{x})}{\partial\mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_M} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_L}{\partial x_1} & \frac{\partial f_L}{\partial x_2} & \dots & \frac{\partial f_L}{\partial x_M} \end{bmatrix} \quad (\text{B.7})$$

Quadratic forms. A *quadratic form* is a scalar valued function of the vector \mathbf{x} , defined through a symmetric matrix $A \in \mathcal{R}^{M \times M}$. The quadratic form is given by

$$\begin{aligned} Q(\mathbf{x}) &= \mathbf{x}^T A \mathbf{x} \\ &= [x_1 \ x_2 \ \dots \ x_M] \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{12} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \dots & \vdots \\ a_{1M} & a_{2M} & \dots & a_{MM} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix} \\ &= \sum_{i=1}^M \sum_{j=1}^M a_{ij} x_i x_j \end{aligned} \quad (\text{B.8})$$

Note that since A is symmetric, $a_{ij} = a_{ji}$.

If $Q(\mathbf{x}) \geq 0$, then A is called a *positive semidefinite* matrix. If $Q(\mathbf{x}) > 0$ then A is called a *positive definite* matrix. If $Q(\mathbf{x}) \leq 0$, then A is called a *negative semidefinite* matrix. If $Q(\mathbf{x}) < 0$, then A is called a *negative definite* matrix.

The gradient of the quadratic form is given by

$$\nabla Q(\mathbf{x}) = 2 A \mathbf{x} \quad (\text{B.9})$$

Multidimensional Taylor series:

A multivariable continuous function $\phi(\mathbf{x})$, whose derivatives of all orders exist, can be expressed in Taylor series as

$$\begin{aligned} \phi(\mathbf{x} + \delta \mathbf{x}) &= \phi(\mathbf{x}) + \sum_{i=1}^M \frac{\partial \phi(\mathbf{x})}{\partial x_i} \delta x_i + \frac{1}{2!} \sum_{i=1}^M \sum_{j=1}^M \frac{\partial^2 \phi(\mathbf{x})}{\partial x_i \partial x_j} \delta x_i \delta x_j + \dots \\ &= \phi(\mathbf{x}) + [\nabla \phi(\mathbf{x})]^T \delta \mathbf{x} + \frac{1}{2!} \delta \mathbf{x}^T \nabla^2 \phi(\mathbf{x}) \delta \mathbf{x} + \dots \end{aligned} \quad (\text{B.10})$$

where $\delta \mathbf{x} = [\delta x_1, \delta x_2, \dots, \delta x_M]^T$ is an incremental vector, $\nabla \phi(\mathbf{x})$ is the gradient of $\phi(\mathbf{x})$ w.r.t. \mathbf{x} and $\nabla^2 \phi(\mathbf{x})$ is the *Hessian matrix* defined as

$$\begin{aligned} \nabla^2 \phi(\mathbf{x}) &= \nabla [\nabla \phi(\mathbf{x})] \\ &= \begin{bmatrix} \frac{\partial^2 \phi(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 \phi(\mathbf{x})}{\partial x_1 x_2} & \dots & \frac{\partial^2 \phi(\mathbf{x})}{\partial x_1 x_M} \\ \frac{\partial^2 \phi(\mathbf{x})}{\partial x_2 x_1} & \frac{\partial^2 \phi(\mathbf{x})}{\partial x_2^2} & \dots & \frac{\partial^2 \phi(\mathbf{x})}{\partial x_2 x_M} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial^2 \phi(\mathbf{x})}{\partial x_M x_1} & \frac{\partial^2 \phi(\mathbf{x})}{\partial x_M x_2} & \dots & \frac{\partial^2 \phi(\mathbf{x})}{\partial x_M^2} \end{bmatrix} \end{aligned} \quad (\text{B.11})$$

If $\phi(\mathbf{x})$ has an extremum at $\mathbf{x} = \mathbf{x}_0$, then $\nabla\phi(\mathbf{x}_0) = \mathbf{0}$. Neglecting higher order terms in the Taylor series in Eq. (B.10), we have at $\mathbf{x} = \mathbf{x}_0$

$$\phi(\mathbf{x}_0 + \delta\mathbf{x}) = \phi(\mathbf{x}_0) + \frac{1}{2} \delta\mathbf{x}^T [\nabla^2\phi(\mathbf{x}_0)] \delta\mathbf{x}$$

Since the last term in the above expression has a quadratic form, $\phi(\mathbf{x})$ exhibits a minimum at $\mathbf{x} = \mathbf{x}_0$ if the Hessian matrix $\nabla^2\phi(\mathbf{x}_0)$ is positive definite. If the Hessian is negative definite, $\phi(\mathbf{x})$ exhibits a maximum at $\mathbf{x} = \mathbf{x}_0$.

Eigenvalues and Eigenvectors:

Consider minimization of the quadratic form $\mathbf{v}^T \mathbf{A} \mathbf{v}$ subjected to the constraint $\mathbf{v}^T \mathbf{v} = 1$, where $\mathbf{A} \in \mathcal{R}^{M \times M}$. That is, taking the derivative of $(\frac{1}{2} \mathbf{v}^T \mathbf{A} \mathbf{v} - \lambda \mathbf{v}^T \mathbf{v})$ w.r.t. \mathbf{v} and setting it to zero, we get

$$\mathbf{A} \mathbf{v} = \lambda \mathbf{v} \quad (\text{B.12})$$

where λ is called the **Lagrange** multiplier. The set of all vectors $\mathbf{v} \in \mathcal{R}^M$ which satisfy equation (B.12) are called the eigenvectors of \mathbf{A} and the corresponding scalars λ are its eigenvalues.

If \mathbf{A} has M nonzero distinct eigenvalues, \mathbf{A} can be expressed as

$$\mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T = \sum_{i=1}^M \lambda_i \mathbf{v}_i \mathbf{v}_i^T \quad (\text{B.13})$$

where $\mathbf{V} = [\mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_M]$ is the matrix having the eigenvectors as its **columns** and $\mathbf{\Lambda} = \text{diag}[\lambda_1 \lambda_2 \dots \lambda_M]$ is the diagonal matrix of eigenvalues. The representation of \mathbf{A} in Eq. (B.13) is called the **eigendecomposition** of \mathbf{A} .

Singular Value Decomposition:

The Singular Value Decomposition (SVD) can be viewed as a generalization of the eigendecomposition. For a matrix $\mathbf{A} \in \mathcal{R}^{N \times M}$, there exist orthogonal matrices $\mathbf{U} = [\mathbf{u}_1 \mathbf{u}_2 \dots \mathbf{u}_N] \in \mathcal{R}^{N \times N}$ and $\mathbf{V} = [\mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_M] \in \mathcal{R}^{M \times M}$ such that

$$\mathbf{U}^T \mathbf{A} \mathbf{V} = \text{diag}[\sigma_1, \sigma_2, \dots, \sigma_r] \quad (\text{B.14})$$

where $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r \geq 0$, and $r \leq \min(N, M)$ is the rank of the matrix \mathbf{A} . The σ_i s are the singular values of \mathbf{A} and the vectors \mathbf{u}_i , \mathbf{v}_i are the *ith left* singular vector and the *ith right* singular vector, respectively.

Therefore the matrix \mathbf{A} can be written as

$$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T = \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^T \quad (\text{B.15})$$

where

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_r \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \quad (\text{B.16})$$

It can be shown that \mathbf{u}_i and \mathbf{v}_i are the eigenvectors of the square matrices $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$, respectively. That is

$$(\mathbf{A}\mathbf{A}^T)\mathbf{u}_i = \sigma_i^2\mathbf{u}_i$$

and

$$(\mathbf{A}^T\mathbf{A})\mathbf{v}_i = \sigma_i^2\mathbf{v}_i$$

σ_i^2 are the eigenvalues in both the cases.

Solution of Linear Equations:

Let \mathbf{A} is an $N \times M$ matrix, \mathbf{x} is an $M \times 1$ column vector, and \mathbf{b} is an $N \times 1$ column vector. Then for solving a set of N linear equations given by

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (\text{B.17})$$

three cases arise:

Case I: $N = M$. For this case of square matrix the solution to Eq. (B.17) is given by $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, provided \mathbf{A}^{-1} exists, i.e., \mathbf{A} is a full rank matrix.

Case II: $N > M$. The solution for this *overdetermined* case is obtained by solving the following least squares problem

$$\min_{\mathbf{x}} (\mathbf{b} - \mathbf{A}\mathbf{x})^T (\mathbf{b} - \mathbf{A}\mathbf{x}) \quad (\text{B.18})$$

The solution to Eq. (B.18) is given by

$$\mathbf{x}_{LS} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} = \mathbf{A}^+\mathbf{b} \quad (\text{B.19})$$

where \mathbf{A}^+ is called the *pseudoinverse* of \mathbf{A} . The vector

$$\hat{\mathbf{b}} = \mathbf{A}\mathbf{x}_{LS} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} = \mathbf{P}\mathbf{b} \quad (\text{B.20})$$

gives the projection of \mathbf{b} onto the space spanned by the columns of \mathbf{A} . Hence, the matrix \mathbf{P} is called the *projection matrix*.

Case III: $N < M$. There are infinite solutions for this *undetermined* case. The *Minimum Norm Least Squares (MNLS)* solution is obtained by minimising $(\mathbf{x}^T\mathbf{x})$ subjected to the constraint that $\mathbf{A}\mathbf{x} = \mathbf{b}$. That is, the solution is obtained by minimising $[\mathbf{x}^T\mathbf{x} - \Lambda^T(\mathbf{A}\mathbf{x} - \mathbf{b})]$ w.r.t. \mathbf{x} . The

vector \mathbf{A} is the set of **Lagrange** multipliers $[\lambda_1, \lambda_2, \dots, \lambda_N]^T$. The solution to the above constrained optimization problem is given by

$$\mathbf{x}_{MNL} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b} \quad (\text{B.21})$$

Projection Matrix

Consider the vectors $\mathbf{a}_1, \mathbf{a}_2, \mathbf{b} \in \mathcal{R}^M$.

The scalar projection of \mathbf{b} onto $\mathbf{a}_1 = \left(\mathbf{b}, \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|_2} \right)$

The vector projection of \mathbf{b} onto $\mathbf{a}_1 = \left(\mathbf{b}, \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|_2} \right) \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|_2}$

Let $\hat{\mathbf{b}}$ be a projection of vector \mathbf{b} onto the **subspace** spanned by \mathbf{a}_1 and \mathbf{a}_2 . The **subspace** is given by the linear combination of \mathbf{a}_1 and \mathbf{a}_2 , i.e., $x_1\mathbf{a}_1 + x_2\mathbf{a}_2 = \mathbf{A}\mathbf{x}$. Let $\hat{\mathbf{b}} = \mathbf{A}\mathbf{x}_p$. Then the vector $\mathbf{b} - \hat{\mathbf{b}} = \mathbf{b} - \mathbf{A}\mathbf{x}_p$ is orthogonal to **all** the vectors in the **subspace** $\mathbf{A}\mathbf{x}$. Therefore

$$(\mathbf{A}\mathbf{x})^T(\mathbf{b} - \mathbf{A}\mathbf{x}_p) = 0$$

i.e.,

$$\mathbf{x}^T(\mathbf{A}^T\mathbf{b} - \mathbf{A}^T\mathbf{A}\mathbf{x}_p) = 0$$

Since this should be true for **all** \mathbf{x} , we have

$$\mathbf{x}_p = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \quad (\text{B.22})$$

Note that the solution in Eq. (B.22), which has been derived using a geometrical approach, is identical to the least squares solution given in Eq. (B.19). **Thus** the projection of \mathbf{b} onto the space spanned by the columns of \mathbf{A} can be written as

$$\hat{\mathbf{b}} = \mathbf{A}\mathbf{x}_p = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} = \mathbf{P}\mathbf{b}$$

where \mathbf{P} is the *projection matrix*.

B.3 Probability

Sample Set

A set U which consists of **all** possible outcomes of a random experiment is called *sample space*. The sample space corresponds to the universal set.

Event

An event $A \subseteq U$ is a set of possible outcomes. If the events A and B do not have any element in common, then they are called *mutually exclusive* events.

Random Variable

A random variable X is a function that maps the outcome of a random event, *i.e.*, each point of a sample space into real scalar values. A random variable which takes discrete values is called a discrete random variable, and the one which assumes continuous values is called a continuous random variable. A vector random variable X is a **vector** whose components X_i are random variables.

Definition of Probability

Classical or a priori approach: If an event A can occur in h different ways out of a total number of n possible ways, all of which are equally likely, then the probability $P(A)$ of the event A is defined as $\frac{h}{n}$.

Frequency or a posteriori approach: If **after** n repetitions of an experiment, where n is very large, an event A is observed to *occur* in h of these, then the probability $P(A)$ of the event A is defined as $\frac{h}{n}$.

Axioms of Probability

$P(A)$ is defined as the probability of an event A if it satisfies the following three axioms:

$$A1: 0 \leq P(A) \leq 1$$

$$A2: P(U) = 1$$

A3: For any sequence of mutually exclusive events A_1, A_2, \dots

$$P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$$

Important Properties of Probability

1. If ϕ denotes an empty set, then

$$P(\phi) = 0, \quad (\text{B.23})$$

2. If \bar{A} is the complement of A , then

$$P(\bar{A}) = 1 - P(A) \quad (\text{B.24})$$

3. If A and B are any two events, then

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \quad (\text{B.25})$$

Conditional Probability

Let $A, B \subseteq X$ be two events such that $P(A) > 0$. Then the probability of B given that A has **occurred**, i.e., **conditional probability** of B given A , is defined as

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \quad (\text{B.26})$$

Independent Events

The event $A \subseteq X$ is called *independent* of the event $B \subseteq X$ if and only if $P(A|B) = P(A)$.

Bayes Theorem

Let A_1, A_2, \dots, A_n are mutually exclusive events such that $\bigcup_{k=1}^n A_k = U$. If one of the events must occur, then

$$P(A_k|A) = \frac{P(A_k) P(A|A_k)}{\sum_{k=1}^n P(A_k) P(A|A_k)} \quad \forall k = 1, 2, \dots, n \quad (\text{B.27})$$

Probability Distribution

For the discrete case, the function $p(x) = P(X = x)$ is a probability distribution of the random variable X if $p(x) \geq 0$ and $\sum p(x) = 1$. For a continuous case $p(x)$ is called a probability **density** function, if $p(x) \geq 0$ and $\int p(x) dx = 1$.

Expectation or Mean

For a discrete random variable X with the possible values x_1, x_2, \dots, x_n , the *expectation* or *mean* is defined as

$$\mathcal{E}(X) = \sum_{i=1}^n x_i P(X = x_i) \quad (\text{B.28})$$

provided the series converges absolutely.

For a continuous case

$$\mathcal{E}(X) = \int x p(x) dx \quad (\text{B.29})$$

Variance

The variance of a random variable X is

$$\text{Var}(X) = \mathcal{E} [(X - \mathcal{E}(X))^2] \quad (\text{B.30})$$

The positive square root of the variance is called **standard deviation** and is given by

$$\sigma = \sqrt{\text{Var}(X)} = \sqrt{\mathcal{E} [(X - \mathcal{E}(X))^2]} \quad (\text{B.31})$$

Uniform Distribution

A **uniform probability distribution** between a and b for a random variable X is defined as

$$u(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \quad (\text{B.32})$$

The mean and variance of the uniform probability distribution are

$$\mathcal{E}(X) = \frac{1}{2}(a+b) \quad \text{and} \quad \text{Var}(X) = \frac{1}{12}(b-a)^2 \quad (\text{B.33})$$

Binomial Distribution

If each experiment is identical, i.e., each experiment has the same sample space and same probability distribution on its events, then the experiments are called **trials**. Repeated independent trials are called **Bernoulli trials** if there are only two possible outcomes for each trial and their probabilities remain the same throughout the trials. Let q be the probability that an event will occur in a single Bernoulli trial. Then $(1 - q)$ is the probability that the event will fail to occur in any single trial. The probability that the event will happen exactly x times in $n \geq 0$ trials is given by the **binomial distribution** of order n

$$b(x; n, q) = P(X=x) = \binom{n}{x} q^x (1-q)^{n-x} \quad \text{for } x=0, 1, \dots, n \quad (\text{B.34})$$

where the random variable X denotes the number of successes in n trials.

The mean and variance of the binomial distribution are

$$\mathcal{E}(X) = nq \quad \text{Var}(X) = nq(1-q) \quad (\text{B.35})$$

Univariate Gaussian Distribution

The function defined by

$$N(x, \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \quad (\text{B.36})$$

is known as the Gaussian distribution of the random **variable** X with mean μ and variance σ^2 .

Multivariate Gaussian Distribution

An explicit expression of a multivariate Gaussian distribution for a vector random variable \mathbf{X} is

$$N(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^M |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \quad (\text{B.37})$$

where $N(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a notation for a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_M]^T$ and covariance matrix $\boldsymbol{\Sigma}$. The covariance matrix $\boldsymbol{\Sigma}$ is defined as

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \dots & \sigma_{1M} \\ \sigma_{22} & \sigma_{21} & \dots & \sigma_{2M} \\ \vdots & \vdots & \dots & \vdots \\ \sigma_{M1} & \sigma_{M2} & \dots & \sigma_{MM} \end{bmatrix} \quad (\text{B.38})$$

where

$$\sigma_{ij} = \mathcal{E}[(X_i - \mathcal{E}(X_i))(X_j - \mathcal{E}(X_j))], \quad 1 \leq i, j \leq M \quad (\text{B.39})$$

Gaussian Mixture Distribution

In a Gaussian mixture distribution we have a mixture of finite number (say I) of multivariate Gaussian distributions in some proportion with mixture weights λ_i , where

$$\sum_{i=1}^I \lambda_i = 1 \quad \text{and} \quad \lambda_i \geq 0, \quad \text{for all } i \quad (\text{B.40})$$

The equation of the Gaussian mixture distribution with I multivariate Gaussian components is *known* as

$$f(\mathbf{x}) = \sum_{i=1}^I \lambda_i N_i(\mathbf{x}, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \quad (\text{B.41})$$

where $N_i(\mathbf{x}, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is the probability distribution of the i th component with mean vector $\boldsymbol{\mu}_i$ and covariance matrix $\boldsymbol{\Sigma}_i$.

B.4 Similarity Measures

The notion of similarity measure appears in artificial neural networks in different contexts. For example, the performance of a learning algorithm *depends on* the similarity of the patterns in the input data. In this section we will describe some useful measures of pattern similarity.

A broad class of measures of similarity between two **patterns** is based on metric distance measures. A distance measure d between two M -dimensional pattern **vectors** \mathbf{x} and \mathbf{y} is called a metric if it satisfies

- (a) $d(\mathbf{x}, \mathbf{y}) \geq 0$ and $d(\mathbf{x}, \mathbf{y}) = 0$ iff $\mathbf{x} = \mathbf{y}$ (positivity)
- (b) $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ (symmetry)
- (c) $d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{z}, \mathbf{x})$ (triangle inequality)

The most common example of this **kind** of distance measure is Minkowski *r-metric*, which is given as follows:

$$d_r(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^M |x_i - y_i|^r \right)^{1/r} \quad (\text{B.42})$$

The particular cases of the above distance measure are:

- (a) When $r = 1$ and $x_i, y_i \in \{0, 1\}$, d_r refers to Hamming distance.
- (b) When $r = 2$, d_r refers to Euclidean distance.
- (c) When $r = \infty$, d_r refers to *Chebyshev* distance.

Two other **metrics** that are useful in the ANN context are:

- (d) Absolute value distance or city block distance or \mathcal{L}_1 *norm*

$$d_a(\mathbf{x}, \mathbf{y}) = |x_1 - y_1| + |x_2 - y_2| + \dots + |x_M - y_M| \quad (\text{B.43})$$

- (e) Maximum value distance or \mathcal{L}_∞ *norm*

$$d_m(\mathbf{x}, \mathbf{y}) = \max \{ |x_1 - y_1|, |x_2 - y_2|, \dots, |x_M - y_M| \} \quad (\text{B.44})$$

The distance between a pattern vector (\mathbf{x}) and its mean vector ($\boldsymbol{\mu}$) of a Gaussian distribution is described in terms of the following *Mahalanobis* distance.

$$d_M = (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \quad (\text{B.45})$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the distribution. This distance is also used in Eq. (B.37) of the multivariate Gaussian distribution. When $\boldsymbol{\Sigma}$ is a diagonal matrix, d_M becomes the weighted *inner* product.

A similarity measure between two pattern **vectors** need not be a distance measure. For example, the cosine of the angle θ subtended between the vectors \mathbf{x} and \mathbf{y} can serve as a similarity measure. That is

$$S_\theta(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \quad (\text{B.46})$$

The inner **product** $\mathbf{x}^T \mathbf{y}$ alone can serve as a similarity measure. This is also called *cross* correlation.

In some cases, we need to have a similarity measure between two probability distributions $\{p_j\}$ and $\{q_j\}$ for discrete random variables. *Cross entropy* or *Kullback-Leibler measure* is one such measure. The cross entropy is defined as

$$S_e = \sum_j \left(p_j \log \left(\frac{p_j}{q_j} \right) + (1-p_j) \log \left(\frac{1-p_j}{1-q_j} \right) \right) \quad (\text{B.47})$$

The first term of S_e , i.e. $\sum_j p_j \log \left(\frac{p_j}{q_j} \right)$, is also used as a similarity measure. It is known as *cross entropy of the distribution $\{p_j\}$ with respect to the distribution $\{q_j\}$* .

Appendix C

Basics of Gradient Descent Methods

C.1 Mean Squared Error

Gradient descent methods form the basis for many supervised learning laws in artificial neural networks. We introduce the basics of the gradient descent methods [Widrow and Stearns, 1985] by considering a single layer, single unit network with linear output function, namely the Adaline, as shown in Figure C.1. The error in the actual

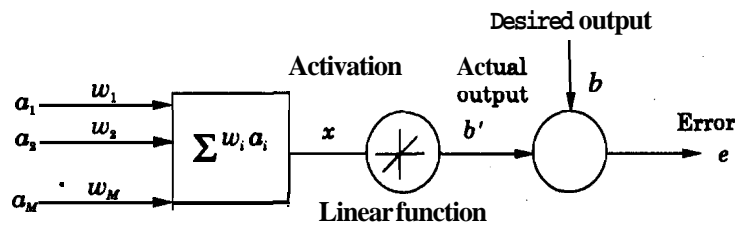


Figure C.1 A single layer single linear unit network (Adaline).

output for a given input vector $\mathbf{a} = (a_1, a_2, \dots, a_M)^T$ is given by

$$\begin{aligned} e &= b - \sum_{i=1}^M w_i a_i \\ &= b - \mathbf{w}^T \mathbf{a} = b - \mathbf{a}^T \mathbf{w} \end{aligned} \quad (\text{C.1})$$

The squared error is given by

$$\begin{aligned} e^2 &= (b - \mathbf{w}^T \mathbf{a})(b - \mathbf{a}^T \mathbf{w}) \\ &= b^2 + \mathbf{w}^T \mathbf{a} \mathbf{a}^T \mathbf{w} - 2b \mathbf{a}^T \mathbf{w} \end{aligned} \quad (\text{C.2})$$

The input vector can be considered as a sample function of a stationary random process [Papoulis, 1990]. Then the mean squared error is given by the expected value of e^2 as follows:

$$E(\mathbf{w}) = \mathbf{E}[e^2] = b^2 + \mathbf{w}^T \mathbf{R} \mathbf{w} - 2\mathbf{p}^T \mathbf{w} = b^2 + \mathbf{w}^T \mathbf{R} \mathbf{w} - 2\mathbf{w}^T \mathbf{p} \quad (\text{C.3})$$

where $R = \mathbf{E}[\mathbf{a}\mathbf{a}^T]$ is an $M \times M$ autocorrelation matrix with $\mathbf{E}[a_i a_j]$ as its (i, j) th element, and $\mathbf{p}^T = \mathbf{E}[b\mathbf{a}^T] = \mathbf{E}[ba_1, ba_2, \dots, ba_M]^T$. To find the weights for minimum E, we take the gradient of the mean squared error $E(\mathbf{w})$. That is

$$\begin{aligned}\nabla &= \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}^T} \\ &= \frac{\partial(\mathbf{w}^T R \mathbf{w})}{\partial \mathbf{w}^T} - \frac{\partial(2\mathbf{w}^T \mathbf{p})}{\partial \mathbf{w}^T} \\ &= 2R\mathbf{w} - 2\mathbf{p}\end{aligned}\quad (\text{C.4})$$

Note that even though E is a scalar, $\partial E / \partial \mathbf{w}^T$ is a vector with components of gradient along each w_i axis in the weight space. Setting the gradient equal to zero, and solving the resulting normal equations for the optimal set of weights \mathbf{w}^* , we get

$$\nabla = 0 = 2R\mathbf{w}^* - 2\mathbf{p}\quad (\text{C.5})$$

Therefore

$$\mathbf{w}^* = R^{-1}\mathbf{p}\quad (\text{C.6})$$

provided that R^{-1} exists. The minimum error is obtained by substituting \mathbf{w}^* for \mathbf{w} in the Eq. (C.3) for E. Therefore, we get after simplification

$$E_{\min} = b^2 - \mathbf{p}^T \mathbf{w}^* = b^2 - \mathbf{w}^{*T} \mathbf{p}\quad (\text{C.7})$$

We can show that the mean squared error in Equation (C.3) is given by

$$\begin{aligned}E(\mathbf{w}) &= E_{\min} + (\mathbf{w} - \mathbf{w}^*)^T R (\mathbf{w} - \mathbf{w}^*) \\ &= E_{\min} + \mathbf{v}^T R \mathbf{v}\end{aligned}\quad (\text{C.8})$$

where

$$\mathbf{v} = \mathbf{w} - \mathbf{w}^* = [v_1, v_2, \dots, v_M]^T\quad (\text{C.9})$$

is a translated weight vector with origin at $\mathbf{w} = \mathbf{w}^*$. This can be proved by using the property that R is a symmetric matrix, i.e., $R^T = R$.

C.2 Properties of the Autocorrelation Matrix R

The following properties of the autocorrelation matrix are useful for studying the properties of the error function $E(\mathbf{w})$ [Widrow and Stearns, 1985].

(a) $R^T = R$, $RR^{-1} = I$ and $(R^{-1})^T = R^{-1}$.

(b) Using the eigenvalues and eigenfunctions of R, we can get the normal form representation of R as $R = Q\Lambda Q^{-1}$, where Λ is a diagonal matrix consisting of the eigenvalues λ_i of R, and Q is a matrix consisting of the eigenvectors of R. That is $Q = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_M]$. We also have $Q^{-1}RQ = \Lambda$.

- (c) The eigenvectors corresponding to the distinct eigenvalues are orthogonal to each other. That is $\mathbf{q}_i^T \mathbf{q}_j = 0$, for all i and j , and $i \neq j$.
- (d) Since R is a real and symmetric matrix, all its eigenvalues are real, and each eigenvalue is greater than or equal to zero.
- (e) If all the eigenvectors are normalized to unit magnitude, then the resulting set of **eigenvectors** are orthonormal. That is $Q Q^T = I = Q Q^{-1}$. Hence $Q^T = Q^{-1}$.

C.3 Properties of the Mean Squared Error $E(\mathbf{w})$

$E(\mathbf{w})$ is a quadratic function of the components of the **vector** \mathbf{w} . That is, in $E(\mathbf{w})$, when expanded, the elements of \mathbf{w} will appear in first and second degrees only. Thus the error surface $E(\mathbf{w})$ is a hyperboloid in the weight space. Since $E(\mathbf{w})$ is a squared error, E_{\min} is the minimum value of the squared error. Therefore we have

$$E(\mathbf{w}) \geq E_{\min} \geq 0$$

and

$$\mathbf{v}^T R \mathbf{v} = E(\mathbf{w}) - E_{\min} \geq 0$$

Note that $\mathbf{v}^T R \mathbf{v}$ is also quadratic **function** of the components of the vector \mathbf{v} . The **error** surface $E(\mathbf{w})$ is a bowl-shaped surface. **Figure C.2**

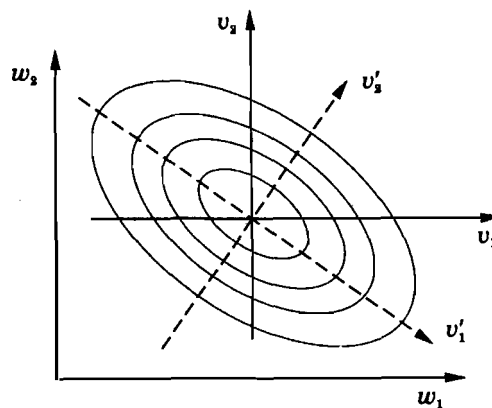


Figure C.2 Constant error contours for the quadratic error surface.

shows constant error contours of an error surface for a two dimensional case. The term $\mathbf{v}^T R \mathbf{v}$ represents a hyperellipse, which in the two dimensional weight space is an ellipse with two principal axes as shown in **Figure C.2**, for different values of the constant in the equation

$$\mathbf{v}^T R \mathbf{v} = \text{constant} \quad (\text{C.10})$$

Any increase in the radius from the $v = 0$ point increases the error. **Therefore**, the gradient of $E(\mathbf{w})$ with respect to v is always positive.

Expressing R as $Q\Lambda Q^T$ in Eq. (C.8), and using $\mathbf{v}' = Q^T\mathbf{v}$, we get

$$E(\mathbf{w}) = E_{\min} + (\mathbf{v}')^T \Lambda \mathbf{v}' \quad (\text{C.11})$$

With the transformed coordinates \mathbf{v}' , the axes are along the two principal axes of the ellipses in the weight space as shown in Figure C.2. It can be shown that the eigenvectors of the matrix R define these principal axes of the hyperellipses formed by the error surface. Since

$$\frac{dE}{d\mathbf{v}'} = 2\lambda\mathbf{v}' = 2 \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} v'_1 \\ v'_2 \end{pmatrix} \quad (\text{C.12})$$

the eigenvalues of R are given by the second derivative of the error surface with respect to the principal axes, i.e.,

$$\frac{\partial^2 E}{\partial v_i'^2} = 2\lambda_i, \quad i = 1, 2 \quad (\text{C.13})$$

We want to find \mathbf{w} for which $E(\mathbf{w})$ is minimum. For a quadratic error surface the minimum of $E(\mathbf{w})$ occurs at a point \mathbf{w}^* in the weight space at which the gradient of the error surface is zero. Therefore the gradient of

$$E(\mathbf{w}) = E_{\min} + (\mathbf{w} - \mathbf{w}^*)^T R (\mathbf{w} - \mathbf{w}^*) \quad (\text{C.14})$$

is given by

$$\nabla = \frac{dE(\mathbf{w})}{d\mathbf{w}} = 2R(\mathbf{w} - \mathbf{w}^*) \quad (\text{C.15})$$

Multiplying both sides with R^{-1} and rearranging, we get

$$\mathbf{w}^* = \mathbf{w} - \frac{1}{2} R^{-1} \nabla \quad (\text{C.16})$$

Note that $\frac{d^2 E(\mathbf{w})}{d\mathbf{w}^2} = 2R$. Thus $\frac{1}{2} R^{-1} \nabla$ can be interpreted as the ratio of the first derivative to the second derivative of the error with respect to the weight in **1-D** case. From Equation (C.16), we note that, starting **from** any initial value of the weight vector \mathbf{w} , the optimum weight vector \mathbf{w}^* can be obtained in one step, provided the first and the second derivatives of the error surface are known at that initial point in the weight space.

C.4 Newton's Gradient Search Method

The optimum weight value can also be captured in an iterative manner by writing

$$\mathbf{w}(m+1) = \mathbf{w}(m) - \eta R^{-1} \nabla, \quad (\text{C.17})$$

where η is a positive constant. This is called *Newton's gradient search method*. This is useful only when the approximate values of the first and second derivatives of the error surface are available at each point.

We can show that the Newton's method converges to the optimal weight \mathbf{w}^* . Let us rewrite Eq. (C.16) as

$$\mathbf{w}^* = \mathbf{w}(m) - \frac{1}{2} R^{-1} \nabla_m \quad (\text{C.18})$$

where $\nabla_m = V$ at $\mathbf{w} = \mathbf{w}(m)$. Therefore from Eqs. (C.17) and (C.18) we get

$$\mathbf{w}(m+1) = \mathbf{w}(m) - 2\eta(\mathbf{w}(m) - \mathbf{w}^*) = \mathbf{w}(m)(1 - 2\eta) + 2\eta\mathbf{w}^* \quad (\text{C.19})$$

Starting with an initial weight of $\mathbf{w}(0)$, we get

$$\begin{aligned} \mathbf{w}(1) &= \mathbf{w}(0)(1 - 2\eta) + 2\eta\mathbf{w}^* \\ &= \mathbf{w}^* + (1 - 2\eta)(\mathbf{w}(0) - \mathbf{w}^*) \\ \mathbf{w}(2) &= \mathbf{w}(1)(1 - 2\eta) + 2\eta\mathbf{w}^* \\ &= \mathbf{w}(0)(1 - 2\eta)^2 + 2\eta\mathbf{w}^*(1 - 2\eta) + 2\eta\mathbf{w}^* \\ &= \mathbf{w}^* + (1 - 2\eta)^2(\mathbf{w}(0) - \mathbf{w}^*) \\ \mathbf{w}(m) &= \mathbf{w}^* + (1 - 2\eta)^m(\mathbf{w}(0) - \mathbf{w}^*) \end{aligned} \quad (\text{C.20})$$

Since $\mathbf{w}(0) - \mathbf{w}^*$ is fixed, $\mathbf{w}(m)$ converges to \mathbf{w}^* , provided $0 < 2\eta \leq 1$, i.e., $0 < \eta \leq 1/2$. The one step solution is obtained for $\eta = 1/2$ as shown in Eq. (C.16).

For a known quadratic error surface $E(\mathbf{w})$, the first and second derivatives are known exactly for all values of \mathbf{w} . Hence the optimum weight vector can be obtained in one step as in Eq. (C.16). But if the error surface $E(\mathbf{w})$, though quadratic, is not known exactly, then the computation needs to be iterative as in Eq. (C.17). If the error surface is not quadratic, then the Newton's method is not guaranteed to converge to the final value, starting from any arbitrary weight value. In the Newton's method the steps do not proceed along the direction of the gradient.

C.5 Method of Steepest Descent

If the weights are adjusted in the direction of the negative gradient at each step, as shown in Figure C.3, then the method is called *steepest descent*. For the method of steepest descent, the weight update is given by

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \mu(-\nabla_m) \quad (\text{C.21})$$

where μ regulates the step size, and V , is the gradient of the error surface at $\mathbf{w} = \mathbf{w}(m)$. Substituting for ∇_m from Eq. (C.18) we get

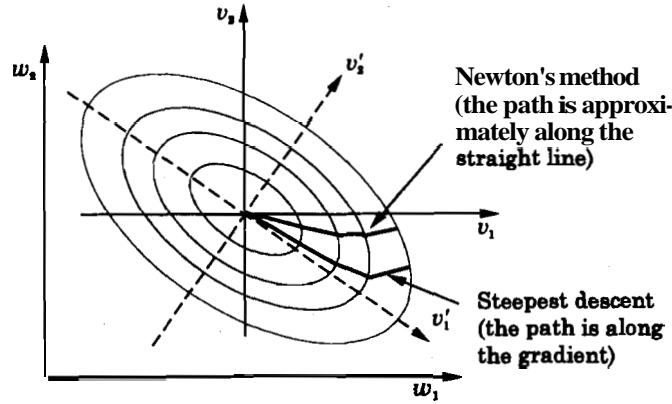


Figure C.3 Illustration of trajectories in the weight space for different gradient descent methods.

$$\begin{aligned} \mathbf{w}(m+1) &= \mathbf{w}(m) - 2\mu R(\mathbf{w}(m) - \mathbf{w}^*) \\ &= (I - 2\mu R)\mathbf{w}(m) + 2\mu R\mathbf{w}^* \end{aligned} \quad (\text{C.22})$$

In terms of $\mathbf{v} = \mathbf{w} - \mathbf{w}^*$ (translation), we get

$$\mathbf{v}(m+1) = (I - 2\mu R)\mathbf{v}(m) \quad (\text{C.23})$$

If we rotate the principal axes by substituting $\mathbf{v} = Q\mathbf{v}'$ in Eq. (C.23), we get

$$Q\mathbf{v}'(m+1) = (I - 2\mu R)Q\mathbf{v}'(m) \quad (\text{C.24})$$

Multiplying both sides by Q^{-1} , we get

$$\begin{aligned} \mathbf{v}'(m+1) &= Q^{-1}[I - 2\mu R]Q\mathbf{v}'(m) \\ &= [Q^{-1}IQ - 2\mu Q^{-1}RQ]\mathbf{v}'(m) \\ &= (I - 2\mu\Lambda)\mathbf{v}'(m) \end{aligned} \quad (\text{C.25})$$

Therefore, starting with $m = 0$, we get

$$\mathbf{v}'(m) = (I - 2\mu\Lambda)^m \mathbf{v}'(0) \quad (\text{C.26})$$

This result will be stable and convergent if

$$\lim_{m \rightarrow \infty} (I - 2\mu\Lambda)^m = 0$$

The convergence condition is satisfied by choosing

$$0 < \mu < \frac{1}{2\lambda_{\max}} \quad (\text{C.27})$$

where λ_{\max} is the largest eigenvalue of R . After convergence

$$\lim_{m \rightarrow \infty} \mathbf{v}'(m) = 0 \quad (\text{C.28})$$

Hence

$$\lim_{m \rightarrow \infty} \mathbf{w}(m) = \mathbf{w}' \quad (\text{C.29})$$

Figure C.3 shows the **trajectory** of the path for the method of steepest descent. The Newton's method converges faster because it uses the information in the R matrix to find a path close to the direct path on the error surface towards \mathbf{E}_{\min} . Note that only the first derivative of $\mathbf{E}(\mathbf{w})$ is required for the steepest descent, whereas both the first and the second derivatives of $\mathbf{E}(\mathbf{w})$ are needed for the Newton's method.

C.6 The LMS Algorithm

Since the gradient of the error surface is not available in general, it needs to be estimated from the available data. If we assume that the input vector $\mathbf{a}(m)$ and the desired output $b(m)$ are the realization of a random process at the m th instant, then the error $e(m) = b(m) - \mathbf{a}^T(m)\mathbf{w}(m)$ is also a random variable.

One method of estimating the gradient is to use an estimate of the gradient of the error by taking differences between short-term averages of $e^2(m)$. But in the LMS algorithm we use each realization $e^2(m)$ itself instead of $\mathcal{E}[e^2(m)]$ as in Eq. (C.3). The estimate of the gradient at each realization is given by

$$\begin{aligned} \hat{\mathbf{v}}_m &= \left[\frac{\partial e^2(m)}{\partial w_1}, \frac{\partial e^2(m)}{\partial w_2}, \dots, \frac{\partial e^2(m)}{\partial w_N} \right]^T \\ &= 2e(m) \left[\frac{\partial e(m)}{\partial w_1}, \frac{\partial e(m)}{\partial w_2}, \dots, \frac{\partial e(m)}{\partial w_N} \right]^T \\ &= -2e(m)\mathbf{a}(m) \end{aligned} \quad (\text{C.30})$$

Using this estimate, the steepest descent algorithm is given by

$$\begin{aligned} \mathbf{w}(m+1) &= \mathbf{w}(m) - \mu \hat{\mathbf{v}}_m \\ &= \mathbf{w}(m) + 2\mu e(m)\mathbf{a}(m). \end{aligned} \quad (\text{C.31})$$

This is called the **LMS** algorithm. Each component of the gradient vector is obtained from a single data sample. Without averaging, the gradient components do contain a large component of noise, but noise is attenuated with time by the weight adaptation process, which acts as a low-pass filter in this respect.

By taking expectation on both sides of Eq. (C.31), it can be shown that the weight **vector** converges in the mean provided $0 < \mu < \frac{1}{2\lambda_{\max}}$,

where λ_{\max} is the largest eigenvalue of \mathbf{R} [Widrow and Stearns, 1985]. But we know that $\lambda_{\max} \leq \text{tr}[\mathbf{A}] = \text{tr}[\mathbf{R}]$. If the input is considered as a signal vector, then $\text{tr}[\mathbf{R}]$ gives sum of the diagonal elements of \mathbf{R} . Each element is of the type $\mathcal{E}[\alpha_k^2]$, which can be viewed as signal power. Therefore, $\text{tr}[\mathbf{R}]$ is equal to the signal power. Hence

$$0 < \mu < \frac{1}{2 \text{ (signal power)}} \quad \mathbf{I} \quad \frac{1}{2\lambda_{\max}} \quad (\text{C.32})$$

This gives an idea for the choice of μ based on the input data. The LMS algorithm is a stochastic gradient descent algorithm. Although the algorithm converges in the mean, the trajectory in the weight space is random.

Table 4.5 in Chapter 4 gives a summary of the gradient search methods discussed in this Appendix.

Appendix D

Generalization in Neural Networks: An overview

In this Appendix we present an overview of the issues of generalization in neural networks. The material in this section is collected from [Neeharika, 1996].

D.1 Concept of Generalization

Generalization is an intuitive concept unique to human learning. For example, we learn the concept of addition of numbers by looking at several examples of addition along with some explanation provided by the teacher. Likewise, we learn the pattern embedded in the written character by observing and by writing several examples of the same character. Thus learning from examples with additional knowledge forms the basis of the concept of generalization.

Generalization by learning **from** examples is possible because of some inherent features **in** the input patterns or because of some **constraints** inherent in the mapping function. Learning, and hence **generalization**, is not possible if we are presented with a set of random data as examples. Therefore **all** problem situations are not generalizable.

After learning we are capable of dealing with new situations such as a new addition problem or a new sample of a character. Our ability to deal with new situations can be evaluated by testing ourselves with several new examples for which we know the answers for comparison. If our performance with this so called test data is better, then we can say that our ability to generalize is also better. Performance of a pattern recognition system depends on its ability to generalize from the training examples. Generalization concept is involved in all pattern recognition tasks, such **as** classification, mapping, storage and clustering. For example, in pattern mapping, it is the smoothness of the mapping function that makes generalization by a network possible. Likewise, in pattern clustering, it is the common feature in each cluster that enables the network to generalize the concept in a given cluster.

Some *Measures* of Generalization

D.2 Some Measures of Generalization

Analytical studies on generalization use models for the learning machine [Blumer et al, 1989; Haussler, 1992; Amari, 1995; Seung et al, 1992]. Various methods of measuring generalization are used in practice [Liu, 1995; Musavi et al, 1994]. We discuss some of the methods in this section.

D.2.1 Kullback-Leibler Measure

The Kullback-Leibler measure (E_{KL}) is given by the following equation:

$$E_{KL} = - \int p(\mathbf{x}, \mathbf{y}) \log \left[\frac{f_{\mathbf{w}}(\mathbf{y} | \mathbf{x}) p(\mathbf{x})}{p(\mathbf{y} | \mathbf{x}) p(\mathbf{x})} \right] d\mathbf{x} d\mathbf{y} \quad (\text{D.1})$$

where $p(\mathbf{y} | \mathbf{x})$ is the class conditional probability distribution of the sample space and $f_{\mathbf{w}}(\mathbf{y} | \mathbf{x})$ is the function approximated by the neural network **after** training using the training set $T_k = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_k, \mathbf{y}_k)\}$ consisting of k examples. The integral in Eq. (D.1) is over the input-output space. The Eq. (D.1) can be written as

$$E_{KL} = - \mathcal{E}[\log (f_{\mathbf{w}}(\mathbf{y} | \mathbf{x}))] + \mathcal{E}[\log (p(\mathbf{y} | \mathbf{x}))] \quad (\text{D.2})$$

where \mathcal{E} is the expectation operator with respect to the random variables (x, y) .

The value of E_{KL} is equal to zero when the function approximated by the neural network is equal to the actual function, **i.e.**, $f_{\mathbf{w}}(\mathbf{y} | \mathbf{x}) = p(\mathbf{y} | \mathbf{x})$. Since the second term in Eq. (D.2) is independent of the weights, the first term can be used to define the generalization error. That is

$$E_r = - \mathcal{E}[\log (f_{\mathbf{w}}(\mathbf{y} | \mathbf{x}))] \quad (\text{D.3})$$

The **Kullback-Leibler** measure is useful for the networks designed for classification purpose. The measure **requires** the knowledge of the underlying probability distribution $p(\mathbf{x}, \mathbf{y})$, which is not known in many cases. Therefore an alternative method of measuring generalization ability is needed. One such measure is the cross-validation measure.

D.2.2 Cross-Validation and Error Rate Measures

Cross-validation is a method of estimating the generalization error by making use of the training and test data [Liu, 1995]. In this method, the generalization **error** E_r in Eq. (D.3) can be estimated using

$$\hat{E}_r = -\frac{1}{k} \sum_{j \neq i} \log (f_{\mathbf{w}_{-i}}(\mathbf{y}_j | \mathbf{x}_j)) \quad (\text{D.4})$$

In the above equation $(\mathbf{x}_j, \mathbf{y}_j) \in T_k$ and \mathbf{w}_{-i} is the weight vector

obtained by using the training set T_k with its i th sample deleted. The method of cross-validation to estimate the generalization error involves training the network several times, each time by deleting a different example from the training set. This is a computationally expensive procedure.

The most commonly used measure of generalization for pattern classification task is the percentage misclassification of the test samples or the **error** rate. This measure is extensively used because it is simple and easy to implement. It can be viewed as a variation of the cross-validation measure.

D.2.3 Other Measures of Generalization

Generalization error can also be measured by the probability that the output for the $(k+1)$ th sample is **misclassified** after the network is trained on k examples of the training set [Anthony and Holden, 1994; Holden and Rayner, 1995]. It is given by

$$e_g(\mathbf{w}, k) = P[f_{\mathbf{w}}(\mathbf{x}_{k+1}) \neq \mathbf{y}_{k+1}] \quad (\text{D.5})$$

where $f_{\mathbf{w}}(\cdot)$ is the output of the neural network with weights \mathbf{w} .

Another measure of generalization is based on the **entropic** error [Amari, 1993]. It is defined as the negative logarithm of the **probability** of correct classification of the $(k+1)$ th pattern. The **entropic** error is given by

$$e_g^*(\mathbf{w}, k) = -\log(1 - e_g(\mathbf{w}, k)) \quad (\text{D.6})$$

It is clear that when the probability of correct classification is one, the value of the **entropic** error is zero.

D.3 Theoretical Studies on Generalization

D.3.1 Learning Models

Theoretical studies on generalization make use of a model of learning. The key idea is to compute the probability that the neural network gives the correct output for new samples after learning from a training set.

Let $\mathbf{z} = (x, y)$ be a sample from the input-output space, so that $p(\mathbf{z}) = p(\mathbf{x}, y)$. The goal in learning is to minimize the risk functional

$$R(\mathbf{w}) = \int Q(\mathbf{z}, \mathbf{w}) dp(\mathbf{z}) \quad (\text{D.7})$$

where $Q(\mathbf{z}, \mathbf{w})$ represents a measure of the loss or discrepancy between the desired response y and the actual response produced by the learning machine (defined by the weights \mathbf{w}) for the input x . If the probability measure $p(\mathbf{z})$ is unknown, the minimization can be carried out on the training set drawn from the input-output space.

All learning **problems** are particular cases of this general problem of minimizing the risk functional based on empirical data. Learning theory addresses the issues of consistency, convergence, generalization and learning algorithm [Vapnik, 1995].

D.3.2 VC Dimension

We consider the issue of generalization of a learning process in some detail [Holden, 1994]. Consider a network which **has** been trained using a set of training examples for a particular problem. If there is a 'high enough' probability that the **actual** error **from** the network for future samples drawn from the same problem is 'small enough', then we say that the network generalizes.

This idea of the concept of generalization is used in the Probably Approximately Correct (PAC) learning theory [Haussler, 1992], which is based on the learning model introduced by Valiant [Valiant, 1984]. We **define** some terms that are essential to understand the theoretical results obtained in the PAC theory in the context of neural networks. In the **following** definitions, \mathcal{F} **denotes** the class of functions that can be implemented by a neural network, f_w represents one of the members of this class for a particular value of weight vector w and S is the input space.

Definition 1 (Dichotomy): Given a finite set $S \subseteq \mathcal{R}^N$ and some function $f_w \in \mathcal{F}$, we define the dichotomy (S^+, S^-) of S , where S^+ and S^- are disjoint subsets of S . Here $S^+ \cup S^- = S$ and $x \in S^+$ if $f_w(x) = 1$, whereas $x \in S^-$ if $f_w(x) = 0$.

Definition 2: The hypothesis h_w associated with the function f_w is the subset of \mathcal{R}^N for which $f_w(x) = 1$, that is,

$$h_w = \{x \in \mathcal{R}^N \mid f_w(x) = 1\} \quad (D.8)$$

The hypothesis space H computed by the neural network is the set given by

$$H = \{h_w \mid w \in \mathcal{R}^{|w|}\} \quad (D.9)$$

where $|w|$ is the total number of weights in the network.

Definition 3: Given a hypothesis space H and a finite set $S \subseteq \mathcal{R}^N$, we define $\Delta_H(S)$ as the set

$$\Delta_H(S) = \{h_w \cap S \mid h_w \in H\} \quad (D.10)$$

We say that S is *shattered* by H , if $\Delta_H(S) = 2^{|S|}$ where $|S|$ is the number of elements of the set S .

Definition 4: Growth function. The growth function, $\Delta_H(i)$, is defined on the set of positive integers as,

$$\Delta_H(i) = \max_{S \subseteq \mathcal{X}^i, |S|=i} (|\Delta_H(S)|) \quad (\text{D.11})$$

The growth function gives the maximum number of distinct dichotomies induced by H for any set of i points.

Definition 5 (Vapnik-Chervonenkis dimension): The *Vapnik-Chervonenkis* dimension or VC dimension of the hypothesis space H , denoted by $\text{VC dim}(H)$, is the largest integer i such that $\Delta_H(i) = 2^i$. In the case when no such i exists, $\text{VC dim}(H)$ is infinity.

Figure D.1 illustrates the shattering of 3 noncollinear points by straight lines. A set of 3 noncollinear points is the largest set of points

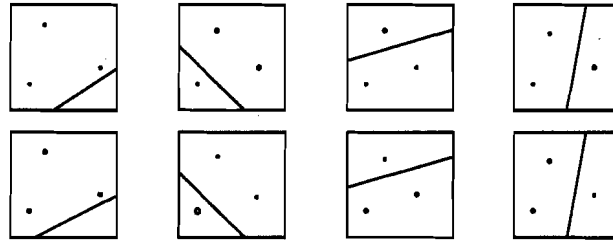


Figure D.1. Shattering of three noncollinear points by straight lines. The VC dimension is three for straight lines in 2-dimensional space on a set of noncollinear points.

that can be shattered in a 2-dimensional space by straight lines. Therefore the VC dimension of the set of straight lines with respect to a set of noncollinear points in a 2-dimensional space is 3.

VC dimension is a combinatorial parameter which measures the expressive power of a network. VC dimension has been used extensively to obtain the generalization ability of a trained network [Blumer et al, 1989; Baum and Haussler, 1989; Sontag, 1992a]. It has been shown that it is not the size of the set of computable functions but the VC dimension of the function that is crucial for good generalization in the context of PAC learning model [Blumer et al, 1989]. The following key result on the bound of the generalization error is given in [Haussler et al, 1994]:

$$\mathcal{E}[e_g(\mathbf{w}, k)] \leq \frac{\text{VC dim}(\mathcal{L})}{k + 1} \quad (\text{D.12})$$

where \mathcal{E} is the expectation operator, \mathcal{L} is the class of target functions (like straight lines in 2-D plane) and k is the number of training patterns.

Experiments conducted by Holden and Niranjana [1994] on real data have shown that the above bound is a moderately good approximation

for the worst case generalization error. The results of bounds based on the VC dimension cannot be used most of the time since it is difficult to calculate the VC dimension. However the calculation of VC dimensions for some classes of functions are reported in [Anthony and Holden, 1994; Sontag, 1992a; Wenocur and Dudley, 1981].

D.3.3 Asymptotic Behaviour of Learning Curves

When the generalization error of a neural network is plotted against the number of training patterns, then the resulting curve is called a learning curve. The behaviour of the learning curve gives an idea about the generalization capability of the trained network [Amari, 1995].

A universal result on the dependence of the **entropic** error $e_g^*(\mathbf{w}, k)$ on the number of training samples is given by [Amari, 1993]

$$\langle e_g^*(\mathbf{w}, k) \rangle \sim \frac{|\mathbf{w}|}{k} \quad (\text{D.13})$$

where $|\mathbf{w}|$ stands for the number of weights, k for the number of training samples and $\langle e_g^*(\mathbf{w}, k) \rangle$ indicates the average over all the training data. **This** result is independent of the architecture of the neural network and the learning algorithm used for training.

D.3.4 Discussion

The VC dimension of a network can be regarded as a measure of capacity or expressive power of a neural network. The number of weights also indicates the capacity of a neural network. The generalization error is directly proportional to the capacity of the network and is inversely proportional to the number of training **patterns**.

In the case of Radial Basis Function Neural Networks (RBFNN), $|\mathbf{w}| - 1 \leq \text{VC dim}(\mathcal{F}) \leq |\mathbf{w}|$, where \mathcal{F} is the family of functions that a network can approximate and $|\mathbf{w}|$ is the number of weights [Anthony and Holden, 1994]. In the case of polynomial basis networks, $\text{VC dim}(\mathcal{F}) = |\mathbf{w}|$. The bounds on the generalization error obtained from computational learning and the behaviour of the learning curves give essentially similar results. But in the computational learning the worst case behaviour of the error is studied, whereas in the learning **curves** case the average behaviour is analyzed [Holden, 1994]. Relationship between these theoretical methods is discussed in [Seung et al, 1992].

D.4 Generalization in the Context of *Feedforward* Neural Networks

Pattern recognition **tasks** are usually complex, and cannot be solved by designing a single algorithm to take care of all the variations in the patterns [Lecun and Bengio, 1995b]. Generalization of a network

depends on the features used for training, whereas the theoretical learning models do not take into account the issue of feature extraction. This is one of the major limitations of the neural networks for the study of generalization.

Despite the above limitation, neural networks perform reasonably well for pattern association problems because of their ability to learn complex mappings in the higher dimensional space. In some cases the generalization performance of a neural network can be improved by manipulating the parameters of the network as follows:

Architecture of neural networks: Choice of an optimum architecture is one of the methods to improve generalization. One way of optimizing the **architecture** is by pruning, which is discussed in detail in the survey paper by Reed [1993].

Size and quality of the training set: A large number of training samples are useful for improving the generalization by a network. One method of increasing the training set data is by introducing noise into the training samples to generate new training examples [Holmstrom and Koistinen, 1993]. A good representation of the training data also improves the generalization [Narendranath, 1995].

Learning algorithm: Methods to accelerate learning are proposed in an effort to train a network with large number of examples [Jean and Wang, 1994].

Criterion for stopping training: Figure D.2 gives **plots** showing the behaviour of training and test error with number of training iterations. Overtraining occurs due to memorization of the training samples by the neural network. There is an increase in the generalization **test error** even though the error on the training set decreases with increase in the number of training iterations. Finding a criterion for stopping the training is a key issue in the generalization in feedforward neural networks.

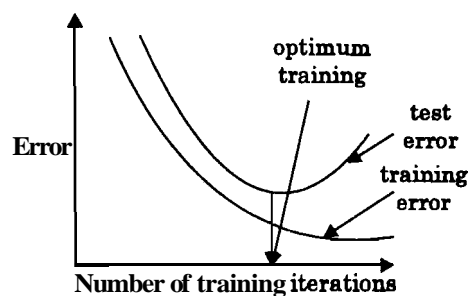


Figure D.2 Graph depicting overtraining. Generalization error is shown as a function of number of iterations. There is an increase in the generalization **test error** even though the error on the training set decreases as the number of training iterations is increased.

Appendix E

Principal Component Neural Networks: An overview

Neural networks have the ability to discover significant features in the input data using *self-organized unsupervised learning* [Haykin, 1994; Linsker, 1988]. The principal component neural network is a self-organizing network which can perform principal component analysis [Matsuoka and Kawamoto, 1994]. In this Appendix we present an overview of the principal component neural networks. The material for this section is obtained from [Sudha, 1996].

E.1 Basics of Principal Component Analysis

From statistical point of view, Principal Component Analysis (PCA) is a method of representing the data points in a **compact** form [Jolliffe, 1986; Hotelling, 1933]. Let us consider a data set $D = \{\mathbf{x} | \mathbf{x} \in \mathcal{R}^N\}$. This data set can be represented as points distributed in an N-dimensional space. The **first** principal component is the direction along which the points have maximum variance. The second principal component is the direction orthogonal to the first component along **which** the variance is maximum for the data points, and so on for the third, fourth, etc. For example, Figure E.1 shows the directions (PC1 and PC2) of the first and second principal components of the data points distributed in a 2-dimensional plane.

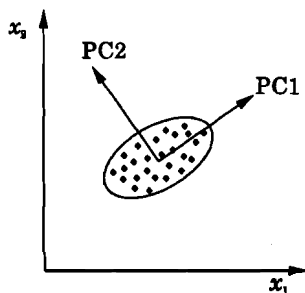


Figure E.1 Directions of the principal components of the data points distributed in a 2-dimensional space.

It is possible to have an effective transformation $\mathbf{x} \rightarrow \mathbf{y}$, where $\mathbf{x} \in \mathcal{R}^N$, $\mathbf{y} \in \mathcal{R}^p$ and $p < N$, when there is redundancy in the data points. This is done by projecting the data points onto the principal subspace formed by the first p principal components, also called *major components* which capture the maximum variations among the points. This forms the basis for *dimensionality reduction*, and the method of data representation is commonly referred to as *subspace decomposition*. Approximation to the data point \mathbf{x} reconstructed with minimum error from the projections \mathbf{y} onto the p largest principal component directions \mathbf{q}_i s is given by

$$\hat{\mathbf{x}} = \sum_{i=1}^p y_i \mathbf{q}_i.$$

The error vector $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}} = \mathbf{x} - \sum_{i=1}^p y_i \mathbf{q}_i$ is orthogonal to the approximating data vector $\hat{\mathbf{x}}$, which is called the *principle of orthogonality*. PCA is similar to the *Karhunen-Loeve transformation* [Devijver and Kittler, 1988] in communication theory. The principal component analysis is a data dependent transformation.

Extraction of the principal components can be done using the covariance matrix, $C = \mathcal{E}[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T]$ of the data set, where $\bar{\mathbf{x}} = \mathcal{E}[\mathbf{x}]$ is mean of the data set. The principal components are the *eigenvectors* of the data covariance matrix C arranged in the descending order of the eigenvalues [Haykin, 1994; Preisendorfer, 1988; Leon, 1990].

E.2 Need for Neural Networks in PCA

In practice we have only an estimate of the covariance matrix due to limited data, whereas the true covariance matrix is the ensemble average of the stochastic process generating the data. Moreover, for a nonstationary process, the principal components may vary with time. Therefore direct computation of the principal components is *difficult*, and is also not likely to be accurate.

On the other hand, a neural network can extract the principal components directly from the data, by incrementally adjusting its weights. Moreover, it is possible to extract the required number of principal components by a neural network, instead of extracting all the components as in the direct computation. For nonstationary processes, the principal components are incrementally adjusted based on the new input to the neural network.

With a single linear unit the simple unsupervised Hebbian learning performs variance maximization, and hence gives the direction of the *first* principal component [Hebb, 1949; Palmieri and Zhu, 1995]. The supervised mean squared error learning for a linear

network can be interpreted as a sum of Hebbian and anti-Hebbian learning **components** [Wang et al, 1995].

The main feature of the linear network is that the energy landscape has a unique global minimum, and the principal component learning converges to the global minimum. Both the gradient descent and the Newton's type methods may get stuck in the saddle points [Haykin, 1994; Hertz et al, 1991]. Thus the principal component learning is the best learning [Baldi and Hornik, 1989] for a linear feedforward neural network.

E.3 Principal Component Neural Networks (PCNN)

E.3.1 Oja's Learning

The drawback of the Hebbian learning for principal component analysis is that the weights may grow indefinitely with training or they may tend to zero. This can be avoided by adding a stabilizing' term. Oja modified the Hebbian learning rule which incorporates the normalization of weights.

For a single linear unit shown in **Figure E.2**, the weight update according to the Hebbian learning is given by

$$\Delta w_j(m) = w_j(m+1) - w_j(m) = \eta y(m)x_j(m) \quad (\text{E.1})$$

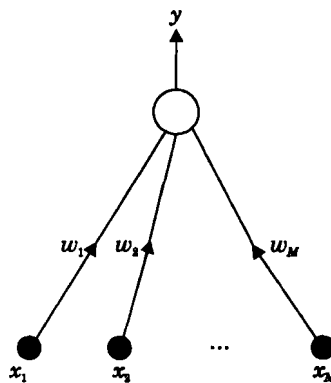


Figure E2 Single linear unit model as a maximum eigenfilter.

where η is the learning rate parameter, $y(m)$ is the output of the linear neuron and $x_j(m)$ is the j th component of the input pattern **vector** at the m th iteration. After incorporating the **normalization** term in the learning rule, the above equation leads to the following **Oja's** learning rule [Oja, 1982; Yan et al, 1994; Zhang and Leung, 1995]:

$$\Delta w_j(m) = \eta y(m) [x_j(m) - y(m)w_j(m)] \quad (\text{E.2})$$

The **Oja's** learning has two feedback terms: (a) The positive feedback or self-amplification term for the growth of the synaptic weight $w_j(m)$ according to the external input $x_j(m)$. (b) The negative feedback term due to the term $-y(m)w_j(m)$ for controlling the growth, thereby resulting in the stabilization of the synaptic weight $w_j(m)$.

The weights converge to the first principal component of the input distribution as shown below:

Substituting $y(m) = \mathbf{x}^T(m)\mathbf{w}(m) = \mathbf{w}^T(m)\mathbf{x}(m)$ in Eq. (E.2), we get

$$\Delta\mathbf{w}(m) = \eta [\mathbf{x}(m)\mathbf{x}^T(m)\mathbf{w}(m) - \mathbf{w}^T(m)\mathbf{x}(m)\mathbf{x}^T(m)\mathbf{w}(m)\mathbf{w}^T(m)]$$

Taking statistical expectation on both sides, for large m , we should get $\mathbf{E}[\Delta\mathbf{w}] = \mathbf{0}$. Therefore,

$$\mathbf{0} = \mathbf{R}\mathbf{q}_0 - (\mathbf{q}_0^T\mathbf{R}\mathbf{q}_0)\mathbf{q}_0$$

where $\mathbf{w}(m) \rightarrow \mathbf{q}_0$ as $m \rightarrow \infty$ and $\mathbf{R} = \mathbf{E}[\mathbf{x}(m)\mathbf{x}^T(m)]$. \mathbf{q}_0 is the eigenvector of the correlation matrix \mathbf{R} corresponding to the largest eigenvalue [Haykin, 1994].

E.3.2 Learning Principal Subspace

Oja extended the single unit case to multiple units to extract the principal **subspace** [Oja, 1989]. The learning algorithm is given by

$$\Delta w_{ij}(m) = \eta y_i(m) \left[x_j(m) - \sum_{k=1}^p w_{kj}(m) y_k(m) \right]$$

where w_{ij} is the weight connecting the j th input with the i th unit. Here the weights will not tend to the eigenvectors but only to a set of rotated basis vectors which span the principal **subspace** corresponding to the first p principal components.

E.3.3 Multiple Principal Component Extraction: Generalized Hebbian Algorithm

By combining the **Oja's** rule and the Gram-Schmidt orthonormalization process, Sanger modified the **subspace** network learning algorithm to compute the first p principal components of a stationary process simultaneously [Sanger, 1989]. A feedforward neural network with a single layer of linear units (M inputs and p outputs) as shown in the Figure E.3 performs the principal component analysis of the input data. The Generalized Hebbian learning Algorithm (GHA) is given by

$$\Delta w_{ij}(m) = \eta y_i(m) \left[x_j(m) - \sum_{k=1}^i w_{kj}(m) y_k(m) \right], \quad \text{for } j = 1, 2, \dots, M$$

and $i = 1, 2, \dots, p$

Principal *Component* Neural Networks (*PCNN*)

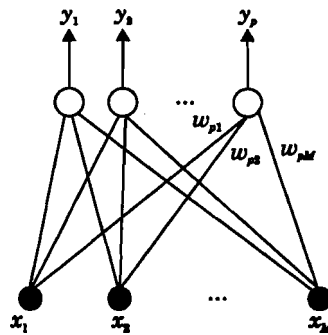


Figure E.3 Single layer of linear units for multiple principal component extraction.

and the output $y_i(m)$ of the i th unit is

$$y_i(m) = \sum_{j=1}^M w_{ij}(m)x_j(m)$$

In the GHA the modified form of the input vector is given by

$$\hat{\mathbf{x}}_i(m) = \mathbf{x}(m) - \sum_{k=1}^{i-1} \mathbf{w}_k(m)y_k(m). \quad (\text{E.3})$$

- (a) For the first unit, $i = 1$ and $\hat{\mathbf{x}}_1(m) = \mathbf{x}(m)$. The GHA reduces to the **Oja's** learning rule. So it extracts the **first** principal component.
- (b) For the second unit, $i = 2$ and $\hat{\mathbf{x}}_2(m) = \mathbf{x}(m) - \mathbf{w}_1(m)y_1(m)$. The second unit sees an input vector $\hat{\mathbf{x}}_2(m)$ in which the component corresponding to the first eigenvector of the correlation matrix \mathbf{R} has been removed. So the second unit extracts the first principal component of $\hat{\mathbf{x}}_2(m)$ which is equivalent to the second principal component of $\mathbf{x}(m)$.
- (c) Proceeding in this fashion, the outputs of the units extract the principal components of $\mathbf{x}(m)$ in the decreasing order of the eigenvalues.

E.3.4 Adaptive Principal Component Extraction

Principal components can be extracted one by one recursively. By including *anti-Hebbian* feedback connections [Palmieri et al., 1993] in the network, the outputs of the **units** define a coordinate system in which there are no correlations even when the incoming signals have strong correlations. Foldiak [Foldiak, 1989] developed a procedure which uses anti-Hebbian connections between every pair of network

outputs to **orthogonalize** the weight vectors. Kung and Diamantaras developed an algorithm called Adaptive Principal Component Extraction (APEX) for recursive computation of the principal components based on a sequential training scheme which uses anti-Hebbian weights from the already trained units to the unit that is currently being trained [Kung and Diamantaras, 1990; Kung and Diamantaras, 1994]. Using this scheme, one can adaptively increase the number of units needed for the principal component extraction. The architecture of the APEX network is shown in the **Figure E.4**.

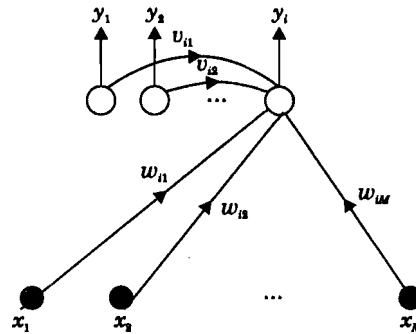


Figure E.4 APEX network architecture for multiple principal component extraction.

There are two kinds of synaptic connections in the network:

- (a) Feedforward connections from the input to each of the units which operate in accordance with the Hebbian learning rule. They are excitatory and therefore provide self-amplification.
- (b) Lateral connections to a unit from the outputs of the previous units, which operate in accordance with anti-Hebbian learning rule, which has the effect of making them inhibitory.

The output of the i th unit is given by

$$y_i(m) = \mathbf{w}_i^T(m)\mathbf{x}(m) + \mathbf{v}_i^T(m)\mathbf{y}_{i-1}(m)$$

where the **feedforward** weight vector $\mathbf{w}_i(m) = [w_{i1}(m), \dots, w_{iM}(m)]^T$, the feedback weight vector $\mathbf{v}_i(m) = [v_{i1}(m), \dots, v_{i(i-1)}(m)]^T$ and the feedback signal vector $\mathbf{y}_i(m) = [y_1(m), \dots, y_{i-1}(m)]^T$.

The **feedforward** and lateral connection weights are updated as follows:

$$\Delta \mathbf{w}_i(m) = \mathbf{w}_i(m+1) - \mathbf{w}_i(m) = \eta [y_i(m)\mathbf{x}(m) - y_i^2(m)\mathbf{w}_i(m)]$$

$$\Delta \mathbf{v}_i(m) = \mathbf{v}_i(m+1) - \mathbf{v}_i(m) = -\eta [y_i(m)\mathbf{y}_{i-1}(m) - y_i^2(m)\mathbf{v}_i(m)]$$

where the term $y_i(m)\mathbf{x}(m)$ represents the Hebbian learning, and the term $-y_i(m)\mathbf{y}_{i-1}(m)$ represents the anti-Hebbian learning. The

remaining terms are included for the stability of the algorithm. In the following sections some PCNNs designed for specific situations are discussed.

E.3.5 Crosscorrelation Neural Network Model

The neural network models discussed in the previous sections extract the principal components of the autocorrelation matrix of the input data. A crosscorrelation neural network model [Diamantaras and Kung, 1994] performs Singular Value Decomposition (SVD) [Leon, 1990] of the *crosscorrelation matrix* of two signals generated by two different stochastic processes which are related to each other. The principal singular vectors of the crosscorrelation matrix encode the directions in both the spaces of the stochastic processes, that support the major *common features* of both the signals. The learning rule is an extension of the Hebbian rule called the *mutual* or *cross-coupled Hebbian rule*, and it can be considered as a *crosscorrelation asymmetric PCA* problem [Kung, 1993].

The SVD of the crosscorrelation matrix $C = \mathbf{E}[\mathbf{y}\mathbf{x}^T]$ of two stochastic signals, \mathbf{x} and \mathbf{y} is given by $C = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where \mathbf{U} is the matrix containing left singular vectors which span the column space of the matrix C (eigenvectors of CC^T) and \mathbf{V} contains the right singular vectors which span the row space of the matrix C (eigenvectors of C^TC). The mutual Hebbian rule extracts both the left and right singular subspaces.

Consider two linear units as shown in the Figure E.5 with inputs $\mathbf{x} \in \mathcal{R}^M$, $\mathbf{y} \in \mathcal{R}^N$, and outputs

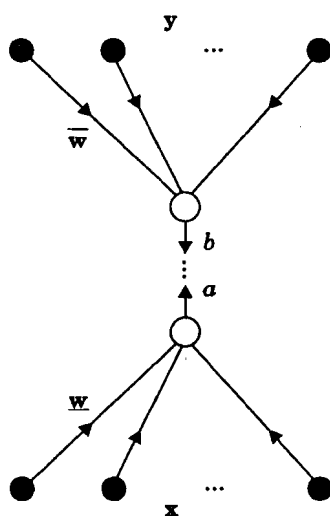


Figure E.5 Crosscorrelation neural network model for performing SVD of crosscorrelation matrix of two stochastic signals \mathbf{x} and \mathbf{y} .

$$a = \underline{\mathbf{w}}^T \mathbf{x} \quad \text{and} \quad b = \overline{\mathbf{w}}^T \mathbf{y}$$

The cross-coupled Hebbian rule that updates the weights of any one of the two units is based on the correlation between the input of this unit and the output of the other unit and hence the name of the rule.

$$\Delta \overline{\mathbf{w}}(m) = \overline{\mathbf{w}}(m+1) - \overline{\mathbf{w}}(m) = \eta a(m) \mathbf{y}(m)$$

$$\Delta \underline{\mathbf{w}}(m) = \underline{\mathbf{w}}(m+1) - \underline{\mathbf{w}}(m) = \eta b(m) \mathbf{x}(m)$$

where η is the learning rate parameter. In order to maintain stability, the weights are normalized and the resultant update rule becomes

$$\Delta \overline{\mathbf{w}}(m) = \overline{\mathbf{w}}(m+1) - \overline{\mathbf{w}}(m) = \eta [\mathbf{y}(m) - \overline{\mathbf{w}}(m) b(m)] a(m)$$

$$\Delta \underline{\mathbf{w}}(m) = \underline{\mathbf{w}}(m+1) - \underline{\mathbf{w}}(m) = \eta [\mathbf{x}(m) - \underline{\mathbf{w}}(m) a(m)] b(m)$$

By maximizing the crosscorrelation cost

$$J = \frac{\mathcal{E}[ba]}{\|\overline{\mathbf{w}}\| \|\underline{\mathbf{w}}\|} = \frac{\mathcal{E}[\overline{\mathbf{w}}^T \mathbf{y} \mathbf{x}^T \underline{\mathbf{w}}]}{\|\overline{\mathbf{w}}\| \|\underline{\mathbf{w}}\|} = \frac{\overline{\mathbf{w}}^T R_{\mathbf{y}\mathbf{x}} \underline{\mathbf{w}}}{\|\overline{\mathbf{w}}\| \|\underline{\mathbf{w}}\|}$$

where $R_{\mathbf{y}\mathbf{x}}$ is the crosscorrelation matrix, the solution for the weight vectors converges to the principal singular vectors [Leon, 1990].

E.3.6 Higher Order Correlation Learning Network

The Oja's learning does not capture the higher order statistics in the input data. A higher order unit [Taylor and Coombes, 1993], which accepts inputs from more than one channel, is capable of capturing the higher order statistics of the input. Figure E.6 shows a higher

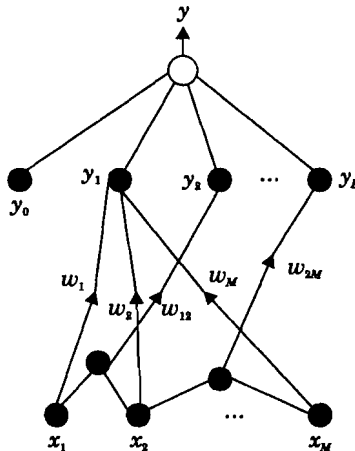


Figure E.6 Higher order neuron model for learning higher order statistics of the input.

order **unit** consisting of a set of higher order connection weights, $w_i, w_{ij}, w_{ijk}, \dots$, **such** that the output of the unit is given by

$$y(m) = \phi \left[\sum_{k=0}^K y_k(m) \right]$$

where

$$y_0(m) = w_0(m),$$

$$y_1(m) = \sum_{j=1}^M w_j(m)x_j,$$

$$y_2(m) = \sum_{i,j=1}^M w_{ij}(m)x_i x_j,$$

x_i denotes the i th component of an M -dimensional input vector x , K is called the order of the unit, ϕ is a nonlinear **function** such as sigmoid.

E.3.7 Nonlinear PCNN and Independent Component Analysis

Normally the PCNN is a single layer linear feedforward neural network. Nonlinear units in the network introduces higher order statistics into computation. The weight vectors become independent of each other and they need not be orthogonal. The network thus performs an Independent Component Analysis (ICA) [Comon, 1994; Cardoso, 1989; Karhunen and Joutsensalo, 1995]. This helps in separating the independent subsignals from their mixture. The nonlinear learning algorithm of ICA may get caught easily in a local minimum.

ICA provides independence, whereas PCA provides only **decorrelation** [Jutten and Herault, 1988]. The principal component basis **vectors** are orthogonal, whereas the ICA basis **vectors** may not be orthogonal. Principal components can be ordered according to their eigenvalues. But in the case of ICA, the coordinates are independent of each other. ICA involves higher order statistical moments while PCA considers only the second order moments. PCA is useful for data compression applications, whereas ICA is useful for signal separation problems.

A simple illustration of the difference between PCA and ICA is given in Figure E.7. Consider a 2-dimensional plane where the data points are distributed inside a parallelogram [Burel, 1992]. PCA finds orthogonal coordinate axes (PC1 and PC2) where the maximum dispersion is obtained on the first axis. The coordinate axes of ICA (IC1 and IC2) are fully independent. Knowledge of IC1 does not give any information about IC2.

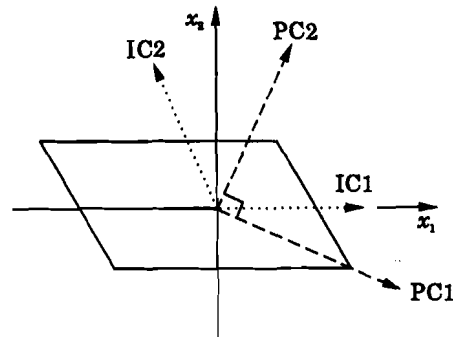


Figure E.7 Comparison of principal component analysis and independent component analysis.

A **summary** of the principal component neural networks is given in Table **E.1**.

Table E.1 Summary of Principal Component Neural Networks

1. A linear unit model as a maximum eigenfilter

- Oja's learning rule: A normalized Hebbian learning algorithm.
- Extracts the first Principal Component (PC).

2. Principal subspace extraction with a layer of neurons

- Oja's p-unit learning algorithm
- Extracts p-dimensional **subspace** with p units.

3. Multiple principal component extraction

- Generalized Hebbian learning algorithm: **Sanger's** rule.
- Extracts the first p **PCs** using a single layer linear feedforward neural network with p units.

4. Adaptive principal component extraction

- Computes **PCs** one by one recursively.
- Anti-Hebbian lateral connections in the output.

5. Crosscorrelation neural network model

- **Cross-coupled** Hebbian rule.
- Performs SVD of the crosscorrelation matrix of two stochastic signals.

6. Higher order correlation learning network

- Learns the higher order statistics of the input data.

7. Nonlinear PCNN

- Nonlinear learning algorithm.
 - Performs Independent Component Analysis.
 - Used for blind separation of independent source signals from their mixture in the received signal.
-

Applications of PCNN

E.4 Applications of PCNN

Applications of PCNN are based on two kinds of data: (a) Statistical data in which the data vector is considered as a point in an N-dimensional space. (b) Temporal data in which the data vector is a segment of sampled signal.

E.4.1 General Applications

These applications consider the statistical data.

Data compression: The dimensionality reduction property of PCA forms the basis for data compression.

Compensation of misalignment of an image. The misalignment of an image due to rotation **and/or** translation is compensated by finding the principal eigenvector of the image and aligning it with the new coordinate system.

PCA as a preprocessor: The projections of a data vector onto the principal components are **uncorrelated** to each other. When these components are given as input to a neural network classifier, the convergence of the network improves [Veckovnik et al, 1990].

Evaluation of feature extraction techniques: If the data set is made up of aggregate of several clusters, the separability of the clusters can be improved using **the** projections of the clusters onto the principal axes.

Subspace-based classification: Different classes of patterns have different sets of principal components. The patterns of a class tend to have larger projections on their own class components than any other class components.

Generalization measure: Generalization here means how well a new pattern can be reconstructed [Baldi and Homik, 1989]. The amount of distortion in the new pattern can be interpreted as the distance of the pattern point to the principal subspace.

Curve and surface fitting: Conventional total least square curve fitting problem can be reduced to finding the minimum eigenvalue and its corresponding normalized eigenvector of the input covariance matrix [Xu and Suen, 1992]. Higher order neural networks can implement nonlinear decision boundaries [Taylor and Coombes, 1993].

Noise cancellation by crosscorrelation neural network: In some adaptive control applications, the crosscorrelation matrix represents the unknown plant *transfer function* from inputs to outputs. The

crosscorrelation neural network model can be potentially used for *filtering applications* [Diamantaras and Kung, 1994] if we have *a priori* knowledge of noise present in a signal.

E.4.2 Applications Specific to Signal Processing

In signal processing applications the data is a temporal data. Many of the frequency estimation algorithms are based on the eigendecomposition of the signals [Kay, 1988; Marple, 1987]. PCNN finds application in the problem of frequency estimation. The signal and noise subspaces of the observed signal space can be estimated by eigendecomposition of the autocorrelation matrix of observed signal [Lee, 1992; Kung, 1993]. In the eigendecomposition of the autocorrelation of a signal with M complex sinusoids, the first M eigenvectors corresponding to the large eigenvalues span the signal **subspace** and the remaining span the noise **subspace** [Lee, 1992; Kung, 1993; Tufts and Kumaresan, 1982; van der Veen, 1993]. Methods for estimating the frequencies by signal **subspace** are called *principal component frequency estimation*. In the *noise subspace frequency estimation*, the property that the noise **subspace** is perpendicular to the signal **subspace** is applied [Kay, 1988; Marple, 1987]. By reconstructing the signal from the projections of the signal onto the signal **subspace** eigenvectors, the noise in the signal is considerably reduced. Thus PCNN can be applied for noise suppression.

We can estimate the principal components of the input signal using PCNN, and these estimated components can then be used for frequency estimation algorithms such as MUSIC, Bartlett or Pisarenko harmonic decomposition [Kay, 1988; Marple, 1987; Karhunen and Joutsensalo, 1991]. Recently, it was found that the PCNN can be made to perform independent component analysis by introducing nonlinearity in the learning algorithm [Karhunen and Joutsensalo, 1994]. The resultant network can be used for blind separation of independent sources from an observed signal. This is useful in sonar and speech for extracting different frequency components present in the signal and hence **tracking** the changes in these frequencies [Sudha, 1996].

Appendix F

Current Trends in Neural Networks

Over the past few years there are attempts to combine ANN models with other well-established paradigms, like evolutionary computation, fuzzy logic, rough sets and chaos. In this Appendix, we briefly discuss how these paradigms are being **fused** with the existing ANN models.

F.1 Evolutionary Computation

Evolutionary Computation (EC) [Fogel, 1994] is a methodology that encompasses a variety of population-based problem solving techniques which mimic the natural process of Darwinian evolution. Current research in the evolutionary computation has resulted in powerful and versatile problem-solving mechanisms for global searching, adaptation, learning and optimization for a variety of pattern recognition tasks. The main techniques in evolutionary computation are genetic algorithms [Holland, 1975; Goldberg, 1989], genetic programming [Koza, 1992], evolutionary strategies [Schwefel, 1981] and evolutionary programming [Fogel et al, 1966; Fogel, 1991; Fogel, 1995]. Genetic algorithms deal with chromosomal operators, genetic programming stresses on operators on general hierarchical structures, evolution strategies emphasize on the behavioural changes at the **individual** level, and evolutionary programming focusses on the behavioural changes at the level of species. The common factor underlying all these techniques is the emphasis on an ensemble of solution structures, and on the evaluation and evolution of these structures via specialized operators similar to a biological system in response to an ever changing environment. Specifically, all the techniques maintain a population of trial solutions, impose random changes to those solutions, and incorporate selection to determine which solutions are to be retained for future generation and which are to be removed **from** the pool of trial solutions. From a mathematical point of view, all the EC techniques can be **considered** as controlled, parallel, stochastic search, optimization techniques.

Since the learning methods used in ANN depend on the optimization of some objective function, it is possible to employ the methodology of EC for learning the weights, for evolving the network architecture, for developing a learning rule, for selection of an input feature and so on [Yao, 1993; Bornholdt and Graudenz, 1992]. For instance, in a MLFFNN the gradient-based local search methods can be substituted by EC for determining the weights [Porto et al, 1995; Saravanan and Fogel, 1995; Miller et al, 1989]. In some cases it may be possible to exploit both the local search methods (like the gradient descent) and the global search methods (like EC) simultaneously [Renders and Flasse, 1996]. The advantages of the local search methods are better accuracy and fast computation. The disadvantages of the local search methods are stagnation at some suboptimal solutions and sensitivity to the initialization of weights. On the other hand EC is a global search method which can avoid the local optima and the initialization problems [Sarkar and Yegnanarayana, 1997a]. However, EC can be extremely slow in convergence to a good solution. This is because EC uses minimal *a priori* knowledge, and does not exploit available local information [Renders and Flasse, 1996]. In fact EC is good for exploration, whereas the gradient descent methods are good for exploitation. Yao and Liu [Yao and Liu, 1997] have proposed a method to evolve the topology (architecture and weights) of a MLFFNN by using both the evolutionary programming and backpropagation algorithm simultaneously.

EC-based techniques are successfully applied to configure RBF networks for improving generalization [Whitehead, 1996; Billings and Zheng, 1995]. In [Angeline et al, 1994] the authors have used EC to configure recurrent neural networks. Jockusch and Ritter [Jockusch and Ritter, 1994] have introduced a training strategy to determine the number of units for a SOM network automatically. EC has also been used to find the optimal number of clusters present in the input data [Sarkar and Yegnanarayana, 1996; Sarkar et al, 1997e]. The clustered output can be used to construct a probabilistic neural network [Sarkar and Yegnanarayana, 1997b]. Attempts are being made to explore the EC approach for simultaneously learning the weights and evolving the architecture of a neural network [Yao, 1993]. The problem of large search space for this type of problem can be addressed by using parallel machines to implement the search operation [Bhattacharya and Roysam, 1994]. The search operation can also be made more efficient and less time consuming by using adaptive EC operators [Sarkar and Yegnanarayana, 1997a].

For some ANN related problems, EC appears to be a more powerful optimization tool than the simulated annealing (SA), since SA is a sequential search operation whereas EC is a parallel search algorithm. In fact, we can say that EC is more than a parallel search. Parallel search starts with a number of different paths and continues

until all the search paths get stuck in blind alleys or any one of them finds the solution. EC also starts with P different paths, but it tries to generate new paths that are better than the current paths. Thus the EC-based search may be more efficient than the SA-based search [Porto et al, 1995].

F.2 Fuzzy Logic

The concept of fuzzy sets was first introduced by L. Zadeh in 1965 [Zadeh, 1965] to represent vagueness present in human reasoning. Fuzzy sets can be considered as a generalization of the classical set theory. In a classical set an element of the universe either belongs to or does not belong to the set. Thus the belongingness of an element is crisp. In a fuzzy set the belongingness of an element can be a continuous variable. Mathematically, a fuzzy set is a mapping (known as membership function) from the universe of discourse to $[0, 1]$. The higher the membership value of an input pattern to a class, the more is the belongingness of the pattern to the class. The membership function is usually designed by taking into consideration the requirements and constraints of the problem. One may obtain the membership function from an expert (subjective computation) or from the data (objective computation) [Bezdek and Pal, 1992]. Fuzzy logic deals with reasoning with fuzzy sets and fuzzy numbers. It is to be noted that fuzzy uncertainty is different from probabilistic uncertainty [Klir and Folger, 1993; Klir and Yuan, 1995].

ANNs adopt numerical computations for learning. But numerical quantities lack representative power in situations where the information is expressed in linguistic terms only [Lin and Lu, 1995]. The linguistic information can be incorporated using the membership function values of the fuzzy sets. Use of the concepts of fuzzy sets in ANNs is also supported by the fad that human reasoning does not employ precise mathematical formulation [Pal and Majumder, 1986]. Specifically, the fuzzy set theory can be used in ANN at various levels such as the input, output and target, and also for the weights, basis functions and the output functions. Introduction of fuzzy set theory into the perceptron learning algorithm makes the decision boundary a soft one, so that the class labels of the input patterns can change slowly from one class to another class, rather than abruptly [Keller and Hunt, 1985].

In [Sarkar et al, 1998; Pal and Mitra, 1992] the network outputs are interpreted as fuzzy membership values. Learning laws are derived by minimizing a fuzzy objective function in a gradient descent manner. In [Sarkar and Yegnanarayana, 1997d] the concept of cross entropy was extended to incorporate fuzzy set theory. Incorporation of fuzziness in the objective functions led to better classification in many cases.

A neural network reinforcement learning algorithm, with

linguistic critic signals like *good*, bad, is proposed in [Lin and Lu, 1995]. The network was able to process and learn numerical information as well as linguistic information in a control application.

In [Chung and Lee, 1994], three existing competitive learning algorithms, namely the unsupervised competitive learning, learning vector quantization, and frequency sensitive competitive learning, are fuzzified to form a class of fuzzy competitive learning algorithms. Unlike the crisp counterpart, where only one output unit wins, here all the output units win with different degrees. Thus the concept of win has been formulated as a fuzzy membership function. It has been observed that this scheme leads to better convergence and better classification performance.

In [Tsao et al, 1994] **Kohonen's** clustering network has been generalized to its fuzzy counterpart. One advantage of this approach is that the final weight vectors do not depend on the sequence of presentation of the input vectors. Moreover, the method uses a systematic approach to determine the learning rate parameter and size of the neighbourhood.

A fuzzy adaptive resonance theory model capable of rapid learning of recognition categories in response to arbitrary sequence of binary input patterns is proposed in [Carpenter et al, 1991c]. This upgradation from binary **ART1** to fuzzy ART is achieved by converting the crisp logical operators used in the binary ART to the corresponding fuzzy logical operators. As a result of this upgradation, learning becomes fast and also the previously learned memories are not erased rapidly in response to fluctuations in the input.

In [Wang and Mendel, 1992] the authors have proposed fuzzy basis functions to design an RBF network which can accept both numerical inputs as well as fuzzy linguistic inputs. In [Pedrycz, 1992] Pedrycz has proposed a neural network model based on fuzzy logical connectives. Instead of using linear basis function, he has utilized fuzzy aggregation operators. This technique has been extended to a more general case where the inhibitory and excitatory characteristics of the inputs are captured using direct and complemented (i.e., negated) input signals [Pedrycz and Rocha, 1993; Hirota and Pedrycz, 1994]. The advantage of this approach is that the problem specific *a priori* knowledge can be incorporated into the network.

In another development, Ishibuchi et al have proposed a learning algorithm where the *a priori* knowledge in terms of fuzzy if-then rules can be incorporated along with the information supplied by the numerical data [Ishibuchi et al, 1993]. This type of approach has been used for both function approximation and classification. Fuzzy set theory has also been employed to speed up the training of an ANN. In [Choi et al, 1992], a fuzzy rule base is used to dynamically adapt the learning rate and momentum parameters of a **MLFFNN** with backpropagation learning algorithm.

F.3 Rough Sets

In many classification tasks the aim is to **form** classes of objects which may not be significantly different. These indiscernible or indistinguishable objects are useful to build knowledge base pertaining to the task. For example, if the objects are classified according to colour (red, black) and shape (triangle, square and circle), then the indiscernible classes are: red triangles, black squares, red circles, etc. **Thus** these two attributes make a partition in the set of **objects**. Now if two red triangles with different areas belong to different classes, then it is impossible for anyone to classify these two red triangles based on the given two attributes. This **kind** of uncertainty is **referred** to as rough uncertainty [Pawlak, 1982; Pawlak et al, 1995]. Pawlak **formulated** the rough uncertainty in terms of rough sets. The rough uncertainty is completely avoided if we can successfully extract all the essential features to represent different **objects**. But it may not be possible to guarantee this as our knowledge about the system generating the data is limited. It must be noted that rough uncertainty is different **from** fuzzy uncertainty [Dubois and Prade, 1992].

In this section we briefly describe the **formulation** of rough sets. In any classification problem, two input training patterns \mathbf{x}_r and \mathbf{x}_s (where $\mathbf{x}_r, \mathbf{x}_s \in X$) are indiscernible with respect to the **qth** feature when the **qth** component of these two patterns have the same value. Mathematically, it can be stated as $\mathbf{x}_r R^q \mathbf{x}_s$ iff $x_{r,q} = x_{s,q}$, where R^q is a binary relation over $X \times X$. Obviously, R^q is an equivalence relation that partitions the universal set X into different equivalence classes. Instead of taking only one feature, if we consider any two features (say pth and **qth**), then we obtain some other equivalence relation R^{pq} and a new set of equivalence classes. This idea can be generalized to take all known features into consideration. Let R be an equivalence relation on the universal set X and $X|R$ denote the family of all equivalence classes induced on X by R . One such equivalence class in $X|R$ that contains $x \in X$ is designated by $[\mathbf{x}]_R$. In any classification problem the objective is to approximate the given class $A \subseteq X$ by $X|R$. For the class A , we can define the lower $\underline{R}(A)$ and upper $\overline{R}(A)$ approximations, which approach A as closely as possibly from inside and outside, respectively [Klir and Yuan, 1995]. Here, $\underline{R}(A) = \cup \{[\mathbf{x}]_R | [\mathbf{x}]_R \subseteq A, x \in X\}$ is the union of all equivalence classes in $X|R$ that are contained in A , and $\overline{R}(A) = \cup \{[\mathbf{x}]_R | [\mathbf{x}]_R \cap A \neq \phi, x \in X\}$ is the union of all equivalence classes in $X|R$ that overlap with A . A rough set $R(A) = \{\underline{R}(A), \overline{R}(A)\}$ is a representation of the given set A by $\underline{R}(A)$ and $\overline{R}(A)$. The set difference $\overline{R}(A) - \underline{R}(A)$ is a rough description of the boundary of A by the equivalence classes of $X|R$. The approximation is free of rough uncertainty if $\overline{R}(A) = \underline{R}(A)$. When all the patterns from an equivalence class do not have the same class

label, rough ambiguity is generated as a manifestation of the one-to-many relationship between that equivalent class and the class labels to which the patterns belong.

In ANN design one critical problem is to determine how many input units are essential. Obviously, it depends on the number of features present in the input data. Using rough sets it may be possible to decrease the dimensionality of the input without losing any information. A set of features is sufficient to classifi. all the input patterns if the rough ambiguity, i.e., the quantity $(\underline{R}(A) - \underline{R}(A))$, for this set of features is equal to zero. Using this quantity it is possible to select a proper set of features from the given data [Pawlak et al, 19881.

In a **classification** task all the features need not carry equal weightage. Hence to facilitate the training as well as to improve the accuracy of classification, it is better to give different weightage or importance for each input feature. Suitable weightage can be derived using the ideas of rough sets [Sarkar and Yegnanarayana, 1997c].

The training of an ANN can be accelerated if the weights of the networks are initially close to the desired ones. For this purpose knowledge extracted from the training data through rough sets can be used to initialize the ANN [Banerjee et al, 1997]. In [Pawlak et al, 1995] it was shown that for a classification task the number of **hidden** units needed in a MLFFNN is equal to the minimal number of features needed to represent the data set without increasing the rough uncertainty.

F.4 Chaos

In many physical systems there appears to be no relationship between cause and effects. In these cases the uncertainty of the system behaviour cannot be predicted using the standard statistical methods. The apparent randomness may in fact be generated by **small** differences in the initial values of the physical systems. Since the whole process is absolutely deterministic, this type of uncertainty is termed as **deterministic chaos** or simply **chaos** [Crutchfield et al, 1986].

Chaotic dynamics is known to exist in biological neural **net-**works due to nonlinear processing units and their interaction due to complex feedback mechanism [Harth, 1983]. The stability-plasticity phenomenon in the biological neural network is attributed to its ability to convert the neural dynamics **from** highly ordered state to chaotic state and vice versa. In order to realize some form of human reasoning capability on machines it may be necessary to exploit the phenomenon of chaos existing in the artificial neural networks. But the sensitivity of feedback networks to initial input conditions makes it difficult to come with any long term predictions about the behaviour of the networks [Parker and Chua, 1987].

The dynamics of a feedback neural network is studied in terms

Chaos

of the equilibrium states that the network reaches starting from some initial states. The equilibrium states can be characterized in terms of fixed point stability, oscillatory stability and chaotic stability. The regions of oscillatory stability are termed as attractors. There can be several other types of attractors like quasi-periodic and strange attractors. The strange **attractor** is an example of chaotic attractor Wasserman, **1993**. In the state space the orbits of the strange attractors sometimes diverge. But, the divergence cannot continue forever as the state space is finite. So the attractors fold over onto themselves. These stretching and folding operations continue repeatedly, creating folds within folds. Consequently, chaotic attractors generate a fractal-like structure that reveals more and more as it is increasingly magnified [**Mandlebrot, 1983**]. This stretching operation **systematically** removes the initial information and makes small scale uncertainty large. The folding operation also removes the initial information. **If** we know the initial state of the network with some uncertainty (due to measurement error), **after** a short period of time the initial uncertainty covers the entire **attractor** and all predictive power is lost. **Thus** there exists no relationship between the past and the future, or the cause and the effects.

There are several avenues to exploit chaos under the ANN paradigm. It is claimed that several limitations of **ANNs** are due to its grossly oversimplified structure. For example, the output of an artificial neuron is smooth, whereas the output of a biological neuron forms a train of pulses. Hence, using **Hodgkin-Huxley** type cell equations, attempts are being made to create complex artificial neuron models to exploit the chaos generated by the cell equations. Freeman and his co-workers have demonstrated that different **kinds** of stimuli in animal cortex can be represented as chaotic attractors [**Yao et al, 1990**]. They have successfully developed an artificial olfactory model, where the artificial neurons exploit chaos for its functioning [**Eisenberg et al, 1989**].

Attempts are also being made to control the chaotic behaviour of an artificial neuron [**Hsu et al, 1996**; Sompolinsky et al, **1988**; **Hayashi, 1994**; Ott et al, **1990**; Hunt, **1991**]. The chaotic variables may be neuron output activities and the control parameters may be synaptic weights or external inputs [**Blondeau et al, 1992**]. In many cases the proposed models do not have any direct physiological interpretation [**Blondeau and Rivest, 1992**].

Scientists are recently analyzing the chaotic dynamics of feedback networks [**Wang, 1996**]. They are employing the periodic attractors embedded in each chaotic attractor to store input patterns. Following this strategy, in [**Adachi and Aihara, 1997**] a chaotic associative memory was constructed. It has been observed that this type of model has the possibility to store a large number of spatio-temporal patterns [**Andreyev et al, 1996**].

Bibliography

- [1] IEEE Computer, Oct. 1996.
- [2] E. **Aarts** and J. Korst, Simulated Annealing and Boltzmann Machines, New York: John Wiley, 1989.
- [3] D.H. Ackley, G.E. **Hinton**, and T.J. Sejnowski, "A learning algorithm for Boltzmann machines", Cognitive Sci., vol. 9, pp. 147–169, 1985.
- [4] M. Adachi and K. Aihara, "Associative dynamics in a chaotic neural network", Neural Networks, vol. 10, no. 1, pp. 83–98, Jan. 1997.
- [5] L. Adleman, "Molecular computation of solutions to combinatorial problems", Science, no. 266, pp. 1021–1024, 1994.
- [6] I. Aleksander and H. Morton, An Introduction to Neural Computing, London: Chapman and Hall, 1990.
- [7] S. Amari, "A theory of adaptive pattern classifiers", IEEE *Trans. Electronic Computers*, vol. EC-16, pp. 299–307, 1967.
- [8] S. Amari, "Neural theory of association and concept-formation", Biol. Cybernet., vol. 26, pp. 175–185, 1977.
- [9] S. Amari, "A universal theorem on learning curves", Neural Networks, vol. 6, no. 2, pp. 161–166, 1993.
- [10] S. Amari, "Learning and statistical inference", in The Handbook of Brain Theory and Neural Networks (**M.A. Arbib**, ed.), Cambridge, MA: MIT Press, pp. 522–526, 1995.
- [11] D.J. **Amit**, H. Gutfreund, and H. Sompolinsky, "Spin-glass models of neural networks", Physical Review Letters A, vol. 32, pp. 1007–1018, 1985.
- [12] D.J. **Amit**, Modeling Brain Function: The World of Attractor Neural Networks, Cambridge: Cambridge University Press, 1989.
- [13] D.J. **Amit**, H. Gutfreund, and H. Sompolinsky, "Statistical mechanics of neural networks near saturation", Annals of Physics, vol. 173, pp. 30–67, 1987.

- [14] S. Anderson, J.W.L. **Merrill**, and R. Port, "Dynamic speech categorization with recurrent networks", in Proceedings of the 1988 Connectionist Models Summer School (**D.S.** Touretzky, G.E. **Hinton**, and T.J. Sejnowski, **eds.**), Morgan **Kaufmann**, San Mateo, CA, pp. **398–406**, 1989.
- [15] J. Anderson, J. Silverstein, S. Ritz, and R. Jones, "Distinctive features, categorical perception, and probability learning: Some applications of a neural model", *Psychological Review*, vol. 84, pp. **413–451**, 1977.
- [16] Y.V. Andreyev, Y.L. Belsky, A.S. Dmitriev, and D.A. Kuminov, "Information processing using dynamic chaos: Neural network implementation", *IEEE Trans. Neural Networks*, vol. 7, pp. 290–299, Mar. 1996.
- [17] **P.J.** Angeline, G.M. Saunders, and J.B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks", *IEEE Trans. Neural Networks*, vol. 5, pp. 54–65, Jan. 1994.
- [18] M. Anthony and S.B. **Holden**, "Quantifying generalization in linearly weighted neural networks", *Complex Systems*, vol. 8, pp. 91–144, 1994.
- [19] W. **Ashby**, *Design for a Brain*, New York: Wiley & Sons, 1952.
- [20] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima", *Neural Networks*, vol. 2, no. 1, pp. 53–58, 1989.
- [21] D.H. Ballard, "Modular learning in hierarchical neural networks", in *Computational Neuroscience* (**E.L.** Schwartz, **ed.**), Cambridge, MA: MIT Press, pp. 139–153, 1990.
- [22] M. **Banerjee**, S. Mitra, and S.K. Pal, "Knowledge-based fuzzy MLP with rough sets", in *IEEE International Conference on Neural Networks* (Houston, USA), June 9–12, 1997.
- [23] E. **Barnard** and D. Casasent, "Invariance and neural nets", *IEEE Trans. Neural Networks*, vol. 2, pp. 498–508, Sept. 1991.
- [24] J.A. **Barnden**, "Artificial intelligence and neural networks", in *The Handbook of Brain Theory and Neural Networks*, (**M.A.** **Arbib**, **ed.**), Cambridge, MA: MIT Press, pp. 98–102, 1995.
- [25] A.G. **Barto**, "Reinforcement learning and adaptive critic methods", in *Handbook of Intelligent Control* (**D.A.** White and D.A. Sofge, **eds.**), New York: Van Nostrand-Reinhold, pp. **469–491**, 1992.
- [26] A.G. **Barto** and P. **Anandan**, "Pattern recognizing stochastic learning automata", *IEEE Trans. Systems, Man and Cybernetics*, vol. 15, no. 3, pp. 360–375, 1985.

- [27] A.G. **Barto**, R. Sutton, and C. Anderson, "Neuron-like adaptive elements that can solve difficult learning control problems", *IEEE Trans. Systems, Man and Cybernetics*, vol. SMC-13, pp. 834–846, 1983.
- [28] R. **Battiti**, "First- and second-order methods for learning: Between steepest descent and Newton's method", *Neural Computation*, vol. 4, no. 2, pp. 141–166, 1992.
- [29] E. Baum and D. Haussler, "What size net gives valid generalization?", *Neural Computation*, vol. 1, no. 1, pp. 151–160, 1989.
- [30] W.G. **Baxt**, "Use of artificial neural network for data analysis in clinical decision-making: The diagnosis of acute coronary occlusion", *Neural Computation*, vol. 2, no. 4, pp. 480–489, 1990.
- [31] R.K. Belew, "When both individuals and populations search: Adding simple learning to genetic algorithm", in *Proceedings of third International Conference on GA* (George Mason University), pp. 34–41, June 1989.
- [32] J.C. Bezdek, "A review of probabilistic, fuzzy and neural models for pattern recognition", in *Fuzzy Logic and Neural Network Handbook* (C.H. Chen, ed.), U.S.A., McGraw-Hill, pp. 2.1–2.33, 1996.
- [33] J.C. Bezdek and S.K. Pal, *Fuzzy Models for Pattern Recognition*, Eds., New York: IEEE Press, 1992.
- [34] A.K. Bhattacharya and B. Roysam, "Joint solution of low, intermediate, and high level vision tasks by evolutionary optimization: Application to computer vision at low SNR", *IEEE Trans. Neural Networks*, vol. 5, pp. 83–95, Jan. 1994.
- [35] G.L. Bilbro, W.E. Snyder, S.J. Garnier, and J.W. Gault, "Mean field annealing: A formalism for constructing GNC-like algorithms", *IEEE Trans. Neural Networks*, vol. 3, no. 1, pp. 131–138, 1992.
- [36] S.A. Billings and G.L. Zheng, "Radial basis function network configuration using genetic algorithms", *Neural Networks*, vol. 8, no. 6, pp. 877–890, 1995.
- [37] K. Binder and D.W. Heerman, *Monte Carlo Simulation in Statistical Mechanics*, Berlin: Springer-Verlag, 1988.
- [38] F.C. Blondeau and G. Chauvet, "Stable, oscillatory and chaotic regimes in the dynamics of small neural networks with delay", *Neural Networks*, vol. 5, no. 5, pp. 735–744, 1992.
- [39] A.L. Blum and R. Rivest, "Training a 3-node neural network is NP-complete", *Neural Networks*, vol. 5, no. 1, pp. 117–128, 1992.

- [40] A. **Blumer**, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth, "Learnability and Vapnik-Chervonenkis dimension", *Journal of the American Computing Machinery*, vol. 36, no. 4, pp. 929–965, Oct. 1989.
- [41] S. **Bornholdt** and D. Graudenz, "General asymmetric neural networks and structure design by genetic algorithms", *Neural Networks*, vol. 5, no. 2, pp. 327–334, 1992.
- [42] N.K. Bose and F. Liang, *Neural Network Fundamentals with Graphs, Algorithms and Applications*, McGraw-Hill, Int. Editions, 1996.
- [43] D.G. Bounds, P.J. Lloyd, B. **Mathew**, and G. **Wadell**, "A multilayer perceptron network for the diagnosis of low back pain", in *Proceedings of IEEE International Conference on Neural Networks*, vol. II (San Diego, CA), pp. 481–489, IEEE, New York, 1988.
- [44] A.C. Bovik, M. Clark, and W.S. Geisler, "Multichannel texture analysis using localized spatial filters", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, pp. 55–73, Jan. 1990.
- [45] D.S. Broomhead and D. **Lowe**, "Multivariate functional interpolation and adaptive networks", *Complex Systems*, vol. 2, pp. 321–355, 1988.
- [46] G. **Burel**, "Blind separation of sources: A nonlinear neural algorithm", *Neural Networks*, vol. 5, no. 6, pp. 937–948, 1992.
- [47] C.J.C. Burges, "Handwritten digit string recognition", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 447–450, 1995.
- [48] G. Burke, "Neural networks: Brainy way to trade?", *Futures*, pp. 34–36, Aug. 1992.
- [49] E.R. Caianiello, "Outline of a theory of thought-processes and thinking machines", *J. Theor. Biology*, vol. 1, pp. 204–235, 1961.
- [50] J.F. **Cardoso**, "Source separation using higher order moments", *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 4, pp. 2109–2112, 1989.
- [51] S.H. Cameron, "An estimate of the complexity requisite in a universal decision network", *Wright Air Development Division, Report 60-600*, 1960.
- [52] G.A. Carpenter, "Neural network models for pattern recognition and associative memory", *Neural Networks*, vol. 2, pp. 243–257, 1989.
- [53] G.A. Carpenter and S. Grossberg, "ART2: Self-organization of stable category recognition codes for analog input patterns", *Applied Optics*, vol. 26, no. 23, pp. 4919–4930, 1987.

- [54] G.A. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organization neural network", *ZEEE Computer*, vol. 21, no. 3, pp. 77-88, 1988.
- [55] G.A. Carpenter and S. Grossberg, "**ART3**: Hierarchical search using chemical transmitters in **self-organising** pattern recognition architectures", *Neural Networks*, vol. 3, no. 2, pp. 129-152, 1990.
- [56] G.A. Carpenter and S. Grossberg, "Learning, categorization, rule formation, and prediction by fuzzy neural networks", in *Fuzzy Logic and Neural Network Handbook* (C.H. Chen, ed.), New York, **McGraw-Hill** Inc., pp. 1.3-1.45, 1996.
- [57] G.A. Carpenter, S. Grossberg, and J.H. Reynolds, "**ARTMAP**: Supervised real-time learning and classification of **non-stationary** data by a **self-organising** neural network", *Neural Networks*, vol. 4, no. 5, pp. 565-588, 1991a.
- [58] G.A. Carpenter, S. Grossberg, and D.B. Rosen, "**ART2-A**: An adaptive resonance algorithm for rapid category learning and recognition", *Neural Networks*, vol. 4, no. 4, pp. 493-504, 1991b.
- [59] G.A. Carpenter, S. Grossberg, and D.B. Rosen, "Fuzzy **ART**: Fast stable learning and categorization of analog patterns by an adaptive resonance system", *Neural Networks*, vol. 4, pp. 759-771, 1991c.
- [60] EL. Casselman, D.F. Freeman, D.A. Kerringan, S.E. Lane, N.H. Millstrom, and W.G. Nichols, Jr., "A neural network-based passive sonar detection and classification design with a low false alarm rate", in *ZEEE Conference on Neural Networks for Ocean Engineering* (Washington, DC), pp. 49-55, 1991.
- [61] C. Chandrasekhar, *Neural Network Models for Recognition of Stop-Consonant-Vowel (SCV) Segments in Continuous Speech*, PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India, Apr. 1996.
- [62] C. Chandrasekhar and B. Yegnanarayana, "Recognition of **stop-consonant-vowel (SCV)** segments in continuous speech using neural network models", *Journal of Institution of Electronics and Telecommunication Engineers (IETE)*, vol. 42, pp. 269-280, July-October 1996.
- [63] R. Chellappa, B.S. Manjunath, and T. Simchony, "Texture segmentation with neural networks", in *Neural Networks for Signal Processing* (B. Kosko, ed.), Englewood Cliffs, NJ: **Prentice-Hall**, 1992.
- [64] C.H. Chen, *Fuzzy Logic and Neural Network Handbook*, New York: **McGraw-Hill** Inc, 1996.

- [65] V. Cherkassky and N. Vassilas, "Performance of back propagation **networks** for associative database retrieval", in Proceedings of International Joint Conference on Neural Networks, Washington D.C., vol. I, pp. 77-84, 1989.
- [66] J. Cheung and M. Omidvar, "Mathematical analysis of learning behaviour of neuronal models", in Proceedings of the 1987 IEEE **Conference** on Neural Information Processing Systems— Natural and Synthetic (D.Z. Anderson, **ed.**), New York, American Institute of Physics, pp. **165–173**, 1988.
- [67] J.J. Choi, R.J. **Arabshahi**, R.J. Marks, and T.P. Caudell, "Fuzzy parameter adaptation in neural systems", in Proceedings of IEEE International Joint Conference on Neural Networks, vol. 1 (Baltimore, **MD**), pp. 232–238, 1992.
- [68] F.L. Chung and T. Lee, "Fuzzy competitive learning", Neural Networks, vol. 7, no. 3, pp. 539–552, 1994.
- [69] M.A. Cohen, H. **Franco**, N. Morgan, D. Rumelhart, and V. Abrash, "Context-dependent multiple distribution phonetic modeling with **MLPs**", in Advances in Neural Information Processing Systems (**S.J. Hanson**, J.D. **Cowan**, and C.L. Giles, eds.), (**San Mateo, CA**), Morgan Kaufmann, pp. 649–657, 1993.
- [70] M.A. Cohen and S. Grossberg, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks", IEEE **Trans.** Systems, Man and Cybernetics, vol. 13, no. 5, pp. 815–826, 1983.
- [71] R. Cole, M. Fanty, Y. Muthuswamy, and M. **Gopalakrishna**, "Speaker-independent recognition of spoken English letters", in Proceedings of the International Joint Conference on Neural Networks (**San Diego, CA**), 1992.
- [72] P. Comon, "Independent component analysis: A new concept?", Signal Processing, vol. 36, no. 3, pp. 287-314, 1994.
- [73] F.S. Cooper, "Acoustics in human communication: Evolving ideas about the nature of speech", **J. Acoust. Soc. Amer.**, vol. 68, pp. 18–21, 1980.
- [74] C. Cortes and J.A. Hertz, "A network system for image segmentation", in International Joint Conference on Neural Networks, vol. I (Washington), IEEE, New York, pp. 121–127, 1989.
- [75] J.D. **Cowan** and D.H. Sharp, "Neural nets", Quarterly Reviews of Biophysics, vol. 21, pp. **365–427**, 1988.
- [76] J.P. Crutchfield, J.D. Farmer, N.H. Packard, and R.S. Shaw, "Chaos", Scientific American, vol. 225, no. 6, pp. 38–49, 1986.

- [77] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.
- [78] J.G. Daugman, "Uncertainty relation for resolution in space, spatial-frequency, and orientation optimized by two-dimensional visual cortical filters", *J. Opt. Soc. Amer.*, vol. 2, pp. 1160–1169, July 1985.
- [79] J.G. Daugman, "Complete discrete 2-D **Gabor** transforms by neural network for image analysis and compression", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1169–1179, July 1988.
- [80] S.B. Davis and P. Mermelstein, "Comparison of parametric representations of monosyllabic word recognition in continuously spoken sentences", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 28, pp. 357–366, Aug. 1980.
- [81] "DEctalk DTC01", Programmer's reference manual, Digital Equipment Corporation, 1983.
- [82] P.A. Devijver and J. Kittler, *Pattern Recognition, A Statistical Approach*, NJ: Prentice Hall Inc., 1982.
- [83] P.A. Devijver and J. Kittler, "Feature extraction based on the **Karhunen-Loeve** expansion", in *Pattern Recognition—A Statistical Approach*, NJ: Prentice-Hall, Inc., ch. 9, pp. 301–341, 1982.
- [84] K.I. Diamantaras and S.Y. Kung, "Cross-correlation neural network models", *IEEE Trans. Signal Processing*, vol. 42, pp. 3218–3223, Nov. 1994.
- [85] K.I. Diamantaras and S.Y. Kung, *Principal Component Neural Networks: Theory and Applications*, John Wiley & Sons, 1996.
- [86] M. Dorigo, V. Maniezzo, and A. Coloni, "Ant system: Optimization by a colony of cooperating agents", *IEEE Trans. Systems, Man and Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.
- [87] H.L. Dreyfus, *What Computers Can't Do: A Critique of Artificial Reason*, New York: Harper and Row, 1972.
- [88] H.L. Dreyfus, *What Computers Still Can't Do*, Cambridge, MA: MIT Press, 1992.
- [89] S.E. Dreyfus, "Expert system: How far can they go?", *Artificial Intelligence Magazine*, vol. 10, no. 2, pp. 65–76, 1989.
- [90] R.C. Dubes, "How many clusters are best? An **experiment**", *Pattern Recognition*, vol. 20, no. 6, pp. 645–663, 1987.
- [91] D. Dubois and H. Prade, "Putting rough sets and fuzzy sets together", in *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Set Theory* (R. Slowinski, ed.), Dordrecht: **Kluwer** Academic Publishers, 1992.

- [92] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, New York: John Wiley and Sons, 1973.
- [93] R.O. Duda and E.H. Shortliffe, "Expert systems research", *Science*, vol. 220, pp. 261–268, 1983.
- [94] R. Durbin and D. Willshaw, "An analogue approach to the traveling salesman problem using an elastic net method", *Nature*, vol. 326, pp. 689–691, April 1987.
- [95] J. Eisenberg, W.J. Freeman, and B. Burke, "Hardware architecture of a neural network model simulating pattern recognition by the olfactory bulb", *Neural Networks*, vol. 2, no. 4, pp. 315–325, 1989.
- [96] J.L. Elman, "Finding structure in time", *Cognitive Sci.*, vol. 14, pp. 179–211, 1990.
- [97] S.E. Fahlman, "Fast learning variations on backpropagation: An empirical study", in *Proc. 1988 Connectionist Models Summer School* (D.S. Touretzky, G.E. Hinton, and T.J. Sejnowski, eds.), San Mateo, CA: Morgan Kaufmann, pp. 38–51, 1989.
- [98] E.A. Feigenbaum, P. McCorduck, and H.P. Nii, *The Rise of the Expert Company*, New York: Times Books, 1988.
- [99] J.L. Flanagan, *Speech Analysis, Synthesis and Perception*, 2nd ed., New York: Springer-Verlag, 1972.
- [100] R. Fletcher and C.M. Reeves, "Function minimization by conjugate gradients", *IEEE Computer*, vol. 7, pp. 149–154, 1964.
- [101] D.B. Fogel, *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*, Needham, MA: Ginn Press, 1991.
- [102] D.B. Fogel, "An introduction to simulated evolutionary optimization", *IEEE Trans. Neural Networks*, vol. 5, pp. 3–14, Jan. 1994.
- [103] D.B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Learning*, Piscataway, IEEE Press, 1995.
- [104] L.J. Fogel, A.J. Owens, and M.J. Walsh, *Artificial Intelligence through Simulated Evolution*, New York: John Wiley, 1966.
- [105] P. Foldiak, "Adaptive network for optimal linear feature extraction", *IEEE Proceedings of the International Joint Conference on Neural Networks*, vol. 1, pp. 401–405, 1989.
- [106] J.A. Freeman and D.M. Skapura, *Neural Networks: Algorithms, Applications and Programming Techniques*, Reading, MA: Addison-Wesley, 1991.

- [107] K. Fukushima, "Cognitron: A self-organizing multilayered neural network", *Biol. Cybernet.*, vol. 20, pp. 121–136, 1975.
- [108] K. Fukushima, "**Neocognitron**: A self-organizing neural network for a mechanism of pattern recognition unaffected by a shift in position", *Biol. Cybernet.*, vol. 36, pp. 193–202, 1980.
- [109] K. Fukushima, "A neural network for visual pattern recognition", *IEEE Computer*, vol. 21, pp. **65–75**, March 1988.
- [110] K. Fukushima and N. Wake, "Handwritten alphanumeric character recognition by the **neocognitron**", *IEEE Trans. Neural Networks*, vol. 2, pp. 355–365, May 1991.
- [111] K. Funahashi, "On the approximate realization of continuous mappings by neural networks", *Neural Networks*, vol. 2, no. 3, pp. 183–192, 1989.
- [112] D. Gabor, "Communication theory and cybernetics", *IRE Trans. on Circuit Theory*, vol. CT-1, pp. 19–31, 1954.
- [113] S.I. Gallant, "Connectionist expert systems", *J. Assoc. Comput. Machinery*, vol. 31, no. 2, pp. 152–169, 1988.
- [114] S.I. Gallant, *Neural Network Learning and Expert Systems*, Cambridge, MA: MIT Press, 1993.
- [115] S.I. Gallant, "Expert systems and decision systems using neural networks", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. **377–380**, 1995.
- [116] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, 1984.
- [117] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the **bias/variance** dilemma", *Neural Computation*, vol. 4, pp. 1–58, 1992.
- [118] R.J. Glauber, "Time-dependent statistics of the Ising model", *J. Math. Phys.*, vol. 4, pp. 294–307, 1963.
- [119] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading MA: Addison Wesley, 1989.
- [120] M. Gori and A. Tesi, "On the problem of local minima in backpropagation", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, pp. 76–86, 1992.
- [121] R.M. Gray, "Vector Quantization", *IEEE ASSP Magazine*, vol. 1, pp. 4–29, Apr. 1984.
- [122] R.M. Gray, *Entropy and Information Theory*, New York: Springer-Verlag, 1990.

- [123] M. Greenberger, *Computers and the World of the Future*, Cambridge: MIT Press, 1962.
- [124] H. Greenspan, R. Goodman, and R. Chellappa, "Texture analysis via unsupervised and supervised learning", in *Proceedings of the International Joint Conference on Neural Networks*, vol. 1 (Piscataway, NJ), IEEE, pp. 639–644, 1991.
- [125] S. Grossberg, "Some networks that can learn, remember, and reproduce any number of complicated space-time patterns", *J. Math. Mech.*, vol. I, no. 19, pp. 53–91, 1969.
- [126] S. Grossberg, "Adaptive pattern classification and universal recoding-I. Parallel development and coding of neural detectors", *Biol. Cybernet.*, vol. 23, pp. 121–134, 1976a.
- [127] S. Grossberg, "Adaptive pattern classification and universal recoding-II. Feedback, oscillation, olfaction, and illusions", *Biol. Cybernet.*, vol. 23, pp. 187–207, 1976b.
- [128] S. Grossberg, *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognitron and Motor Control*, Boston, MA: Reidel, 1982.
- [129] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures", *Neural Networks*, vol. 1, pp. 17–61, 1988.
- [130] S. Grossberg, "Are there universal principles of brain computation?", in *International Conference on Neural Networks*, (Washington, DC), pp. 49–55, IEEE, 1996.
- [131] I. Guyon, "Applications of neural networks to character recognition", *Int. J. Pattern Recognition and Artificial Intelligence*, vol. 5, pp. 353–382, 1991.
- [132] M. Hagiwara, "Multidimensional associative memory", in *Proceedings of the International Joint Conference on Neural Networks*, vol. I (Washington, DC), pp. 3–6, 1990.
- [133] H. Haken, *Synergetic Computers and Cognition*, Berlin and New York: Springer, 1991.
- [134] H. Haken, "Cooperative phenomena", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 261–266, 1995.
- [135] J.B. Hampshire and B. Pearlmutter, "Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function", in *Connectionist models summer school* (D.S. Touretzky, J.L. Elman, T.J. Sejnowski, and G.E. Hinton, eds.), San Mateo, CA: Morgan Kaufmann, 1990.
- [136] J. Harrington, "Acoustic cues for automatic recognition of English consonants", in *Aspects of Speech Technology* (M. Jack and J. Laver, eds.), Edinburgh: Edinburgh University Press, pp. 69–143, 1988.

- [137] E. **Harth**, 'Order and chaos in neural system: An approach to the dynamics of higher brain function', *IEEE Trans. Systems, Man and Cybernetics*, vol. 13, pp. 782–798, **Sept.-Oct.**, 1983.
- [138] M.H. Hassoun, *Fundamentals of Artificial Neural Networks*, Cambridge, MA: MIT Press, 1995.
- [139] D. Haussler, "Decision theoretic generalizations of the PAC model for neural net and other learning applications", *Information and Computation*, vol. 100, pp. 78–150, 1992.
- [140] D. Haussler, M. **Kearns**, and R. Schapire, "Bounds on the sample complexity using information theory", *Machine Learning*, vol. 14, pp. 83–113, 1994.
- [141] Y. **Hayashi**, 'Oscillatory neural network and learning of continuously transformed patterns', *Neural Networks*, vol. 7, no. 2, pp. 219–232, 1994.
- [142] S. **Haykin**, *Adaptive Filter Theory*, 2nd ed., Englewood Cliffs, NJ: Prentice-Hall, 1991.
- [143] S. **Haykin**, *Neural Networks: A Comprehensive Foundation*, New York: Macmillan College Publishing Company Inc., 1994.
- [144] D.O. Hebb, *The Organization of Behaviour: A Neuropsychological Theory*, New York: Wiley, 1949.
- [145] R. Hecht-Nielsen, "Counterpropagation networks", *Applied Optics*, vol. 26, pp. 4979–4984, 1987.
- [146] R. Hecht-Nielsen, "Applications of counterpropagation networks", *Neural Networks*, vol. 1, pp. 131–139, 1988.
- [147] R. Hecht-Nielsen, *Neuro Computing*, Reading, MA: **Addison-Wesley**, 1990.
- [148] F. Hergert, W. Finnoff, and H.G. Zimmermann, "A comparison of weight elimination methods for reducing complexity in neural networks", in *Proceedings of the International Joint Conference on Neural Networks*, vol. III (Baltimore), IEEE, New York, pp. **980–987**, 1992.
- [149] J.A. Hertz, A. Krogh, and G.I. Thorbergsson, "Phase transitions in simple learning", *J. Phys. A*, vol. 22, pp. 2133–2150, 1989.
- [150] J.A. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, New York: Addison-Wesley, 1991.
- [151] J.A. Hertz, "Computing **with** attractors", in *The Handbook of Brain Theory and Neural Networks* (**M.A. Arbib, ed.**), Cambridge, MA: MIT Press, pp. 230–234, 1995.

- [152] G.E. **Hinton** and T.J. Sejnowski, "Learning and relearning in Boltzmann machines", in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. I (D.E. Rumelhart, J.L. **McClelland**, and the PDP Research Group, eds.), Cambridge, MA, MIT Press, pp. 282-317, 1986.
- [153] G.E. **Hinton**, "Deterministic Boltzmann learning performs steepest descent in weight space", *Neural Computation*, vol. 1, pp. 143-150, 1989.
- [154] K. **Hirota** and W. Pedrycz, "ORAND neurons in modeling fuzzy set connectives", *IEEE Trans. Neural Networks*, vol. 2, pp. 151-161, May 1994.
- [155] A.L. Hodgkin and A.F. **Huxley**, "A quantitative description of membrane current and its application to conduction and excitation in nerve", *J. Physiology*, vol. 117, pp. 500-544, 1952.
- [156] S.B. **Holden**, *On the Theory of Generalization and Self-Structuring in Linearly Weighted Connectionist Networks*, PhD thesis, Corpus Christi College, Cambridge University Engineering Department, Trumpington Street, Cambridge CB2 1PZ, Jan. 1994.
- [157] S.B. **Holden** and M. Niranjan, "On the practical applicability of VC dimension bounds", Tech. Rep. **CUED/F-INFENG/TR.155**, University of Cambridge, Cambridge University Engineering Department, Trumpington Street, Cambridge CB2 1PZ, Oct. 1994.
- [158] S.B. **Holden** and P.J.W. Rayner, "Generalization and PAC learning: Some new results for the class of generalized single layer networks", *IEEE Trans. Neural Networks*, vol. 6, pp. 368-380, Mar. 1995.
- [159] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
- [160] J.H. Holland, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule based systems", in *Machine Learning: An Artificial Intelligence Approach*, (R. Michalski, J. Carbonell, and T. Michell, eds.), vol. 2 (San Mateo, CA), Morgan Kaufmann, pp. 593-623, 1986.
- [161] L. Holmstrom and P. Koistinen, "Using additive noise in back propagation training", *IEEE Trans. Neural Networks*, vol. 3, pp. 24-38, Jan. 1993.
- [162] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational capabilities", in *Proceedings of the National Academy of Sciences (USA)*, vol. 79, pp. 2554-2558, Nov. 1982.

- [163] J.J. Hopfield, "Neurons with graded responses have collective computational properties like those of two-state neurons", in Proceedings of the National Academy of Sciences (*USA*), vol. 81, pp. 3088-3092, 1984.
- [164] J.J. Hopfield and D.W. Tank, "Neural computation of decisions in optimization problems", *Biol. Cybernet.*, vol. 52, pp. 141-152, 1985.
- [165] H. Hotelling, "Analysis of a complex of statistical variables into principal components", *J. Educational Psychology*, vol. 24, pp. 417-441, 498-520, 1933.
- [166] C.C. Hsu, D. Gobovic, M.E. Zaghloul, and H.H. Szu, "Chaotic neuron models and their VLSI circuit implementation", *IEEE Trans. Neural Networks*, vol. 7, pp. 1339-1350, Nov. 1996.
- [167] M. Hu, "Visual pattern recognition by moment invariants", *IRE Transactions on Information Theory*, vol. IT-8, pp. 179-187, Feb. 1962.
- [168] Z. Huang and A. Kuh, "A combined self-organizing feature map and multilayer perceptron for isolated word recognition", *IEEE Trans. Signal Processing*, vol. 40, pp. 2651-2657, Nov. 1992.
- [169] W.Y. Huang and R.P. Lippmann, "Neural nets and traditional classifiers", in *Neural Information Processing Systems* (D.Z. Anderson, ed.), (Denver, 1987), American Institute of Physics, New York, pp. 387-396, 1988.
- [170] D.H. Hubel and T.N. Wiesel, "Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex", *J. Physiology*, vol. 160, pp. 106-154, 1962.
- [171] E.R. Hunt, "Stabilizing high period in a chaotic systems: The diode resonator", *Physical Review Letters*, vol. 67, pp. 1953-1955, 1991.
- [172] J.E. Hunt and D.E. Cooke, "Learning using an artificial immune system", *J. Networks and Computer Applications*, vol. 19, pp. 189-212, 1996.
- [173] K.J. Hunt, D. Sbarbaro, R. Zbikowski, and P.J. Gawthrop, "Neural networks for control systems—A survey", *Automatica*, vol. 28, pp. 1083-1112, 1992.
- [174] D.R. Hush and B.G. Horne, "Progress in supervised neural networks: Whats new since Lippmann", *IEEE Signal Processing Magazine*, vol. 10, pp. 8-39, Jan. 1993.
- [175] D.R. Hush, J.M. Salas, and B. Horne, "Error surfaces for multilayer perceptrons", in *International Joint Conference on Neural Networks*, vol. 1 (Seattle), IEEE, New York, pp. 759-764, 1991.

- [176] J.N. Hwang, "Textured images: Modeling and segmentation*", in *The Handbook of Brain Theory and Neural Networks*, (M.A. Arbib, ed.) Cambridge, MA: MIT Press, pp. 971–976, 1995.
- [177] H. Ishibuchi, R. Fujioka, and H. Tanaka, "Neural networks that learn from fuzzy if-then rules", *IEEE Trans. Fuzzy Systems*, vol. 1, pp. 85–97, May 1993.
- [178] R.A. Jacobs, "Increased rates of convergence through learning rate adaptation*", *Neural Networks*, vol. 1, no. 4, pp. 295–407, 1988.
- [179] R.A. Jacobs and M.I. Jordan, "A competitive modular connectionist architecture*", in *Advances in Neural Information Processing Systems*, vol. 3 (R.P. Lippmann, J.E. Moody, and D.J. Touretzky, eds.), San Mateo, CA: Morgan Kaufmann, pp. 767–773, 1991.
- [180] J.S.N. Jean and J. Wang, "Weight smoothing to improve network generalization", *IEEE Trans. Neural Networks*, vol. 5, pp. 752–763, Sept. 1994.
- [181] S. Jockusch and H. Ritter, "Self-organizing maps: Local competition and evolutionary optimization*", *Neural Networks*, vol. 7, no. 8, pp. 1229–1240, 1994.
- [182] E.M. Johansson, F.U. Dowla, and D.M. Goodman, "Back-propagation learning for multilayer feedforward neural networks using the conjugate gradient method*", Report UCRL-JC-104850, Lawrence Livermore National Laboratory, Livermore, CA, 1990.
- [183] I.T. Jolliffe, *Principal Component Analysis*, New York: Springer-Verlag, 1986.
- [184] M.I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine*", in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (Amherst, MA), Erlbaum, Hillsdale, pp. 531–546, 1986.
- [185] M.I. Jordan, "Serial order: A parallel, distributed processing approach*", in *Advances in Connectionist theory: Speech* (J.L. Elman and D.E. Rumelhart, eds.), (New Jersey), Erlbaum, Hillsdale, 1989.
- [186] R.L. Jordan, B.L. Huneycutt, and M. Werner, "The SIR-C/X-SAR synthetic aperture radar system*", *Proc. IEEE*, vol. 79, pp. 827–838, June 1991.
- [187] J.S. Judd, *Neural Network Design and the Complexity of Learning*, Cambridge, MA: MIT Press, 1990.

- [188] C. Jutten and J. Herault, "Independent components analysis (INCA) versus principal components analysis", in *Signal Processing IV: Theories and applications* (J.L. Lacoume, A. Chehikian, N. Martin, and J. Malbos, eds.), EURASIP, Elsevier Science Publishers B.V. (North Holland), pp. 643–646, 1988.
- [189] L. Kanal, "Patterns in pattern recognition: 1968–1974", *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 697–722, Nov. 1974.
- [190] J. Karhunen and J. Joutsensalo, "Robust MUSIC based on direct signal subspace estimation", *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 5, pp. 3357–3360, 1991.
- [191] J. Karhunen and J. Joutsensalo, "Representation and separation of signals using nonlinear PCA type learning", *Neural Networks*, vol. 7, no. 1, pp. 113–127, 1994.
- [192] S.M. Kay, *Modern Spectral Estimation*, New Jersey: Prentice-Hall, 1988.
- [193] J.D. Keeler, D.E. Rumelhart, and W.K. Leow, "Integrated segmentation and recognition of hand-printed numerals", in *Advances in Neural Information Processing Systems* (R.P. Lippmann, J.E. Moody, and D.S. Touretzky, eds.), vol. 3 (Denver 90), Morgan Kaufmann, San Mateo, CA, pp. 557–563, 1991.
- [194] J.M. Keller and D.J. Hunt, "Incorporating fuzzy membership functions into the perceptron algorithms", *IEEE Trans. Pattern Analysis and Machine Intelligence*, pp. 693–699, July/August 1985.
- [195] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing", *Science*, vol. 220, pp. 671–680, 1983.
- [196] D.H. Klatt, "Software for a cascade/parallel formant synthesizer", *J. Acoust. Soc. Amer.*, vol. 67, pp. 971–995, Mar. 1980.
- [197] G.J. Klir and T.A. Folger, *Fuzzy Sets, Uncertainty and Information*, Englewood Cliffs, NJ: Prentice-Hall, 1993.
- [198] G.J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic—Theory and Applications*, Englewood Cliffs, NJ: Prentice-Hall, 1995.
- [199] A.H. Klopff, "A drive-reinforcement model of single neuron function: An alternative to the Hebbian neuronal model", in *Proceedings of the American Institute of Physics: Neural Networks Comput.* (Snowbird, UT), pp. 265–270, 1986.
- [200] C. Koch, J. Marroquin, and A. Yuille, "Analog 'neuronal' networks in early vision", in *Proceedings of the National Academy of Sciences, USA*, vol. 83, pp. 4263–4267, 1986.

- [201] T. Kohonen, "A class of randomly organized associative memories", *Acta Polytechnic Scandanavica*, vol. E1 25, 1971.
- [202] T. Kohonen, "Analysis of simple self-organizing process", *Biol. Cybernet.*, vol. 44, pp. 135–140, 1982a.
- [203] T. Kohonen, "Self-organized formation of topologically correct feature maps", *Biol. Cybernet.*, vol. 43, pp. 59–69, 1982b.
- [204] T. Kohonen, "Dynamically expanding context, with application to, the correction of symbol strings in the recognition of continuous speech", in *Proceedings of the 8th International Conference on Pattern Recognition*, (Washington, DC), IEEE Computer Society Press, pp. 1148–1151, 1986.
- [205] T. Kohonen, "The 'neural' phonetic typewriter", *IEEE Computer*, vol. 27, pp. 11–22, Mar. 1988.
- [206] T. Kohonen, *Self-organization and Associative Memory*, 3rd ed., Berlin: Springer-Verlag, 1989.
- [207] T. Kohonen, "Improved versions of learning vector quantization", in *Proceedings of the International Joint Conference on Neural Networks (San Diego)*, IEEE Neural Networks Council, pp. 545–550, 1990a.
- [208] T. Kohonen, "The self-organizing map", *Proc. IEEE*, vol. 78, pp. 1464–1480, 1990b.
- [209] T. Kohonen, "New developments of learning vector quantization and the self-organizing map", in *Symposium on Neural Networks: Alliances and Perspectives in Senri 1992 (SYNAPSE '92)*, (Osaka, Japan), 1992.
- [210] T. Kohonen, "Learning vector quantization", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 537–540, 1995.
- [211] T. Kohonen, *Self-Organizing Maps*, 2nd ed., Berlin: Springer-Verlag, 1997.
- [212] B. Kosko, "Adaptive bidirectional associative memories", *Applied Optics*, vol. 26, no. 23, pp. 4947–4960, 1987.
- [213] B. Kosko, "Bidirectional associative memories", *IEEE Trans. Systems, Man and Cybernetics*, vol. 18, no. 1, pp. 49–60, 1988.
- [214] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Englewood Cliffs, NJ: Prentice-Hall, 1992.
- [215] J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, Massachusetts: MIT Press, 1992.

- [216] A.H. **Kramer** and A. Sangiovanni-Vincentelli, "Efficient parallel learning algorithms for neural networks", in *Advances in Neural Information Processing Systems* (D.S. Touretzky, ed.), vol. 1 (San Mateo, CA), pp. 40–48, 1989.
- [217] S. Kullback, *Information Theory and Statistics*, New York: Dover, 1968.
- [218] S.Y. Kung, *Digital Neural Networks*, New Jersey: Prentice-Hall, 1993.
- [219] S.Y. Kung and K.I. Diamantaras, "A neural network learning algorithm for adaptive principal component extraction (APEX)", in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, Albuquerque, NM, USA, vol. 3, pp. 861–864, Apr. 1990.
- [220] S.Y. Kung and K.I. Diamantaras, "Adaptive principal component extraction (APEX) and applications", *IEEE Trans. Signal Processing*, vol. 42, pp. 1202–1217, May 1994.
- [221] A. Lapedes and R. Farber, "How neural networks works", in *Neural Information Processing Systems* (D.Z. Anderson, ed.), (Denver), American Institute of Physics, New York, pp. 442–456, 1988.
- [222] Y. LeCun, "Learning process in an asymmetric threshold network", in *Disordered systems and biological organization*, (E. Bienenstock, F.F. **Soulie**, and G. Weisbuch, eds.), Berlin, Heidelberg: Springer, pp. 233–240, 1986.
- [223] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. **Hubbard**, and L.D. **Jackel**, "Handwritten digit recognition with a back-propagation network", in *Advances in Neural Information Processing Systems* (D.S. Touretzky, ed.), vol. 2 (Denver), Morgan Kaufmann, San Mateo, CA, pp. 396–404, 1990.
- [224] Y. LeCun and Y. Bengio, "Convolution networks for images, speech, and time series", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 255–258, 1995a.
- [225] Y. LeCun and Y. Bengio, "Pattern recognition", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 711–715, 1995b.
- [226] Y. Lee, S.H. Oh, and M.W. Kim, "The effect of initial weights on premature saturation in backpropagation learning", in *International Joint Conference on Neural Networks*, vol. 1 (Seattle), IEEE, New York, pp. 765–770, 1991.
- [227] H.B. Lee, "Eigenvalues and eigenvectors of covariance matrices for signals closely spaced in frequency", *IEEE Trans. Signal Processing*, vol. 4, pp. 2518–2535, Oct. 1992.

- [228] S.J. Leon, *Linear Algebra with Applications*, New York: Macmillan Publishing Company, 1990.
- [229] C.T. Lin and C.S.G. Lee, *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*, New Jersey: Prentice-Hall Inc., 1996.
- [230] C.T. Lin and Y.C. Lu, "A neural fuzzy system with linguistic teaching signals", *IEEE Trans. Fuzzy Systems*, vol. 3, pp. 169–189, May 1995.
- [231] R. Linsker, "From basic network principles to neural architecture", in *Proceedings of the National Academy of Sciences*, vol. 83 (USA), pp. 7508–7512, 8390–8394, 877943783, 1986.
- [232] R. Linsker, "Self-organization in a perceptual network", *IEEE Computer*, vol. 21, pp. 105–117, Mar. 1988.
- [233] R.P. Lippmann, "An introduction to computing with neural nets", *IEEE ASSP Magazine*, vol. 4, pp. 4–22, Apr. 1987.
- [234] R.P. Lippmann, "Review of neural networks for speech recognition", *Neural Computation*, vol. 1, no. 1, pp. 1–38, 1989.
- [235] W. Little, "The existence of persistent states in the brain", *Math. Biosciences*, vol. 19, pp. 101–120, 1974.
- [236] W. Little and G. Shaw, "Analytical study of the memory storage capacity of a neural network", *Math. Biosciences*, vol. 39, pp. 281–290, 1978.
- [237] Y. Liu, "Unbiased estimate of generalization error and model selection in neural network", *Neural Networks*, vol. 8, no. 2, pp. 215–219, 1995.
- [238] D. Lowe, "Radial basis function networks", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 779–782, 1995.
- [239] D.J.C. MacKay, "Bayesian methods for supervised neural networks", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 144–149, 1995.
- [240] B. Mandelbrot, *The Fractal Geometry of Nature*, Sanfrancisco: Freeman, 1983.
- [241] J. Mantas, "Methodologies in pattern recognition and image analysis: A brief survey", *Pattern Recognition*, vol. 20, no. 1, pp. 1–6, 1987.
- [242] A. Marcus and A.V. Dam, "User-interface developments for the nineties", *IEEE Trans. Comput.*, vol. 24, pp. 49–57, 1991.

- [243] S.L. Marple, *Digital Spectral Analysis with Applications*, Prentice-Hall, 1987.
- [244] K. Matsuoka and M. Kawamoto, "A neural network that self-organizes to perform three operations related to principal component analysis", *Neural Networks*, vol. 7, no. 5, pp. 753–765, 1994.
- [245] J.L. McClelland, "Retrieving general and specific information from stored knowledge of specifics", in *Proceedings of the Third Annual Meeting of the Cognitive Science Society*, pp. 170–172, 1981.
- [246] J.L. McClelland and D.E. Rumelhart, *Parallel Distributed Processing*, vol. 2, Cambridge, MA: MIT Press, 1986.
- [247] J.L. McClelland and D.E. **Rumelhart**, *Explorations in Parallel Distributed Processing*, Cambridge MA: MIT Press, 1988.
- [248] W.S. **McCulloch** and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.
- [249] R. **McGough**, "Fidelity's Bradford Lewis takes aim at indexes with his 'neural **network**' computer program", *The Wall Street Journal*, Oct. 1992.
- [250] C. Mead, "A sensitive electronic photoreceptor", in *1985 Chapel Hill Conference on Very Large Scale Integration*, pp. 463–471, Mar. 1985.
- [251] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, "**Equation** of state calculations by fast computing **machines**", *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.
- [252] P.K. Mhaswade, "A neural network approach for information retrieval from spelled names", Master's thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, Jan. 1997.
- [253] G.F. Miller, P.M. Todd, and S.U. Hegde, "Designing neural networks using genetic algorithms", in *Proceedings of the Third International Conference on Genetic Algorithms and their Applications* (J.D. Schaffer, ed.), San Mateo, CA: Morgan Kaufmann, pp. 379–384, 1989.
- [254] A.A. Minai and R.J. Williams, "**Backpropagation** heuristics: A study of the extended delta-bar-delta algorithm", in *International Joint Conference on Neural Networks*, vol. 1 (**San Diego, CA**), pp. 595–600, 1990.
- [255] M.L. Minsky, *Theory of neural-analog reinforcement systems and its application to the brain-model problem*, **PhD** thesis, Princeton University, Princeton, **NJ**, 1954.

- [256] M. Minsky, "Steps toward artificial intelligence", in Proceedings of the IRE, vol. 49, pp. 5–30, 1961.
- [257] M. Minsky and O. Selfridge, "Learning in random nets*", in Information Theory: Fourth London Symposium, (C. Cherry, ed.), (London), Butterworths, 1961.
- [258] M.L. Minsky and S.A. **Papert**, Perceptrons, Cambridge, MA: MIT Press, 1969.
- [259] M.L. Minsky and S.A. **Papert**, Perceptrons, expanded ed., Cambridge, MA: MIT Press, 1990.
- [260] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units*", Neural Computation, vol. 1, pp. 281–294, 1989.
- [261] D.P. Morgan and C.L. Scofield, Neural Networks and Speech Processing, Boston: **Kluwer** Academic Publishers, 1991.
- [262] M.C. **Mozer**, "A focused back-propagation algorithm for temporal pattern recognition", Complex Systems, vol. 3, pp. 349–381, 1989.
- [263] B. Muller and J. Reinhardt, Neural Networks, New York: Springer-Verlag, 1990.
- [264] B. Muller and J. Reinhardt, Neural Networks: An Introduction, Physics of Neural Networks, New York: Springer-Verlag, 1991.
- [265] K. Murakami and T. Aibara, "An improvement on the **Moore-Penrose** generalized inverse associative memory", IEEE *Trans. Systems, Man and Cybernetics*, vol. SMC 17, pp. 699–706, July-August 1987.
- [266] S. Muroga, Threshold Logic and its Applications, New York: Wiley Interscience, 1971.
- [267] M.T. Musavi, K.H. Chan, D.M. Hummels, and K. Kalantri, "On the generalization ability of neural network classifiers", IEEE *Trans. Pattern Analysis and Machine Intelligence*, vol. 16, pp. 659–663, June 1994.
- [268] K.S. Narendra and S. Mukhopadhyay, "Intelligent control using neural networks", IEEE Control Systems Magazine, vol. 12, no. 2, pp. 11–18, 1992.
- [269] K.S. Narendra and K. **Parthasarathy**, "Identification and control of dynamical systems using neural networks", IEEE *Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.
- [270] M. Narendranath, "Transformation of vocal **tract** characteristics for voice conversion using artificial neural networks", Master's thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, Nov. 1995.

- [271] M. Narendranath, H.A. Murthy, S. Rajendran, and B. Yegnanarayana, "Transformation of formants for voice conversion using artificial neural networks", *Speech Communication*, vol. 16, pp. 206–216, Feb. 1995.
- [272] N.M. Nasrabadi and R.A. King, "Image coding using Vector Quantization: A Review", *ZEEE Trans. Communications*, vol. 36, pp. 957–971, 1988.
- [273] A. Neeharika, "Generalization capability of feedforward neural networks for pattern recognition tasks", Master's thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, 1996.
- [274] D.H. Nguyen and B. Widrow, "Neural networks for self-learning control systems", *ZEEE Control Systems Magazine*, pp. 18–23, Apr. 1990.
- [275] N. Nilsson, *Learning Machines*, New York: **McGraw-Hill**, 1965.
- [276] S.J. Nowlan, "Maximum likelihood competitive learning", in *Advances in Neural Information Processing Systems*, vol. 2, (D.S. Touretzky, ed.) (Denver), Morgan Kaufmann, San Mateo, CA, pp. 574–582, 1990.
- [277] E. Oja, "A simplified neuron model as a principal component analyzer", *J. Math. Biology*, vol. 15, pp. 267–273, 1902.
- [278] E. Oja, "Neural networks, principal components, and subspaces", *Znt. J. Neural Systems*, vol. 1, no. 1, pp. 61–68, 1989.
- [279] E. Ott, C. Grebogi, and J.A. Yorke, "Controlling chaos", *Physical Review Letters*, vol. 64, pp. 1196–1199, 1990.
- [280] S.K. Pal and D.D. Majumder, *Fuzzy Mathematical Approach to Pattern Recognition*, New York: Wiley (**Halsted Press**), 1986.
- [281] S.K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification", *ZEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992.
- [282] F. Palmieri, J. Zhu, and C. Chang, "Anti-Hebbian learning in topologically constrained linear networks: A tutorial", *ZEEE Trans. Neural Networks*, vol. 4, pp. 748–761, Sept. 1993.
- [283] F. Palmieri and J. Zhu, "Self-association and Hebbian learning in linear neural networks", *ZEEE Trans. Neural Networks*, vol. 6, pp. 1165–1184, Sept. 1995.
- [284] Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Reading, MA: Addison-Wesley, 1989.
- [285] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 3rd ed., New York: **McGraw-Hill**, 1991.

- [286] T.S. Parker and L.O. Chua, "Chaos: A tutorial for engineers", Proceedings of IEEE, vol. 75, pp. 982–1008, Aug. 1987.
- [287] D. Partridge, "The case for inductive programming", IEEE Computer, pp. 36–41, Jan. 1997.
- [288] E. Parzen, "On estimation of a probability density function and mode", Annals of Mathematical Statistics, vol. 33, pp. 1065–1076, 1962.
- [289] Z. Pawlak, "Rough sets", Int. J. Comput. Infom. Sci., vol. 11, pp. 341–356, 1982.
- [290] Z. Pawlak, Rough Sets: Theoretical Aspects of Reasoning about Data, Dordrecht: **Kluwer**, 1991.
- [291] Z. Pawlak, J.G. Busse, R. Slowinsky, and W. Ziarko, "Rough sets", Communications of the ACM, vol. 38, pp. 89–95, Nov. 1995.
- [292] Z. Pawlak, S.K.M. Wong, and W. Ziarko, "Rough sets: Probabilistic verses **deterministic** approach", Int. J. Man-Machine Studies, vol. 29, pp. 81–95, 1988.
- [293] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Reading, MA. Addison-Wesley, 1984.
- [294] B.A. Pearlmutter, "Learning state space trajectories in recurrent neural networks", Neural Computation, vol. 1, pp. 263–269, 1989.
- [295] W. Pedrycz, "Fuzzy neural networks with reference neurons as pattern classifiers", IEEE *Trans.* Neural Networks, vol. 3, pp. 770–775, Sept. 1992.
- [296] W. Pedrycz and A.F. **Rocha**, "Fuzzy-set based models of neurons and knowledge-based networks", IEEE *Trans.* Fuzzy Systems, vol. 1, pp. 254–266, Nov. 1993.
- [297] R. **Penrose**, "A generalized inverse for matrices", in Proceedings of the Cambridge Philosophical Society, vol. 51, pp. 406–413, 1955.
- [298] P. Peretto, "Collective properties of neural networks: A statistical physics approach", Biol. Cybernet., vol. 50, pp. 51–62, 1984.
- [299] D.H. **Perkel**, B. Mulloney, and R.W. Budelli, "Quantitative methods for predicting neuronal behaviour", Neuroscience, vol. 6, no. 5, pp. 823–837, 1981.
- [300] C. Peterson, "Parallel distributed approaches to combinatorial optimization: Benchmark studies on travelling salesman problem", Neural Computation, vol. 2, pp. 261–269, April 1990.
- [301] C. Peterson and J.R. Anderson, "A mean field theory learning algorithm for neural networks", Complex Systems, vol. 1, pp. 995–1019, 1987.

- [302] G.E. Peterson and H.L. Barney, "Control methods used in a study of vowels", *J. Acoust. Soc. Amer.*, vol. 24, no. 2, pp. 175–184, 1952.
- [303] C. Peterson and B. Soderberg, "A new method of mapping optimization problems onto neural networks", *Int. J. Neural Systems*, vol. 1, pp. 3–22, April 1989.
- [304] C. Peterson and B. Soderberg, "Neural optimization", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 617–621, 1995.
- [305] D.S. Plaut, S.J. Nowlan, and G.E. Hinton, "Experiments on learning by back propagation", Technical Report **CMU-CS-86-126**, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1986.
- [306] T. Poggio and F. Girosi, "Networks for approximation and learning", *Proc. ZEEE*, vol. 78, no. 9, pp. 1481–1497, 1990.
- [307] T. Poggio, V. Torre, and C. Koch, "Computational vision and regularization theory", *Nature*, vol. 317, pp. 314–319, Sept. 1985.
- [308] E. Polak and G. Ribiere, "Note sur la convergence de methods de directions conjures", *Revue Francaise Information Recherche Operationnelle*, vol. 16, pp. 35–43, 1969.
- [309] W. Porto, D.B. Fogel, and L.J. Fogel, "Alternative neural network training methods", *ZEEE Expert*, pp. 16–22, June 1995.
- [310] M.J.D. Powell, "Radial basis function approximations to polynomials", in *Numerical Analysis 1987 Proceedings* (Dundee, UK), pp. 223–241, 1988.
- [311] R.W. Preisendorfer, *Principal Component Analysis in Meteorology and Oceanography*, New York: Elsevier, 1988.
- [312] L.R. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition", *Proc. ZEEE*, vol. 77, pp. 257–286, Feb. 1989.
- [313] L.R. Rabiner and B.H. Juang, *Fundamentals of Speech Recognition*, Englewood Cliffs, New Jersey: Prentice-Hall, 1993.
- [314] P.P. Raghu, H.M. Chouhan, and B. Yegnanarayana, "Multi-spectral texture classification using neural network", in *Proceedings of the National Conference on Neural Networks*, (Anna University, Madras, India), pp. 1–10, Nov. 1993.
- [315] P.P. Raghu, *Artificial Neural Network Models for Texture Analysis*, **PhD** thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India, Nov. 1995.

- [316] P.P. Raghu, R. Poongodi, and B. Yegnanarayana, "A combined neural network approach for texture classification", *Neural Networks*, vol. 8, no. 5, pp. 975–987, 1995.
- [317] P.P. Raghu and B. Yegnanarayana, "Segmentation of **gabor**-filtered textures using deterministic relaxation", *IEEE Trans. Image Processing*, vol. 5, pp. 1625–1636, **Dec.** 1996.
- [318] P.P. Raghu and B. Yegnanarayana, "**Multispectral** image classification using **gabor** filters and stochastic relaxation neural network", *Neural Networks*, vol. 10, pp. 561–572, Apr. 1997.
- [319] P.P. Raghu, R. Poongodi, and B. Yegnanarayana, "Unsupervised texture classification using vector quantization and deterministic relaxation neural network", *IEEE Trans. Image Processing*, Oct. **1997a**.
- [320] A. Rangarajan, R. Chellappa, and B.S. Manjunath, "Markov random fields and neural networks with applications to early vision", in *Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections* (I.K. Sethi and A.K. Jain, eds.), New York: **Elsevier** Science, 1991.
- [321] A. Rangarajan and R. Chellappa, "Markov random field models in image processing", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 564–567, 1995.
- [322] A. Ravichandran and B. Yegnanarayana, "A two stage neural network for translation, rotation and size-invariant visual pattern recognition", in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 4, (Toronto, Canada), pp. 2393–2396, May 1991.
- [323] A. Ravichandran, "Object recognition from degraded images using neural networks", Master's thesis, Department of Computer Science and Engineering, Indian Institute of **Technology**, Madras, Mar. 1993.
- [324] A. Ravichandran and B. Yegnanarayana, "Studies on object recognition from degraded images using neural networks", *Neural Networks*, vol. 8, no. 3, pp. 481–488, 1995.
- [325] R. Reddy, "Foundations and grand challenges of artificial intelligence", *Artificial Intelligence Magazine*, pp. 9–21, Winter 1988.
- [326] R. Reddy, "The challenge of artificial intelligence", *IEEE Computer*, pp. 86–98, Oct. 1996.
- [327] J.M. Renders and S.P. Flasse, "Hybrid methods using genetic algorithms for global optimization", *IEEE Trans. Systems, Man and Cybernetics*, vol. 26, pp. 243–258, April 1996.

- [328] R. Reed, "Pruning algorithms—A survey", *IEEE Trans. Neural Networks*, vol. 4, pp. 740–747, Sept. 1993.
- [329] E. Rich and K. Knight, *Artificial Intelligence*, New Delhi: Tata McGraw-Hill Publishing Company Limited, 1991.
- [330] M.D. Richard and R.P. Lippmann, "Neural network classifiers estimate Bayesian a posteriori probabilities", *Neural Computation*, vol. 3, pp. 461–483, 1991.
- [331] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", *Psychological Review*, vol. 65, pp. 386–408, 1958.
- [332] D.E. Rumelhart and D. Zipser, "Feature discovery by competitive learning", *Cognitive Sci.*, vol. 9, pp. 75–112, 1985.
- [333] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, Cambridge, MA: MIT Press, 1986.
- [334] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation", in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1 (D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, eds.), Cambridge, MA: MIT Press, pp. 318–362, 1986a.
- [335] D.E. Rumelhart, P. Smolensky, J.L. McClelland, and G.E. Hinton, "Schemata and sequential thought processes in PDP models", in *Parallel Distributed Processing: Explorations in Microstructure of Cognition*, vol. 2 (J.L. McClelland, D.E. Rumelhart, and the PDP Research Group, eds.), Cambridge, MA: MIT Press, pp. 7–57, 1986b.
- [336] A.P. Russo, "Neural networks for sonar signal processing", in *IEEE Conference on Neural Networks for Ocean Engineering*, vol. 51 (Washington, DC), 1991.
- [337] E. Säckinger, B.E. Boser, J. Bromely, Y. LeCun, and L.D. Jackel, "Application of the ANNA neural network chip to high-speed character recognition", *IEEE Trans. Neural Networks*, vol. 3, pp. 498–505, May 1992.
- [338] P. Salamon, J.D. Nulton, J. Robinson, J. Petersen, G. Ruppeiner, and L. Liao, "Simulated annealing with constant thermodynamic speed", *Computer Physics Communications*, vol. 49, pp. 423–428, 1988.
- [339] T.D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network", *Neural Networks*, vol. 2, no. 12, pp. 459–473, 1989.
- [340] N. Saravanan and D.B. Fogel, "Evolving neural control systems", *IEEE Expert*, pp. 23–27, June 1995.

- [341] M. Sarkar and B. Yegnanarayana, "A clustering algorithm using evolutionary programming", in Proceedings of IEEE International Conference on Neural Networks (Washington D.C.), June 1996.
- [342] M. Sarkar and B. Yegnanarayana, "Feedforward neural networks configuration using evolutionary programming", in Proceedings of IEEE International Conference on Neural Networks (Houston, USA), June 9–12, **1997a**.
- [343] M. Sarkar and B. Yegnanarayana, "An evolutionary programming-based probabilistic neural network construction technique", in Proceedings of IEEE International Conference on Neural Networks (Houston, USA), June 9–12, **1997b**.
- [344] M. Sarkar and B. Yegnanarayana, "Rough-fuzzy set theoretic approach to evaluate the importance of input features in classification", in Proceedings of IEEE International Conference on Neural Networks (Houston, USA), June 9–12, **1997c**.
- [345] M. Sarkar and B. Yegnanarayana, "Incorporation of fuzzy classification properties into backpropagation learning algorithm", in Proceedings of IEEE International Conference on Fuzzy Systems (Barcelona, Spain), July 1–5, **1997d**.
- [346] M. Sarkar, B. Yegnanarayana, and D. Khemani, "Evolutionary programming based hard clustering", Pattern Recognition Letters, vol. 18, pp. 975–986, **1997e**.
- [347] M. Sarkar, B. Yegnanarayana, and D. Khemani, "**Feedforward** neural networks with fuzzy backpropagation learning algorithm for classification", to appear in Pattern Recognition Letters, 1998.
- [348] R. Schalkoff, Pattern Recognition: Statistical, Structural and Neural Approaches, New York: John Wiley & Sons, 1992.
- [349] P. Schumacher and J. Zhang, "Texture classification using neural networks and discrete wavelet transform", in Proceedings of the International Conference on Image Processing, vol. 3, (Piscataway, NJ), IEEE, pp. 903–907, 1994.
- [350] H.P. Schwefel, Numerical Optimization of Computer Models, Chichester: John Wiley, 1981.
- [351] T.J. Sejnowski, "Strong covariance with nonlinearly interacting neurons", J. Math. Biology, vol. 4, pp. **303–321**, 1977.
- [352] T.J. Sejnowski and C.R. Rosenberg, "Parallel networks that learn to pronounce English text", Complex Systems, vol. 1, pp. 145–168, 1987.
- [353] H.S. Seung, H. Sompolinsky, and N. Tishby, "Statistical mechanics of learning from examples", Physical Review A, vol. 45, pp. 6056–6091, Apr. 1992.

- [354] C.E. Shannon, "A mathematical theory of communication", Bell System *Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.
- [355] P.K. **Simpson**, *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations*, Elmsford, NY: Pergamon Press, 1990.
- [356] P.K. **Simpson**, "Fuzzy min-max classification with neural networks", *Heuristics Journal of Knowledge Engineering*, no. 4, pp. 1–9, 1991.
- [357] P.K. **Simpson**, "Foundations of neural networks", in *Artificial Neural Networks: Paradigms, Applications and Hardware Implementations* (E. Sanchez-Sinencio and C. Lau, eds.), New York: IEEE Press, pp. 3–24, 1992.
- [358] S. **Singhal** and L. Wu, "Training feed-forward networks with the extended Kalman algorithm", in *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 2 (Glasgow, Scotland), pp. 1187–1190, 1989.
- [359] H. Sompolinsky, A. Crisanti, and H.J. **Sommers**, "Chaos in random neural networks", *Physical Review Letters*, vol. 61, pp. 259–262, 1988.
- [360] E.D. **Sontag**, "Feedforward nets for interpolation and classification", *J. Computing System Sciences*, vol. 45, pp. 20–48, 1992a.
- [361] E.D. **Sontag**, "Feedback stabilization using two-hidden-layer nets", *IEEE Trans. Neural Networks*, vol. 3, pp. 981–990, Nov. 1992b.
- [362] C.M. Soukoulis, K. **Levin**, and G.S. Grest, "Irreversibility of metastability in spin-glasses. I. **Ising** model", *J. Physical Review*, vol. B28, pp. 1495–1509, 1983.
- [363] D.F. Specht, "Probabilistic neural networks for classification, mapping, or associative memory", in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 1, IEEE Press, New York, pp. 525–532, June 1988.
- [364] D.F. Specht, "Probabilistic neural networks", *Neural Networks*, vol. 3, no. 1, pp. 109–118, 1990.
- [365] D.F. Specht, "**A** general regression neural **network**", *IEEE Trans. Neural Networks*, vol. 2, pp. 568–576, Nov. 1991.
- [366] K. Steinbuch, "Die **lernmatrix**", *Kybernetik*, vol. 1, pp. 36–45, 1961.
- [367] K. Steinbuch and U.A.W. **Piske**, "Learning matrices and their applications": *IEEE Trans. Electronic Computers*, vol. EC-12, pp. 846–862, 1963.

- [368] W.S. Stornetta, T. Hogg, and B.A. Huberman, "A dynamical approach to temporal **pattern** processing", in Neural Information Processing Systems (**D.Z. Anderson, ed.**), (Denver), American Institute of Physics, New York, pp. 750–759, 1988.
- [369] G. **Strang**, Linear Algebra and its Applications, New York: Academic Press, 1980.
- [370] N. Sudha, "Principal component neural networks for applications in signal processing", Master's thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, 1996.
- [371] R.S. Sutton, *Temporal* Credit Assignment in Reinforcement Learning, **PhD** thesis, University of Massachusetts, **Amherst**, 1984.
- [372] **R.S. Sutton** and A.G. **Barto**, "Towards a modern theory of adaptive networks: Expectation and prediction", Psychological Review, vol. 88, pp. 135–170, 1981.
- [373] R.S. Sutton, A.G. **Barto**, and R.J. Williams, "Reinforcement learning is direct adaptive optimal control", in Proceedings of the American Control Conference (Boston, MA), pp. 2143–2146, 1991.
- [374] R.S. Sutton, A.G. **Barto**, and R.J. Williams, "Reinforcement learning is direct adaptive optimal control", IEEE Control Systems Magazine, vol. 12, pp. 19–22, 1992.
- [375] H. Szu, "Fast simulated annealing", in Neural Networks for Computing (**J.S. Denker, ed.**) American Institute of Physics, New York: Snowbird, pp. 420–425, 1986.
- [376] C.L. **Tan**, T.S. **Quah**, and H.H. Teh, "An artificial neural network that models human decision making", IEEE Computer, pp. 64–70, Mar. 1996.
- [377] D.W. Tank and J.J. Hopfield, "Neural computation by concentrating information in time", in Proceedings of the National Academy of Sciences, vol. 84, (USA), pp. 1896–1900, **1987a**.
- [378] D.W. Tank and J.J. Hopfield, "Concentrating information in time: Analog neural networks with applications to speech recognition problems", in IEEE First International Conference on Neural Networks, vol. IV (**M. Caudill and C. Butler, eds.**) (San **Diego**), IEEE, New York, pp. 455–468, **1987b**.
- [379] J.G. Taylor and S. Coombes, "Learning higher order correlations", Neural Networks, vol. 6, no. 3, pp. 423–427, 1993.
- [380] A.N. **Tikhonov**, "On regularization of Ill-Posed Problems", Doklady Akademii Nauk USSR, vol. 153, pp. 49–52, 1973.

- [381] A.N. **Tikhonov** and V.Y. **Arsenin**, *Solutions of Ill-Posed Problems*, Washington, DC: **W.H. Winston**, 1977.
- [382] K. **Torkkola**, K. Kangas, J. Utela, S. **Kaski**, M. **Kokkonen**, M. Kurimo, and T. Kohonen, "Status report of the finnish phonetic typewriter project", in *Proceedings of the International Conference on Artificial Neural Networks*, Amsterdam: Elsevier, pp. 771–776, 1991.
- [383] E.C.K. **Tsao**, J.C. Bezdek, and N.R. Pal, "Fuzzy kohonen clustering algorithm", *Pattern Recognition*, vol. 27, no. 5, pp. 757–767, 1994.
- [384] **D.W. Tufts** and R. Kumaresan, "Singular value decomposition and improved frequency estimation using linear prediction", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 30, pp. 671–675, Aug. 1982.
- [385] L.H. Ungar, "Forecasting", in *The Handbook of Brain Theory and Neural Networks* (**M.A. Arbib**, ed.), Cambridge, MA: MIT Press, pp. 399–403, 1995.
- [386] A. Uttley, "Conditional probability machines and conditioned reflexes", in *Automata Studies* (**C. Shannon** and **J. McCarthy**, eds.), Princeton: Princeton University Press, pp. 253–276, 1956a.
- [387] A. Uttley, "Temporal and spatial patterns in a conditional probability machine", in *Automata Studies* (**C. Shannon** and **J. McCarthy**, eds.), Princeton: Princeton University Press, pp. 277–285, 1956b.
- [388] L.G. Valiant, "A theory of the learnable", *Communications of the ACM*, vol. 27, pp. 1134–1142, Nov. 1984.
- [389] L.G. Valiant, *Circuits of the Mind*, Oxford University Press, 1994.
- [390] A. van der Veen, E.F. Deprettere, and A.L. Swindlehurst, "Subspace-based signal analysis using singular value decomposition", *Proc. IEEE*, vol. 81, pp. 1275–1308, Sept. 1993.
- [391] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, Boston, MA: **Kluwer Academic Publishers**, 1988.
- [392] V. Vapnik, "Learning and generalization: Theoretical bounds", in *The Handbook of Brain Theory and Neural Networks* (**M.A. Arbib**, ed.), Cambridge, MA: **MIT Press**, pp. 516–522 1995.
- [393] M. Vidyasagar, *A Theory of Learning and Generalization: With Applications to Neural Networks and Control Systems* (Communications and Control Engineering), Springer-Verlag, 1997.

- [394] A. Visa, "A texture **classifier** based on neural network principles", in IEEE Proceedings of the International Joint Conference on Neural Networks, vol. 1 (**San Diego, CA**), IEEE, pp. **491–496**, 1990.
- [395] C. von der Malsburg, "Self-organization of orientation sensitive cells in the striate cortex", *Kybernetik*, vol. 14, pp. **85–100**, 1973.
- [396] J. von Neumann, *The Computer and the Brain*, New Haven, **CT**: Yale University Press, 1958.
- [397] G. Vrckovnik, T. Chung, and C.R. Carter, "Classifying impulse radar waveforms using principal components analysis and neural networks", IEEE Proceedings of the International Joint Conference on Neural Networks, vol. 1, pp. 69–74, 1990.
- [398] G. Wahba, "Generalization and regularization in nonlinear learning systems", in *The Handbook of Bmin Theory and Neural Networks* (**M.A. Arbib, ed.**), Cambridge, MA: MIT Press, pp. **426–430**, 1995.
- [399] A. Waibel, "Modular construction of time-delay neural networks for speech recognition", *Neural Computation*, vol. 1, pp. **39–46**, 1989.
- [400] A. Waibel, T. Hanazawa, G. **Hinton**, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks", *IEEE Dans. Acoust., Speech, Signal Processing*, vol. 37, pp. 328–339, March 1989.
- [401] **L.X.** Wang, "Oscillatory and chaotic dynamics in neural networks under varying operating conditions", *IEEE Dans. Neural Networks*, vol. 7, pp. 1382–1388, Nov. 1996.
- [402] Y.F. Wang, J.B. Cruz, and J.H. Mulligan, "**Two** coding strategies for bidirectional associative memory", *IEEE Dans. Neural Networks*, vol. 1, pp. 81–92, March **1990a**.
- [403] Y.F. Wang, J.B. Cruz, and J.H. Mulligan, "On multiple training for bidirectional associative memories", *IEEE Dans. Neural Networks*, vol. 1, pp. 275–276, Sept. **1990b**.
- [404] Y.F. Wang, J.B. Cruz, and J.H. Mulligan, "Guaranteed recall of all training pairs for bidirectional associative memory", *IEEE Dans. Neural Networks*, vol. 2, pp. 559–567, Nov. 1991.
- [405] C. Wang, J.M. Kuo, and J.C. **Principe**, "**A** relation between Hebbian and MSE learning", in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 5, pp. 3363–3366, 1995.
- [406] **L.X.** Wang and J.M. **Mendel**, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning", *IEEE Trans. Neural Networks*, vol. 3, pp. 807–814, Sept. 1992.

- [407] P.D. Wasserman, "Combined **backpropagation/Cauchy** machine", *Neural Networks Supplement: INNS Abstracts*, vol. 1, p. 556, 1988.
- [408] P.D. Wasserman, *Neural Computing, Theory and Practice*, New York: Van **Nostrand** Reinhold, 1989.
- [409] P.D. Wasserman, *Advanced Methods in Neural Computing*, New York: Van **Nostrand** Reinhold, 1993.
- [410] A.S. Weigend and N.A. Gershenfeld, "Results of the time series prediction competition at the Santa Fe Institute", in *Proceedings of the IEEE International Conference on Neural Networks*, vol. **III** (San Francisco), IEEE, New York, pp. **1786–1793**, 1993.
- [411] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman, "Generalization by weight-elimination with application to forecasting", in *Advances in Neural Information Processing Systems* (R.P. Lippmann, J.E. Moody, and D.S. Touretzky, eds.), vol. 3 (Denver), Morgan **Kaufmann**, San Mateo, CA, pp. 875–882, 1991.
- [412] N. Wiener, *Cybernetics*, New York: John Wiley & Sons, 1948.
- [413] R.S. **Wenocur** and M.R. Dudley, "Some special Vapnik-Chervonenkis classes", *Discrete Mathematics*, vol. 33, pp. 313–318, 1981.
- [414] P.J. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioural sciences*, **PhD** thesis, Harvard University, Cambridge, MA, 1974.
- [415] P.J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*, New York: John Wiley, 1994.
- [416] L.F.A. **Wessels** and E. **Barnard**, "Avoiding false local minima by proper initialization of connections", *IEEE Trans. Neural Networks*, vol. 3, pp. 899–905, Nov. 1992.
- [417] H. White, "Learning in artificial neural networks: A statistical perspective", *Neural Computation*, vol. 1, no. 4, pp. **425–464**, 1989.
- [418] B.A. Whitehead, "Genetic evolution of radial basis function coverage using orthogonal niches", *IEEE Trans. Neural Networks*, vol. 7, pp. 1525–1528, Nov. 1996.
- [419] B. **Widrow**, "Generalization and information storage in networks of Adaline 'neurons'", in *Self-Organizing Systems* (**M.C.** Yovitz, G.T. Jacobi, and G. Goldstein, eds.), Washington, DC: Spartan Books, pp. **435–461**, 1962.
- [420] B. **Widrow** and M.E. Hoff, "Adaptive switching circuits", *IRE WESCON Convention Record*, vol. 4, pp. 96–104, Aug. 1960.

- [421] B. Widrow and S.D. Stearns, Adaptive Signal Processing, Englewood Cliffs, NJ: Prentice-Hall Inc., 1985.
- [422] A. Wieland and R. Leighton, "Geometric analysis of neural network capabilities", in First IEEE International Conference on Neural Networks, vol. 3 (**San Diego, CA**), IEEE, New York, pp. 385–392, 1987.
- [423] R.J. Williams, "Toward a theory of reinforcement-learning connectionist systems", Technical Report, NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA, 1988.
- [424] R.J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", Machine Learning, vol. 8, pp. 229–256, 1992.
- [425] R.J. Williams and D. Zipser, "Experimental analysis of the real-time recurrent learning algorithm", Connection Science, vol. 1, pp. 87–111, 1989.
- [426] D. Willshaw, Models of distributed associative memory, PhD thesis, University of Edinburgh, 1971.
- [427] G.Y. Wilson and G.S. Pawley, "On the stability of the travelling salesman problem algorithm of Hopfield and Tank", Biol. Cybernet., vol. 58, pp. 63–70, 1988.
- [428] R. Wolf and J. Platt, "Postal address block location using a convolutional locator network", in Advances in Neural Information Processing Systems, vol. 6 (**J. Cowan**, G. Tesauro, and **J. Alspector**, eds.), (**San Mateo, CA**), Morgan Kaufmann, pp. 745–752, 1994.
- [429] L. Xu, E. Oja, and C.Y. Suen, "Modified Hebbian learning for curve and surface fitting", Neural *Networks*, vol. 5, no. 3, pp. 441–457, 1992.
- [430] W.Y. Yan, U. Helmke, and J. B. Moore, "Global analysis of Oja's flow for neural networks*", IEEE *Trans.* Neural Networks, vol. 5, pp. 674–683, Sept. 1994.
- [431] Y. Yao and W.J. Freeman, "Models of biological pattern recognition with spatially chaotic dynamics", Neural Networks, vol. 3, pp. 153–170, 1990.
- [432] X. Yao, "Evolutionary artificial neural networks", Int. J. Neural Systems, vol. 4, no. 3, pp. 203–222, 1993.
- [433] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks", IEEE *Trans.* Neural Networks, vol. 8, May 1997.
- [434] B. Yegnanarayana, "Artificial neural networks for pattern recognition", Sadhana, vol. 19, pp. 189–238, Apr. 1994.

- [435] B. Yegnanarayana, D. Khemani, and M. Sarkar, "Neural networks for contract bridge bidding", *Sadhana*, vol. 21, pp. 395–413, June 1996.
- [436] B. Yegnanarayana, C.P. Mariadassou, and P. Saini, "Signal reconstruction from partial data for sensor array imaging applications", *Signal Process*, vol. 19, pp. 139–149, Feb. 1990.
- [437] B. Yegnanarayana, S. Rajendran, V.R. Ramachandran, and A.S. Madhukumar, "Significance of knowledge sources for a text-to-speech system for Indian languages", *Sadhana*, vol. 19, pp. 147–169, Feb. 1994.
- [438] Y.O. Yoon, R.W. Brobst, P.R. Bergstresser, and L.L. Peterson, "A desktop neural network for dermatology diagnosis", *J. Neural Network Computing*, pp. 43–52, Summer 1989.
- [439] A.L. Yuille, "Constrained optimization and the elastic net", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 250–255, 1995.
- [440] L.A. Zadeh, "Fuzzy sets", *Information and Control*, vol. 8, pp. 338–353, 1965.
- [441] A.D. Zapranis and A.N. Refenes, "Investment management: Tactical asset allocation", in *The Handbook of Brain Theory and Neural Networks* (M.A. Arbib, ed.), Cambridge, MA: MIT Press, pp. 491–495, 1995.
- [442] Q. Zhang and Y.W. Leung, "Energy function for the one-unit Oja algorithm", *IEEE Trans. Neural Networks*, vol. 6, pp. 1291–1293, Sept. 1995.
- [443] J.M. Zurada, *Introduction to Artificial Neural Systems*, Singapore: Info Access and Distribution, 1992.

Author Index

A

- Aarts**, E., 192, 193, 399, 427
Abrash, V., 404
Ackley, D.H., 22, 23, 65, 183, 399
Adachi, M., 397, 399
Adleman, L., 335, 399
Aibara, T., **94**, 145, 418
Aihara, K., 397, 399
Aleksander, I., 110, 158, 160, 162, 179, 182, 399
Alspector, J., 430
Amari, S., 22, 70, 373, 374, 377, 399
Amit, D.J., 22, 152, 174, 175, 399
Anandan, P., 64, 400
Anderson, C., 195, 401
Anderson, **D.Z.**, 404, 411, 415, 426
Anderson, J., 400
Anderson, J.R., 296, 420
Anderson, S., 22, 269, 400
Andreyev, Y.V., 335, 397, 400
Angeline, P.J., 392, 400
Anthony, M., 374, 377, 400
Arabshahi, R.J., 404
Arbib, Michael A., 399, 400, 402, 407, 408, 409, 422, 427, 428, 431
Arsenin, V.Y., 132, 242, 427
Ashby, W., 22, 400

B

- Baldi**, P., 381, 389, 400
Ballard, **D.H.**, 134, 400
Banerjee, M., 396, 400
Barnard, E., 127, 285, 400, 429
Barnden, John A., 13, 400
Barney, H.L., 309, 310, 421
Barto, **A.G.**, **22**, **59**, **64**, **134**, **400**, **401**, 426
Battiti, R., 129, 131, 401

- Baum, E., 376, 401
Baxt, W.G., 333, 401
Belew, R.K., 335, 401
Belsky, Y.L., 400
Bengio, Y., 321, 322, 377, 415
Bergstresser, P.R., 431
Bezdek, J.C., 9, 393, 401, 427
Bhattacharya, A.K., 392, 401
Bienenstock, E., 407, 415
Bilbro, G.L., 173, 401
Billings, S.A., 392, 401
Binder, K., 190, 194, 401
Blondeau, F.C., 397, 401
Blum, A.L., 133, 401
Blumer, A., 373, 376, 402
Bornholdt, S., 392, 402
Bose, **N.K.**, 140, 259
Boser, B., 415, 423
Bounds, D.G., 334, 402
Bovik, A.C., 325, 402
Brobst, R.W., 431
Bromely, J., 423
Broomhead, D.S., 22, 247, 402
Budelli, R.W., 420
Burel, G., 387, 402
Burges, C.J.C., 324, 402
Burke, B., 406
Burke, G., 334, 402
Busse, J.G., 420
Butler, C., 426

C

- Caianiello, E.R., 22, 402
Cameron, S.H., 107, 402
Carbonell, J., 410
Cardoso, J.F., 387, 402
Carpenter, G.A., 258, 259, 262, 273, 394, 402, 403
Carter, C.R., 428

Author Index

Casasent, D., 285, 400
Casselman, F.L., 134, 403
Caudell, T.P., 404
Caudill, M., 426
Chan, K.H., 418
Chandrasekhar, C., 312, 314, 321, 403
Chang, C., 419
Chauvet, G., 401
Chehikian, A., 413
Chellappa, R., 292,324,403,408,422
Chen, C.H., 333, 334, 401, 403
Cherkassky, V., 293, 404
Cherry, C., 418
Cheung, J., 59, 404
Choi, J.J., 394, 404
Chouhan, H.M., 421
Chua, L.O., 396, 420
Chung, F.L., 394, 404
Chung, T., 428
Clark, M., 402
Cohen, **M.A.**, 68, 69, 73, 134, 148, 404
Cole, R., 307, 404
Colorni, A., 405
Comon, P., 387, 404
Cooke, D.E., 335, 411
Coombes, S., 386, 389, 426
Cooper, F.S., 5, 404
Cortes, C., 303, 404
Cowan, J., 13, 404, 430
Crisanti, A., 425
Crutchfield, J.P., 396, 404
Cruz, J.B., 428
Cybenko, G., 133, 246, 405

D

Dam, A. Van., 1, 416
Darken, C., 247, 418
Daugman, J.G., 292, 325, 405
Davis, S.B., 309, 310, 405
Denker, J.S., 415, 426
Deprettere, E.F., 427
Devijver, P.A., 9, 380, 405
Diamantaras, K.I., 384, 385, 390, 405, 415
Dmitriev, A.S., 400
Dorigo, M., 335, 405
Doursat, R., 407

Dowla, F.U., 412
Dreyfus, H.L., 1, 4, 306, 405
Dreyfus, S.E., 405
Dubes, R.C., 254, 405
Dubois, D., 395, 405
Duda, R.O., 6, 333, 406
Dudley, M.R., 377, 429
Durbin, R., 300, 406

E

Ehrenfeucht, A., 402
Eisenberg, J., 397, 406
Elman, J.L., 267, 406, 408, 412

F

Fahlman, S.E., 129, 406
Fanty, M., 404
Farber, R., 269, 415
Farmer, J.D., 404
Feigenbaum, E.A., 4, 406
Finnoff, W., 409
Flanagan, J.L., 307, 406
Flasse, S.P., 392, 422
Fletcher, R., 130, 406
Fogel, D.B., 335, 391, 392, 406, 423
Fogel, L.J., 406, 421
Foldiak, P., 383, 406, 421
Folger, T.A., 393, 413
Franco, H., 404
Freeman, D.F., 403
Freeman, J.A., 147, 153, 155, 213, 215, 217, 230, 259, 266, 406
Freeman, W.J., 406, 430
Fujioka, R., 412
Fukushima, K., 22,271,272,323,407
Funahashi, K., 246, 407

G

Gabor, D., 22, 407
Gallant, S.I., 333, 407
Gamier, S.J., 401
Gault, J.W., 401
Gawthrop, P.J., 411
Geisler, W.S., 402
Gelatt, C.D., 413

Geman, D., 192, 295, 407
Geman, S., 192, 256, 295, 407
Gershenfeld, N.A., 269, 429
 Giles, C.L., 404
 Girosi, F., 22, 248, 421
 Glauber, R.J., 295, 407
Gobovic, D., 411
 Goldberg, D.E., 391, 407
Goldstein, G., 429
 Goodman, D.M., 412
 Goodman, R., 408
Gopalakrishna, M., 404
Gori, M., 134
 Graudenz, D., 392, 402
 Gray, R.M., 185, 304, 407
 Grebogi, C., 419
 Greenberger, M., 5, 408
 Greenspan, H., 324, 408
 Grest, G.S., 425
 Grossberg, S., 30, 43, **48, 60, 68, 69**,
 73, 148, 258, 259, 262, 266, 273,
 402, 403, 404, 408
Gutfreund, H., 399
Guyon, I., 120, 125, 408

H

Hagiwara, M., 239, 240, 408
Haken, H., 292, 408
 Hampshire, J.B., 132, 408
Hanazawa, T., 428
 Hanson, S.J., 404
Harrington, J., 310, 408
 Hart, P.E., 6, 406
Harth, E., 396, 409
 Hassoun, M.H., 31, 36, 53, 65, 138,
 139, 231, 232, 409
 Haussler, D., 373, 375, 376, 401, 402,
 409
Hayashi, Y., 397, 409
Haykin, S., 120, 127, 128, 131, 132,
 134, **193, 200, 209, 230**, 231, 249,
250, 251, 253, 276, 296, 313, 379,
 380, 381, 382, 409
 Hebb, D.O., 21, 22, 32, 58, 380, 409
Hecht-Nielsen, R., **22, 91, 97, 99, 256**,
 351, 409
Heerman, D.W., 190, 401
 Hegde, **S.U.**, 417
Helmke, U., 430

Henderson, D., 415
Herault, J., 387, 413
 Hergert, F., 409
 Hertz, J.A., 69, 139, 151, 154, 155,
 157, 170, 175, 194, 199, 200, 209,
 224, 230, 231, 232, 266, 293, 296,
 301, 303, 304, 339, 381, 404, 409
Hinton, G., **22, 23, 195, 399, 400, 406**,
 408, 410, 421, 423, 428
Hirota, K., 394, 410
Hodgkin, A.L., 50, 397, 410
 Hoff, M.E., 22, 23, 97, 429
Hogg, T., 426
Holden, S.B., 374, 375, 376, 377, 400
 Holland, J.H., 1, 4, 391, 410
 Holmstrom, L., 378, 410
 Hopfield, J.J., 22, 23, 46, 152, **155**,
 156, 267, 299, 410, 411, 426
 Home, B., 120, 134, 411
 Hornik, K., 381, 389, 400
 Hotelling, H., 379, 411
 Howard, R.E., 415
 Hsu, C.C., 397, 411
 Hu, M., 205, 411
 Huang, W.Y., 310, 411
 Huang, Z., 229, 411
Hubbard, W., 415
Hubel, D.H., 224, 411
 Huberman, B.A., 426, 429
 Hummels, D.M., 418
 Huneycutt, B.L., 412
 Hunt, D.J., 393, 413
 Hunt, E.R., 397, 411
 Hunt, J.E., 335, 411
 Hunt, K.J., 306, 411
 Hush, D.R., 120, 126, 134, 411
Huxley, A.F., 50, 397, 410
Hwang, J.N., **324**, 412

I

Ishibuchi, H., 132, 394, 412

J

Jack, M., 408
Jackel, L.D., 415, 423
 Jacobi, G.T., 429
 Jacobs, R.A., 128, 134, 412

Author Index

Jain, A.K., 422
Jean, J.S.N., 378, 412
Jockusch, S., 392, 412
Johansson, E.M., 412
Jolliffe, I.T., 379, 412
Jones, R., 400
Jordan, M.I., 267, 268, 412
Jordan, R.L., 332, 412
Joutsensalo, J., 387, 390, 413
Juang, B.H., 307, 421
Judd, J.S., 133, 412
Jutten, C., 387, 413

K

Kalantri, K., 418
Kanal, L., 8, 413
Kangas, K., 427
Karhunen, J., 387, 390, 413
Kaski, S., 427
Kawamoto, M., 379, 417
Kay, S.M., 390, 413
Kearns, M., 409
Keeler, J.D., 324, 413
Keller, J.M., 393, 413
Kerrigan, D.A., 403
Khemani, D., 424, 431
Kim, M.W., 415
King, R.A., 222, 304, 419
Kirkpatrick, S., 22, 179, 293, 413
Kittler, J., 9, 380, 405
Klatt, D.H., 308, 413
Klir, G.J., 393, 395, 413
Klopf, A.H., 22, 60, 66, 413
Knight, K., 1, 423
Koch, C., 303, 413, 421
Kohonen, T., 22, 222, 224, 225, 267, 292, 293, 304, 305, 309, 414, 427
Koistnen, P., 378, 410
Kokkonen, M., 427
Korst, J., 192, 399
Kosko, B., 22, 41, 42, 47, 61, 71, 73, 238, 239, 403, 414
Koza, J.R., 391, 414
Kramer, A.H., 131, 415
Krogh, A., 409
Kuh, A., 411
Kullback, S., 132, 415
Kumaresan, R., 390, 427
Kuminov, D.A., 400

Kung, S.Y., 200, 219, 221, 230, 231, 246, 339, 384, 390, 405, 415
Kuo, J.M., 428
Kurimo, M., 427

L

Lacoume, J.L., 413
Lane, S.E., 403
Lang, K., 428
Lapedes, A., 269, 415
Lau, C., 425
Laver, J., 408
LeCun, Y., 134, 194, 321, 322, 323, 377, 415, 423
Lee, C.S.G., 24, 335, 416
Lee, H.B., 390, 415
Lee, T., 394, 404
Lee, Y., 126, 415
Leighton, R., 132, 430
Leon, S.J., 380, 385, 386, 416
Leow, W.K., 413
Leung, Y.W., 381, 431
Levin, K., 425
Liang, P., 140, 259, 402
Liao, L., 423
Lin, C.T., 24, 335, 393, 394, 416
Linsker, R., 22, 224, 379, 416
Lippmann, R.P., 112, 255, 256, 281, 282, 307, 310, 326, 411, 412, 413, 416, 423, 429
Little, W., 22, 416
Liu, Y., 373, 392, 416, 430
Lloyd, P.J., 402
Lowe, D., 243, 247, 251, 402, 416
Lu, Y.C., 393, 394, 416

M

MacKay, D.J.C., 248, 416
Madhukumar, A.S., 431
Majumder, D.D., 393, 419
Malbos, J., 413
Mandlebrot, B., 397, 416
Maniezzo, V., 405
Manjunath, B.S., 403, 422
Mantas, J., 8, 416
Marcus, A., 1, 416
Mariadassou, C.P., 431

Author Index

Marks, R.J., 404
Marple, S.L., 390, 417
Marroquin, J., 413
Martin, N., 413
Mathew, B., 402
Matsuoka, K., 379, 417
McCarthy, J., 427
McClelland, J.L., **4, 20, 293, 316, 317**,
324, 341, 342, 343, 344, 345, 346,
347, 349, 410, 417, 423
McCorduck, P., 406
McCulloch, W.S., 21, 22, **26, 27**, 417
McGough, R., 334, 417
Mead, C., 22, 417
Mendel, J.M., 394, 428
Mermelstein, P., 309, 310, **405**
Merrill, J.W.L., 400
Metropolis, N., 190, 295, 417
Mhaswade, P.K., 292, 293, 417
Michalski, R., 410
Michell, T., 410
Miller, G.F., 392, 417
Millstrom, N.H., 403
Minai, A.A., 128, 417
Minsky, M., **21, 22, 23, 109, 123, 134**,
241, 417, 418
Mitra, S., 307, 400, 419
Moody, J., 247, 412, 413, 418, 429
Moore, J.B., 430
Morgan, **D.P.**, 334, 418
Morgan, N., 404
Morton, H., 110, 158, 160, 162, 179,
182, 399
Mozer, M.C., 267, 418
Mukhopadhyay, S., 306, 418
Muller, B., 16, 168, 170, 171, 293,
296, 418
Mulligan, J.H., **428**
Mulloney, B., 420
Murakami, K., 94, 145, 418
Muroga, S., 107, 418
Murthy, H.A., 419
Musavi, M.T., 373, 418
Muthuswamy, Y., 404

N

Narendra, K.S., 134, 269, 277, 305,
306, 418
Narendranath, M., 307, 378, 418, 419

Nasrabadi, N.M., 222, 304, 419
Neeharika, A., 372, 419
Nguyen, D.H., 305, 419
Nichols Jr., W.G., 403
Nii, H.P., 406
Nilsson, N., 22, 419
Niranjan, M., 376, 410
Nowlan, S.J., 309, 419, 421
Nulton, J.D., 423

O

Oh, **S.H.**, 415
Oja, E., 66, 208, 209, 381, 382, 419,
430
Ornidvar, M., 59, 404
Ott, E., 397, 419
Owners, A.J., 406

P

Packard, N.H., 404
Pal, N.R., 427
Pal, **S.K.**, 307, 393, 400, 401, 419
Palmer, R.G., 409
Palmieri, F., 380, 383, 419
Pao, Y.H., 99, 419
Papert, S.A., 22, 23, 109, 123, 134,
241, 418
Papoulis, A., 152, 167, 364, 419
Parker, T.S., 396, 420
Parthasarathy, K., 134, 269, 277, 305,
418
Partridge, D., 4, 420
Parzen, E., 255, 420
Pawlak, Z., 335, 395, 396, 420
Pawley, G.S., 299, 430
Pearl, J., 1, 420
Pearlmutter, B., 132, 271, 408, 420
Pedrycz, W., 394, 410, 420
Penrose, R., 92, 420
Peretto, P., 22, 420
Perkel, D.H., 46, 420
Petersen, J., 423
Peterson, C., 195, 293, 295, 296, 298,
420, 421
Peterson, G.E., 309, 310, 421
Peterson, L.L., 431
Piske, U.A.W., 65, 425

Pitts, W., 21, 22, 26, 27, 417
 Platt, J., 324, 430
 Plaut, D.S., 129, 421
Poggio, T., 22, 242, 248, 421
 Polak, E., 130, 421
 Pollack, J.B., 400
 Poongodi, R., 422
 Port, R., 400
Porto, W., 392, 393, 421
 Powell, M.J.D., 247, 421
 Prade, H., 395, 405
Preisendorfer, R.W., 380, 421
Principe, J.C., 428

Q

Quah, T.S., 426

R

Rabiner, L.R., 307, 309, 421
 Raghu, P.P., 229, 324, 325, 326, 327,
 328, 330, 333, 421, 422
Rajendran, S., 419
 Ramachandran, V.R., 431
Rangarajan, A., 292, 324, 422
 Ravichandran, A., 281, 283, 285,
 422
Rayner, P.J.W., 374, 410
 Reddy, R., 4, 306, 422
 Reed, R., 378, 423
 Reeves, C.M., 130, 406
 Refenes, A.N., 334, 431
 Reinhardt, J., 16, 168, 170, 171, 293,
 296, 418
 Renders, J.M., 392, 422
 Reynolds, J.H., 403
 Ribiere, G., 130, 421
 Rich, E., 1, 423
 Richard, M.D., 255, 256, 324, 423
Ritter, H., 392, 412
 Ritz, S., 400
Rivest, R., 133, 397, 401
 Robinson, J., 423
Rocha, A.F., 394, 420
 Rosen, D.B., 403
 Rosenberg, C.R., 134, 307, 424
 Rosenblatt, F., 22, 23, 27, 423
 Rosenbluth, A.W., 417

Rosenbluth, M.N., 417
 Roysm, B., 392, 401
Rumelhart, D., **4, 20, 22, 23, 117, 125,**
 129, 221, 270, 271, 293, 316, 317,
 324, 341, 342, 343, 344, 345, 346,
 347, 348, 349, 410, 412, 413, 417,
 423, 429
 Ruppeiner, G., 423
 Russo, A.P., 120, 423

S

Sackinger, E., 322, 323, 423
Saini, P., 431
 Salmon, P., 192, 423
 Salas, J.M., 411
 Sanchez-Sinencio, E., 425
 Sanger, T.D., 382, 409, 423
Sangiovanni-Vincentelli, A., 131,
 415
 Saravanan, N., 392, 423
 Sarkar, M., **335**, 392, 393, 396, 424,
 431
 Saunders, G.M., 400
 Sbarbaro, D., 411
 Schaffer, J.D., 417
 Schalkoff, R., 9, 424
 Schapire, R., 409
 Schumacher, P., 324, 424
 Schwartz, E.L., 400
 Schwefel, H.P., 391, 424
 Scofield, C.L., 334, 418
Sejnowski, T.J., 22, 23, 59, 134, 307,
 399, 400
Selfridge, O., 22, 418
Sethi, I.K., 422
 Seung, H.S., 373, 377, 424
 Shannon, C., 22, 425, 427
 Sharp, D.H., 13, 404
 Shaw, G., 416
 Shaw, R.S., 404
 Shikano, K., 428
 Shortliffe, E.H., 333, 406
 Silverstein, J., 400
 Simchony, T., 403
Simpson, P.K., **19, 29, 58, 65, 66, 233,**
 234, 425
 Singhal, S., 131, 425
 Skapura, D.M., 147, 153, 155, 213,
 215, 217, 230, 259, 266, 406

Slowinski, R., 405, 420
 Smolensky, P., 423
 Snyder, W.E., 401
Soderberg, B., 293, 298, 421
 Sofge, D.A., 400
Sommers, H.J., 425
 Sompolinsky, H., 157,397,399,424,
 425
Sontag, E.D., 133, 376, 377, 425
Soukoulis, C.M., 173, 425
Soulie, F.F., 415
 Specht, D.F., 255, 425
 Steams, S.D., 364,371, 430
 Steinbuch, K., 22, 65, 425
Stornetta, W.S., 267, 426
Strang, G., 92, 426
Sudha, N., 379, 390,426
 Suen, C.Y., 389, 430
 Sutton, R., 22, 59, 63, 64; 134, 401,
 426
Swindlehurst, A.L., 427
 Szu, H., 65, 411, 426

T

Tan, C.L., 333, 426
Tanaka, H., 412
Tank, D.W., 267, 299, 411, 426
 Taylor, J.G., 386, 389, 426
Teh, H.H., 426
 Teller, A.H., 417
 Teller, E., 417
 Tesauro, G., 430
Tesi, A., 134, 407
Thorbergsson, G.I., 409
Tikhonov, A.N., 22,132,242,426,427
 Tishby, N., 424
 Todd, P.M., 417
'Ibrkkola, K., 309, 427
'Ibrre, V., 421
'Iburetzky, D.J., 412
'Iburetzky, D.S., 400, 406, 408, 413,
 415, 419, 429
Tsao, E.C.K., 394, 427
 Tufts, D.W., 390, 427

U

Ungar, L.H., 334, 427

Utela, J., 427
 Uttley, A., 22, 427

V

Valiant, L.G., 135, 375, 427
 van der Veen, A., 390, 427
 van **Laarhoven**, P.J.M., 193, 427
Vapnik, V., 375, 427
 Vassilas, N., 293, 404
 Vecchi, M.P., 413
 Vidyasagar, M., 135, 427
 Visa, A., 324, 428
 von der Malsburg, C., 22, **221**,
 428
 von **Neumann**, J., 22, 428
Vrckovnik, G., 389, 428

W

Wadell, G., 402
 Wahba, G., 248, 428
 Waibel, A., **267**, 311, 312, 428
 Wake, N., 407
 Walsh, M.J., 406
 Wang, C., 381, 428
 Wang, J., 378, 412
 Wang, L.X., 394, 397, 428
 Wang, Y.F., 239, 428
Warmuth, M.K., 402
 Wasserman, P.D., 65, 134, 135, 255,
 397, 429
 Weigend, A.S., 269, 429
 Weiner, N., 22, 429
 Weisbuch, G., 415
Wenocur, R.S., 377, 429
Werbos, P.J., 22, 117, 429
 Werner, M., 412
 Wessels, L.F.A., 127, 429
 White, D.A., 400
 White, H., 255, 429
 Whitehead, B.A., 392, 429
Widrow, B., 22, 23, 28, 97, 305, 364,
 371, 419, 429, 430
 Wieland, A., 132, 430
 Wiesel, T.N., 224, 411
 Williams, R.J., 22, 64, 128,417, 423,
 426, 430
 Willshaw, D., 22, 300, 406, 430

Author Index

Wilson, G.V., 299, 430
Wolf, R., 324, 430
Wong, S.K.M., 420
Wu, L., 131, 425

X

Xu, L., 389, **430**

Y

Yan, W.Y., 381, 430
Yao, X., 392, 430
Yao, Y., 397, 430
Yegnanarayana, B., 6, 281, 286, 290,
308, 313, 326, 333, 336, 392, 393,
396, 403, 419, 421, 422, 424, 430,
481
Yoon, Y.O., 334, 431

Yorke, J.A., 419
Yovitz, M.C., 429
Yuan, B., 393, 396, 413
Yuille, A., 293, 413, 431

Z

Zadeh, L.A., 393, 431
Zaghloul, M.E., 411
Zapranis, A.D., 334, 431
Zbikowski, R., 411
Zhang, J., 324, 424
Zhang, Q., 381, 431
Zheng, G.L., 392, 401
Zhu, J., 380, 419
Ziarko, W., 420
Zimmermann, H.G., 409
Zipser, D., 221, 423, 430
Zurada, J.M., **27, 31, 35, 67, 100, 240,**
271, 276, 306, 338, 431

Subject Index

2

2D convolution, 325

A

A posteriori probability, 122, 131, 133, 251, 324, 358
maximum (MAP), 329

A priori, 358

A priori knowledge, 244, 394

ABAM theorem, 71

Absolute value distance, 362

Absolutely stable, 69

Absolutely stable states, 172

Accretive, 6, 77, 85, 100, 146

Acoustic features, 306

Acoustic-phonetic knowledge, 316

Activation

state, 24, 40

value, 24

vector, 90

Activation and synaptic dynamics, 40–73

distinction, 54

Activation dynamics, 1, 25, 40, 143

additive models, 44–47

shunting models, 48–52

stochastic models, 51

Actual output, 91, 96

Adaline, 28

learning, 29

model, 23, 29

Adaptive

autoassociative network, 70

BAM (ABAM), 71, 239

clustering, 262

formation of desired weights, 97

heteroassociative network, 71

learning, 97

principal component extraction (APEX), 383

resonance theory (ART) (see ART models)

vector quantization (see Vector Quantization)

Additive activation **dynamics** models, 44–47

autoassociative, 46

heteroassociative, 46

summary, 47

Adjoint differential operator, 249

Afferent, 271

Affine transformation, 99

AI (see Artificial Intelligence)

Algorithmic approach, 4

Alternate proof, 104

All-Class-One-Network(**ACON**), 313

Alveolar, 314

Analog patterns, 239, 262

Analog VLSI, 23

Analysis of

competitive learning network, 202–211

feature mapping network, **223–228**

feedback layer, 211–218

pattern association networks, 90–99

pattern classification networks, **99–113**

pattern clustering networks, 218

pattern mapping networks, **113–135**

self-organization network (see feature mapping network)

Animal cortex, 397

ANN models (see basic **ANN** models)

Annealing

deterministic, 194, 330

fast, 192

Subject Index

- mean-field, 196
- simulated, 165, 296
- stochastic, 880
- Annealing process, 178
- Annealing schedule, 179, 181, 190, 192, 298
- Antiformants, 806
- Anti-Hebbian learning, 381, 384
- Antiresonances, 306
- Application areas, 279, 306–333
 - decision making, 333
 - image processing, 321
 - speech, 806
- Applications level, 11, 13, 886
- Applications of ANN, 278
 - application areas, 279, 306
 - direct application, 279, 280
 - organization of topics, 279
- Approximate mapping, 114
- Approximate representation of environment, 184
- Approximation of function, 114, 122
- Approximation of MLP, 242
- Approximation to desired weight matrix, 97
- Architecture, 183, 378
 - ANN, 2
 - ART, 258
 - BAM, 236
 - MAM, 239
 - TDNN, 311
- Architecture level, 12, 233, 886
- Architectures for
 - complex PR tasks, 233
 - pattern variability, 271
 - temporal pattern recognition, 265
- ART models, 22, 258
 - ART1, 259, 262
 - ART2, 262
 - ARTMAP, 262
 - fuzzy ARTMAP, 262
- Articulators, 806
 - jaw, 806
 - lips, 806
 - tongue, 806
 - velum, 806
- Articulatory parameters, 306
- Artificial ant system, 335
- Artificial Intelligence (AI), 1, 20
 - limitations, 20
 - systems, 2
- Artificial Neural Networks (ANN), 1, 20, 99
 - architectures, 2
 - functional units, 1
 - limitations, 20
 - models (*see* basic ANN models)
- Artificial neuron, 26, 997
 - models, 26
- Artificial olfactory model, 997
- Aspirated, 312
- Associative reward-penalty learning, 64
- Association
 - auto, 77
 - hetero, 77
 - units, 27
- Associative memory, 21, 235, 262, 279, 291
 - auto, 31, 236
 - bidirectional (BAM), 31, 47, 236
 - capacity, 239
 - hetero, 236
 - linear, 90, 236
 - multidirectional (MAM), 289
 - recurrent, 267
 - temporal, 240
- Asymmetric PCA, 886
- Asymptotic approximation, 131
- Asynchronous update, 150
- Attentional subsystem, 259
- Attractors, 897
- Auditory processing mechanism, 307
- Augmented
 - input vector, 101
 - weight vector, 101
- Autoassociative memory, 31, 236
 - desirable characteristics, 236
- Autoassociation, 76
 - problem, 83
 - task, *ldd*
- Autocorrelation matrix, 116, 127, 201, 365
 - eigenvalues, 127, 207, 365
 - eigenvectors, 207, 365
 - properties, 365
- Automated inspection, 333
- Autonomous dynamical system, 41
- Avalanche structure, 266
- Average
 - clamped free energy, 194
 - energy, 178
 - value of output, 168

- Axon, 16
 Axon hillock, 42
 BAM, 236
- B**
- Backpropagation, 117, 392
 applications, 134
 batch mode, 127
 convergence, 122, 125
 generalization, 122
 gradient descent, 117
 learning rate, 122, 127
 local minimum, 122, 132
 momentum, 122, 129
 network, 134
 output function, 124
 pattern mode, 127
 recurrent network, 270
 scaling, 122
 stopping criterion, 121, 126
 through time, 270
 weight change, 180
 weight initialization, 181
 Backpropagation learning, 23, 117, 121
 algorithm, 121
 description and features, 123
 discussion, 120
 issues, 121
 summary, 121
 Backward propagation, 271
 Bandwidth, 325
 Basic ANN models, 76
 Basic competitive learning (see competitive learning)
 Basic learning laws, 31, 67
 discussion, 35
 summary, 35
 Basics level, 11, 336
 Basis of attraction, 69, 147, 157, 292
 Basis function
 Gaussian, 254
 linear, 246
 nonlinear, 245
 radial, 22, 245
 Basis function formulation, 252
 classification problem, 251
 function approximation, 247
 Bayes classification, 255
 Bayes theorem, 248, 328, 359
 Behavioural changes, 391
 Bernoulli trials, 360
 Bias input, 99, 245
 Bias of unit, 319
 Bibliographic information, 292
 Bidding
 card game, 290
 problem, 280, 290
 sequence, 280
 Bidirectional associative memory (**BAM**), 31, 47, 236
 Bilabial, 314
 Binary
 input, 106, 281
 output, 106
 output function, 158, 206
 pattern, 237
 relation, 395
 Binomial distribution, 152, 360
 Biological neural network, 1, 15, 19, 233, 396
 features, 15
 Biological system, 391
 Bipolar
 output function, 107
 patterns, 151, 237
 states, 150, 294
 Bit pattern, 241
 Blind separation, 390
Boltzmann-Gibb's distribution, 170
 Boltzmann-Gibb's law, 186
 Boltzmann learning, 65, 184, 191
 algorithm for implementation, 191
 algorithm for recall from partial input, 192
 Boltzmann learning law, 143, 184
 implementation, 188
 issues, 190
 summary, 189
 Boltzmann Machine (**BM**), 22, 143, 183
 architecture, 183
 Boolean functions, 138, 241
 linear threshold, 138
 polynomial, 138
 quadratic, 138
 Bottom-up competitive learning, 258
 Bottom-up connections, 259
 Bounded output, 216

Subject Index

Bowl shape, 366
Brain State-in-Box (**BSB**), 22
Brain vs. computer, 19
Brittleness, 1
Building blocks, 76

C

C-cells, 272
CV units, 310
CV utterances, 280
Capacity, 148
 BAM, 239
 Hopfield model, 151
 memory, 236
 storage, 151
Categories, **53**
 input patterns, 207
 learning, 53, 234
 learning laws, **53**
Category learning tasks, 258
Cauchy machine, 192
Cauchy-Schwartz inequality, 103
Cell body, 16
Centroids of clusters, 246
Cepstral coefficients, 310
Chaos, 391, 396
Chaotic
 associative memory, 397
 attractors, 397
 behaviour, 157, 397
 dynamics, 396
 nature, 335
 neurons, 335
 regions of equilibrium, **148**
 stability, 69, 397
 state regions, 157
 variables, 397
Character recognition, 134
Chebyshev distance, 362
Chromosomal operators, 391
City block distance, 362
Clamped
 free energy, 190, 194
 input unit, 346
 phase, 191
 units, 186
Class **label**, 242, 251
Classical set theory, 393
Classification problem, 243, 251
Closeness of features, 147
Closeness property, 242, 252
Club in card game, 290
Cluster centres, 256
Cluster spread, 256
Clustering
 algorithm, 253
 analog signals, 262
 binary patterns, 262
 network, 394
Co-occurrence patterns, 346
Coarticulation, 307
Code compression, 262
Codebook, 304
Cohen-Grossberg theorem, 70
Cohen-Grossberg-Koskotheorem, 71
Collective computation, 16
Combination network, 202
Combination of SOM and classification, 229
Combining evidence, 316
Combining multiple evidence, 321
Committed units, 260
Common features, 241
Common sense knowledge, 4
Communication theory, 380
Competitive **dynamical** systems, 69
Competitive layer, 202, 211
Competitive learning, 60, 201, 218
 deterministic, 60
 differential, 61
 leaky learning, 221
 linear, 61
 linear differential, 62
 methods of implementing, 219
 minimal learning, 221
 random differential, 62
 random linear, 61
 random linear differential, 62
 standard, 220
Competitive learning algorithm, 220, 304
Competitive learning methods, 222
 summary, 222
Competitive learning neural networks (**CLNN**), 76, 201, 219
 components, 203
Complex
 artificial neuron model, 397
 decision surfaces, 133
 nonlinear mapping, 53

- Complexity of
 - problem, 122
 - surfaces, 241
 - symbol set, 284
- Compression of data, 304
- Computation of weights, 91, 161
- Computational learning, 377
- Computer and brain, 19
 - fault tolerance, 19
 - processing, 19
 - size and capacity, 19
 - speed, 19
 - storage of information, 19
- Computer architecture
 - von **Neumann**, 4
- Computer memory, 341
- Computer technology, 1
- Concepts of rooms, 347
- Conditional
 - expectation, 255
 - probability, 248, 359
 - probability distribution, 132
- Conductance, 18
- Confusable set, 312
- Conjugate gradient method, 121, 130
- Connectedness** problem, 110
- Connectionist expert system, 333
- Connections
 - excitatory, 18, 317
 - feed forward, 384
 - inhibitory, 18, 317
 - lateral, 384
- Connectivity of neurons, 18
 - number of synapses, 18
- Consonant-Vowel (**CV**) segment, 279, 310
- Constant error contours, 366
- Constrained optimization problem, 357
- Constrain satisfaction (**CS**) model, 312, 316, 344
 - initialization, 315
 - operation of, 315
 - performance, 315
- Content addressable memory, 6, 31, 236, 291, 344
- Context units, 268
- Continuous
 - Hopfield** model, 155
 - nonlinear function, 112
 - nonlinear output function, 155
 - perceptron learning, 33, 63
 - time recurrent network, 271
- Contract Bridge, 280
- Control application, 279, 305, 394
- Control in learning, 259
- Controller, 271
- Convergence, 68
- Convergence in the mean, 117, 371
- Convex
 - hull, 107
 - region, 110
 - set, 107
- Convolutional layer, 322
- Convolutional networks, 322
- Correlation, 188
 - learning law, 33, 68
 - matching, 281, 284
 - term, 55
- Cost function, 131, 293
- Counterpropagation**, 22, 256
- Counterpropagation networks (**CPN**), 256
- Coupled nonlinear differential equations, 296
- Covariance matrix, 66, 208, 253, 319, 361, 380
- Credit assignment, 22, 64
 - fixed, 64
 - probabilistic, 64
 - problem, 22
 - structural, 64
 - temporal, 64
- Credit scoring, 334
- Crisp logical operators, 394
- Criteria for grouping, 314
 - SCV** classes, 312, 314
- Critical temperature, 174
- Crosscorrelation, 362
- Cross entropy measure, 363
- Cross-validation, 122, 134, 243, 373
- Crosscorrelation matrix, 385
- Crosscorrelation neural networks, 385
- Crosscoupled Hebbian **rule**, 385
- Cultural evolution, 335
- Current trends in NN, 391-397
 - chaos, 396
 - evolutionary computation, 391
 - fuzzy logic, 393
 - rough sets, 395
- Cursive script, 8, 323

Subject Index

- Cursive writing, 323
- Curve fitting, 389
- Cybenko** theorem, 133
- Cybernetics, 22
- Cycle, 123, 190, 344

- D**
- Darwinian evolution, 391
- Data, 5
- Data compression, 258, 389
- Data-dependent transformation, 380
- Decision
 - boundaries, 102
 - criterion, 321
 - regions, 112
 - surfaces, 121
- Decision making, 279, 333
- DECTalk**, 308
- Decay term, 45, 47, 55, 58, 205
- Decoding scheme, 234
- Decorrelation, 389
- Deep energy minima, 176
- Deep features, 244
- Default assignment, 344
- Deformation, 280
- Deformed patterns, 271
- Delay units, 265
- Delta learning, 32, 63, 68, 117, 123
- Delta-bar-delta learning, 128
- Delta-delta learning, 128
- Dendrites, 16
- Dental, 314
- Derivative measurement, 117
- Designing energy minima, 164
- Desired
 - input patterns, 188
 - mapping, 115
 - output, 91, 115
- Determination of weights by
 - learning, 94
 - matrix inversion, 254
- Deterministic
 - analog states, 295
 - Boltzmann machine, 190
 - case, 59, 172
 - chaos, 396
 - learning, 54
 - modelling, 324
 - relaxation, 299
 - update, 23, 164
- Device technology, 2
- Diameter-limited, 109
- Diamond in card game, 290
- Dichotomy, 138, 375
 - number of distinct dichotomies, 376
- Differentiable nonlinear output function, 115
- Differential Hebbian learning, 60
- Differential operator, 248, 249
- Dimensionality of input pattern, 88, 99
- Dimensionality reduction, 380
- Direct application, 279, 280
- Discontinuities in images, 301
 - 1-D** case, 302
 - 2-D** case, 303
- Discrete
 - BAM, 236
 - binary cube, 351
 - bipolar cube, 351
 - Hopfield** model, 152
 - N-dimensional space, 99
 - perceptron learning, 32, 63
- Discrimination between patterns, 243
- Disjoint subsets, 375
- Dissimilar patterns, 262
- Distorted patterns, 272
- Distributed memory, 344
- Dividing surface, 111
- Distribution
 - Binomial, 152, 360
 - Boltzmann-Gibbs, 170, 188
 - Gaussian, 152, 360
- Domain expert, 333
- Domain-specific knowledge, 325
- Dominating features, 241
- Drive-reinforcement learning, 23, 66
- Dynamic
 - equilibrium, 167
 - matching, 259
 - sounds, 310
- Dynamically expanding context algorithm, 309
- Dynamics
 - activation (**see** activationdynamics)
 - chaotic, 396
 - neural, 396
 - plant, 269
 - synaptic (**see** synaptic dynamics)

Subject Index

E

Effective **energy**, 296
Efferent, 271
Eigendecomposition, 355, 390
Eigenvalues, 93, 127, 144, 207, 355, 365
Eigenvectors, 144, 207, 298, 355, 365
Elastic ring method, 300
Encoding scheme, 234
Energy
 analysis, 23, 152
 function, 152, 156
 landscape, 152, 147, 176
 minima, 153
 of stable state, 161
 of state, 142, 147
English phoneme, 308
English syllables, 269
Ensemble of solutions, 391
Entropic error, 374, 377
Equilibrium, 51
 state, 23, 69
 statistics, 188
 stochastic networks, 167
Equivalence classes, 395
Equivalence relation, 395
Error
 criterion, 185, 188
 function, 131, 152
 in pattern recall, 145, 165
 rate measure, 373
 second derivative, 367
 signal, 107
 surface, 116, 125, 366
Error backpropagation, **121**
Error-based learning, 262
Error correction learning, 62, 115
 stochastic approximation, 62
Estimate of desired function, 248
Estimated noise level, 98
Estimation of gradient, 117
Estimation of probability distribution, 134, 244
Euclidean distance, 222, 362
Euclidean norm, 353
Even parity, 241
Event, 357
Evolutionary
 computation, 334, 391
 programming, 391
 strategies, 391

Exchange rate forecasting, **334**
Excitation signal, 306
Excitatory
 connection, 317
 feedback, 49
 weights, 24
Expectation, 359
Expert system, 2, 333
Exploitation, 392
Exploration, 392
Exponential kernel representation, 267
Exponentially decaying memory, 268
Extended Kalman-type algorithm, 122
Extensions of backpropagation, 121, 134
External noise, 99

F

Fascimile, 304
False energy minima, 143, 163
Family of local predicates, 108
Fast annealing schedule, 192
Fast learning procedure, 194
Fault detection, 334
Fault tolerance, 5, 16, 19, 236
Feature
 extraction, 285, 389
 formation, 325, 326
 mapping, 7, 76, 202, 223, 226
 1D to **1D**, 226
 2D to **1D**, 226
 2D to 2D, 226
 network, 223
 problem, 86
Feature space, 133, 202, 223
Feature-label interaction, 325
Features for pattern recognition, 8
Features of biological NN, 15, 341
Feedback networks, **23, 76, 142, 294,**
 316
 control, 305,
 desired information, 158
 global pattern **behaviour**, 158
 layer, 210
 retrieval of information, 158
 storage capacity, 53, 151, 157
Feedforward and feedback, 76, **201**

Subject Index

- Feedforward** neural networks, **76, 88**
 - analysis, 88
 - summary of pattern recognition tasks, 73, 89
 - Financial markets, 269
 - Finite state machine, 271
 - Finnish language, 309
 - Firing, 17
 - First order statistics, 253
 - Fixed
 - delays, 269
 - point, 157, 397
 - point equilibrium states, 69
 - points of equilibrium, 148
 - Flip-flop, 271
 - Folding operation, 397
 - Forced units, 186
 - Forecasting
 - applications, 334
 - situations, 269
 - Forgetting term, 56, 189
 - Formant contour, 8
 - Formant data, 309
 - Formants, 8, 306
 - Forward mapping, 258
 - Forward propagation, 271
 - Fractal-like structure, 397
 - Fraud detection, 334
 - Free energy of a system, 171
 - Free parameters of a network, 133
 - Free running, 185, 191
 - Frequency estimation
 - noise subspace, 390
 - principal components, 390
 - Full energy, 190
 - Full free energy, 194
 - Fully recurrent networks, 269
 - Function approximation, 114, 121, 133, 243, 246, 247
 - linear basis functions, 246
 - radial basis functions, 246
 - Function estimation, 244
 - Functional level, 12, 336
 - Functional relation, 88
 - Functional units, 76
 - pattern recognition tasks, 76
 - Fuzzy
 - adaptive resonance theory, 394
 - ARTMAP, 262
 - backpropagation learning law, 132
 - backpropagation networks, 135
 - learning, 54
 - logic, 23, **264**, 334, 335, 391, 393
 - logic connectives, 394
 - logical operators, 394
 - nature of output, 291
 - neural networks, 122
 - numbers, 393
 - objective function, 393
 - representation, 132
 - sets, 393
 - uncertainty, 393
- ## G
- Gabor** filter, 325
 - bandwidth, 325
 - complex sinusoidal grating, 325
 - Gaussian function, 325
 - oriented, 325
 - Gain control process, 259
 - Gain control unit, 261
 - Gain parameter, 156, 261
 - Games, 4
 - Gaussian basis function, 247, 254
 - Gaussian distribution, 152, 251, 360
 - mixture, 251, 361
 - mixture models, 326
 - multivariate, 249, 326
 - univariate**, 360
 - Generalization, 7, 88, 121, 133, 241, 372
 - bound, 374
 - capability, 248
 - concept, 372
 - error, 374, 377
 - feature, 126, 308
 - in NN, 372, 377
 - in pattern recognition tasks, 372
 - measure, 389
 - studies, 373
 - Generalization for classification, 241
 - mapping, 242
 - Generalization in **FFNN**, 377
 - architecture, 378
 - learning algorithm, 378
 - stopping criterion, 378
 - training set, 378

- Generalized
 - delta rule, 23, 63, 117, 123
 - Hebbian law, 210, 382
 - regression NN, 255
 - Genetic
 - algorithms, 391
 - programming, 391
 - Geometric momenta, 285
 - Geometrical arrangement of units, 223
 - Geometric interpretation, 80
 - PR** tasks by CLNN, 85
 - PR** tasks by FBNN, 83
 - PR** tasks by FFNN, 80
 - Geometrical interpretation of hard problems, 110
 - Geometrically restricted regions, 109
 - Gibb's distribution, 325
 - Global
 - behaviour of **ANN**, 68
 - energy, 169
 - features, 272
 - knowledge, 247
 - Lyapunov function, 71
 - minimum, 125
 - pattern behaviour, 158
 - pattern formation, 69
 - searching, 391
 - stability, 44
 - structure, 321
 - Gobally** stable, 69
 - Goodness-of-fit function, 346, 347
 - Goodness-of-fit surface, 348
 - Graceful degradation, 344
 - Gradient descent methods, 116, 125, 364
 - convergence issues, 125
 - LMS** algorithm, 117, 251, 371
 - Newton's method, 116, 130, 368
 - steepest descent, 368
 - summary, 116, 371
 - Gradient
 - error, 107
 - error measure, 192
 - mean-squared error, 365
 - quadratic form, 364
 - reuse method, 128
 - search, 116, 367
 - Gram-Schmidt **orthogonalization**, 382
 - Graph-bipartition problem, 279, 296
 - Graphs, 296
 - Green's function, 249
 - Group of **instars**, 30, 202, 206
 - Group similar patterns, 262
 - Grouping of **SCV** classes, 314
 - Growth function, 376
- ## H
- Hamming distance, 147, 150, 164, 241, 362
 - Hamming network, 281
 - Hand-drawn figures, 271
 - Hand-printed characters, 7, 86, 99
 - Hand-written characters, 271, 288, 321
 - Hard
 - classification, 244
 - leaning problem, 88, 113, 165
 - pattern storage problem, 164
 - problem, 88, 108, 142, 149, 164, 183, 241
 - Hard-limiting threshold function, 48, **100**
 - Hard-limiting threshold units, 108
 - Harmonic decomposition, 390
 - Heart in card game, 290
 - Hebb's law, 18, 21, 32, 95, 150, 173
 - Hebbian learning, 57, 58, 188, 381, 384
 - stochastic version, 59
 - Hebbian unlearning, 188
 - Hessian matrix, 129, 355
 - Heteroassociation, 6
 - Heteroassociative memory, 236
 - Heteroassociative network, 31, 236
 - Heuristic search methods, **1**
 - Hidden layer, 88, 114
 - Hidden Markov model, 309, 324
 - Hidden **units**, 23, 143, 165, 183, 260, 343
 - Hierarchical structure of visual system, 271
 - Hierarchical structures, 391
 - Higher order
 - connections, 387
 - correlation learning network, 386
 - neuron model, 386
 - statistical momenta, 387
 - statistics, 386
 - terms, 129, 155
 - unit, 386

Subject Index

- Hindi, 312
Hinton diagram, 346
Historical development, 21–24
History of neurocomputing, 15
 Table 1.1, 22
Hodgkin-Huxley cell equations, 50, 397
Hopfield model, 23, 149, 188
 algorithm, 151
 continuous, 149
 discrete, 152
 energy analysis, 143
 energy equation, 170
 energy function, 294
Human
 information processing, 11
 memory, 341
 players, 280
 reasoning, 290
Hyperbolic tangent function, 124
Hypercube, 157, 351
 area, 352
 volume, 352
Hyperellipse, 366
Hypersphere, 352
Hypersurface, 110
Hypothesis, 345
Hypothesis space, 375
- I**
- IAC** model, 293, 341
Identity matrix, 92
If-then rules, 333
Ill-posed problem, 132, 242
 solutions, 132
Image
 degradation, 283
 lattice, 303
 pattern recall, 279, 292
 segmentation, 280, 321, 323
 smoothing, 279, 301
Image pixels, 303, 321
 global structure, 321
 local structure, 321
Image processing, 280, 321
Image-specific constraints, 325
Immunity net, 335
Implementation of Boltzmann learning, 188
 issues, 188
Implicit pattern **behaviour**, 126
Independent component analysis, 387
Independent events, 359
Indian languages, 280, 312
Individual level, 391
Inferencing mechanism, 4
Information
 access, 293
 preservation, 22
 retrieval, 279, 293
 theoretic measure, 184, 188
 theory, 22
Inhibitory, 18
 connection, 317
 external input, 46
 feedback, 46, 49, 211
 weights, 24
Initial state, 148
Initial weights, 126, 190, 191
Inner product, 154, 352
Input
 dimensionality, 143
 layer, 90, 203
 matrix, 90
 vector, 90
Input-output pattern pairs, 88, 242
Instance pool, 317, 343
Instantaneous error, 62, 126, 129, 255
Instar, 30
 group of **instars**, 31, 202
 learning law, 34
 network, 202
 processing, 206
 steady activation value, 205
 structure, 257
Integer programming problem, 298
Intelligence, 2, 4
Intelligent decision, 333
Intelligent tasks, 2
Intensity-based methods, 324
Interactive and competition (IAC), 293, 341
Intercity distances, 298
Interconnections, 24
Intermediate layers, 114
Interneuron, 17
Interpolating function, 248
Interpolative, 7, 77
Interpolative recall, 73

- Interpretation of Boltzmann learning, 190
 - Intersection of convex regions, 111
 - Intonation, 307
 - Invariance
 - by structure, 285
 - by training, **285**
 - Invariant
 - feature extraction, 285
 - measures, 285
 - pattern recognition, 284
 - Inverse **Kronecker** delta function, 327
 - Inverse mapping, 258
 - Investment management, 333
 - Ising model, 22
 - Issues in Boltzmann learning, 190
 - annealing schedule, 190, 192
 - implementation of simulated annealing, 190
 - initial weights, 191
 - learning and unlearning, 190
 - learning pattern environment, 190
 - learning rate parameter, 191
 - local** property, 190
 - recall of patterns, 191
 - Iteration index, 119
- J**
- Jacobian matrix, 353
 - Jaw. 306
- K**
- Kalman-type** learning, 131
 - Karhunen-Loeve** transformation, 380
 - Knowledge-based systems, 9
 - Kohonen learning, 223, 225
 - algorithm** for implementation, 226
 - Kohonen mapping, 223
 - Kronecker delta function, 327
 - Kullback-Leibler measure, 363, 373
- L**
- LMS algorithm, 370
 - convergence, 371
 - learning rate parameter, 371
 - trajectory of path, 371
 - Label competition, 325, 326
 - Label-label interaction, 326
 - Lagrange** multipliers, 357
 - Laplace** transform, 305
 - Layers** of processing units, 29
 - Leaky learning law, 221
 - Learning laws, 31, 53
 - algorithm for multilayer FFNN, 117
 - algorithms for PCA, 210
 - anti-Hebbian, 384
 - associated reward and penalty, 64
 - asymptotic behaviour, **377**
 - backpropagation, 121
 - Boltzmann, 189
 - competitive, 222
 - correlation, 33, 68
 - curve, 377
 - delta, 32, 68
 - equation, 31
 - from **examples**, 372
 - function, 66
 - Hebb's**, 32, 67
 - leaky, 221
 - Linsker, 231
 - LMS, 33
 - machine, 22, 374
 - methods, 57
 - models, 374
 - Oja**, 208
 - online, 271
 - pattern environment, 190
 - perceptron, 32, 68
 - principal subspace, 382
 - rate parameter, 89, 97, 127, 221
 - reinforcement, 63
 - Sanger, 209
 - supervised, 32
 - temporal, 54
 - theory, **375**
 - unsupervised, 32
 - Widrow-Hoff, 33, 68
 - with **critic**, 63
 - with teacher, 63
 - Learn matrix, 22
 - Learning vector quantization (**LVQ**), 222, 305
 - Learning with critic, 63, 122
 - Learning with teacher, 63
 - Least Mean Square (**LMS**) learning, 22

Subject Index

- Least square problem, 356
 - Length of sequence, 265
 - Level of species, 391
 - Levels of ANN research
 - application level, 11, 336
 - architectural level, 11, 886
 - basics level, 10, 886
 - functional level, 11, 336
 - problem level, 10, 336
 - Levels of bidding, *Z90*
 - Limitations of backpropagation learning, 121, 134
 - Limitations on local predicates, 109
 - diameter-limited, *I09*
 - order-limited, *I09*
 - Line minimization, 130
 - Linear Algebra, 352
 - Linear associative network, 90
 - Linear autoassociative networks
 - FB network, 146
 - FF networks, 144
 - Linear
 - basis function, 246
 - dependence, 352
 - differential operator, 248
 - equations, 356
 - overdetermined, 356
 - underdetermined, 356
 - hyperplane, 101, *I06*
 - independence, 93, 352
 - inseparability, 108
 - processing units, 88, 143
 - programming problem, 279
 - separability, 88, 107
 - transformations, 99
 - units in feedback layer, 212
 - vector quantization, 305
 - Linearity of output function, 91
 - Linearly
 - independent inputs, 143
 - independent patterns, 88
 - independent vectors, 93
 - separable classes, 99, 102
 - separable functions, 99, 108
 - separable problems, 110
 - Linguistic
 - critic signals, 394
 - form of input, 334
 - information, 393
 - message, 307
 - terms, 893
 - Lips, 806
 - LMS algorithm, 63, 117, 251, *E10*
 - LMS learning law, 29, 33
 - Loading of pattern environment, 197
 - Loading problem, 122
 - I08I*
 - computation, 121
 - features, 272
 - information, 53
 - minima problem, 121, 132, *I78*
 - predicate, *I08*
 - property, 188
 - receptive field, 256, 322
 - structure, 321
 - Logic circuits, *Z1*
 - Logic functions, *Z1*, 38, 122, 134
 - Logical computation, 21
 - Logical inference, 5
 - Logical predicate, 108
 - Logistic function, 124
 - Long term memory (LTM), 25, 212
 - Lower subnet, 282
 - low-pass filter, 370
 - Lyapunov energy function, 55, 70, 157, 238
- ## M
- MLFFNN, 310, 396
 - Machine-printed characters, 90
 - Mahalanobis distance, 362
 - Manner of articulation, 312, 313
 - Mapping function, 242
 - Markov property of simulated annealing, 190, 193
 - Markov random fields, 324
 - Masking of features, 241
 - Match-based learning, 262
 - Matching probability distributions, 178
 - Mathematical preliminaries, 351
 - Matrix identities, 92
 - Matrix transformations, 239
 - Maximum
 - eigenvalue, 208
 - likelihood, 53
 - likelihood classifier, 281
 - value distance, 362
 - McCulloch-Pitts model, 26
 - McCulloch-Pitts neuron, 21

- Mean of input data, 253
 - Mean squared error, 53, 91, 365
 - Mean-field
 - algorithm, 196
 - annealing, 195
 - approximation, 172, 195, 295
 - energy, 195
 - free energy, 195
 - Medical diagnosis, 333
 - Mel-scale, 310
 - Membership function, 393
 - Membrane
 - capacitance, 45
 - potential, 17, 42
 - resistance, 45
 - Memorizing, 7
 - Memory
 - content addressable, 21
 - long term, 25
 - short term, 21, 25
 - Memory function, 235
 - Meshed regions, 111
 - Metric distance measures, 362
 - absolute value distance, 362
 - Chebyshev distance, 362
 - city block distance, 362
 - Euclidean distance, 362
 - Hamming distance, 362
 - maximum value distance, 362
 - Minkowski** r-metric, 362
 - Metric transformation, 284
 - Metropolis algorithm, 190, 295
 - Mexican hat function, 224
 - Min-max learning, 65
 - Minimal
 - ART, 262
 - learning, 221
 - Minimum error, 93, 146
 - Minimum error retrieval, 93
 - Minimum norm solution, 356
 - Mismatch of probabilities, 184
 - Mixture distribution (see Gaussian mixture)
 - Models of
 - activation dynamics, 42
 - computing, 1, 15
 - neural networks, 41
 - neuron, 26
 - synaptic dynamics, 52
 - Modular approach, 312, 313
 - Modular architecture, 134
 - Modular network, 312
 - Momentum constant, 129
 - Momentum term, 121
 - Monotonically increasing function, 156
 - Monte **Carlo** method, 194
 - Motor neuron, 17
 - Multilayer feed forward neural network (**MLFFNN**), 88, 114
 - Multiclass problem, 106
 - Multidimensional patterns, 110
 - Multidirectional associative memory (**MAM**), 236, 239
 - Multidirectionally stable, 240
 - Multilayer perceptron (**MLP**), 110, 113, 133, 241
 - Multilevel network hierarchy, 262
 - Multiple associations, 239
 - Multiple binary output units, 100
 - Multiple principal component extraction, 382
 - Multispectral band imagery, 331
 - Multivariate function approximation, 244
 - Multivariate Gaussian function, 249, 326
 - Murakami result, 94, 145
 - Mutual Hebbian rule, 385
 - Mutually exclusive events, 359
 - Mutually orthogonal vectors, 96
- N**
- N-dimensional
 - Euclidean geometry, 351
 - space, 157
 - Nasal tract, 306
 - Natural language processing, 1
 - Nearest **neighbour**
 - recall, 73
 - stored pattern, 72
 - Negative definite matrix, 354
 - Negative reinforcement, 63
 - Negative **semidefinite** matrix, 364
 - Negative gradient, 107
 - Neighbouring pixel interaction, 324
 - Neighbouring units, 223
 - Neocognitron**, 22, 271, **323**
 - NETtalk**, 280, 307
 - Nerve fibres, 16

Subject Index

- Neural network
 - architectures, 235
 - feedback, 142
 - feedforward, 88
 - models, 41
 - recall, 72
 - Neuro-evolutionary techniques, 335
 - Neuro-fuzzy systems, **335**
 - Neuro-rough synergism, **335**
 - Neuron
 - firing, 17
 - number in brain, 18
 - structure of, 16
 - Neurotransmitter, 18
 - Newton's method, 116, 130, 367
 - Noise
 - cancellation, 389
 - power, 94
 - subspace, 390
 - suppression, 216
 - vector, 93
 - Noise-saturation dilemma, 43, 204
 - Noisy
 - image, 285
 - input, 93
 - pattern, 193
 - Nonautonomous dynamical system, 41
 - Nonconvex regions, 111
 - Nonlinear
 - basis function, 245, 255
 - convolution, 322
 - dynamical systems, 70, 269
 - error surface, 134
 - feature detector, 121
 - feature extraction, 133
 - filters, 318
 - hypersurfaces**, 241
 - optimal** filtering, 131
 - output function, 100, 131
 - PCNN**, 387
 - plant dynamics, 269
 - processing units, 88, 99, 143
 - regression, 255, 333
 - system identification, 122, 131
 - Nonlinearly separable classes, 241
 - Nonparametric nonlinear regression, 334
 - Nonparametric regression problem, 244
 - Nonquadratic error surface, 130
 - Nonstationary input, 117
 - Normal** distribution (see Gaussian distribution)
 - Normalization of **features**, 285
 - Normalized basis function, 252
 - Normalized radial distance, 245
 - Normalizing the weight, 208
 - Notrump** in card game, 290
 - Number of
 - cycles, 191
 - linearly separable classes, 107
 - linearly separable functions, 108
 - trials, 191
- O**
- Objective function, 293
 - Odd parity, 241
 - Oder-limited, 109
 - Off-line learning, 54
 - Oja's** learning, 208, 381
 - Oja's p-unit** rule, 209
 - Olympic game symbols, 280
 - On-centre and off-surround, 48, 202
 - One-Class-One-Network (OCON), 313
 - On-line learning, 54, 271
 - Opening bid in card game, 280, 290
 - Operating range, 43
 - Operation of **ANN**, 1
 - Operation of stochastic network, 175
 - Optical
 - character recognition, 322
 - computers, 4
 - image processing, 296
 - Optimization, 279, 293, 391
 - criterion, 131
 - problems, 155, 293
 - Optimum**
 - choice of weights, 93
 - number of clusters, 254
 - set of weights, 117
 - weight matrix, 145
 - weight value, 104
 - weight vector, 116, 250
 - Order of a unit, 387
 - Oriental** selectivity, 224
 - Orienting subsystem, 259

- Orthogonal
 - inputs, 98, 143
 - unit vectors, 209
 - vectors, 98, 353
 - Orthography, 309
 - Orthonormal, 96, 208
 - Oscillatory
 - regions of equilibrium, 148
 - stable states, 69
 - state regions, 157
 - Outer product, 353
 - Output function, 25
 - binary, 27
 - bipolar, 32
 - continuous, 33
 - discrete, 32
 - linear range, 127
 - ramp, 26
 - saturation region, 127
 - sigmoid**, 26
 - Output
 - layer, 90
 - matrix, 90
 - pattern space, 80
 - signal, 26
 - state, 25
 - vector, 90
 - Outstar**, 30
 - group of, 30
 - learning law, 34
 - structure, 257
 - Overall logical predicate, 108
 - Overdetermined, 356
 - Overlapping frames, 311
 - Overtraining, 378
- P**
- PCNN**, 381
 - applications, 389
 - statistical data, 389
 - temporal data, 390
 - curve fitting, 389
 - data compression, 389
 - feature extraction, 389
 - generalization measure, 389
 - misalignment of image, 389
 - noise suppression, 390
 - preprocessor, 389
 - summary, 390
 - surface fitting, 389
 - PCNN** learning, 381
 - PDP models, 36, 345
 - Parallel and Distributed Processing (**PDP**), 4, 20, 341
 - Parallel computers, 4
 - Parametric level matching, 9
 - Parity problem, 109
 - Partial information, 184
 - Partially recurrent models, 267
 - Partition function, 170
 - Partition process, 326
 - Partitioned graphs, 296
 - Parzen windows, 255
 - Passive
 - decay rate, 45
 - decay term, 56
 - sonar detection, 134
 - Pattern
 - association, **6, 76, 77, 98, 184, 187, 190**
 - classification, 6, 76, 81, 88, 99, 100, 122, 251, 279, 280
 - clustering, 7, 76, 85, 202, 219
 - completion, 184, 190, 192, 265
 - environment, 143, 183
 - environment storage, 85, 183
 - grouping**, 7
 - mapping, 7, 76, 83, 88, 113, 240
 - matching, 9
 - storage, 76, 84, 143, 146, 211
 - variability, 8, 271
 - Pattern and data, 4
 - Pattern recall, 183
 - Pattern recognition tasks, 76, 89
 - Patterns in data, 341
 - Perception, 2
 - by human **beings**, 2
 - by machines, 2
 - Perceptron, 27, 103
 - classification, 113
 - convergence, 28, 102, 113
 - learning law, **28, 32, 101, 106, 113**
 - model, 27
 - multilayer, 110
 - network, 113
 - representation problem, 107, 113
 - single layer, 108, 241
 - Perceptron convergence theorem, 28, 102, 113
 - alternate proof, 104
 - discussion**, 106
 - proof, 102

Subject Index

- Perceptron** learning
 - continuous, 33
 - discrete, 32
 - gradient descent, 106, 113
 - Performance measure, 107
 - Performance of backpropagation learning, 121, 126
 - modular** network, 315
 - subnets**, 315
 - Periodic
 - regions of equilibrium, 148
 - stability, 148
 - Perkel's** model, 46
 - Peterson and Barney data, 309
 - Phoneme
 - classifier, 309
 - code, 307
 - Phoneme-like units, 308
 - Phonetic
 - decoding, 309
 - description, 313
 - transcription, 308
 - typewriter, 267, 280, 308
 - Pitch period, 307
 - Pixels, 281, 303, 325
 - Place of articulation, 314
 - Plain Hebbian learning, 207
 - Plant dynamics, 305
 - Plant transfer function, 305
 - Plasticity in ART, 259
 - Plosive source, 307
 - Polarization, 18
 - Pools of units, 342
 - Poor generalization, 133
 - Population-based problem solving, 391
 - Positional errors, 272
 - Positive definite, 354
 - Post-processor, 313
 - Post-synaptic neuron, 18
 - Post-synaptic potential, 18
 - Postal addresses, 323
 - Power spectrum, 325
 - Prediction of time series, 265
 - Preprocessing of image, 285
 - Preprocessing of input, 241
 - Principal **axes**, **366**
 - Principal component neural network, 379
 - Principal component learning, 66, 209, 381
 - Principle Component Analysis (**PCA**), 209, 379
 - Principle of orthogonality, 380
 - Printed characters, 7, 279, 287
 - Printed text symbols, 265
 - Prior knowledge, 126, 247
 - Probabilistic
 - neural networks, 121, 135, 255, 392
 - uncertainty, 393
 - update, 23, 152, 165
 - Probability, 357
 - a posteriori, 358
 - a priori, 358
 - axioms, 358
 - definition, 358
 - properties, 358
 - Probability density function (see distribution)
 - Probability distribution, 168, 359
 - expectation, 359
 - mean, 359
 - variance, 359
 - Probability distribution of states, 168, 176
 - Probability estimation, 122
 - Probability of
 - error, 149, 152
 - error in recall, 178
 - firing, 165
 - occurrence of patterns, 184
 - transition, 158
 - Probability **theory**, 248
 - Probably Approximate Correct (PAC) learning, 375
 - Problem level, 10, 11, 336
 - Problem of false minima, 163
 - Processing unit, 24
 - Production rules, 262
 - Projection matrix, 357
 - Proof of convergence, 126
 - Prototype vector, 259
 - Pseudoinverse of a matrix, 92, 144, 250
 - Puzzles, 4
- Q**
- Quadratic
 - error function, 130, 366
 - error surface, 130

Subject Index

- Quadratic forms, 92, 354
 - negative definite matrix, 354
 - negative semidefinite matrix, 354
 - positive definite matrix, 354
 - positive semidefinite matrix, 354
 - Quadratic output functions, 215
- R**
- RBF networks, 135, 245, 247, 251
 - architecture, 245
 - classification, 251
 - function approximation, 247
 - Radial basis function (RBF), *ZZ*, 245, 246
 - Random
 - connections, 157
 - initial values, 191
 - number generation, 190
 - patterns, 152
 - weight change, 66
 - Random access memory, 235
 - Random process, 167, 176
 - Random variable, 968
 - Range of activation values, 51
 - Range of influence, 246
 - Rank of matrix, 93
 - Real symmetric matrix, 92
 - Real time recurrent learning, 271
 - Real world problems, 234, 279, 306
 - Reasoning power, 280
 - Recall from
 - noisy input, 190
 - noisy pattern, 193
 - partial input, 190
 - Recency effect, 66
 - Receptive fields, 224
 - Recognition, 6
 - handwritten digits, 322
 - speech patterns, 267
 - Recurrent NN, 269, 992
 - Recurrent network models, 267
 - Recursive procedure, 97
 - Redundancies in patterns, 242
 - Reference patterns, 9
 - Refinements of backpropagation learning, 191
 - Region of traversal, 176
 - Regions of trajectories, 176
 - Regression estimate, 255
 - Regularization, 122, 247
 - methods, *ZZ*
 - networks, 196
 - parameter, 247
 - problem, *ZA9*
 - solution, *ZA9*
 - term, 247
 - theory, *ZZ*
 - Reinforcement
 - function, 61
 - signal, 63
 - Reinforcement learning, *ZZ*, 63, 134, 393
 - negative reinforcement, 64
 - positive reinforcement, 64
 - Relative entropy, 53
 - Relative spacing of energy minima, 148
 - Relaxation strategies
 - deterministic, 330
 - stochastic, 990
 - Relaxation of Hopfield model, 324
 - Remote sensing, 331
 - Replicated convolutional networks, 324
 - Representation of input, 241
 - Representation problem, 107
 - Requirements of learning laws, 53
 - Reset parameter, 261
 - Reset value, 261
 - Residual mismatch, 184
 - Resolution, 283, 323
 - Resonances of vocal tract, 8
 - Resonant state, *Z69*
 - Resting potential, 17, 45
 - Restriction on input patterns, 99
 - Retina, 108
 - Retrieval
 - by key, 341
 - Avu* name, 344
 - from partial description, 344
 - with noisy clues, 341
 - Risk functional, 374
 - Robustness, 6, 16
 - Room descriptors, 345
 - Rosenblatt's perceptron, 27
 - Rotation, 99, 280
 - Rough sets, 395
 - Rough uncertainty, 335, 996
 - Rule-based expert system, 334
 - Rule-based text-to-speech system, 308

S

- S-cells, 272
- SCV classes, 312, 314
- SVD expression for pseudoinverse, 94
- SVD of crosscorrelation matrix, 385
- Sample
 - function, 167, 364
 - set, 357
 - space, 357
- Sanger's** rule, 209, 382
- Saturation model, 48
- Scaling, 99, 122, 280
- Search methods
 - controlled, 391
 - global, 392
 - gradient-descent, **364–371**, 392
 - parallel, 391
 - stochastic, 391
- Second order
 - derivatives, 131
 - methods, 122
 - statistics, 253
- Segmental features, 307
- Selective attention feature, 272
- Self-amplification, 382
- Self-feedback, 188
- Self-organization, 22, 202, 262
 - learning, 379
 - network, 225, 300
- Self-stabilizing, 262
- Sensor** array imaging, 281
- Sensory mechanism, 5
- Sensory units, 27
- Sequence of patterns, 265
- Sequence** recognition, 265
- Sequence reproduction, 265
- Sequential model, 8
- Set of inequalities, 100
- Shattering, 376
- Shifted patterns, 271
- Short time memory (**STM**), 25, 40, 85, 202, 212
- Short-time **characteristics**, 307
- Short-time segment, 307
- Shunting activation, 48, 50, 204
 - general form, 50
 - summary, 51
- Sigmoid function, 26, 112, 155
- Sigmoidal nonlinearity, 124
- Signal power, 371
- Signal processing, 390
- Signal separation, 387
- Similarity
 - matrix, 316
 - measure, 260, 361
- Simulated annealing, 22, 65, 143, 165, 178, 179, 349, 392
- Single layer perceptron, 106, 241
- Singular subspaces, 385
- Singular value decomposition (SVD), 92, 144, 355
- Singular vectors
 - left, 355, 385
 - right, 355, 385
- Size-normalization, 321
- Skin diseases diagnosis, 334
- Slow convergence, 134
- Slow learning, 143
- Smoothed surface, 301
- Smoothness constraint, 244, 247
- Smoothness in mapping function, 242
- Smoothness property, 242
- Soft constraints, 299
- Software, 2
- Softwiring, 224
- SOM network, 225, 392
- Sonar, 134, 390
- Sound units in speech, 265, 307
- Space displacement neural networks, 324
- Space filling characteristic, 227
- Spade in card game, 290
- Sparse data, 285
- Sparse encoding, 65
- Spatial
 - correlation, 321
 - pattern, 235, 265
 - relations in features, 147
 - transformation, 285
- Spatio-temporal pattern, 235, 266, 397
- Speaker identification, 307
- Spectral features, 307
- Spectral vector, 310
- Speech, 1, 4, 99, 134, 306, 390
 - production knowledge, 318
 - recognition, 307
 - spectra, 7
 - synthesis, 134, 307
- Speech-like signals, 267
- Speed, 19

- Spin glasses. 22
- Spontaneous generalization, 344
- Spurious stable states, 183
- Square norm, 92
- Stability, 68
 - chaotic, 69, 397
 - fixed point, 69, 157
 - in **ART**, 259
 - in stochastic networks, 172
 - of patterns, 69
 - oscillatory, 69, 397
 - theorems, 42
 - thermal equilibrium, 172
- Stability and convergence, 42, 68
- Stability-plasticity dilemma, 8, 258, 396
- Stable state, 55, 69, 150
- State at thermal equilibrium, 170
- State of energy minima, 148
- State of network, 147
- State space, 25
 - depth of energy minima, 148
 - relative spacings of energy minima, 148
- State transition
 - diagram, 158, 179
 - probability matrix, 181
- Static
 - equilibrium, 167
 - pattern, 310
 - spatial pattern, 265
- Stationary
 - probabilities, 170, 295
 - probability distribution, 177
 - random process, 364
- Statistical machines, 23
- Statistical mechanics, 170
- Steady
 - activation stat., 40, 55
 - state, 45, 55
 - weight state, 40
- Steepest descent method, 368
- Stereovision matching, 296
- Stochastic, 25, 42, 51, 165, 324, 330, 391
 - activation models, 51
 - differential equation, **59**
 - equilibrium, 168
 - gradient descent, 62, 121, 134, 371
 - learning algorithms, 134
 - learning, 54, 65
 - network, 165, 167, 175
 - process, 51
 - scalar, 51
 - vector, 51
 - relaxation, 299
 - unit, 22
 - update, 143, 164, 165, 295
 - update law, 167
- Stock prices, 334
- Stop-Consonant-Vowel (**SCV**) **utterances**, 312
- Stopping criterion, 121, 126, 378
- Storage capacity, 53, 151, 157
- Strange **attractors**, 397
- Stretching operation, 397
- Structural
 - learning, 54
 - stability**, 42, 44
- Subjective computation, 393
- Submatrices, 93
- Subnet**, 313
- Suboptimal solution, 117, 250
- Subsampling, 323
- Subsignals, 387
- Subspace** decomposition, 380
- Summary of
 - backpropagation learning algorithm, 121
 - gradient search methods, 116
 - perceptron** learning, 113
- Summing part, 24
- Supervised learning, 6, 32
- Supervised vector quantization, 223
- Supervisory mode, 115
- Suprasegmental features, 307
- Surface fitting, 389
- Syllable, 310
- Symbolic processing, 5
- Symmetric
 - matrix, 366
 - weights, 149, 153
- Synapse, 16
- Synaptic connection, 18
- Synaptic dynamics, 25, 40, 52
 - discrete-time implementation, 56
 - model, 52
- Synaptic equilibrium, 59
- Synaptic junctions, 16
- Synaptic strength, **18**
- Synchronous update, 150, 237
- Syntactic pattern recognition, 9
- System identification, 134

Subject Index

T

- Tapped delay line, 265
- Tasks with backpropagation, 122
- Taylor series, 129, 354
 - multidimensional, 354
- Temperature parameter, 166, 181
- Template matching, 9
- Temporal
 - association, 265
 - aseociative memory, 240
 - correlations, 265
 - learning, 54
 - pattern, 8, 265
 - pattern recognition, 265
 - pattern vectors, 240
 - sequence, 265
- Temporary pattern storage, 85, 212
- Terminology** of ANN, 24
- Test patterns, 9
- Test error, 378
- Texture classes, 326
- Texture classification, 279, 321, 324
- Texture features, 324
 - deterministic modelling, 324
 - stochastic modelling, 324
- Texture label, 326
- Texture segmentation, 324
- Texture-based scheme, 324
- Theorems for function approximation, 246
- Theoretical machine, 22
- Thermal averages, 170
- Thermal equilibrium, 168, 181, 295
- Threshold function
 - linear, 138
 - polynomial, 138
 - quadratic, 138
- Threshold value, 101
- Time constant, **18**
- Time correlation, 265
- Time registration, 266
- Time sequences, 271
- Time-delay neural networks (TDNN), 311
- Time-series prediction, 269
- Top-down **outstar** learning, 259
- Top-down weights, 259
- Topological mapping, 224
- Topology of ANN, 29
- Topology preserving map, ~~224~~
- Total error, 91
- Total** error surface, 117
- Tongue, 306
- Trace of a square matrix, 92
- Tracking frequency components, 390
- Training, 127
 - batch mode, 127
 - instars** of CPN, 257
 - outstars** of CPN, 257
 - pattern mode, 127
 - process**, 89
 - samples, 89, 377
- Training data, 117, 378
- Trajectory, **25, 53, 147, 167, 176, 368**
- Transformation invariant object recognition, 288
- Transient
 - phenomenon, 176
 - region, 176
- Transition probabilities, 180
- Translation, 99, 280
- Travelling salesman problem, 279, 298
 - elastic ring method, 300
 - optimization method, 296
- Traversal in the landscape, 167
- Trends in computing, 2
- Trial** solutions, 391
- Truck backer-upper problem, 270
- Turbulent flow, 307
- Two-class problem, 101
- Two-layer networks, 110

U

- Unaspirated, 312
- Unclamped** condition, 195
- Uncommitted units, 259
- Unconditionally stable, 238
- Unconstrained optimization, 121, 131
- Understanding, 5
- Uniform distribution, 190, 360
- Unique solution, 100
- Unit
 - higher-order, 386
 - sensory, 27
- Universal approximation theorem, 133, 247
- Unrepresentable problems, 108
- Unstable states, 153

Unstable weight values, 208
Unsupervised learning, 7, 32
Unvoiced, 307, 314
Update, 25
 asynchronous, 25
 deterministic, 25
 stochastic, 25
 synchronous, 25
Upper **subnet**, 282

V

Validation, 373
VLSI, 3
VC dimension, 122, 375
Variance maximization, 380
Variance of input data, 127
Vector quantization (**VQ**), 86, 279, 304
Vigilance parameter, 259
Vigilance test, 260
Visible units, 183, 343
Vision, **1**
Visual clues, 284
Visual pattern recognition, 271
Vocal folds, 306
Vocal tract, 306
Voiced, 306, 312

Vowel classification, 279, 309
Vowels, 314

W

Weak constraints, 299, 345
Weight
 decay, 248
 matrix, 90, 95, 236
 sharing, 322
 space, 25, 40, 53
 state, 40
 update, 107
 vector, 25; **90**, 207
Weighted inner product, 362
Weighted matching problem, 296
Weights by computation, 91
Well-posed problems, 242
Widrow's learning law, 97
Widrow-Hoff's LMS algorithm, 68
Winner, 202
Winning unit, 61, 202, 257, 284
Winner-take-all, 34, 60, 218, 261

X

XOR problem, 241

ARTIFICIAL NEURAL NETWORKS

■

B. YEGNAWARAYANA

Designed as **an introductory** level textbook on **Artificial** Neural Networks at the postgraduate and senior undergraduate **levels in any branch** of engineering, **this self-contained** and well-organized book highlights the **need** for new models of computing based on the fundamental principles of neural networks.

Professor Yegnanarayana compresses, into the covers of a single volume, his several years of rich experience, in teaching **and** research in the areas of speech processing, image **processing, artificial** intelligence and neural networks. He gives a masterly analysis of such topics as Basics of **artificial neural** networks, Functional units of artificial neural networks **for pattern** recognition tasks, **Feedforward** and Feedback neural **networks**, and **Architectures** for complex pattern recognition tasks. Throughout, the emphasis is on the pattern processing feature of the neural networks. Besides, the presentation of real-world applications provides a practical thrust to the discussion.

The fairly large number of diagrams, the detailed Bibliography, and the provision of Review Questions and Problems at the end of each chapter should prove to be of considerable assistance to the reader. Besides students, practising engineers and research scientists would cherish this book which treats the emerging and exciting area of artificial neural **networks** in a rigorous yet lucid fashion.

B. YEGNANARAYANA, Ph.D., is Professor, Department of Computer Science and Engineering, Indian Institute of Technology Madras. A Fellow of the Indian National Science Academy, Fellow of the Indian Academy of Sciences and Indian National Academy of Engineering, Prof. Yegnanarayana has published several papers in reputed national and international journals. His areas of interest include signal processing, speech and image **processing**, and neural networks.

To learn more about
Prentice-Hall of India products,
please visit  at : www.phindia.com

Rs. 275.00

ISBN 81-203-1253-8

