The McGraw-Hill Companies

Computer Engineering Series

INTRODUCTION TO NEURAL NETWORKS USING MATLAB 6.0

S N SIVANANDAM • S SUMATHI • S N DEEPA

Tata McGraw-Hill

Copyright © 2006, by Tata McGraw-Hill Publishing Company Limited

Second reprint 2006
RZLYCDRKRQLYC

ISBN 0-07-059112-1

Published by the Tata McGraw-Hill Publishing Company Limited,
7 West Patel Nagar, New Delhi 110 008, typeset in Times at Script Makers,
19, A1-B, DDA Market, Paschim Vihar, New Delhi 110 063 and
printed at S.P. Printers, E-120, Sector-7, Noida

Cover: Shree Ram Enterprises

Cover Design: Kapil Gupta, Delhi

The McGraw-Hill Companies

Contents

Preface

The world we live in is becoming ever more reliant on the use of electronic gadgets and computers to control the behavior of real world resources. For example, an increasing amount of commerce is performed without a single bank note or coin ever being exchanged. Similarly, airports can safely land and send off aeroplanes without even looking out of a window. Another, more individual, example is the increasing use of electronic personal organizers for organizing meetings and contacts. All of these examples share a similar structure; multiple parties (e.g. aeroplanes, or people) come together to coordinate their activities in order to achieve a common goal. It is not surprising, then, that a lot of research is being done on how the mechanics of the coordination process can be automated using computers. This is where neural networks come in.

Neural networks are important for their ability to adapt. Neural nets represent entirely different models from those related to other symbolic systems. The difference occurs in the way the nets store and retrieve information. The information in a neural net is found to be distributed throughout the network and not localized. The nets are capable of making memory associations. They can handle a large amount of data, fast and efficiently. They are also fault tolerant, i.e. even if a few neurons fail, it will not disable the entire system.

The paradigm of artificial neural networks, developed to emulate some of the capabilities of the human brain, has demonstrated great potential for various low-level computations and embodies salient features such as learning, fault-tolerance, parallelism and generalization. Neural networks, comprising processing elements called neurons, are capable of coping with computational complexity, non-linearity and uncertainty. In view of this versatility of neural networks, it is believed that they hold great potential as building blocks for a variety of behaviors associated with human cognition. However, the subjective phenomena such as reasoning and perceptions are often regarded beyond the domain of the neural network theory. Neural networks can deal with imprecise data and ill-defined activities; thus they offer low-level computational features.

About the Book

Neural networks, at present is a much sought-after topic, among academicians as well as program developers. This book is designed to give a broad, yet an in-depth overview of the field of neural networks. The principles of neural networks are discussed in detail, including information and useful knowledge available for various network processes. The various algorithms and solutions to the problems given in the book are well balanced and pertinent to the neural networks research projects, labs and for college and university level studies. The modern aspects of neural networks have been introduced right from the basic principles and discussed in an easy-to-understand manner, so that a beginner to the subject is able to grasp the concept of soft networks with minimal effort.

The wide variety of worked-out examples relevant to the neural network area, will help in reinforming the concepts explained. The solutions to the problems are programmed using MATLAB 6.0 and the simulated results are given. The MATLAB neural network toolbox is provided in the Appendix for easy reference.

This book provides the neural network architecture, algorithms and application procedure-oriented structures to help the reader move into the world of neural networks with ease. It also presents application of neural networks to a wide variety of fields of current interest. A few field projects are also included.

Who will Benefit

This book would be an ideal text for undergraduate students of Computer Science, Information Technology, Electrical and Electronics and Electronics and Communication engineering for their course on Neural Networks. Those pursuing MCA and taking a course on Neural Networks will find the book useful. Programmers involved in neural network applications programming will also benefit from this book.

Organization

The book includes 16 chapters altogether. The chapters are organized as follows:

Chapter 1 gives an introduction to Neural Networks Techniques. An overview of MATLAB is also discussed.

The preliminaries of the Artificial Neural Network are described in **Chapter 2**. The discussion is based on the development of artificial neural net, comparison between the biological neuron and the artificial neuron, the basic building blocks of a neural net and the terminologies used in neural net. The summary of notations is given at the end of the chapter.

Chapter 3 deals with the fundamental models of an artificial neural net. The basics of McCulloch Pitts neuron and the Hcbb net along with the concept of linear separability is given. The learning rules used in neural networks are also described in detail in this chapter.

Chapter 4 provides information regarding the Perceptron Neural Net. The architecture and algorithm of the perceptron neural net was explained along with suitable example problems. An introduction to multi layer perceptron is given.

The basic architecture and algorithm along with examples for Adaline and Madaline nets are described in **Chapter 5**.

Chapter 6 discusses pattern association nets. Pattern association nets include auto association, hetero association and bi-directional associative memory net. The learning rules used for pattern association are also given.

Feedback network is described in **Chapter 7**. The chapter mainly provides information regarding Discrete Hopfield and Continuous Hopfield nets. Their architecture, algorithm and application procedure along with solved examples are discussed in this chapter.

Chapter 8 gives details on feed forward nets. The feed forward nets described here are the Back Propagation Algorithm and the Radial Basis Function Network. Both the networks are described with their architecture, algorithm and example problem. The merits and demerits of back propagation algorithm are also included.

Chapter 9 deals with competitive nets. The nets that come under this category are self-organizing feature map, learning vector quantization, Max net, Mexican Hat, Hamming net. All these networks are discussed in detail with their features in this chapter.

The Counter Propagation Net (CPN) used for data compression is discussed in **Chapter 10.** The two types of CPN, full CPN and forward only CPN are discussed along with their architecture and algorithms.

Chapter 11 describes the features of Adaptive Resonance Theory (ART). The types of ART, ART network and ART2 network are described with their respective architecture, algorithms and example problems.

The information regarding the special nets like Boltzmann machine, cascade correlation, spatio temporal network, simulated annealing, optical neural net, Cauchy machine, Gaussian machine, cognitron, neo cognitron, Boltzmann machine with learning, etc. are given in **Chapter 12.**

Chapter 13 discusses the application of neural network in arts, biomedicine, industrial and control area, data mining, robotics, pattern recognition, etc. with case studies.

Chapter 14 presents the applications of various special networks dealt in Chapter 12.

Few projects related to pattern classification, system identification using different networks with MATLAB programs are discussed in **Chapter 15.**

Chapter 16 gives a brief introduction to Fuzzy Systems and Hybrid Systems (Fuzzy Neural Hybrid and Neural Fuzzy Hybrid).

The appendices include the neural network MATLAB tool box.

In conclusion, we hope that the reader will find this book a truly helpful guide and a valuable source of information about the neural networks principles and their numerous practical applications. Critical comments and suggestions from the readers are welcome as they will help us improve the future editions of the book.

S N Sivanandam

S Sumathi

S N Deepa

Acknowledgements

First of all, the authors would like to thank the Almighty for granting them perseverance and achievements.

Dr S N Sivanandam, Dr S Sumathi and S N Deepa wish to thank Mr V Rajan, Managing Trustee, PSG Institutions, Mr C R Swaminathan, Chief Executive, and Dr S Vijayarangan, Principal, PSG College of Technology, Coimbatore, for their whole-hearted cooperation and encouragement provided for this endeavor.

The authors are grateful for the support received from the staff members of the Electrical and Electronics Engineering and Computer Science and Engineering departments of their college.

Dr Sumathi owes much to her daughter S Priyanka, who cooperated even when her time was being monopolized with book work. She feels happy and proud for the steel-frame support rendered by her husband. She would like to extend whole-hearted thanks to her parents and parents-in-law for their constant support. She is thankful to her brother who has always been the "Stimulator" for her progress.

Mrs S N Deepa wishes to thank her husband Mr T S Anand, her daughter Nivethitha T S Anand and her family for the support provided by them.

Thanks are also due to the editorial and production teams at Tata McGraw-Hill Publishing Company Limited for their efforts in bringing out this book.

What You Will Learn

- Scope of neural networks and MATLAB.
- How neural network is used to learn patterns and relationships in data.
- The aim of neural networks.
- About fuzzy logic .
- Use of MATLAB to develop the applications based on neural networks.

CHAPTER 1

Introduction to Neural Networks

1.1 Neural Processing

Artificial neural networks are the result of academic investigations that use mathematical formulations to model nervous system operations. The resulting techniques are being successfully applied in a variety of everyday business applications.

Neural networks (NNs) represent a meaningfully different approach to using computers in the workplace. A neural network is used to learn patterns and relationships in data. The data may be the results of a market research effort, a production process given varying operational conditions, or the decisions of a loan officer given a set of loan applications. Regardless of the specifics involved, applying a neural network is substantially different from traditional approaches.

Traditionally a programmer or an analyst specifically 'codes' for every facet of the problem for the computer to 'understand' the situation. Neural networks do not require explicit coding of the problems. For example, to generate a model that performs a sales forecast, a neural network needs to be given only raw data related to the problem. The raw data might consist of history of past sales, prices, competitors' prices and other economic variables. The neural network sorts through this information and produces an understanding of the factors impacting sales. The model can then be called upon to provide a prediction of future sales given a forecast of the key factors.

These advancements are due to the creation of neural network learning rules, which are the algorithms used to 'learn' the relationships in the data. The learning rules enable the network to 'gain knowledge' from available data and apply that knowledge to assist a manager in making key decisions.

What are the Capabilities of Neural Networks?

In principle, NNs can compute any computable function, i.e. they can do everything a normal digital computer can do. Especially anything that can be represented as a mapping between vector spaces can be approximated to arbitrary precision by feedforward NNs (which is the most often used type).

In practice, NNs are especially useful for mapping problems, which are tolerant of some errors, have lots of example data available, but to which hard and fast rules cannot easily be applied. However, NNs are, as of now, difficult to apply successfully to problems that concern manipulation of symbols and memory.

Who is Concerned with Neural Networks?

Neural Networks are of interest to quite a lot of people from different fields:

- Computer scientists want to find out about the properties of non-symbolic information processing with neural networks and about learning systems in general.
- Engineers of many kinds want to exploit the capabilities of neural networks in many areas (e.g. signal processing) to solve their application problems.
- Cognitive scientists view neural networks as a possible apparatus to describe models of thinking and conscience (High-level brain function).
- Neuro-physiologists use neural networks to describe and explore medium-level brain function (e.g. memory, sensory system).
- Physicists use neural networks to model phenomena in statistical mechanics and for a lot of other tasks.
- Biologists use Neural Networks to interpret nucleotide sequences.
- Philosophers and some other people may also be interested in Neural Networks to gain knowledge about the human systems namely behavior, conduct, character, intelligence, brilliance and other psychological feelings. Environmental nature and related functioning, marketing business as well as designing of any such systems can be implemented via Neural networks.

1.2 Neural Networks — An Overview

The development of Artificial Neural Network started 50 years ago. Artificial neural networks (ANNs) are gross simplifications of real (biological) networks of neurons. The paradigm of neural networks, which began during the 1940s, promises to be a very important tool for studying the structure-function relationship of the human brain. Due to the complexity and incomplete understanding of biological neurons, various architectures of artificial neural networks have been reported in the literature. Most of the ANN structures used commonly for many applications often consider the behavior of a single neuron as the basic computing unit for describing neural information processing operations. Each computing unit, i.e. the artificial neuron in the neural network is based on the concept of an ideal neuron. An ideal neuron is assumed to respond optimally to the applied inputs. However, experimental studies in neuro-physiology show that the response of a biological neuron appears random and only by averaging many observations it is possible to obtain predictable results. Inspired by this observation, some researchers have developed neural structures based on the concept of neural populations.

In common with biological neural networks, ANNs can accommodate many inputs in parallel and encode the information in a distributed fashion. Typically the information that is stored in a neural net is shared by many of its processing units. This type of coding is in sharp contrast to traditional memory schemes, where a particular piece of information is stored in only one memory location. The recall process is time consuming and generalization is usually absent. The distributed storage scheme provides many advantages, most important of them being the redundancy in information representation. Thus, an ANN can undergo partial destruction of its structure and still be able to function well. Although redundancy can also be built into other types of systems, ANN has a natural way of implementing this. The result is a natural fault-tolerant system which is very similar to biological systems.

The aim of neural networks is to mimic the human ability to adapt to changing circumstances and the current environment. This depends heavily on being able to learn from events that have happened in the past and to be able to apply this to future situations. For example the decisions made by doctors are rarely based on a single symptom because of the complexity of the human body; since one symptom could signify any number of problems. An experienced doctor is far more likely to make a sound decision than a trainee, because from his past experience he knows what to look out for and what to ask, and may have etched on his mind a past mistake, which he will not repeat. Thus the senior doctor is in a superior position than the trainee. Similarly it would be beneficial if machines, too, could use past events as part of the criteria on which their decisions are based, and this is the role that artificial neural networks seek to fill.

Artificial neural networks consist of many nodes, i.e. processing units analogous to neurons in the brain. Each node has a node function, associated with it which along with a set of local parameters determines the output of the node, given an input. Modifying the local parameters may alter the node function. Artificial Neural Networks thus is an information-processing system. In this information-processing system, the elements called **neurons**, process the information. The signals are transmitted by means of connection links. The links possess an associated weight, which is multiplied along with the incoming signal (net input) for any typical neural net. The output signal is obtained by applying activations to the net input.

The neural net can generally be a single layer or a multi-layer net. The structure of the simple artificial neural net is shown in Fig. 1.1.

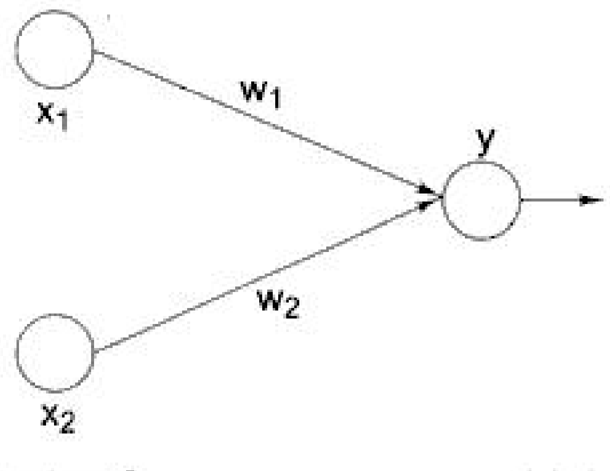


Fig. 1.1 *A Simple Artificial Neural Net*

Figure 1.1 shows a simple artificial neural net with two input neurons (x_1, x_2) and one output neuron (y). The inter-connected weights are given by w_1 and w_2. In a single layer net there is a single layer of weighted interconnections.

A typical multi-layer artificial neural network, abbreviated as MNN, comprises an input layer, output layer and hidden (intermediate) layer of neurons. MNNs are often called layered networks. They can implement arbitrary complex input/output mappings or decision surfaces separating different patterns. A three-layer MNN is shown in Fig. 1.2, and a simplified block diagram representation in Fig. 1.3. In a MNN, a layer of input units is connected to a layer of hidden units, which is connected to the layer of output units. The activity of neurons in the input layer represents the raw information that is fed into the network. The activity of neurons in the hidden layer is determined by the activities of the input neurons and the connecting weights between the input and hidden units. Similarly, the behavior of the output units depends on the activity of the neurons in the hidden layer and the connecting weights between the hidden and the output layers. This simple neural structure is interesting because neurons in the hidden layers are free to construct their own representation of the input.

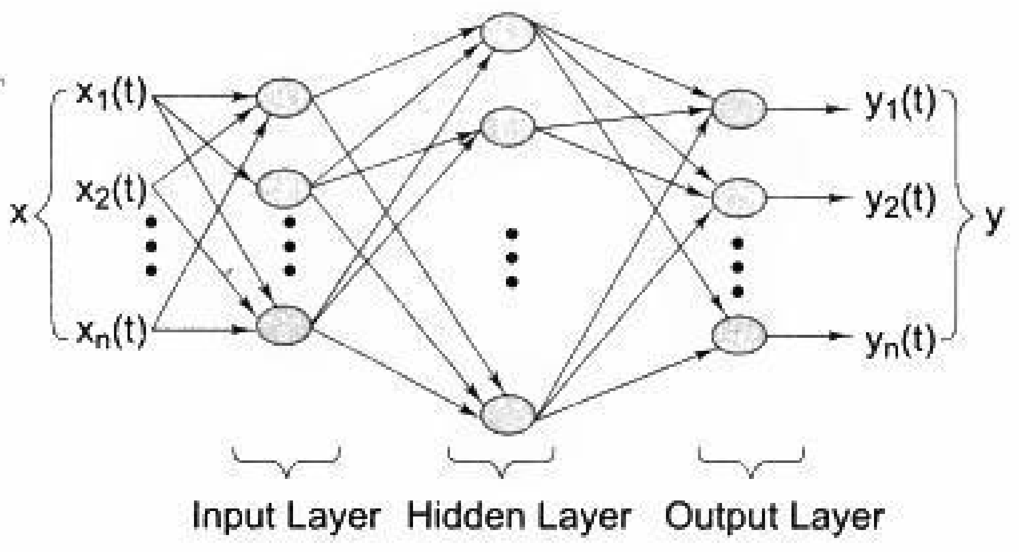


Fig. 1.2 *A Densely Interconnected Three-layered Static Neural Network. Each Shaded Circle, or Node, Represents an Artificial Neuron*

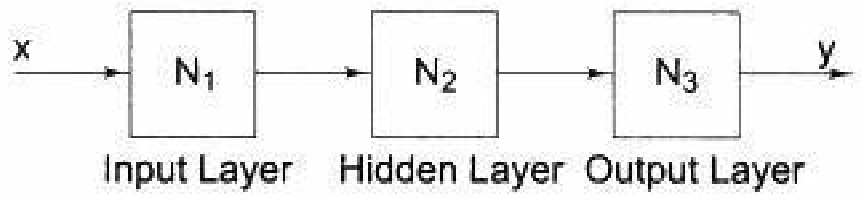


Fig. 1.3 *A Block Diagram Representation of a Three-layered MNN*

MNNs provide an increase in computational power over a single-layer neural network unless there is a nonlinear activation function between layers. Many capabilities of neural networks, such as nonlinear functional approximation, learning, generalization, etc are in fact performed due to the nonlinear activation function of each neuron.

ANNs have become a technical folk legend. The market is flooded with new, increasingly technical software and hardware products, and many more are sure to come. Among the most popular hardware implementations are Hopfield, Multilayer Perceptron, Self-organizing Feature Map, Learning Vector Quantization, Radial Basis Function, Cellular Neural, and Adaptive Resonance Theory (ART) networks, Counter Propagation networks, Back Propagation networks, Neo-cognitron, etc. As a result of the existence of all these networks, the application of the neural network is increasing tremendously.

Thus artificial neural network represents the major extension to computation. They perform the operations similar to that of the human brain. Hence it is reasonable to expect a rapid increase in our understanding of artificial neural networks leading to improved network paradigms and a host of application opportunities.

1.3 The Rise of Neurocomputing

A majority of information processing today is carried out by digital computers. This has led to the widely held misperception that information processing is dependent on digital computers. However, if we look at cybernetics and the other disciplines that form the basis of information science, we see that information processing originates with living creatures in their struggle to survive in their environments, and that the information being processed by computers today accounts for only a small part — the automated portion — of this. Viewed in this light, we can begin to consider the possibility of information processing devices that differ from conventional computers. In fact, research aimed at realizing a variety of different types of information processing devices is already being carried out, albeit in the shadows of the major successes achieved in the realm of digital computers. One direction that this research is taking is toward the development of an information processing device that mimics the structures and operating principles found in the information processing systems possessed by humans and other living creatures.

Digital computers developed rapidly in and after the late 1940's and after originally being applied to the field of mathematical computations, have found expanded applications in a variety of areas, like text (word), symbol, image and voice processing, i.e. pattern information processing, robotic control and artificial intelligence. However, the fundamental structure of digital computers is based on the principle of sequential (serial) processing, which has little if anything in common with the human nervous system.

The human nervous system, it is now known, consists of an extremely large number of nerve cells, or neurons, which operate in parallel to process various types of information. By taking a hint from the structure of the human nervous system, we should be able to build a new type of advanced parallel information processing device.

In addition to the increasingly large volumes of data that we must process as a result of recent developments in sensor technology and the progress of information technology, there is also a growing requirement to simultaneously gather and process huge amounts of data from multiple sensors and other sources. This situation is creating a need in various fields to switch from conventional computers that process information sequentially, to parallel computers equipped with multiple processing elements aligned to operate in parallel to process information.

Besides the social requirements just cited, a number of other factors have been at work during the 1980's to prompt research on new forms of information processing devices. For instance, recent neurophysiological experiments have shed considerable light on the structure of the brain, and even in fields such as cognitive science, which study human information processing processes at the macro level, we are beginning to see proposals for models that call for multiple processing elements aligned to operate in parallel. Research in the fields of mathematical science and physics is also concentrating more on the mathematical analysis of systems comprising multiple elements that interact in complex ways. These factors gave birth to a major research trend aimed at clarifying the structures and operating principles inherent in the information processing systems of human beings and other animals, and constructing an information processing device based on these structures and operating principles. The term **'neurocomputing'** is used to refer to the information engineering aspects of this research.

1.4 MATLAB – An Overview

Dr. Cleve Moler, chief scientist at MathWorks, Inc., originally wrote MATLAB to provide easy access to matrix software developed in the LINPACK and EISPACK projects. The first version was written in the late 1970s for use in courses in matrix theory, linear algebra, and numerical analysis. MATLAB is therefore built upon a foundation of sophisticated matrix software, in which the basic data element is a matrix that does not require predimensioning.

MATLAB is a product of The MathWorks, Inc. and is an advanced interactive software package specially designed for scientific and engineering computation. The MATLAB environment integrates graphical illustrations with precise numerical calculations, and is a powerful, easy-to-use, and comprehensive tool for performing all kinds of computations and scientific data visualization. MATLAB has proven to be a very flexible and useful tool for solving problems in many areas. MATLAB is a high-performance language for technical computing. It integrates computation, visualization and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical areas of application of MATLAB include:

- Math and computation
- Algorithm development
- Modeling, simulation and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic element is an array that does not require dimensioning. This helps in solving many computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or FORTRAN. Mathematics is the common language of science and engineering. Matrices, differential equations, arrays of data, plots and graphs are the basic building blocks of both applied mathematics and MATLAB. It is the underlying mathematical base that makes MATLAB accessible and powerful. MATLAB allows expressing the entire algorithm in a few dozen lines, to compute the solution with great accuracy in about a second. Therefore it is especially helpful for technical analysis, algorithm prototyping and application development.

MATLAB's two-and three-dimensional graphics are object oriented. MATLAB is thus both an environment and a matrix/vector-oriented programming language, which enables the user to build own reusable tools. The user can create his own customized functions and programs (known as M-files) in MATLAB code. The Toolbox is a specialized collection of M-files for working on particular classes of problems. MATLAB Documentation Set has been written, expanded and put online for ease of use. The set includes online help, as well as hypertext-based and printed manuals. The commands in MATLAB are expressed in a notation close to that used in mathematics and engineering. There is a very large set of these commands and functions, known as MATLAB M-files. As a result, solving problems through MATLAB is faster than the other traditional programming. It is easy to modify the functions since most of the M-files can be opened and modified. For ensuring high performance, the MATLAB software has been written in optimized C and coded in assembly language.

The main features of MATLAB can be summarized as:

- Advance algorithms for high-performance numerical computations, especially in the field of matrix algebra.
- A large collection of predefined mathematical functions and the ability to define one's own functions.
- Two- and three-dimensional graphics for plotting and displaying data.
- A complete online help system.
- Powerful, matrix/vector-oriented, high-level programming language for individual applications.

- Ability to cooperate with programs written in other languages and for importing and exporting formatted data.
- Toolboxes available for solving advanced problems in several application areas.

Figure 1.4 shows the main features and capabilities of MATLAB.

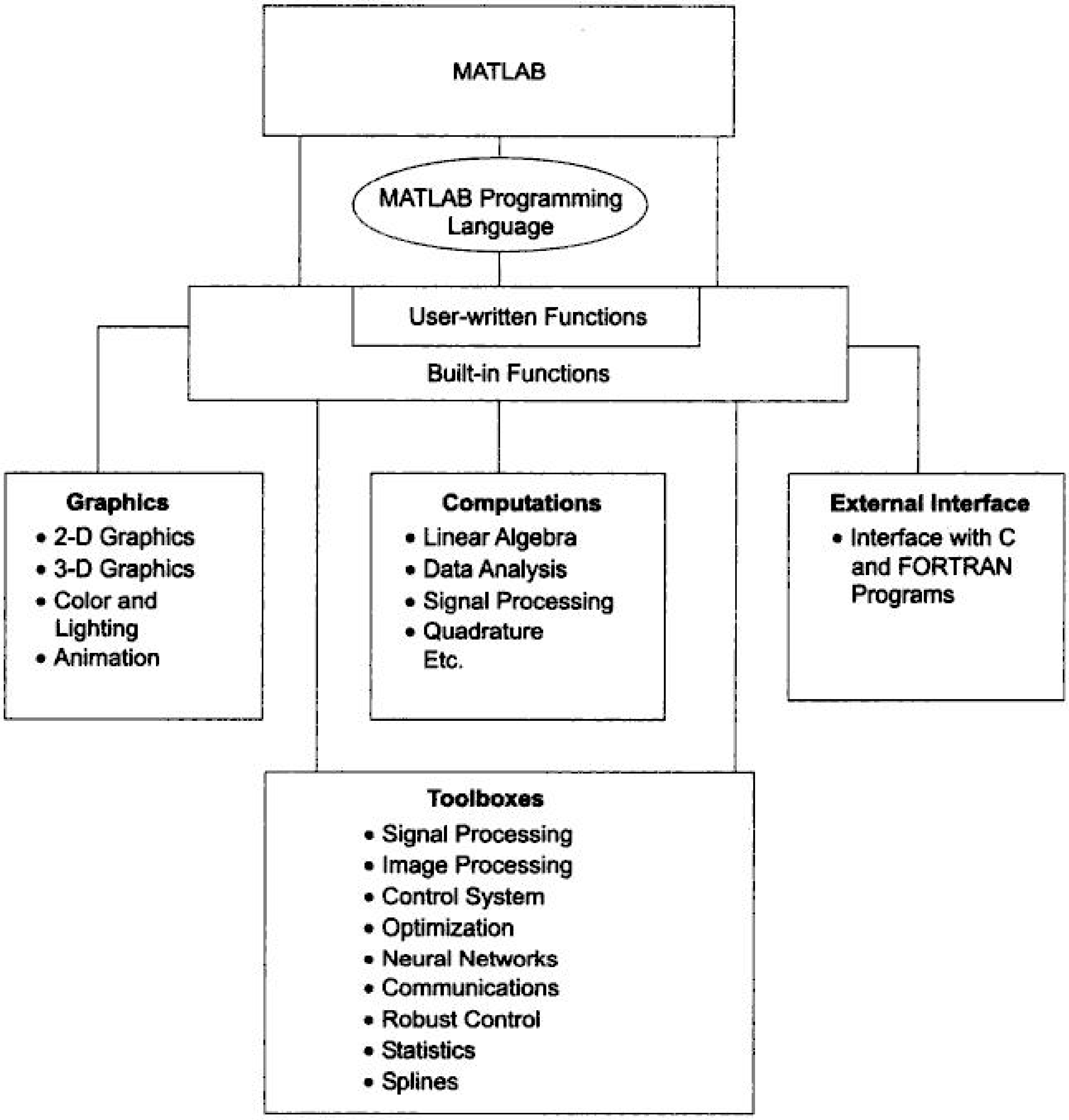


Fig. 1.4 *Features and Capabilities of MATLAB*

An optional extension of the core of MATLAB called SIMULINK is also available. SIMULINK means SIMUlating and LINKing the environment. SIMULINK is an environment for simulating linear and non-linear dynamic systems, by constructing block diagram models with an easy to use graphical user interface.

SIMULINK is a MATLAB toolbox designed for the dynamic simulation of linear and non-linear systems as well as continuous and discrete-time systems. It can also display information graphically. MATLAB is an interactive package for numerical analysis, matrix computation, control system design, and linear system analysis and design available on most CAEN (Computer Aided Engineering Network) platforms (Macintosh, PCs, Sun, and Hewlett-Packard). In addition to the standard functions provided by MATLAB, there exist a large set of Toolboxes, or collections of functions and procedures, available as part of the MATLAB package. Toolboxes are libraries of MATLAB functions used to customize MATLAB for solving particular class of problems. Toolboxes are a result of some of the world's top researches in specialized fields. They are equivalent to prepackaged 'off-the-shelf' software solution for a particular class of problem or technique. It is a collection of special files called M-files that extend the functionality of the base program. The various Toolboxes available are:

- *Control System:* Provides several features for advanced control system design and analysis.
- *Communications:* Provides functions to model the components of a communication system's physical layer.
- *Signal Processing:* Contains functions to design analog and digital filters and apply these filters to data and analyze the results.
- *System Identification:* Provides features to build mathematical models of dynamical systems based on observed system data.
- *Robust Control:* Allows users to create robust multivariable feedback control system designs based on the concept of the singular-value Bode plot.
- *Simulink:* Allows you to model dynamic systems graphically.
- *Neural Network:* Allows you to simulate neural networks.
- *Fuzzy Logic:* Allows for manipulation of fuzzy systems and membership functions.
- *Image Processing:* Provides access to a wide variety of functions for reading, writing, and filtering images of various kinds in different ways.
- *Analysis:* Includes a wide variety of system analysis tools for varying matrices.
- *Optimization:* Contains basic tools for use in constrained and unconstrained minimization problems.
- *Spline:* Can be used to find approximate functional representations of data sets.
- *Symbolic:* Allows for symbolic (rather than purely numeric) manipulation of functions.
- *User Interface Utilities:* Includes tools for creating dialog boxes, menu utilities, and other user interactions for script files.

MATLAB has been used as an efficient tool, all over this text to develop the applications based on Neural Nets.

Review Questions

1.1 How did neurocomputing originate?

1.2 What is a multilayer net? Describe with a neat sketch.

1.3 State some of the popular neural networks.

1.4 Briefly discuss the key characteristics of MATLAB.

1.5 List the basic arithmetic operations that can be performed in MATLAB.

1.6 What is the necessity of SIMULINK package available in MATLAB?

1.7 Discuss in brief about the GUI toolbox feature of MATLAB.

1.8 What is meant by toolbox and list some of the toolboxes available for MATLAB?

CHAPTER 2

Introduction to Artificial Neural Networks

What You Will Learn

- The preliminaries of Artificial Neural Networks.
- Definition of an artificial neuron.
- The development of neural networks.
- Comparison between the biological neuron and artificial neuron based on speed, fault tolerance, memory, control mechanism, etc.
- The method of setting the value for the weights and how it enables the process of learning or training.
- Various methods of training, viz. supervised training, unsupervised training and reinforcement training.
- Basic building blocks of the artificial neural network, i.e. network architecture, setting the weights, activation function, etc.
- The activation function used to calculate the output response of a neuron.
- Summary of notations used all over in this text.

2.1 Introduction

The basic preliminaries involved in the Artificial Neural Network (ANN) are described in this chapter. A brief summary of the history of neural networks, in terms of the development of architectures and algorithms, the structure of the biological neuron is discussed and compared with the artificial neuron. The basic building blocks and the various terminologies of the artificial neural network are explained towards the end of the chapter. The chapter concludes by giving the summary of notations, which are used in all the network algorithms, architectures, etc. discussed in the forthcoming chapters.

2.2 Artificial Neural Networks

Artificial neural networks are nonlinear information (signal) processing devices, which are built from interconnected elementary processing devices called neurons.

An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in union to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

ANN's are a type of artificial intelligence that attempts to imitate the way a human brain works. Rather than using a digital model, in which all computations manipulate zeros and ones, a neural network works by creating connections between processing elements, the computer equivalent of neurons. The organization and weights of the connections determine the output.

A neural network is a massively parallel-distributed processor that has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process, and,
2. Inter-neuron connection strengths known as synaptic weights are used to store the knowledge.

Neural networks can also be defined as parameterized computational nonlinear algorithms for (numerical) data/signal/image processing. These algorithms are either implemented on a general-purpose computer or are built into a dedicated hardware.

Artificial Neural Networks thus is an information-processing system. In this information-processing system, the elements called as neurons, process the information. The signals are transmitted by means of connection links. The links possess an associated weight, which is multiplied along with the incoming signal (net input) for any typical neural net. The output signal is obtained by applying activations to the net input.

An artificial neuron is characterized by:

1. Architecture (connection between neurons)
2. Training or learning (determining weights on the connections)
3. Activation function

All these are discussed in detail in the forthcoming subsections. The structure of the simple artificial neural network is shown in Fig. 2.1.

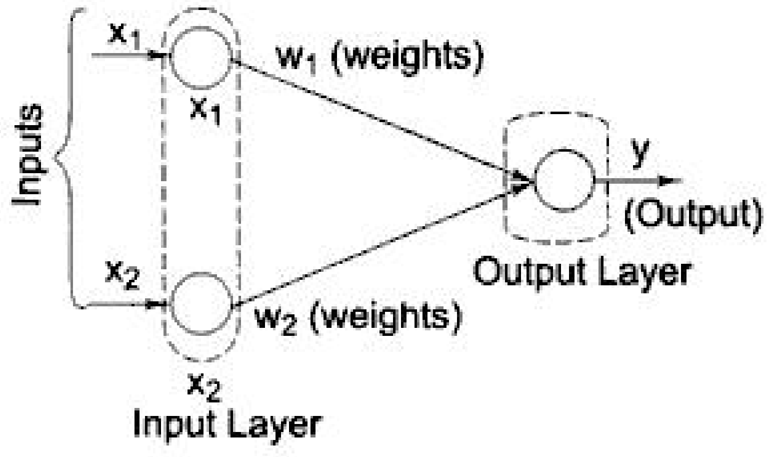


Fig. 2.1 *A Simple Artificial Neural Net*

Figure 2.1 shows a simple artificial neural network with two input neurons (x_1, x_2) and one output neuron (y). The inter connected weights are given by w_1 and w_2. An artificial neuron is a *p*-input single-output signal-processing element, which can be thought of as a simple model of a non-branching biological neuron. In Fig 2.1, various inputs to the network are represented by the mathematical symbol, x (n). Each of these inputs are multiplied by a connection weight. These weights are represented by w(n). In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then delivered as output. This process lends itself to physical implementation on a large scale in a small package. This electronic implementation is still possible with other network structures, which utilize different summing functions as well as different transfer functions.

Why Artificial Neural Networks?

The long course of evolution has given the human brain many desirable characteristics not present in Von Neumann or modern parallel computers. These include

- Massive parallelism,
- Distributed representation and computation,
- Learning ability,
- Generalization ability,
- Adaptivity,
- Inherent contextual information processing
- Fault tolerance, and
- Low energy consumption.

It is hoped that devices based on biological neural networks will posses some of these desirable characteristics. Modern digital computers outperform humans in the domain of numeric computation and related symbol manipulation. However, humans can effortlessly solve complex perceptual problems (like recognizing a man in a crowd from a mere glimpse of his face) at such a high speed and extent as to dwarf the world's fastest computer. Why is there such a remarkable difference in their performance? The biological neural system architecture is completely different from the Von Neumann architecture (see Table 2.1). This difference significantly affects the type of functions each computational model can best perform.

Numerous efforts to develop "intelligent" programs based on Von Neumann's centralized architecture have not resulted in any general-purpose intelligent programs. Inspired by biological neural networks, ANNs are massively parallel computing systems consisting of an extremely large number of simple processors with many interconnections. ANN models attempt to use some "organizational" principles believed to be used in the human brain.

Table 2.1 *Von Neumann Computer* **Versus** *Biological Neural System*

	Von Neumann Computer	*Biological Neural System*
Processor	complex High speed One or a few	Simple Low speed A large number
Memory	Separate from a processor Localized Noncontent addressable	Integrated into Processor Distributed Content addressable
Computing	Centralized Sequential Stored programs	Distributed Parallel Self-learning
Raliability	Very vulnerable	Robust
Expertise	Numerical and symbolic manipulations	Perceptual problems
Operating environment	Well-defined, well-constrained	Poorly defined, unconstrained

Either humans or other computer techniques can use neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, to extract patterns and detect trends that are too complex to be noticed. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.

Other advantages include:

1. *Adaptive learning:* An ability to learn how to do tasks based on the data given for training or initial experience.
2. *Self-organization:* An ANN can create its own organisation or representation of the information it receives during learning time.
3. *Real-time operation:* ANN computations may be carried out in parallel, using special hardware devices designed and manufactured to take advantage of this capability.
4. *Fault tolerance via redundant information coding:* Partial destruction of a network leads to a corresponding degradation of performance. However, some network capabilities may be retained even after major network damaǫe due to this feature.

2.3 Historical Development of Neural Networks

The historical development of the neural networks can be traced as follows:

- **1943—McCulloch and Pitts: start of the modern era of neural networks**
  This forms a logical calculus of neural networks. A network consists of sufficient number of neurons (using a simple model) and properly set synaptic connections can compute any computable

function. A simple logic function is performed by a neuron in this case based upon the weights set in the McCulloch–Pitts neuron. The arrangement of neuron in this case may be represented as a combination of logic functions. The most important feature of this type of neuron is the concept of threshold. When the net input to a particular neuron is greater than the specified threshold by the user, then the neuron fires. Logic circuits are found to use this type of neurons extensively.

- **1949—Hebb's book "The organization of behavior"**
  An explicit statement of a physiological learning rule for *synaptic modification* was presented for the first time. Hebb proposed that the connectivity of the brain is continually changing as an organism learns differing functional tasks, and that neural assemblies are created by such changes.

  Hebb's work was immensely influential among psychologists. The concept behind the Hebb theory is that if two neurons are found to be active simultaneously the strength of connection between the two neurons should be increased. This concept is similar to that of the correlation matrix learning.
- **1958—Rosenblatt introduces Perceptron (Block [1962], Minsky and Papert [1988])**
  In Perceptron network the weights on the connection paths can be adjusted. A method of iterative weight adjustment can be used in the Perceptron net. The Perceptron net is found to converge if the weights obtained allow the net to reproduce exactly all the training input and target output vector pairs.
- **1960—Widrow and Hoff introduce adaline**
  ADALINE, abbreviated from Adaptive Linear Neuron uses a learning rule called as Least Mean Square rule or Delta rule. This rule is found to adjust the weights so as to reduce the difference between the net input to the output unit and the desired output. The convergence criteria in this case are the reduction of mean square error to a minimum value. This delta rule for a single layer net can be called a precursor of the backpropagation net used for multi-layer nets. The multi-layer extensions of Adaline formed the Madaline [Widrow and Lehr, 1990].
- **1982—John Hopfield's networks**
  Hopfield showed how to use "Ising spin glass" type of model to store information in dynamically stable networks. His work paved the way for physicists to enter neural modeling, thereby transforming the field of neural networks. These nets are widely used as associative memory nets. The Hopfield nets are found to be both continuous valued and discrete valued. This net provides an efficient solution for the 'Travelling Sales-man Problem'.
- **1972—Kohonen's Self-Organizing Maps (SOM)**
  Kohonen's Self-Organizing Maps are capable of reproducing important aspects of the structure of biological neural nets. They make use of data representation using topographic maps, which are common in the nervous systems. SOM also has a wide range of applications. It shows how the output layer can pick up the correlational structure (from the inputs) in the form of the spatial arrangement of units. These nets are applied to many recognition problems.
- **1985—Parker, 1986—Lecum**
  During this period the backpropagation net paved its way into the Neural Networks. This method propagates the error information at the output units back to the hidden units using a generalized delta rule. This net is basically a multilayer, feed forward net trained by means of backpropagation. Originally, even though the work was performed by Parker (1985) the credit of publishing this net goes to Rumelhart, Hinton and Williams (1986). Backpropogation net emerged as the most popular

learning algorithm for the training of multilayer perceptrons and has been the workhorse for many neural network applications.

- **1988—Grossberg**
  Grossberg developed a learning rule similar to that of Kohonen, which is widely used in the Counter Propagation net. This Grossberg type of learning is also used as outstar learning. This learning occurs for all the units in a particular layer; no competition among these units is assumed.
- **1987, 1990—Carpenter and Grossberg**
  Carpenter and Grossberg invented Adaptive Resonance Theory (ART). ART was designed for both binary inputs and the continuous valued inputs. The design for the binary inputs formed ART1, and ART2 came into being when the design became applicable to the continuous valued inputs. The most important feature of these nets is that the input patterns can be presented in any order.
- **1988—Broomhead and Lowe** developed Radial Basis Functions (RBF). This is also a multi-layer net that is quiet similar to the back propagation net.
- **1990—Vapnik** developed the support vector machine.

2.4 Biological Neural Networks

A biological neuron or a nerve cell consists of **synapses, dendrites, the cell body (or hillock), and the axon**. The "building blocks" are discussed as follows:

- The synapses are elementary signal processing devices
  - A synapse is a biochemical device, which converts a pre-synaptic electrical signal into a chemical signal and then back into a post-synaptic electrical signal.
  - The input pulse train has its amplitude modified by parameters stored in the synapse. The nature of this modification depends on the type of the synapse, which can be either inhibitory or excitatory.
- The postsynaptic signals are aggregated and transferred along the dendrites to the nerve cell body.
- The cell body generates the output neuronal signal, a spike, which is transferred along the axon to the synaptic terminals of other neurons.
- The frequency of firing of a neuron is proportional to the total synaptic activities and is controlled by the synaptic parameters (weights).
- The pyramidal cell can receive 104 synaptic inputs and it can fan-out the output signal to thousands of target cells—a connectivity difficult to achieve in the artificial neural networks.

In general the function of the main elements can be given as,

Dendrite	–	Receives signals from other neurons
Soma	–	Sums all the incoming signals
Axon	–	When a particular amount of input is received, then the cell fires. It transmits signal through axon to other cells.

The fundamental processing element of a neural network is a neuron. This building block of human awareness encompasses a few general capabilities. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then outputs the final result. Figure 2.2 shows the relationship of these four parts.

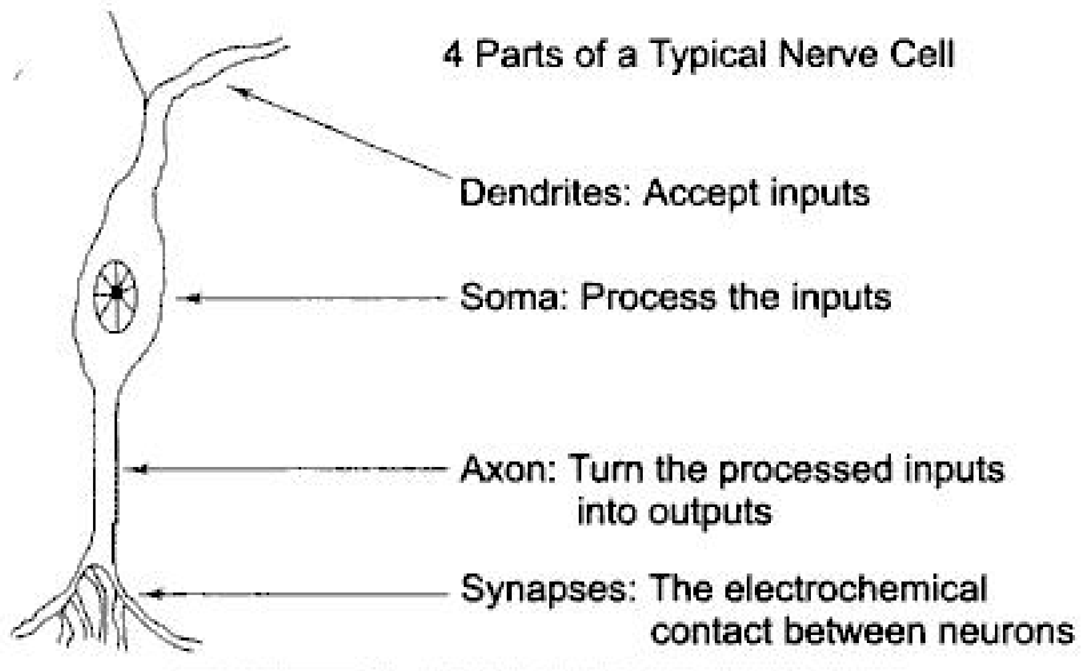

4 Parts of a Typical Nerve Cell
Dendrites: Accept inputs
Soma: Process the inputs
Axon: Turn the processed inputs into outputs
Synapses: The electrochemical contact between neurons

Fig. 2.2 *A Biological Neuron*

The properties of the biological neuron pose some features on the artificial neuron, They are:

1. Signals are received by the processing elements. This element sums the weighted inputs.
2. The weight at the receiving end has the capability to modify the incoming signal.
3. The neuron fires (transmits output), when sufficient input is obtained.
4. The output produced from one neuron may be transmitted to other neurons.
5. The processing of information is found to be local.
6. The weights can be modified by experience.
7. Neurotransmitters for the synapse may be excitatory or inhibitory.
8. Both artificial and biological neurons have inbuilt fault tolerance.

Figure 2.3 and Table 2.2 indicate how the biological neural net is associated with the artificial neural net.

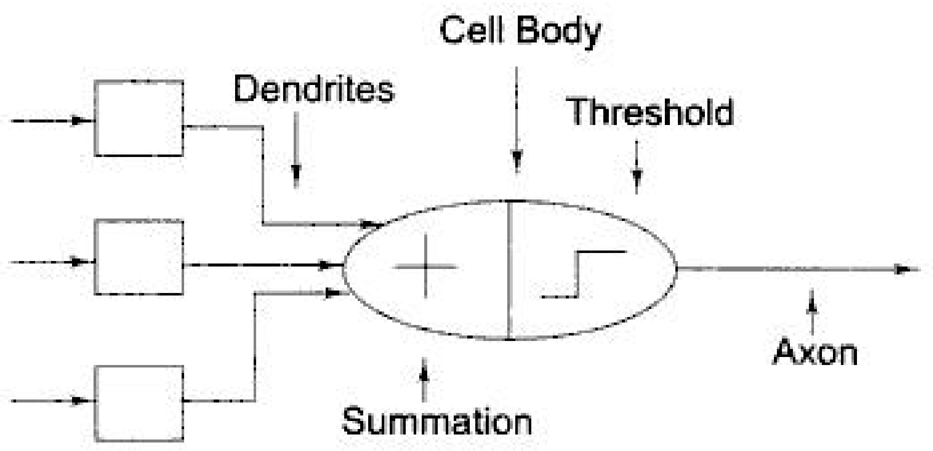


Fig. 2.3 *Association of Biological Net with Artificial Net*

Table 2.2 *Associated Terminologies of Biological and Arifical Neural Net*

Biological Neural Network	*Artificial Neural Network*
Cell Body	Neurons
Dendrite	Weights or interconnections
Soma	Net input
Axon	Output

2.5 Comparison Between the Brain and the Computer

The main differences between the brain and the computer are:

- Biological neurons, the basic building blocks of the brain, are slower than silicon logic gates. The neurons operate in milliseconds, which is about six orders of magnitude slower than the silicon gates operating in the nanosecond range.
- The brain makes up for the slow rate of operation with two factors:
  - a huge number of nerve cells (neurons) and interconnections between them. The human brain contains approximately 10^{14} to 10^{15} interconnections.
  - the function of a biological neuron seems to be much more complex than that of a logic gate.
- The brain is very energy efficient. It consumes only about 10–16 joules per operation per second, comparing with 10-6 joules per operation per sec, for a digital computer.
- The brain is a highly complex, non-linear, parallel information processing system. It performs tasks like pattern recognition, perception, motor control, many times faster than the fastest digital computers.
- Consider an efficiency of the visual system which provides a representation of the environment which enables us to interact with the environment. For example, a complex task of perceptual recognition, e.g. recognition of a familiar face embedded in an unfamiliar scene can be accomplished in 100–200 ms, whereas tasks of much lesser complexity can take hours if not days on conventional computers.
- As another example consider an efficiency of the SONAR system of a bat. SONAR is an active echolocation system. A bat SONAR provides information about the distance from a target, its relative velocity and size, the size of various features of the target, and its azimuth and elevation. The complex neural computations needed to extract all this information from the target echo occur within the brain, which has the size of a plum. The precision and success rate of the target location is rather impossible to match by RADAR or SONAR engineers.

2.6 Comparison Between Artificial and Biological Neural Network

Table 2.3 shows the major differences between the biological and the artificial neural network.

Feed Forward Net

Feed forward networks may have a single layer of weights where the inputs are directly connected to the outputs, or multiple layers with intervening sets of hidden units (see Fig. 2.4). Neural networks use hidden units to create internal representations of the input patterns. In fact, it has been shown that given enough hidden units, it is possible to approximate arbitrarily any function with a simple feed forward network. This result has encouraged people to use neural networks to solve many kinds of problems.

1. *Single layer net:* It is a feed forward net. It has only one layer of weighted interconnections. The inputs may be connected fully to the output units. But there is a chance that none of the input units and output units are connected with other input and output units respectively. There is also a case where, the input units are connected with other input units and output units with other output units. In a single layer net, the weights from one output unit do not influence the weights for other output units.
2. *Multi layer net:* It is also a feed forward net i.e., the net where the signals flow from the input units to the output units in a forward direction. The multi-layer net pose one or more layers of nodes between the input and output units. This is advantageous over single layer net in the sense that, it can be used to solve more complicated problems.

Competitive Net

The competitive net is similar to a single-layered feed forward network except that there are connections, usually negative, between the output nodes. Because of these connections the output nodes tend to compete to represent the current input pattern. Sometimes the output layer is completely connected and sometimes the connections are restricted to units that are close to each other (in some neighborhood). With an appropriate learning algorithm the latter type of network can be made to organize itself topologically. In a topological map, neurons near each other represent similar input patterns. Networks of this kind have been used to explain the formation of topological maps that occur in many animal sensory systems including vision, audition, touch and smell

Recurrent Net

The fully recurrent network is perhaps the simplest of neural network architectures. All units are connected to all other units and every unit is both an input and an output. Typically, a set of patterns is instantiated on all of the units, one at a time. As each pattern is instantiated the weights are modified. When a degraded version of one of the patterns is presented, the network attempts to reconstruct the pattern.

Recurrent networks are also useful in that they allow networks to process sequential information. Processing in recurrent networks depends on the state of the network at the last time step. Consequently, the response to the current input depends on previous inputs. Figure 2.4 shows two such networks: the simple recurrent network and the Jordan network.

2.7.2 Setting the Weights

The method of setting the value for the weights enables the process of learning or training. The process of modifying the weights in the connections between network layers with the objective of achieving the expected output is called training a network. The internal process that takes place when a network is trained is called learning. Generally, there are three types of training as follows.

Binary Step Function

The function is given by,

$$f(x) = \begin{cases} 1; & \text{if} \quad f(x) \geq \theta \\ 0; & \text{if} \quad f(x) < \theta \end{cases}$$

The threshold 'θ' is discussed in the following sections. Mostly single layer nets use binary step function for calculating the output from the net input. The binary step function is also called as threshold function or Heaviside function. Figure 2.7 shows a binary step function.

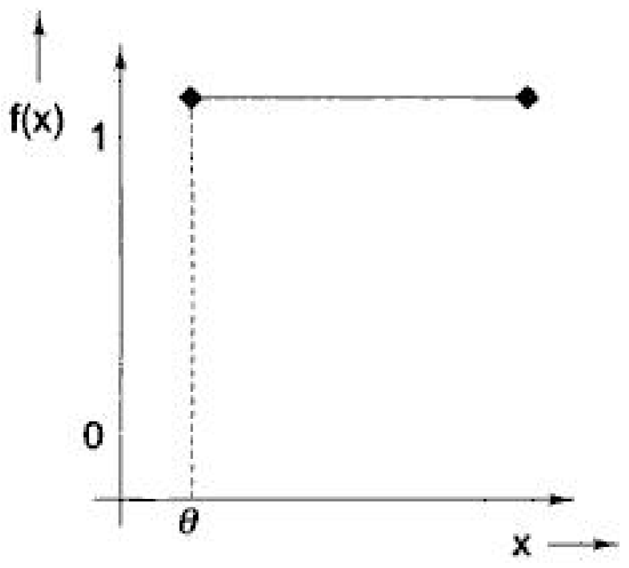


Fig. 2.7 | *Binary Step Function*

2.8.3 Sigmoidal Functions

These functions are usually S-shaped curves. The hyperbolic and logistic functions are commonly used. These are used in multilayer nets like back propagation network, radial basis function network etc. There are two main types of sigmoidal functions:

Binary Sigmoidal Function

This is also called logistic function. It ranges between 0 to 1.

$$f(x) = \frac{1}{1 + \exp(-\sigma x)}$$

where, σ is called the steepness parameter. If $f(x)$ is differentiated we get,

$$f'(x) = \sigma f(x) [1 - f(x)].$$

Figure 2.8 shows the binary sigmoidal function

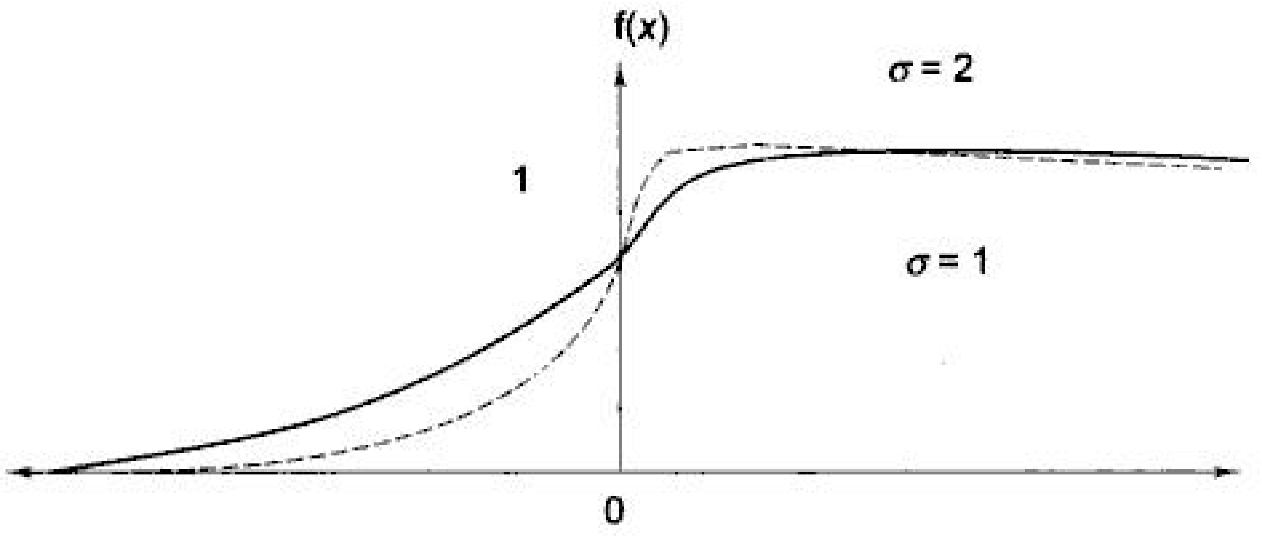


Fig. 2.8 | *Binary Sigmoidal Function*

Biploar Sigmoidal Function

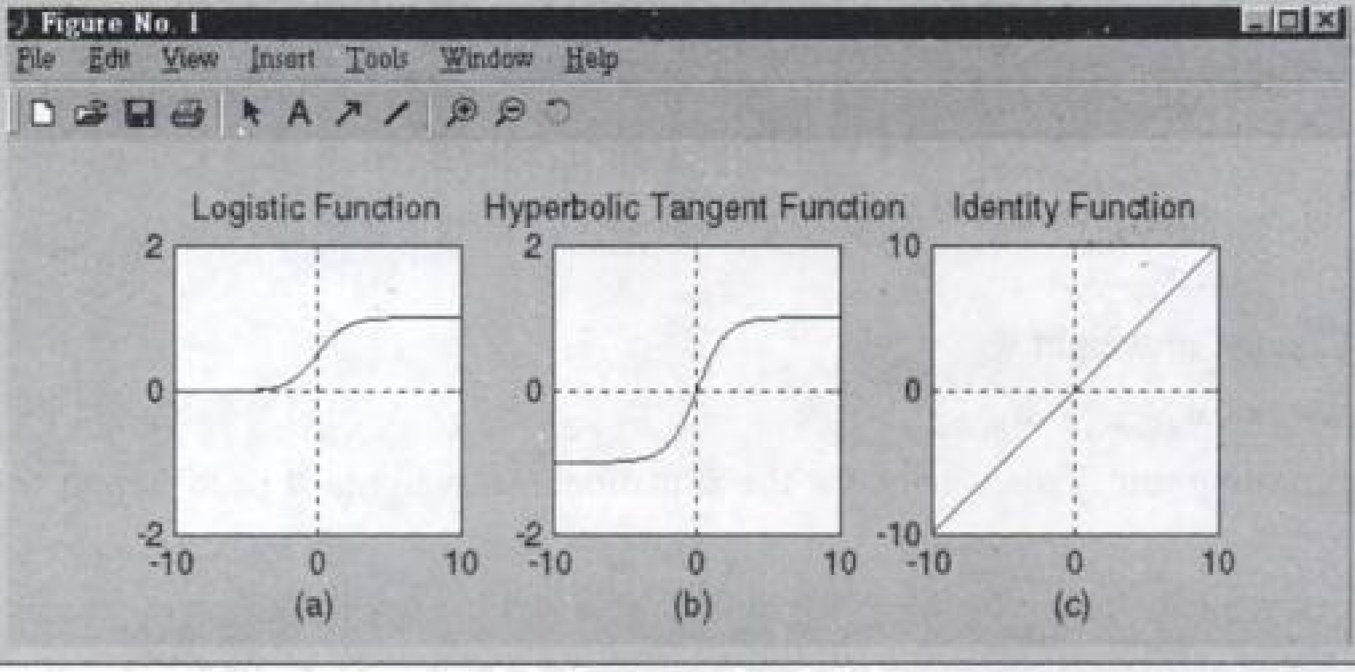
The desired range here is between + 1 and – 1. This function is related to the hyperbolic tangent function. The bipolar sigmoidal function is given as,

$$b(x) = 2f(x) - 1$$

```
y3 = x;
subplot(231); plot(x, y1); grid on;
axis([min(x) max(x) -2 2]);
title('Logistic Function');
xlabel('(a)');
axis('square');
subplot(232); plot(x, y2); grid on;
axis([min(x) max(x) -2 2]);
title('Hyperbolic Tangent Function');
xlabel('(b)');
axis('square');
subplot(233); plot(x, y3); grid on;
axis([min(x) max(x) min(x) max(x)]);
title('Identity Function');
xlabel('(c)');
axis('square');
```

`Output`

The output response is given as



Example 2.2 **If the net input to an output neuron is 0.64, calculate its output when the activation function is (a) binary sigmoidal (b) bipolar sigmoidal.**

Solution Net input to the neuron = 0.64.

(a) For binary activation function,

$$Y = f(\text{netinput}) = \frac{1}{1 + e^{-0.64}} = 0.6548$$

(b) For bipolar activation function,

$$Y = f(\text{netinput}) = \frac{2}{1 + e^{-0.64}} - 1 = 0.3095$$

$x_{n+1} \ldots x_{n+m}$ are inhibitory denoted by '– p'. The McCulloch-Pitts neuron Y has the activation function,

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{-in} \geq \theta \\ 0 & \text{if } y_{-in} < \theta \end{cases}$$

where θ is the threshold and y_{-in} is the total net input signal received by neuron Y.

The threshold θ should satisfy the relation.

$$\theta > nw - p$$

This is the condition for absolute inhibition.

The McCulloch-Pitts neuron will fire if it receives k or more excitatory inputs and no inhibitory inputs, where

$$k_w \geq \theta > (k-1)\,w.$$

Solved Examples

Example 3.1 Generate the output of logic AND function by McCulloch-Pitts neuron model.

Solution The AND function returns a true value only if both the inputs are true, else it returns a false value. '1' represents true value and '0' represents false value.

The truth table for AND function is,

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

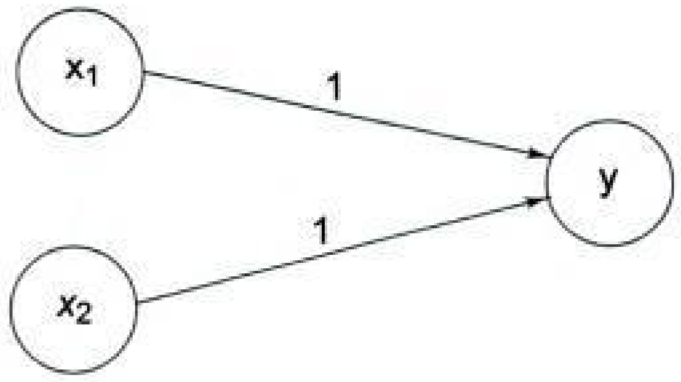


Fig. 3.2 McCulloch-Pitts Neuron to Perform Logical AND Function

A McCulloch-Pitts neuron to implement AND function is shown in Fig. 3.2. The threshold on unit Y is 2.

The output Y is,

$$Y = f(y_{in})$$

The net input is given by

$$y_{in} = \sum_i \text{weights} * \text{input}$$

$$y_{in} = 1 * x_1 + 1 * x_2$$

$$y_{in} = x_1 + x_2$$

From this the activations of output neuron can be formed.

$$Y = f(y_{in}) = \begin{cases} 1 & \text{if} \quad y_{-in} \geq 2 \\ 0 & \text{if} \quad y_{-in} < 2 \end{cases}$$

The activations of z_1 and z_2 are given as,

$$z_1 = (z_{in-1}) = \begin{cases} 1 & \text{if} \quad z_{in-1} \geq 1 \\ 0 & \text{if} \quad z_{in-1} < 1 \end{cases}$$

$$z_2 = (z_{in-2}) = \begin{cases} 1 & \text{if} \quad z_{in-2} \geq 1 \\ 0 & \text{if} \quad z_{in-2} < 1 \end{cases}$$

The calculation of net input and activations of z_1 and z_2 are shown below.

$z_1 = (x_1 \text{ ANDNOT } x_2)$ $\quad z_{in-1} = x_1w_1 + x_2w_2$

x_1	x_2	z_{in-1}	z_1
		$w_1 = 1, w_2 = -1$	
1	1	0	0
1	0	1	1
0	1	−1	0
0	0	0	0

$z_2 = (x_2 \text{ ANDNOT } x_1)$ $\quad z_{in-2} = x_1w_1 + x_2w_2$

x_1	x_2	Z_{in-2}	z_2
		$w_1 = -1, w_2 = 1$	
1	1	0	0
1	0	−1	0
0	1	1	1
0	0	0	0

The activation for the output unit y is 1.

$$y = f(y_{in}) = \begin{cases} 1 & \text{if} \quad y_{in} \geq 1 \\ 0 & \text{if} \quad y_{in} < 1 \end{cases}$$

Presenting the input patterns (z_1 and z_2) and calculating net input and activations gives output of XOR.

Here, $\quad y_{in} = z_1w_1 + z_2w_2$

z_1	z_2	y_{in}	$y = z_1$ or z_2
		$w_1 = 1, w_2 = 1$	
0	0	0	0
1	0	1	1
0	1	1	1
0	0	0	0

Thus the Exclusive-OR function is realized.

Example 3.6 Consider the neural network of McCulloch-Pitts neuron shown in Fig. 3.7. Each neuron (other than the input neurons N_1 and N_2) has a threshold of 2.

(a) Define the response of neuron N_5 at time t in terms of the activations of the input neurons, N_1 and N_2 at the appropriate time.

(b) Show that the activation of each neuron that results from an input signal of $N_1 = 1, N_2 = 0$ at $t = 0$.

```
w2=input('weight w2=');
disp('Enter Threshold Value');
theta=input('theta=');
y=[0 0 0 0];
x1=[0 0 1 1];
x2=[0 1 0 1];
z=[0 0 1 0];
con=1;
while con
  zin=x1*w1+x2*w2;
  for i=1:4
      if zin(i)>=theta
          y(i)=1;
      else
          y(i)=0;
      end
  end
  disp('Output of Net');
  disp(y);
  if y==z
      con=0;
  else
     disp('Net is not learning enter another set of weights and Threshold value');
          w1=input('weight w1=');
          w2=input('weight w2=');
          theta=input('theta=');
   end
end
disp('Mcculloch-Pitts Net for ANDNOT function');
disp('Weights of Neuron');
disp(w1);
disp(w2);
disp('Threshold value');
disp(theta);
```

Output

```
Enter weights
Weight w1=1
weight w2=1
Enter Threshold Value
theta=0.1
Output of Net
     0     1     1     1
Net is not learning enter another set of weights and Threshold value
Weight w1=1
weight w2=-1
theta=1
```

This type of synapse is called Hebbian synapse. The four key mechanisms that characterize a Hebbian synapse are time dependent mechanism, local mechanism, interactive mechanism and correlational mechanism.

The simplest form of Hebbian learning is described by,

$$\Delta w = x_i\, y$$

This Hebbian learning rule represents a purely feed forward, unsupervised learning. It states that if the cross product of output and input is positive, this results in increase of weight, otherwise the weight decreases.

In some cases, the Hebbian rule needs to be modified to counteract unconstrained growth of weight values, which takes place when excitations and response consistently agree in sign. This corresponds to the Hebbian learning rule with saturation of weights at a certain preset level.

3.3.2 Perceptron Learning Rule

For the perceptron learning rule, the learning signal is the difference between the desired and actual neuron's response. This type of learning is supervised.

The fact that the weight vector is perpendicular to the plane separating the input patterns during the learning processes, can be used to interpret the degree of difficulty of training a perceptron for different types of input.

The perceptron learning rule states that for a finite '*n*' number of input training vectors,

$$x(n) \quad \text{where} \quad n = 1 \text{ to } N$$

each with an associated target value,

$$t(n) \quad \text{where} \quad n = 1 \text{ to } N$$

which is +1 or – 1, and an activation function $y - f(y_{-in})$, where,

$$y = \begin{cases} 1 & \text{if} \quad y > \vartheta \\ 0 & \text{if} \quad -\vartheta \le y_{in} \le \vartheta \\ -1 & \text{if} \quad y_{in} < -\vartheta \end{cases}$$

the weight updation is given by

if $y \neq t$, then

$$w_{new} = w_{old} + tx$$

if $y = t$, then there is no change in weights.

The perceptron learning rule is of central importance for supervised learning of neural networks. The weights can be initialized at any values in this method.

There is a perceptron learning rule convergence theorem which states, "If there is a weight vector w^* such that $f(x(p)\ w^*) = t(p)$ for all p, then for any starting vector w_1 the perceptron learning rule will converge to a weight vector that gives the correct response for all training patterns, and this will be done in a finite number of steps".

3.3.3 Delta Learning Rule (Widrow-Hoff Rule or Least Mean Square (LMS) Rule)

The delta learning rule is also referred to as Widrow-Hoff rule, named due to the originators (Widrow and Hoff, 1960). The delta learning rule is valid only for continuous activation functions and in the

the dot product or Euclidean norm. Euclidean norm is most widely used because dot product may require normalization.

3.3.5 Out Star Learning Rule

Out star learning rule can be well explained when the neurons are arranged in a layer. This rule is designed to produce the desired response t from the layer of n neurons. This type of learning is also called as Grossberg learning.

Out star learning occurs for all units in a particular layer and no competition among these units are assumed. However the forms of weight updates for Kohonen learning and Grossberg learning are closely related.

In the case of out star learning

$$\Delta w_{jk} = \begin{cases} \propto (y_k - w_{jk}) & \text{if neuron j wins the competition} \\ 0 & \text{if neuron j losses the competition} \end{cases}$$

The rule is used to provide learning of repetitive and characteristic properties of input–output relationships. Though it is concerned with supervised learning, it allows the network to extract statistical properties of the input and output signals. It ensures that the output pattern becomes similar to the undistorted desired output after repetitively applying on distorted output versions. The weight change here will be a times the error calculated.

3.3.6 Boltzmann Learning

The learning is a stochastic learning. A neural net designed based on this learning is called Boltzmann learning. In this learning, the neurons constitute a recurrent structure and they work in binary form. This learning is characterized by an energy function, E, the value of which is determined by the particular states occupied by the individual neurons of the machine, given by,

$$E = \frac{-1}{2} \sum_i \sum_i w_{ij} x_j x_i \quad i \neq j$$

where x_i is the state of neuron i and w_{ij} is the weight from neuron i to neuron j. The value $i \neq j$ means that none of the neurons in the machine has self feedback. The operation of machine is performed by choosing a neuron at random.

The neurons of this learning process are divided into two groups; visible and hidden. In visible neurons there is an interface between the network and the environment in which it operates but in hidden neurons, they operates independent of the environment. The visible neurons might be clamped onto specific states determined by the environment, called as clamped condition. On the other hand, there is free-running condition, in which all the neurons are allowed to operate freely.

3.3.7 Memory Based Learning

In memory based learning, all the previous experiences are stored in a large memory of correctly classified input–output examples: $(x_i, t_i)_{i=1}^{N}$ where x_i is the input vector and t_j is the desired response. The desired response is a scalar.

For the 3rd and 4th epoch the separating line remains the same, hence this line separates the boundary regions as shown in Fig. 3.9.

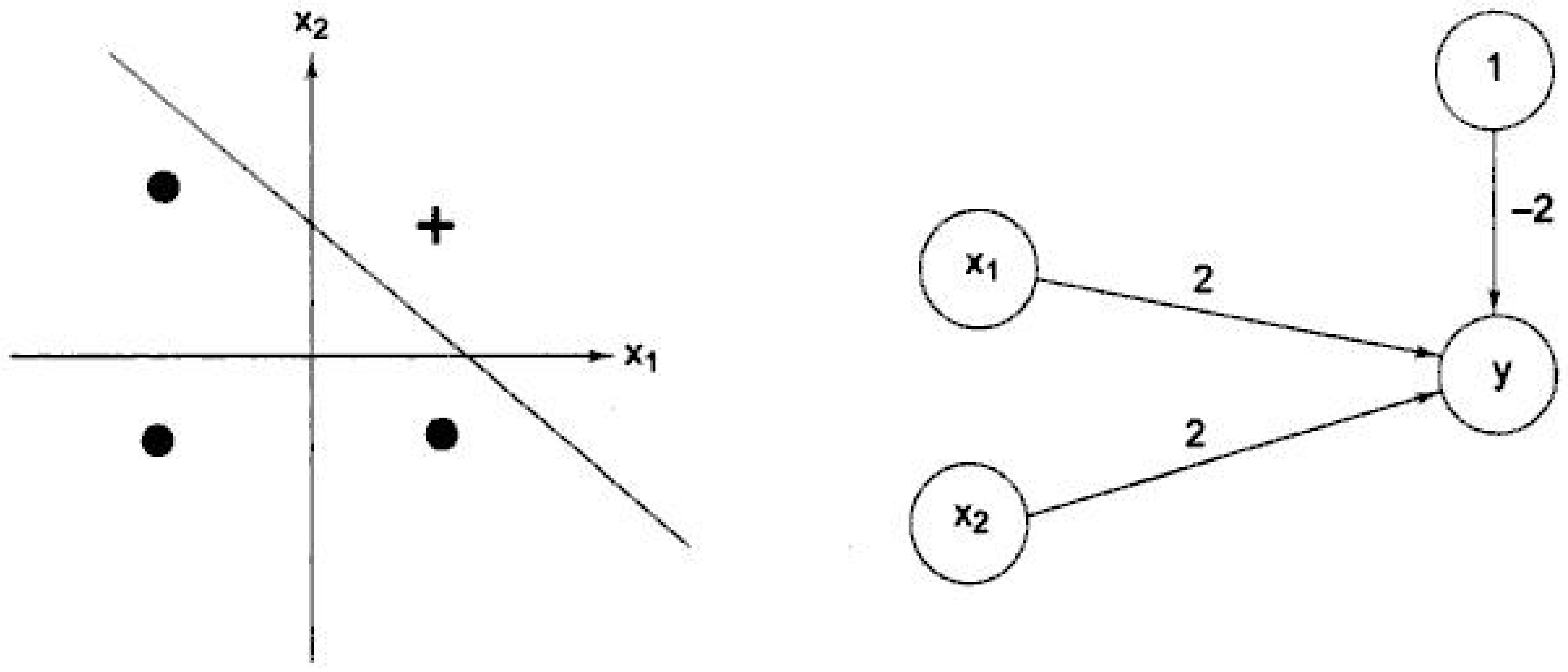


Fig. 3.9 *Hebb Net for AND Function*

The same procedure can be repeated for generating the logic function OR, NOT, AND NOT etc.

Example 3.10 Apply the Hebb net to the training patterns that define XOR function with bipolar input and targets.

Solution

Input			Target
x_1	x_2	b	y
1	1	1	− 1
1	− 1	1	1
− 1	1	1	1
− 1	− 1	1	− 1

By Hebb training algorithm, assigning initial values of the weights w_1 & w_2 to be zero and bias to be zero.

$w_1 = w_2 = 0$ and $b = 0$

Input			Target	Weight Changes			Weights		
$(x_1$	x_2	$b)$	y	Δw_1	Δw_2	Δb	w_1	w_2	B
							(0	0	0)
1	1	1	− 1	− 1	− 1	− 1	− 1	− 1	− 1
1	− 1	1	1	+ 1	− 1	1	0	− 2	0
− 1	1	1	1	− 1	1	1	− 1	− 1	1
− 1	− 1	1	− 1	1	1	− 1	0	0	0

The weight changes are called using,

$\Delta w_i = x_i\, y$ and $\Delta b = y$

Solution The '*' symbol indicates that there exist a '+ 1' and '.' Symbol indicates that there exist a '– 1'.

The input is given by, E = [1 1 1 1 1 – 1 – 1 – 1 1 1 1 1 1 – 1 – 1 – 1 1 1 1 1]
F = [1 1 1 1 1 – 1 – 1 – 1 1 1 1 1 1 – 1 – 1 – 1 1 – 1 – 1 – 1];

The MATLAB program is given as follows

Program

```
%Hebb Net to classify two dimensional input patterns
clear;
clc;
%Input Patterns
E=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1];
F=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 -1 -1 -1];
x(1,1:20)=E;
x(2,1:20)=F;
w(1:20)=0;
t=[1 -1];
b=0;
for i=1:2
    w=w+x(i,1:20)*t(i);
    b=b+t(i);
end
disp('Weight matrix');
disp(w);
disp('Bias');
disp(b);
```

Output

```
Weight matrix
  Columns 1 through 18
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  2
  Columns 19 through 20
     2    2
Bias
       0
```

Summary

The fundamental models of the artificial neural network was discussed in this chapter. The models were used to generate the logic functions like AND, OR, XOR etc. The linear seperability concept to obtain the decision boundary of the regions and the Hebb rule for the pattern classification problem was illustrated. The learning rules used in various networks for weight updation process were also derived in this chapter.

CHAPTER

4

Perceptron Networks

What You Will Learn

- How the perceptron learning rule is better than the Hebb rule.
- Layer structure in the original perceptrons.
- Learning and training algorithms in the perceptron network .
- Architecture, algorithm and the application procedure of the perceptron net.
- Derivation of perceptron algorithm for several output classes .
- Applications of multilayer perceptrons.

4.1 Introduction

Frank Rosenblatt [1962], and Minsky and Papert [1988], developed large class of artificial neural networks called Perceptrons. The perceptron learning rule uses an iterative weight adjustment that is more powerful than the Hebb rule. The perceptrons use threshold output function and the McCulloch-Pitts model of a neuron. Their iterative learning converges to correct weights, i.e. the weights that produce the exact output value for the training input pattern. The original perceptron is found to have three layers, sensory, associator and response units as shown in Fig. 4.1.

Step 5: Compute the activation output of each output unit $y_{-inj} = b_j + \sum_i x_i w_i$ for $j = 1$ to m.

$$y_j = f(y_{-inj}) = \begin{cases} 1, & \text{if} \quad y_{-inj} > 0 \\ 0, & \text{if} \quad -\theta \le y_{-inj} \le \theta \\ -1, & \text{if} \quad y_{-inj} < -\theta \end{cases}$$

Step 6: The weights and bias are to be updated for $j = 1$ to m and $i = 1$ to n.

If $y_j \neq t_j$ and $x_i \neq 0$, then

$$w_{ij(\text{new})} = w_{ij(\text{old})} + \alpha t_j x_i$$
$$b_j(\text{new}) = b_j(\text{old}) + \alpha t_j.$$

else if $\quad y_j = t_j$

$$w_{ij(\text{new})} = w_{ij(\text{old})}$$
$$b_j(\text{new}) = b_j(\text{old}).$$

That is, the biases and weights remain unchanged.

Step 7: Test for stopping condition.

The stopping condition may be the weight changes.

Solved Examples

Example 4.1 Develop a perceptron for the AND function with bipolar inputs and targets.

Solution The training pattern for AND function can be,

Input			Target
X_1	X_2	b	t
1	1	1	1
− 1	1	1	− 1
1	− 1	1	− 1
− 1	− 1	1	− 1

Step 1: Initial weights $w_1 = w_2 = 0$ and $b = 0, \alpha = 1, \theta = 0$.

Step 2: Begin computation.

Step 3: For input pair (1, 1): 1, do Steps 4–6

Step 4: Set activations of input units

$$x_i = (1, 1).$$

Step 5: Calculate the net input.

$$y_{-in} = b + \sum x_i w_i = 0 + 1 \times 0 + 1 \times 0 = 0$$

Applying the activation,

$$y = f(y_{-in}) = \begin{cases} 1, & \text{if} \quad y_{-in} > 0 \\ 0, & \text{if} \quad -0 \le y_{-in} \le 0 \\ -1, & \text{if} \quad y_{-in} < -0 \end{cases}$$

Therefore $y = 0$.

Input					Net	Output	Target	Weight Changes					Weights				
x_1	x_2	x_3	x_4	1	y_{in}	y	t	Δw_1	Δw_2	Δw_3	Δw_4	Δb	w_1	w_2	w_3	w_4	b
													(0	0	0	0	0)
Epoch 1:																	
1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	−1	1	3	1	−1	−1	−1	−1	1	−1	0	0	0	2	0
−1	1	−1	−1	1	0	0	1	−1	1	−1	−1	1	−1	1	−1	−1	1
1	−1	−1	1	1	−1	−1	−1	0	0	0	0	0	−1	1	−1	−1	1
Epoch 2:											initial →		0	0	0	2	0
1	1	1	1	1	2	1	1	0	0	0	0	0	0	0	0	2	0
1	1	1	−1	1	−2	−1	−1	0	0	0	0	0	0	0	0	2	0
		Initial →											−1	1	−1	−1	1
−1	1	−1	−1	1	5	1	1	0	0	0	0	0	−1	1	−1	−1	1
1	−1	−1	1	1	−1	−1	−1	0	0	0	0	0	−1	1	−1	−1	1

The final weights from Epoch 1 are used as the initial weights for Epoch 2. Thus the output is equal to target by training for suitable weights.

Testing the response of the net

The final weights are,

For the 1st set of input, $w_1 = 0$, $w_2 = 0$, $w_3 = 0$, $w_4 = 2$, $b = 0$, and

For the 2nd set of input, $w_1 = -1$, $w_2 = 1$ $w_3 = -1$, $w_4 = -1$, $b = 1$

The net input is, $y_{in} = b + \sum x_i w_i$

For the 1st set of inputs,

(i) (1 1 1 1 1)

$$y_{-in1} = 0 + 0 \times 1 + 0 \times 1 + 0 \times 1 + 2 \times 1 = 2 > 0$$

Applying activation, $y_1 = f(y_{-in1}) = 1.$

(ii) (1 1 1 − 1 1)

$$y_{-in1} = 0 + 0 \times 1 + 0 \times 1 + 0 \times 1 + 2 \times -1 = -2 < 0$$

Applying activation, $y_2 = f(y_{-in2}) = -1.$

For 2nd set of inputs,

(i) (− 1 1 − 1 − 1 1)

$$y_{-in1} = 1 + -1 \times -1 + 1 \times 1 + -1 \times -1 + -1 \times -1 + 1 \times 1 = 5 > 0$$

Applying activation, $y_1 = f(y_{-in1}) = 1$

(ii) (1 − 1 − 1 1 1)

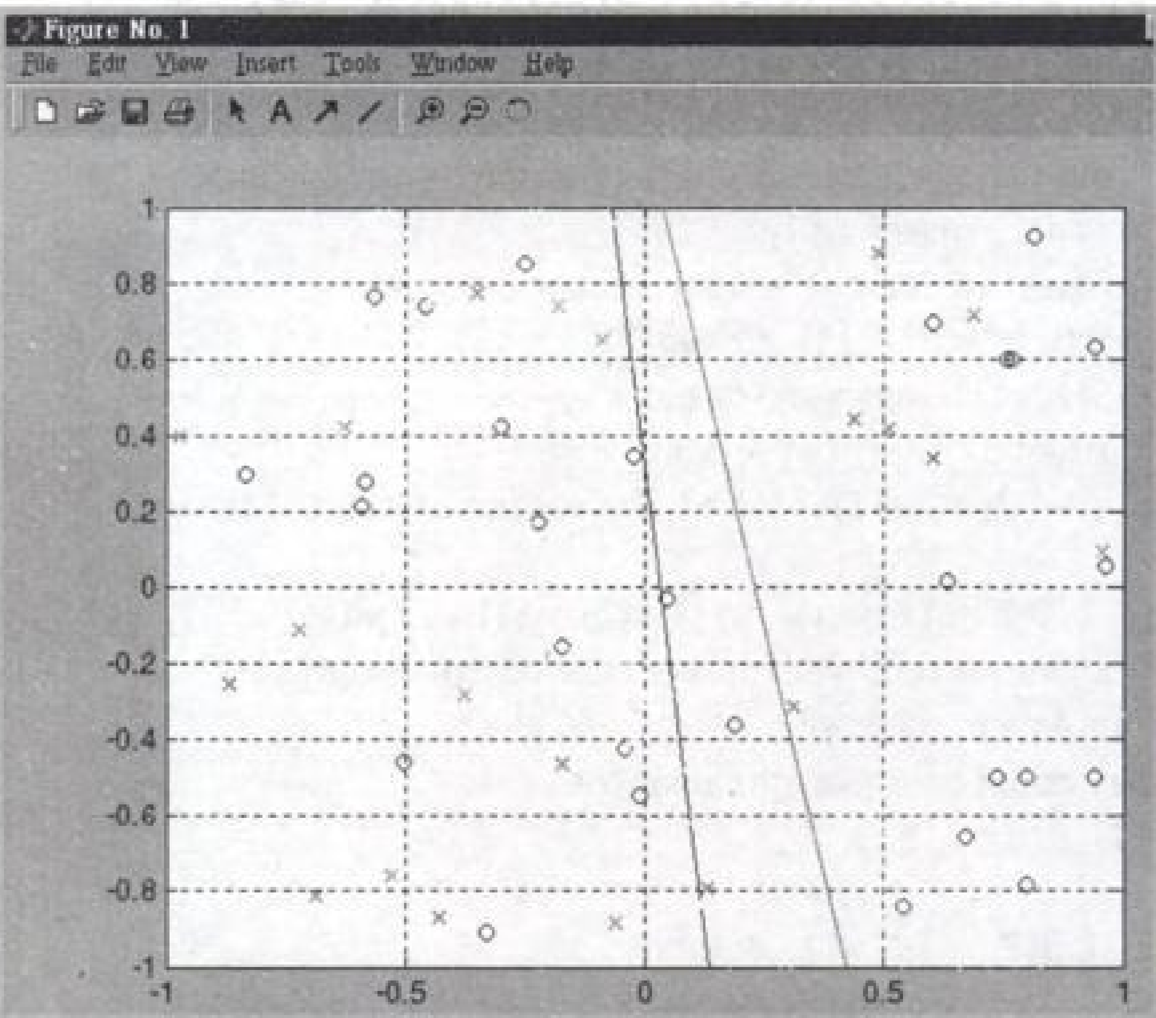
$$y_{-in2} = 1 + -1 \times 1 + 1 \times -1 + -1 \times -1 + -1 \times 1 + -1 \times 1 = -1 < 0$$

Applying activations, $y_2 = f(y_{-in2}) = -1.$

```
            0 1 0 0 1;
            1 1 1 1 1;
            1 0 1 1 1;
            0 1 1 1 1;
            1 0 1 1 1;
            1 1 1 0 0;
            0 1 0 1 0;
            1 0 0 1 1;
            1 1 1 1 1;
            1 1 1 1 0;
            1 1 1 1 1;]
```

Program

```
clear;
clc;
cd=open('reg.mat');
input=[cd.A';cd.B';cd.C';cd.D';cd.E';cd.F';cd.G';cd.H';cd.I';cd.J']';
for i=1:10
   for j=1:10
       if i==j
          output(i,j)=1;
       else
          output(i,j)=0;
       end
   end
end
for i=1:15
   for j=1:2
       if j==1
         aw(i,j)=0;
       else
         aw(i,j)=1;
       end
   end
end
test=[cd.K';cd.L';cd.M';cd.N';cd.O']';
net=newp(aw,10,'hardlim');
net.trainparam.epochs=1000;
net.trainparam.goal=0;
net=train(net,input,output);
y=sim(net,test);
x=y';
```



Output for Perceptron Learning Law Example

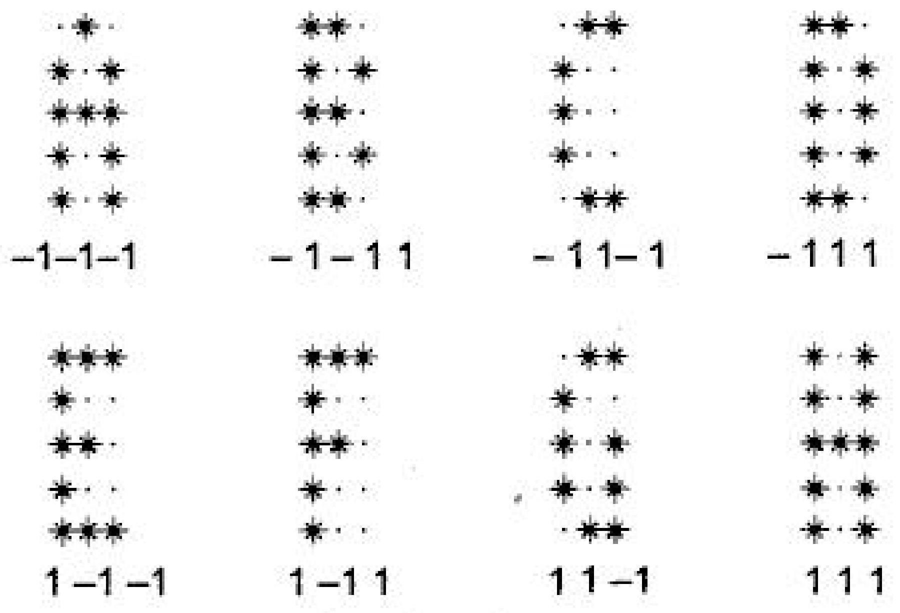
Example 4.8 **With a suitable example simulate the perceptron learning network and separate the boundaries. Plot the points assumed in the respective quadrants using different symbols for identification.**

Solution **Plot the elements as square in the first quadrant, as star in the second quadrant, as diamond in the third quadrant, as circle in the fourth quadrant. Based on the learning rule draw the decision boundaries.**

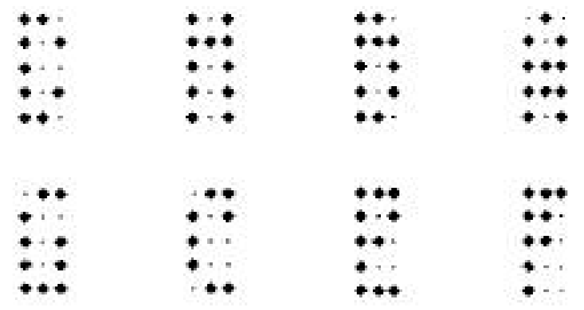
Program

```
Clear;
p1=[1 1]'; p2=[1 2]'; %- class 1. first quadrant when we plot the elements. square
p3=[2 -1]'; p4=[2 -2]'; %- class 2. 4th quadrant when we plot the elements. circle
p5=[-1 2]'; p6=[-2 1]'; %- class 3. 2nd quadrant when we plot the elements.star
p7=[-1 -1]'; p8=[-2 -2]';% - class 4. 3rd quadrant when we plot the elements.diamond
%Now. lets plot the vectors
hold on
plot(p1(1).p1(2).'ks'.p2(1).p2(2).'ks'.p3(1).p3(2).'ko'.p4(1).p4(2).'ko')
plot(p5(1).p5(2).'k*'.p6(1).p6(2).'k*'.p7(1).p7(2).'kd'.p8(1).p8(2).'kd')
grid
hold
axis([-3 3 -3 3])%set nice axis on the figure
t1=[0 0]'; t2=[0 0]'; %- class 1. first quadrant when we plot the elements. square
t3=[0 1]'; t4=[0 1]'; %- class 2. 4th quadrant when we plot the elements. circle
t5=[1 0]'; t6=[1 0]'; %- class 3. 2nd quadrant when we plot the elements.star
t7=[1 1]'; t8=[1 1]';% - class 4. 3rd quadrant when we plot the elements.diamond
%lets simulate perceptron learning
```

Example 4.9 Write a MATLAB program for pattern classification using perceptron network. Test train the net with a noisy pattern. Form the input vectors and noisy vectors from patterns as shown below and store it in a *.mat file.



Input vectors

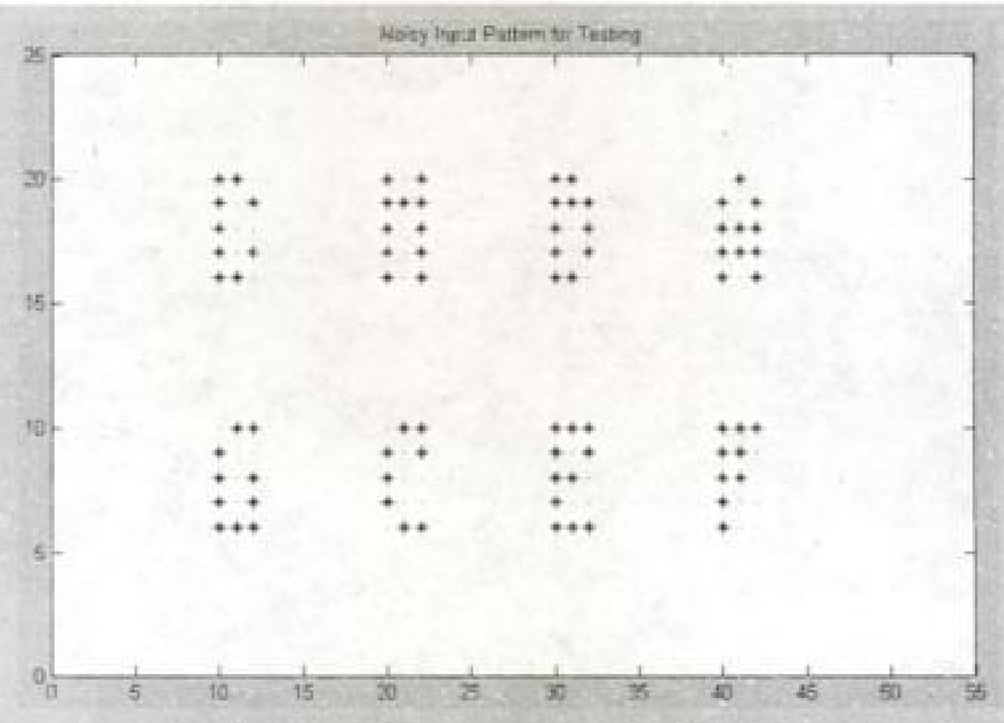


Noisy vectors

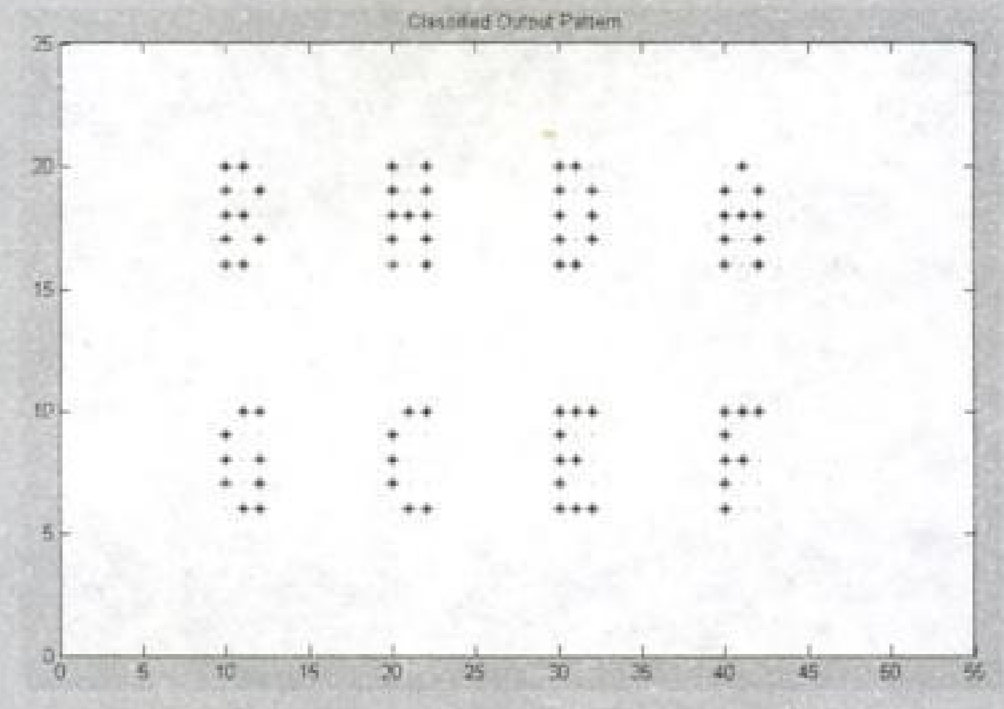
Solution The input vectors and the noisy vectors are stored in a mat file, say class.mat, and the required data is taken from the file. Here a subfunction called charplot.m is used. The MATLAB program for this is given below.

Program

```
%Perceptron for pattern classification
clear;
clc;
%Get the data from file
data=open('class.mat');
x=data.s;     %input pattern
t=data.t;     %Target
ts=data.ts;   %Testing pattern
n=15;
m=3;
%Initialize the Weight matrix
w=zeros(n,m);
b=zeros(m,1);
%Intitalize learning rate and threshold value
alpha=1;
```



Noisy Input Pattern used for Training



Classified Output Pattern

4.3 Brief Introduction to Multilayer Perceptron Networks

Multilayer perceptron networks is an important class of neural networks. The network consists of a set of sensory units that constitute the input layer and one or more hidden layer of computation modes. The input signal passes through the network in the forward direction. The network of this type is called multilayer perceptron (MLP).

The multilayer perceptrons are used with supervised learning and have led to the successful back-propagation algorithm. The disadvantage of the single layer perceptron is that it cannot be extended to multi-layered version. In MLP networks there exists a non-linear activation function. The widely used non-linear activation function is logistic sigmoid function. The MLP network also has various layers of hidden neurons. The hidden neurons make the MLP network active for highly complex tasks. The layers of the network are connected by synaptic weights. The MLP thus has a high computational efficiency.

rule and are applied to various neural network applications. The weights on the interconnections between the adaline and madaline networks are adjustable. The adaline and madaline networks are discussed in detail in this chapter.

5.2 Adaline

Adaline, developed by Widrow and Hoff [1960], is found to use bipolar activations for its input signals and target output. The weights and the bias of the adaline are adjustable. The learning rule used can be called as Delta rule, Least Mean Square rule or Widrow-Hoff rule. The derivation of this rule with single output unit, several output units and its extension has been dealt already in Section 3.3.3. Since the activation function is an identity function, the activation of the unit is its net input.

When adaline is to be used for pattern classification, then, after training, a threshold function is applied to the net input to obtain the activation.

The activation is,

y = ● ● ●

The adaline unit can solve the problem with linear separability if it occurs.

5.2.1 Architecture

The architecture of an adaline is shown in Fig. 5.1.

The adaline has only one output unit. This output unit receives input from several units and also from bias; whose activation is always +1. The adaline also resembles a single layer network as discussed in Section 2.7. It receives input from several neurons. It should be noted that it also receives input from the unit which is always '+1', called as bias. The bias weights are also trained in the same manner as the other weights. In Fig. 5.1, an input layer with $x_1 \cdots x_i \cdots x_n$ and bias, an output layer with only one output neuron is present. The link between the input and output neurons possess weighted interconnections. These weights get changed as the training progresses.

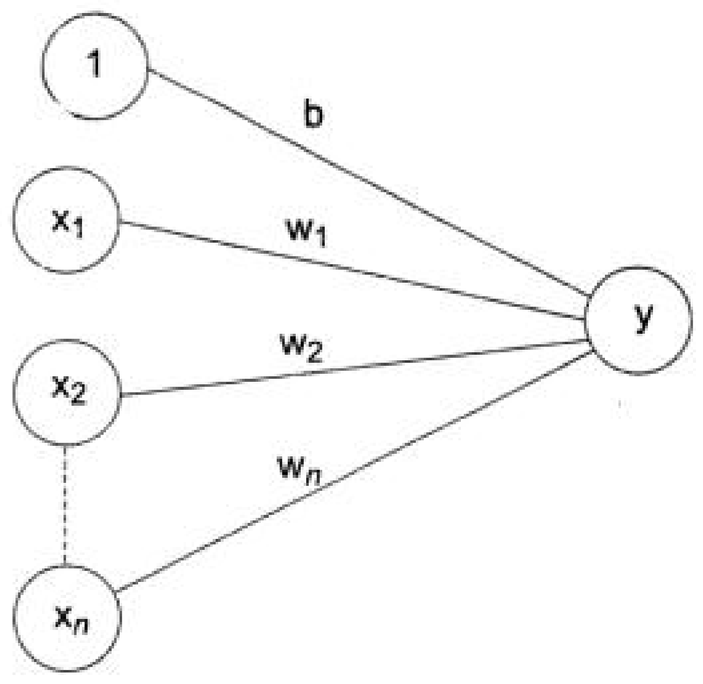


Fig. 5.1 Architecture of an Adaline

5.2.2 Algorithm

Basically, the initial weights of adaline network have to be set to small random values and not to zero as discussed in Hebb or perceptron networks, because this may influence the error factor to be considered. After the initial weights are assumed, the activations for the input unit are set. The net input is calculated based on the training input patterns and the weights. By applying delta learning rule discussed in 3.3.3, the weight updation is being carried out. The training process is continued until the error, which is the difference between the target and the net input becomes minimum. The step based training algorithm for an adaline is as follows:

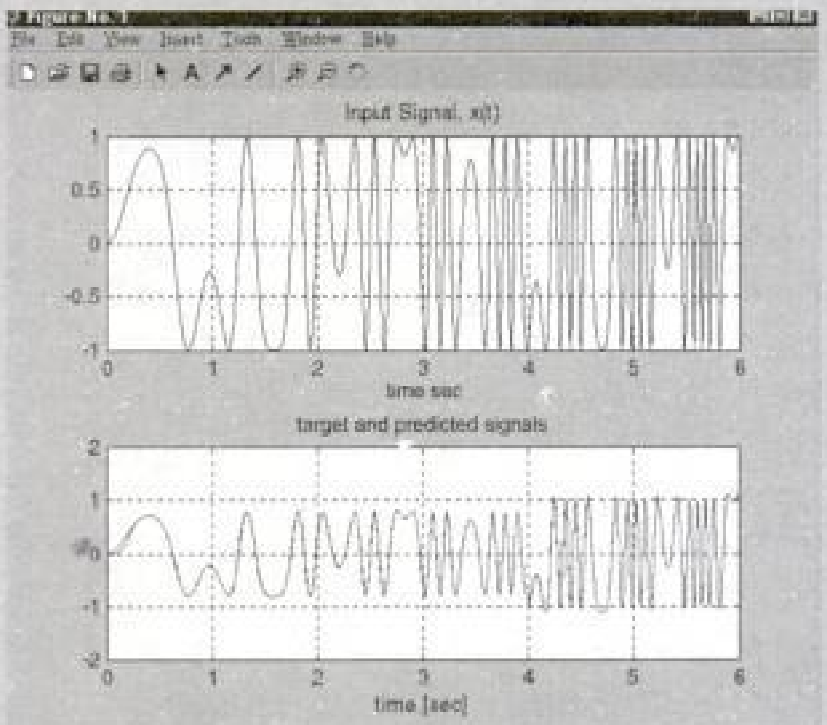
Program

```
clear all;
clc;
disp('Adaline network for OR function Bipolar inputs and targets');
%input pattern
x1=[1 1 -1 -1];
x2=[1 -1 1 -1];
%bias input
x3=[1 1 1 1];
%target vector
t=[1 1 1 -1];
%initial weights and bias
w1=0.1;w2=0.1;b=0.1;
%initialize learning rate
alpha=0.1;
%error convergence
e=2;
%change in weights and bias
delw1=0;delw2=0;delb=0;
epoch=0;
while(e>1.018)
   epoch=epoch+1;
   e=0;
   for i=1:4
      nety(i)=w1*x1(i)+w2*x2(i)+b;
      %net input calculated and target
      nt=[nety(i) t(i)];
      delw1=alpha*(t(i)-nety(i))*x1(i);
      delw2=alpha*(t(i)-nety(i))*x2(i);
      delb=alpha*(t(i)-nety(i))*x3(i);
      %weight changes
      wc=[delw1 delw2 delb]
      %updating of weights
      w1=w1+delw1;
      w2=w2+delw2;
      b=b+delb;
      %new weights
      w=[w1 w2 b]
      %input pattern
      x=[x1(i) x2(i) x3(i)];
      %printring the results obtained
      pnt=[x nt wc w]
   end
   for i=1:4
      nety(i)=w1*x1(i)+w2*x2(i)+b;
      e=e+(t(i)-nety(i))^2;
   end
end
```

```
eps = zeros(size(d)) ;    % memory allocation for eps
eta = 0.2  ; % learning rate/gain
w = 2*(rand(1, p)  -0.5) ; % Initialisation of the weight vector
for n = 1:N  %  learning loop
      y(n) = w*X(:,n) ;     % predicted output signal
      eps(n) = d(n) - y(n) ; % error signal
      w = w + eta*eps(n)*X(:,n)' ;
      if n == N1-1, w1 = w ;
     end
end
figure(1)
subplot(2,1,1)
plot(t, xt), grid, title('Input Signal, x(t)'), xlabel('time sec')
subplot(2,1,2)
plot(t, d, 'b', t, y, '-r'), grid, ...
title('target and predicted signals'), xlabel('time [sec]')
figure(2)
plot(t, eps), grid, title(['prediction error for eta = ', num2str(eta)]), ...
xlabel('time [sec]')
 [b1; w1]
[b2;  w]
```

```
Output
 [b1; w1]
 ans =
    1.0000  -0.6000   0.4000
    0.2673   0.9183  -0.3996
 [b2;  w]
 ans =
    0.9000  -0.5000   0.7000
    0.1357   1.0208  -0.0624
```

The corresponding responses are shown in the following figures.



Target and Predicted Signals along with the Input Signal

This architecture consists of two hidden adalines and one output adaline. The input x_1 and x_2 determine the output signal of hidden adalines z_1 and z_2. The output adaline is denoted by 'y'. z_1 and z_2 provide net computational capabilities that are not in the single layer neural net. Adaline was a special net which had only one output unit, but combination of adalines results in a madaline network as shown in Fig. 5.2. This architecture resembles the multilayer feed forward network discussed in Section 2.7. There can be any number of hidden layers between the input an the output layer but this increases the computation of the network. In this architecture, both the hidden units have separate bias connections, along with the input connections. The output from the hidden neurons are linked to the output neuron, where the final output of the network is calculated.

5.3.2 MRI Algorithm

This algorithm was formed by Widrow and Hoff in 1960. Here the weights of the hidden adaline units are only adjusted while the weights of the output unit are fixed. First, the initialization of the weights between the input and the hidden units is done (which are small positive random values). The input is presented, and based on the weighted interconnections the net input is calculated for both the hidden units. Then by applying the activations the output is obtained for both the hidden units. With this as input for the output layer, and constant weighted inter-connections acting between the hidden and output layer, the net input to the output neuron is found. By means of the net input, applying the activations the final output of the net is calculated. Then this compared with the target, and suitable weight updations are performed.

Parameters

The weights into y, v_1 and v_2 are fixed as 0.5 with bias b_3 as 0.5.

The activations function for units z_1, z_2 and y is given by,

$$f(p) = \begin{cases} 1, & \text{if } p \geq 0 \\ 0, & \text{if } p < 0 \end{cases}$$

The training algorithm is as follows: The algorithm is stated for the architecture in Fig. 5.2.

Step 1: Initialize weights and bias, set learning rate $\propto$.

$v_1 = v_2 = 0.5$ and $b_3 = 0.5$. Other weights may be small random values.

Step 2: When stopping condition is false do Steps 3–9.

Step 3: For each bipolar training pair s : t, do Steps 4–8.

Step 4: Set activations of input units:

$x_i = s_i$ for $i = 1$ to n

Step 5: Calculate net input of hidden adaline units.

$z_{-in1} = b_1 + x_1 w_{11} + x_2 w_{21}$

$z_{-in2} = b_2 + x_1 w_{12} + x_2 w_{22}$

Step 6: Find output of hidden adaline unit using activation mentioned above.

$z_1 = f(z_{-in1})$

$z_2 = f(z_{-in2})$

$$
\begin{aligned}
w_{12}\,(\text{new}) &= w_{12}(\text{old}) + \acute{\alpha}(-1-z_{in2}) \times 2 \\
&= 0.1 + 0.5\,(-1-0.45)\,1 \\
&= -0.625 \\
b_1\,(\text{new}) &= b_1(\text{old}) + \acute{\alpha}(-1-z_{in1}) \\
&= 0.3 + 0.5\,(-1-0.55) \\
&= -0.475 \\
w_{21}(\text{new}) &= w_{21}(\text{old}) + \acute{\alpha}(-1-z_{in1}) \times 2 \\
&= 0.2 + 0.5\,(-1-0.55)1 \\
&= -0.575 \\
w_{22}\,(\text{new}) &= w_{22}\,(\text{old}) + \acute{\alpha}(-1-z_{in2}) \times 2 \\
&= 0.2 + 0.5(-1-0.45)\,1 \\
&= -0.525 \\
b_2\,(\text{new}) &= b_2\,(\text{old}) + \acute{\alpha}(-1-z_{in2}) \\
&= 0.15 + 0.5\,(-1-0.45) \\
&= -0.575
\end{aligned}
$$

Step 9: Test stopping condition

This completes the 1st Epoch, 1st training pair presentation.

This process is repeated until the weight converges.

For this XOR function, the weight converges within 3 epochs, as tabulated below.

Inputs				t	z_{in1}	z_{in2}	w_{11}	w_{21}	b_1	w_{12}	w_{22}	b_2	z_1	z_2	y_{in}	y
	x_1	x_2	b													
Epoch 1	1	1	1	–1	0.55	0.45	–0.725	–0.575	–0.475	–0.625	–0.525	–0.575	1	1	1.5	1
	1	–1	1	1	–0.625	–0.675	0.0875	–1.3875	0.3375	–0.625	–0.525	–0.575	–1	–1	–0.5	–1
	–1	1	1	–1	–1.1375	–0.475	0.0875	–1.3875	0.3375	–1.3625	0.2125	0.1625	–1	–1	–0.5	–1
	–1	–1	1	–1	1.6375	1.3125	1.4065	–0.0685	–0.9815	–0.207	1.369	–0.994	1	1	1.5	1
Epoch 2	1	1	1	–1	0.3565	0.168	0.7285	–0.7465	–1.6595	–0.791	–0.207	–1.578	1	1	1.5	1
	1	-1	1	1	–0.1845	–3.154	1.3205	–1.339	–1.068	–0.791	0.785	–1.578	–1	–1	–0.5	–1
	–1	1	1	–1	–3.728	–0.002	1.3205	–1.339	–1.068	–1.292	0.785	–1.077	–1	–1	–0.5	–1
	–1	–1	1	–1	–1.0495	–1.071	1.3205	–1.339	–1.068	–1.292	1.286	–1.077	–1	–1	–0.5	–1
Epoch 3	1	1	1	–1	–1.0865	–1.083	1.3205	–1.339	–1.068	–1.292	1.286	–1.077	–1	–1	-0.5	–1
	1	–1	1	1	1.5915	–3.655	1.3205	–1.339	–1.068	–1.292	1.286	–1.077	1	–1	0.5	1
	–1	1	1	–1	–3.728	1.501	1.3205	–1.339	–1.068	–1.292	1.286	–1.077	–1	1	0.5	1
	–1	–1	1	–1	–1.0495	–1.071	1.3205	–1.339	–1.068	–1.292	1.286	–1.077	–1	–1	–0.5	–1

In Epoch 3 all t = y. Hence even if further iteration is done, the weights will remain the same. Thus madaline network is formed for XOR function.

5.22 Write a MATLAB program using adaline net for AND function with binary input and binary target.

5.23 Write a MATLAB program to generate XOR function using adaline with bipolar input and bipolar target.

5.24 Write a MATLAB program using madaline net for AND function with bipolar inputs and targets.

E is a function of all the weights. The partial derivative of E with respect to weight W_{ij} is a gradient of E and is given by

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \sum_{j=1}^{m} (t_j - y_j)^2 = \frac{\partial}{\partial W_{ij}} [t_j - y_j]^2$$

t_j does not depend on W_{ij}, only y_j depends on W_{ij}.

We know that,

$$y_{-inj} = \sum_{i=1}^{n} x_i w_{ij}$$

$$y_j = f(y_{-inj})$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} (t_j - f(y_{-inj}))^2$$

$$= 2(t_j - f(y_{-inj})) \frac{\partial f(y_{-inj})}{\partial W_{ij}}$$

Pattern is,

$$E = \sum_{j=1}^{m} (t_j - y_j)^2$$

E is a function of all the weight. The partial derivative of E with respect to weight W_{ij} is a gradient of E and is given by

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \sum_{j=1}^{m} (t_j - y_j)^2$$

$$= \frac{\partial}{\partial W_{ij}} (t_j - y_j)^2$$

As y_j only depends on weight W_{ij} the equation can be written as

$$= -2(t_j - y_j) \frac{\partial (y_j)}{\partial W_{ij}}$$

We know that

$$y_{-inj} = \sum_{i=1}^{n} x_i w_{ij}$$

$$Y_j = f(y_{-inj})$$

$$\frac{\partial (y_j)}{\partial W_{ij}} = \frac{\partial f(y_{-inj})}{\partial W_{ij}}$$

$$w_4 = S_4^T t_4 = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}[1 \quad 0] = \begin{bmatrix}0&0\\1&0\\0&0\\0&0\end{bmatrix}$$

Step 3: The weight matrix to store the entire four patterns is the sum of the weight matrix for each stored pattern.

$$W = w_1 + w_2 + w_3 + w_4$$

$$w = \begin{bmatrix}1&0\\1&0\\0&0\\0&0\end{bmatrix} + \begin{bmatrix}0&1\\0&1\\0&1\\0&0\end{bmatrix} + \begin{bmatrix}0&0\\0&0\\1&0\\1&0\end{bmatrix} + \begin{bmatrix}0&0\\1&0\\0&0\\0&0\end{bmatrix} = \begin{bmatrix}1&1\\2&1\\1&1\\1&0\end{bmatrix}$$

Thus the weight matrix has been determined using outer products method of Hebb rule.

Example 6.2 A hetero associative network is given. Find the weight matrix and test the network with the training input vectors.

$s_1 = (1\ 1\ 0\ 0)$	$t_1 = (1\ 0)$
$s_2 = (0\ 1\ 0\ 0)$	$t_2 = (1\ 0)$
$s_3 = (0\ 0\ 1\ 1)$	$t_3 = (0\ 1)$
$s_4 = (0\ 0\ 1\ 0)$	$t_4 = (0\ 1)$

Solution

Step 1: Initialize weight using Hebb outer product rule method

$$s_1 = (1\ 1\ 0\ 0) \qquad t_1 = (1\ 0)$$

$$w_1 = s_1^T t_1$$

$$= \begin{bmatrix}1\\1\\0\\0\end{bmatrix}[1 \quad 0] = \begin{bmatrix}1&0\\1&0\\0&0\\0&0\end{bmatrix}$$

$$w_2 = s_2^T t_2; \; s_2 = [0\ 1\ 0\ 0] \qquad t_2 = [1\ 0]$$

$$w_2 = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}[1 \quad 0] = \begin{bmatrix}0&0\\1&0\\0&0\\0&0\end{bmatrix}$$

Solution

Method I

The weight matrix w_{ij} is given by

$$w_{ij} = \sum (2S_i(p) - 1)(2t_j(p) - 1)$$

$w_{11} = (2S_1(1) - 1)(2t_1(1) - 1) + (2S_1(2) - 1)(2t_1(2) - 1) + (2S_1(3) - 1)(2t_1(3) - 1) + (2S_1(4) - 1)(2t_1(4) - 1)$

$w_{11} = (1)(1) + (-1)(1) + (-1)(-1) + (-1)(-1)$

$= 2$

$w_{12} = (2S_1(1) -1)(2t_2(1) - 1) + (2S_1(2) - 1)(2t_2(2) - 1) + (2S_1(3) - 1)(2t_2(3) - 1) + (2S_1(4) - 1)(2t_2(3) - 1)$

$= (1)(-1) + (-1)(-1) + (-1)(1) + (-1)(1)$

$= -1 + 1 -1 = -2$

$w_{21} = (2S_2(1) - 1)(2t_1(1) -1) + (2S_2(2) - 1)(2t_1(2)-1) + (2S_2(3) - 1)(2t_1(3) - 1) + (2S_2(4) - 1)(2t_1(4) - 1)$

$= (1)(1) + (1)(1) + (-1)(-1) + (-1)(-1)$

$= 4$

$w_{22} = (2S_2(1) - 1)(2t_2(1) -1) + (2S_2(2) - 1)(2t_2(2) - 1) + (2S_2(3) - 1)(2t_2(3) - 1) + (2S_2(4) - 1)(2t_2(4) - 1)$

$= (1)(-1) + (1)(-1) + (-1)(1) + (-1)(1)$

$= -4$

$w_{31} = (2S_3(1) - 1)(2t_1(1) -1) + (2S_3(2) - 1)(2t_1(2) - 1)+(2S_3(3) - 1)(2t_1(3) - 1) + (2S_3(4) - 1)(2t_2(4) - 1)$

$= (-1)(1) + (+1)(-1) + (1)(-1) + (1)(-1)$

$= -4$

$w_{32} = (2S_3(1) - 1)(2t_2(1) -1) + (2S_3(2) - 1)(2t_2(2) - 1) + (2S_3(3) - 1)(2t_2(3) - 1) + (2S_3(4) - 1)(2t_2(4) - 1)$

$= (-1)(-1) + (-1)(-1) + (1)(1) + (1)(1)$

$= 4$

$w_{41} = (2S_4(1) - 1)(2t_1(1) - 1) + (2S_4(2) - 1)(2t_1(2) - 1) + (2S_4(3) - 1)(2t_1(3)-1) + (2S_4(4) - 1)(2t_1(4) - 1)$

$= (-1)(1) + (-1)(1) + (1)(-1) + (-1)(-1)$

$= -2$

$w_{42} = (2S_4(1) - 1)(2t_2(1) -1) + (2S_4(2) - 1)(2t_2(2) - 1) + (2S_4(3) - 1)(2t_2(3) - 1) + (2S_4(4) - 1)(2t_2(4) - 1)$

$= (-1)(-1) + (-1)(-1) + (1)(1) + (-1)(1)$

$= 2$

Method II

Step 1: Convert binary data to bipolar data by changing 0 to –1

Step 2: Use outer product method to find the weight matrix.

$$w = S^T_{(P)}\ t(P)$$

Converting binary data to bipolar data

$S(1) = (1\ 1\ -1\ -1)$ $\qquad$ $t(1) = (1\ -1)$

$S(2) = (-1\ 1\ -1\ -1)$ $\qquad$ $t(2) = (1\ -1)$

Step 4: $[y_{-in1} \ y_{-in2}] = x(2)w$

$$= [1 \ 1 \ 1 \ 0]\begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix}$$

$$= [2 + 4 - 4 + 0 \quad -2 - 4 + 4 + 0]$$

$$= [2 \quad -2]$$

Step 5: $f(2) = 1 \qquad f(-2) = 0$

$$[f(2) \ f(-2)] = [1 \ 0]$$

This net associates a known output pattern with the input.

Example 6.6 Consider hetero associative net in Example 6.4 with input that is not similar to the training input, i.e. it differs by at least two or more components. Test the response of the network.

Solution

Step 1: Initialize weights

$$\begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix}$$

Step 2: For the input pattern, follow Steps 3 to 5

Step 3: $x_1 = [1 \ 1 \ 1 \ 1]$

Step 4: $[y_{-in1} \ y_{-in2}] = x(1)w$

$$= [1 \ 1 \ 1 \ 1]\begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix}$$

$$= [2 + 4 - 4 - 2 \quad 2 - 4 + 4 + 2]$$

$$= [0 \quad 0]$$

Step 5: $f(2) = 0 \qquad f(-2) = 0$

$$[f(2) \ f(-2)] = [0 \ 0]$$

The output is not one of the outputs with which the net was trained. In other words the net does not recognize the pattern.

$$y_{in-j} = [-1 \quad -1 \quad -1 \quad -1]\begin{bmatrix} 4 & -4 \\ -4 & 4 \\ -2 & 2 \\ 2 & -2 \end{bmatrix} = [0 \quad 0]$$

y = [0 0] which is an unknown response.

Observations

(1) Thus with binary test vectors, with both mistakes and missings in two components, the net is not able to recognize the stored vectors.

(2) By changing to bipolar patterns, the net was able to recognize with missing in two components but could not recognize mistakes in two components.

(3) Bipolar is better than binary.

Example 6.9 Write a M-file to calculate the weights for the following patterns using hetero associative neural net for mapping four input vectors to two output vectors.

S1	S2	S3	S4	t1	t2
1	1	0	0	1	0
1	0	1	0	1	0
1	1	1	0	0	1
0	1	1	0	0	1

Solution For the pattern association net, the weight matrix can be calculated either by the Hebbian learning rule or using outer products rule. The MATLAB program for calculating the weight matrix is as follows

Program

```
%Hetro associative neural net for mapping input vectors to output vectors
clc;
clear;
x=[1 1 0 0;1 0 1 0;1 1 1 0;0 1 1 0];
t=[1 0;1 0;0 1;0 1];
w=zeros(4,2);
for i=1:4
   w=w+x(i,1:4)'*t(i,1:2);
end
disp('weight matrix');
disp(w);
```

Output

```
weight matrix
     2  1
     1  2
     1  2
     0  0
```

Hence total weight of three vectors to be stored is $\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2 + \mathbf{W}_3$

$$= \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & -1 & 1 \\ -1 & 0 & 1 & -1 \\ -1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

First Vector

Step 1: To initialize weight matrix.

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Step 2: For testing the first stored input vector follow Steps 3 to 5.

Step 3: $X = [1\ -1\ 1\ -1]$

Step 4: $y_{-in} = xw$

$$= [1 \quad -1 \quad 1 \quad -1] \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

$$= [0 + 1 - 1 + 1 \quad -1 + 0 - 1 + 1 \quad -1 + 1 + 0 + 1 \quad -1 + 1 - 1 + 0]$$

$$= [1\ -1\ 1\ -1]$$

Step 5: $y = f\,[1\ -1\ 1\ -1] = (1\ -1\ 1\ -1)$

The response vector y is the same as stored vector. Hence the input vector is recognized as "known vector".

Second Vector

Step 2: For testing the second stored input vector do Step 3 to 5

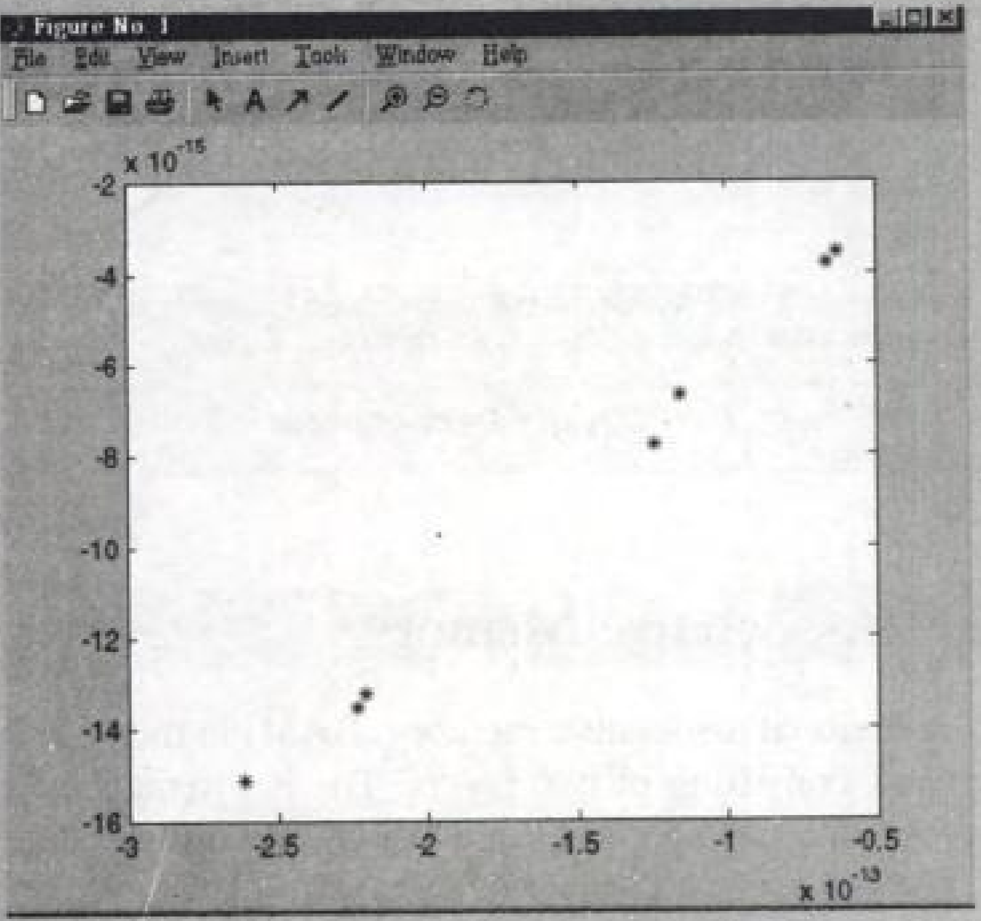
Step 3: $X = [-1\ 1\ 1\ -1]$

Step 4: $Y_{-in} = XW$

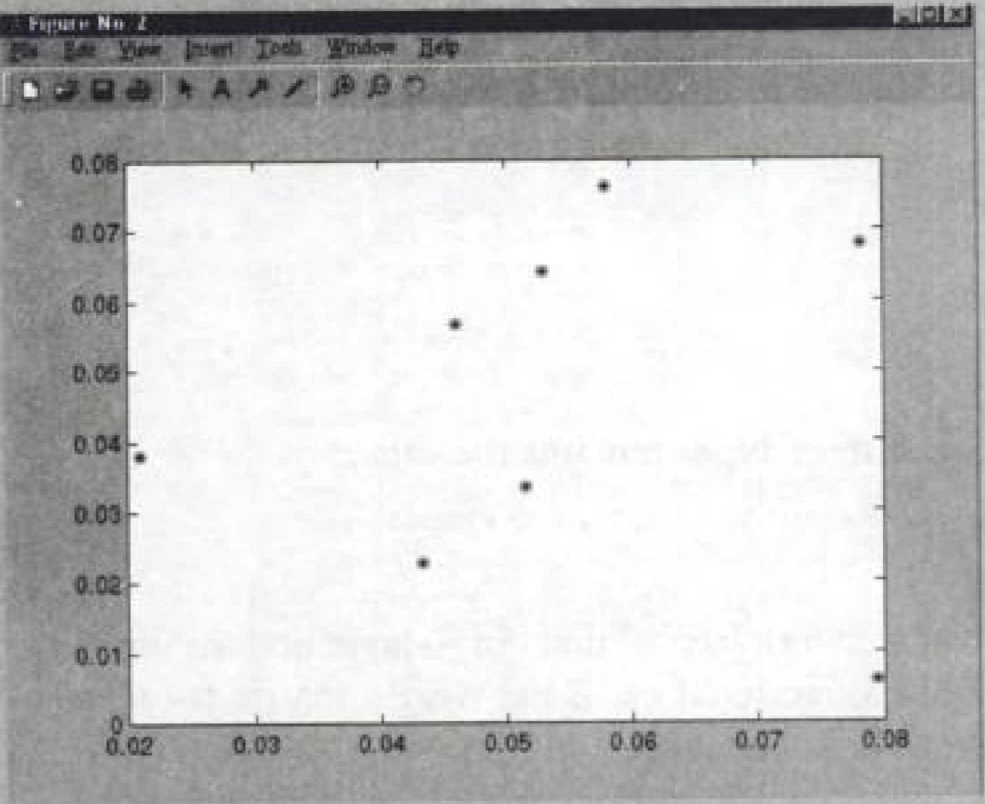
$$= [1 \quad -1 \quad 1 \quad -1] \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

```
criterion =
 1.2085e- 012
criteriontest =
  1.0131
output =
  0.0727  0.0838  0.0370  0.0547  0.0695  0.0795  0.0523  0.0173
  0.0309  0.0568  0.0703  0.0445  0.0621  0.0957  0.0880  0.0980
```

The response of the errors are shown graphically as,



Plot of the Errors



Plot of Errors in Output

$$f(y_{inj}) = \frac{1}{1 + \exp(-y_{inj})}$$

If bias is included in calculating the net input then

$$f(y_{inj}) = b_j + \sum x_i w_{ij}$$

X-layer

The logistic sigmoid activation function is given by

$$f(x_{inj}) = \frac{1}{1 + \exp(-x_{inj})}$$

If bias is included in calculating the net input then

$$x_{in\,i} = b_i + \sum y_j w_{ij}$$

The **memory (storage) capacity** of the BAM is min (n, m) where,

n is the number of units in x-layers

m is the number of units in y-layers.

according to Haines and Hecht-Neilsen (1988). It should be noted that this could be extended to min (2^n, 2^m) if appropriate non-zero threshold value is chosen for each unit.

6.5.3 Application Algorithm

From the training process, obtain the final weights. The input patterns are presented to both X-layer and Y-layer. The net input and the activations to the Y-layer are calculated. Then the signals are sent to X-layer and here its net input and activations are found. In this manner, the bi-directional associative memory is tested for its perfomance. The algorithm for the bi-directional memory net is given as follows:

Step 1: Initialize the weight to store a set of P vectors. Initialize all activations to 0.

Step 2: For each testing input follow Steps 3–8.

Step 3: Set activation of X-layer to current input pattern.

Step 4: Input pattern y is presented to the Y-layer.

Step 5: While activations are not converged follow Steps 6–8.

Step 6: Activation unit in Y-layer and net input are computed as,

Compute the net input $y_{-inj} = \sum_i w_{ij} x_i$. Compute activations. $y_i = f(y_{-inj})$.

Send signals to the X-layer.

Step 7: Update activation unit in X-layer. Net input is computed by $x_{-inj} = \sum_i w_{ij} y_j$. Then compute the activations $x_i = f(x_{-ini})$. Send signals to the Y-layer.

Step 8: Test for convergence.

The stopping condition may be that the activation vectors x and y have reached equilibrium. The activation function applied in steps 6 and 7 is based on the discussions in Section 6.5.2.

Now since the algorithm specifies that if net input to a unit is zero the activation of the unit remains unchanged we get

$[1\ \ 1\ \ 1\ \ 1\ \ -1\ \ -1\ \ 1\ \ 1\ \ 1\ \ 1\ \ -1\ \ 1\ \ 1\ \ 1\ \ 1]$ which is pattern E. Also,

$$x_i = \begin{cases} 1, & \text{if} \quad x_{ini} > 0 \\ x_i, & \text{if} \quad x_{ini} = 0 \\ -1, & \text{if} \quad x_{ini} < 0 \end{cases}$$

Example 6.22 (a) Use the Hebb rule to find the weight matrix to store the following (binary) input output pattern.

$$x(1) = (1\ \ 0\ \ 1) \quad r(1) = (1\ \ 0)$$
$$x(2) = (0\ \ 1\ \ 0) \quad r(2) = (0\ \ 1)$$

(b) Using the binary step function (with threshold 0) as the activation function for both layers, test the response of your network in both directions of each binary training pattern. Initial activation of the other layers is set to zero.

(c) Using the bipolar step function (with threshold 0) as the activation function for both layers. Convert the training patterns to bipolar form and test the network response in both directions again. Initial activation as in part (b).

(d) Test the response of your network on each of the following noisy versions of the bipolar form of the training patterns.

(A)	(0	−1	1)
(B)	(0	0	1)
(C)	(1	0	0)
(D)	(−1	0	−1)
(E)	(−1	0	0)
(F)	(0	0	−1)
(G)	(1	0	−1)
(H)	(1	0)	
(I)	(0	1)	

Solution

(a) **Step 1:** To find weight matrix

$x(1) = (1\ \ 0\ \ 1)$ $\qquad\qquad$ $y(1) = (1\ \ 0)$

$$w_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} [+1\ \ -1] = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix}$$

E. (−1 0 0)

$$Y_{in-j} = [-1 \quad 0 \quad 0]\begin{bmatrix} 2 & -2 \\ -2 & 2 \\ 2 & -2 \end{bmatrix}$$

$$= [-2 \quad 2]$$

$$Y_j = [-1 \quad 1]$$

which is the correct response.

F. (0 0 −1)

$$Y_{in-j} = [0 \quad 0 \quad -1]\begin{bmatrix} 2 & -2 \\ -2 & 2 \\ 2 & -2 \end{bmatrix}$$

$$= [-2 \quad 2]$$

$$Y_j = [-1 \quad 1]$$

which is the correct response.

G. (1 0)

$$X_{i-in-i} = [1 \quad 0]\begin{bmatrix} 2 & -2 & 2 \\ -2 & 2 & -2 \end{bmatrix}$$

$$X_i = f(X_{i-in-i}) = f[2 \quad -2 \quad 2]$$

$$= [1 \quad -1 \quad 1]$$

which is one of the stored vector. Hence it is the correct response.

H. (0 1)

$$X_{i-in-i} = [0 \quad 1]\begin{bmatrix} 2 & -2 & 2 \\ -2 & 2 & -2 \end{bmatrix}$$

$$X_i = [-2 \quad 2 \quad -2]$$

$$= [-1 \quad 1 \quad -1]$$

which is the correct response.

Example 6.23 Write a MATLAB program to find the weight matrix in bipolar form for the Bi-directional Associative Memory (BAM) network based on the following binary input output pairs.

S(1) = (1 1 0) t(1) = (1 0)

S(2) = (1 0 1) t(2) = (0 1)

Solution The MATLAB program for calculating the weight matrix using BAM network is as follows

S(1)	=	(1 0 0 1)		t(t)	=	(0, 1)
S(2)	=	(1 0 1 0)		t(2)	=	(0, 1)
S(3)	=	(1 1 0 0)		t(3)	=	(1, 0)
S(4)	=	(1 1 1 0)		t(4)	=	(1, 0)

(b) Test the response of the network on various combinations of input patterns with "mistakes" or "missing" data.

6.39 Write a MATLAB program to store A, B, C, D and after training, for testing the noisy version of A, B, C, D. Assume your own matrix representation for forming the alphabets A,B,C,D and the noisy versions. Perform this using hetero associative net.

6.40 Write a MATLAB program to store the following input matrix using hetero associative net.

P(1) = (1, –1, –1, –1)	t(1) = (1, –1)
P(2) = (1, –1, –1)	t(2) = (1, –1)
P(3) = (–1, –1, –1, 1)	t(3) = (–1, 1)
P(4) = (–1, –1, 1, 1)	t(4) = (–1, 1)

Test these inputs by writing another MATLAB program (use the initial weight from the first program).

6.41 Write a MATLAB program to find the weight matrix of an auto associative net to store the vector (1 –1 –1 –1). Test the response of the network by presenting the same pattern and recognize whether it is a known vector or an unknown vector.

6.42 Write an M-file to store the vectors (–1 1 –1 1) and (–1 –1 1 1) in an auto associative net. Find the weight matrix. Test the net with (–1 1 1 1) as input.

6.43 Write a MATLAB program to find the weight matrix in bipolar form for the Bi-directional Associative Memory (BAM) network based on the following binary input output pairs.

S(1) = (1 1 0 1)	t(1) = (1 0)
S(2) = (1 0 1 0)	t(2) = (0 1)

6.44 Write MATLAB program using Hebb rule to store the vectors (1, 1, 1, 1) and (1, 1, –1, –1) in an auto associative net and write to program to test (1, 1, 1, 1), (1, 1, –1, –1) (1, 1, 1, 0). (without setting diagonal elements to zero and with setting diagonal elements to zero).

6.45 Write a MATLAB program to find the weight matrix in bipolar form for the Bi-directional associative memory network for the given input and output vector pains.

S(1) = (0 0 1 1)	t(1) = (0 1)
S(2) = (1 0 1 0)	t(2) = (0 1)
S(3) = (1 1 1 0)	t(3) = (1 0)
S(4) = (1 1 0 0)	t(4) = (1 0)

Using the weights obtained, also test the response of the network with a unit step function (assume activation to be zero).

$P \approx 0.15\, n$ where n is the number of neurons in the net.

If bipolar patterns are used then, $P = \dfrac{n}{2 \log_2 n}$

Energy Function

The discrete net will converge to a stable limit point considering an energy function of the system. It is said that the energy function is a function that is bounded below and is a non-increasing function of the state of the system. The energy function for the discrete Hopfield network is given by

$$E = -0.5 \sum_{i \neq j} \sum_{j} y_i y_j w_{ij} - \sum x_i y_i + \sum_{i} \theta_i y_i$$

The change in energy is due to a change in the state of the neuron and is given by Δ E. It is found that the activation of the net changes by Δy_i. This can be calculated using the following equation.

$$\Delta E = -\left[\sum_{j} y_j w_{ij} + x_i - \theta_i\right] \Delta y_i$$

$$\Rightarrow (\text{net}_i - \theta_i)\Delta y_i$$

where Δy_i is the change in the output of neuron i.

Two cases in which a change Δy_i will occur in the activation of neuron y_i are

If y_i is positive, it will change to zero if

$$x_i + \sum_{j} y_j w_{ij} < \theta_i$$

This gives a negative change for y_i; in this cases $\Delta E < 0$.

If y_i is zero, it will change to positive if

$$x_i + \sum_{j} y_j w_{ji} > \theta_i$$

This gives a positive change for y_i; in this case, $\Delta E < 0$.

From the above two cases we can show that the energy cannot increase i.e. for both positive and negative change in y_i, the value of ΔE is less than zero (negative). As a result, the energy is bounded. So the net must reach a stable equilibrium such that the energy does not change with further iteration. The important aspect of the algorithm is that the energy change depends only on the change in activation of one unit and that the weight matrix can be symmetric with zeros present on the diagonal.

Solved Examples

Example 7.1 Consider a vector (1 0 1 1) to be stored in a net. Test a discrete Hopfield net with mistakes in the first and fourth components (0 0 1 0) of the stored vector.

Solution

Step 1: Initialize weight to store patterns, convert binary input to bipolar to find weight, $W = S^T S$

$$= -0.5[2 - 4 \quad -2 \; 2 \; -2]\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

$E_2 = -0.5 \times 12 = 6$
$E_3 = -10$
$x_1^1 = [1 \; 1 \; 1 \; -1 \; 1]$

Choose unit 4 for updation

$$y_{in4} = x_4 + \sum y_j w_{j4}$$

$$= -1 + [1 \quad 1 \quad 1 \quad -1 \quad 1]\begin{bmatrix} 3 \\ -1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

$$= -1 + 4 = 3 > 0$$

$y_4 = 1$
$x_1^1 = [1 \; 1 \; 1 \; 1 \; 1]$ it is converged
$E_1^1 = -0.5\, x_1^1\, w_T\, x_1^{1T}$

$$= -0.5[1 \; 1 \; 1 \; 1 \; 1]\begin{bmatrix} 0 & -1 & 1 & 3 & 1 \\ -1 & 0 & 1 & -1 & 1 \\ 1 & 1 & 0 & 1 & 3 \\ 3 & -1 & 1 & 0 & 1 \\ 1 & 1 & 3 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$E_1^1 = -10$
$x_2^1 = [1 \; -1 \; -1 \; -1 \; -1]$

Choose unit 4 for activation

$$y_{in4} = -1 + [1 \quad 1 \quad -1 \quad -1 \quad -1]\begin{bmatrix} 0 \\ -1 \\ 1 \\ 3 \\ 1 \end{bmatrix}$$

$$= -1 - 6$$
$$= -7 < 0$$

$y_1 = -1$
$x_3^1 = [-1 \; 1 \; -1 \; -1 \; -1]$

1	1	1	1	1	1	1	1	1	1
1	–1	1	1	1	1	1	1	1	1
1	–1	1	1	1	1	1	1	1	–1
1	–1	–1	1	–1	–1	–1	–1	1	1
1	–1	–1	1	1	1	1	1	1	–1
–1	1	1	1	–1	–1	–1	–1	1	1
1	1	1	1	1	1	1	1	–1	1
1	1	1	1	1	1	1	1	–1	1
1	1	1	1	–1	1	1	1	–1	1
1	1	1	–1	1	–1	1	–1	1	1
1	–1	1	1	1	1	1	1	1	1
1	–1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	–1	1	1	1
–1	–1	–1	1	–1	–1	1	–1	1	–1
–1	–1	–1	1	1	–1	1	1	1	–1
–1	1	1	1	–1	1	–1	–1	–1	1
–1	1	1	1	1	1	1	1	1	1
–1	1	1	1	1	1	1	1	1	1
–1	1	–1	1	1	1	1	1	1	1
–1	–1	–1	–1	–1	–1	1	–1	–1	1
–1	–1	1	1	1	1	1	1	1	1
–1	1	1	1	1	1	–1	1	1	1
–1	1	1	1	1	1	1	1	1	1
–1	–1	–1	1	–1	–1	1	–1	–1	–1
1	–1	–1	1	1	1	1	1	1	1
1	1	1	1	–1	1	–1	1	–1	–1
1	1	1	1	1	1	1	–1	1	1
1	1	1	1	1	1	1	–1	1	1
1	–1	–1	1	1	1	1	–1	1	1
1	–1	–1	–1	1	1	1	1	–1	1
1	1	1	1	–1	–1	–1	–1	1	1
1	1	1	1	–1	–1	1	–1	1	1
1	1	1	1	–1	–1	1	–1	–1	–1
–1	–1	–1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	–1	–1	–1	1	1	1	1	–1	1
1	–1	–1	–1	1	1	–1	1	1	–1
1	–1	–1	–1	1	1	–1	1	1	1
1	–1	1	–1	1	1	–1	1	1	1
1	1	1	–1	1	1	–1	1	–1	1

Step 8: Test for stopping condition.

The values of A, B, C, D, N and α may taken as any constant. A steep sigmoid function is formed for large values of $\propto$, which then approximates a step function. In the case of traveling salesman problem, the value of N is taken much higher than the number of cities, n.

7.4 Relation Between BAM and Hopfield Nets

From the study of Bi-directional associative memory (BAM) net and the Hopfield net (Sections 6.5 and 7.2), it is noted that they are closely related. The Hopfield net can be viewed as an auto associative BAM with the layers X and Y considered as single and the weight matrix with no self-connection. It means that the diagonal elements of the symmetric weight matrix are zero.

Also, it should be noted that the BAM can be viewed as a case of Hopfield net which has all the X and Y layer neurons, but no interconnection is found to exist between the two X-layer neurons or two Y-layer neurons. Hence all the X-layer neurons have to update their activations before any of the Y-layer neurons update theirs and vice versa. The updation within the X-layer and the Y-layer can be done at the same time because a change in activation of an X-layer neuron does not affect the net input to any other X-layer unit and similarly for the Y-layer units.

Summary

In this section we have seen the discrete Hopfield type of feedback network in detail. It is found that each unit is connected to every other unit. It should be noted that in this type of net only one unit updates its activation at a time. Since here the weight updation is asynchronous, it involves the energy function or Lyapunov function. If an energy function can be found for a iterative neural net, then the net will converge to a stable set of activations. Thus discrete Hopfield net is one of the most important feedback networks.

Review Questions

7.1 What is a discrete Hopfield net?

7.2 Define energy (Lyapunov) function.

7.3 Explain the discrete Hopfield net with its architecture.

7.4 State the application algorithm for a discrete Hopfield net.

7.5 Discuss in detail the energy function used in the discrete Hopfield net.

7.6 What is a spurious stable state?

Exercise Problems

7.7 (a) Form the weight matrix of the bipolar vectors shown below using the training algorithm for a discrete Hopfield net

that come under the feed forward type of nets. Some of them have already been explained in previous chapters. One of the most important types of feed forward network is the Back Propagation network, which is discussed in the next section. Radial Basis function is also included in this category. A radial basis function is closely related to the back propagation network except that there is a variation in the function used between both the networks.

8.2 Back Propagation Network (BPN)

Back propagation is a systematic method for training multi-layer artificial neural networks. It has a mathematical foundation that is strong if not highly practical. It is a multi-layer forward network using extend gradient-descent based delta-learning rule, commonly known as back propagation (of errors) rule. Back propagation provides a computationally efficient method for changing the weights in a feed forward network, with differentiable activation function units, to learn a training set of input-output examples. GE Hinton, Rumelhart and R.O. Williams first introduced BPN in 1986. Being a gradient descent method it minimizes the total squared error of the output computed by the net. The network is trained by supervised learning method. The aim of this network is to train the net to achieve a balance between the ability to respond correctly to the input patterns that are used for training and the ability to provide good responses to the input that are similar.

8.2.1 Generalized Delta Learning Rule (or) Back Propagation Rule

The total squared error of the output computed by net is minimized by a gradient descent method known as back propagation or generalized delta rule.

Derivation

Consider an arbitrary activation function f(x). The derivation of activation function is denoted by F(x). Let

$$y_{-\text{in}k} = \sum_i z_i w_{jk}$$

$$z_{-\text{in}J} = \sum_i v_{ij} x_i$$

$$Y_k = f(y_{-\text{in}k})$$

The error to be minimized is

$$E = 0.5 \sum_k [t_k - y_k]^2$$

By use of chain rule we have

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \left(0.5 \sum_k [t_k - y_k]^2 \right)$$

$$= \frac{\partial}{\partial w_{Jk}} \left(0.5 [t_k - t(y_{-\text{in}k})]^2 \right)$$

$$= -[t_k - y_k] \frac{\partial}{\partial w_{Jk}} f(y_{-\text{in}k})$$

$$= -[t_k - y_k] f(y_{-\text{in}k}) \frac{\partial}{\partial w_{Jk}} (y_{-\text{in}k})$$

Back Propagation of Errors

Step 7: Each output unit (y_k, $k = 1, \ldots, m$) receives a target pattern corresponding to an input pattern, error information term is calculated as

$$\delta_k = (t_k - y_k)f(y_{-\text{ink}})$$

Step 8: Each hidden unit (z_j, $j = 1, \ldots, n$) sums its delta inputs from units in the layer above

$$\delta_{-\text{in}j} = \sum_{k=1}^{m} \delta_j w_{jk}$$

The error information term is calculated as

$$\delta_j = \delta_{-\text{in}j}\, f(z_{-\text{in}j})$$

Updation of Weight and Biases

Step 9: Each output unit (y_k, $k = 1, \ldots, m$) updates its bias and weights ($j = 0, \ldots, p$)

The weight correction term is given by

$$\Delta W_{jk} = \alpha\delta_k z_j$$

and the bias correction term is given by

$$\Delta W_{ok} = \alpha\delta_k$$

Therefore, $W_{jk}\text{ (new)} = W_{jk}\text{(old)} + \Delta W_{jk}$, $W_{ok}\text{ (new)} = W_{ok}\text{(old)} + \Delta W_{ok}$

Each hidden unit (z_j, $j = 1, \ldots, p$) updates its bias and weights ($i = 0, \ldots n$)

The weight correction term

$$\Delta V_{ij} = \alpha\delta_j x_i$$

The bias correction term

$$\Delta V_{oj} = \alpha\delta_j$$

Therefore, $V_{ij}\text{(new)} = V_{ij}\text{(old)} + \Delta V_{ij}$, $V_{oj}\text{ (new)} = V_{oj}\text{(old)} + \Delta V_{oj}$

Step 10: Test the stopping condition.

The stopping condition may be the minimization of the errors, number of epochs etc.

8.2.4 Selection of Parameters

For the efficient operation of the back propagation network it is necessary for the appropriate selection of the parameters used for training. The initial value assignments are discussed in detail in this section.

Initial Weights

It will influence whether the net reaches a global (or only a local) minima of the error and if so how rapidly it converges. If the initial weight is too large the initial input signals to each hidden or output unit will fall in the saturation region where the derivative of the sigmoid has a very small value (f(net) = 0). If initial weights are too small, the net input to a hidden or output unit will approach zero, which then causes extremely slow learning. To get the best result the initial weights (and biases) are set to random numbers between –0.5 and 0.5 or between –1 and 1. The initialization of weights (bias) can be done randomly and there is also a specific approach, which is discussed below.

Suppose we start with a weight set for the network corresponding to a point P. If we perform gradient descent, the minimum we encounter is the one at M_l not that at M_g. M_l is called local minima and corresponds to a partial solution for network in response to the training data. M_g is the global minima and unless measures are taken to escape from M_l, M_g will never be reached.

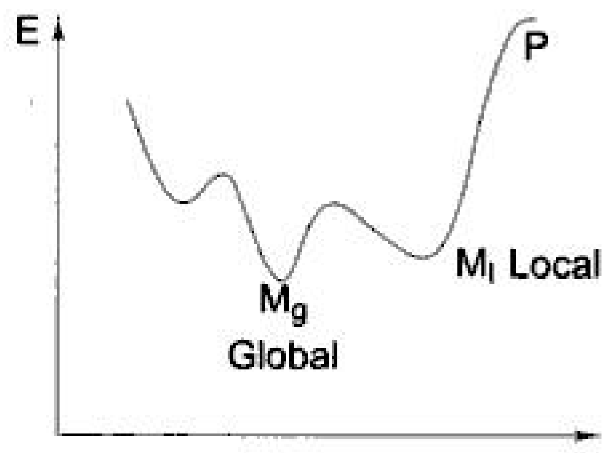


Fig. 8.2 | Error Function

8.2.8 Merits and Demerits of Back Propagation Network

The merits and the demerits of the back propagation network are as listed below:

Merits

1. The mathematical formula present here, can be applied to any network and does not require any special mention of the features of the function to be learnt.
2. The computing time is reduced if the weights chosen are small at the beginning.
3. The batch update of weights exist, which provides a smoothing effect on the weight correction terms.

Demerits

1. The number of learning steps may be high, and also the learning phase has intensive calculations.
2. The selection of the number of hidden nodes in the network is a problem. If the number of hidden neurons is small, then the function to be learnt may not be possibly represented, as the capacity of network is small. If the number of hidden neurons is increased, the number of independent variable of the error function also increases and the computing time also increases rapidly.
3. For complex problems it may require days or weeks to train the network or it may not train at all. Long training time results in non-optimum step size.
4. The network may get trapped in a local minima even though there is a much deeper minimum nearby.
5. The training may sometimes cause temporal instability to the system.

8.2.9 Applications

The back propagation algorithm covers wide area of applications, viz.

1. Optical character recognition
2. Image compression
3. Data compression
4. Load forecasting problems in power system area
5. Control problems
6. Non linear simulation
7. Fault detection problems etc.

This can also be extended to face recognition, avionic problems, etc.

$$\Delta v_{21} = \alpha\Delta_1 x_2 = 0.3 \times -0.0083 \times 0.8 = -0.002$$
$$\Delta v_{22} = \alpha\Delta_2 x_2 = 0.3 \times 0.0071 \times 0.8 = 0.0017$$
$$\Delta v_{13} = \alpha\Delta_3 x_1 = 0.3 \times 0.0361 \times 0.6 = 0.0065$$
$$\Delta v_{23} = \alpha\Delta_3 x_2 = 0.0087$$
$$\Delta v_{31} = \Delta_{32} = \Delta v_{33} = 0$$
$$\Delta v_{01} = \alpha\Delta_1; \;\; \Delta v_{02} = \alpha\Delta_2; \qquad \Delta v_{03} = \alpha\Delta_3$$
$$\Delta v_0 = 0.3 \times -0.0023 \Rightarrow -0.0025$$
$$\Delta v_{02} = 0.3 \times 0.0071 \Rightarrow 0.0021$$
$$\Delta v_{03} = 0.3 \times 0.0361 \Rightarrow 0.0108$$
$$v_{new} = v_{old} + \Delta v_1$$
$$v_{11}(new) = v_{11}(old) + \Delta v_{11} = 2 - 0.0015 = 1.9985$$
$$v_{12}(new) = v_{12}(old) + \Delta v_{13} = 9 + 0.0013 = 1.0013$$
$$v_{13}(new) = v_{13}(old) + \Delta v_{13} = 01 + 0.065 = 0.065$$
$$v_{21}(new) = v_{21}(old) + \Delta v_{21} = 1 - 0.002 = 0.998$$
$$v_{22}(new) = v_{22}(old) + \Delta v_{22} = 2 + 0.0017 = 2.0017$$
$$v_{23}(new) = v_{23}(old) + \Delta_{23} = 2 + 0.0067 = 2.0087$$
$$v_{31}(new) = v_{31}(old) + \Delta_{31} = 0 + 0 = 0$$
$$v_{32}(new) = v_{32}(old) + \Delta_{32} = 3 + 0 = 3$$
$$v_{33}(new) = v_{32}(old) + \Delta_{33} = 1 + 0 = 1$$

$$\therefore\; v = \begin{bmatrix} 1.9985 & 1.0013 & 0.085 \\ 0.998 & 2.0017 & 2.0087 \\ 0 & 3 & 1 \end{bmatrix}$$

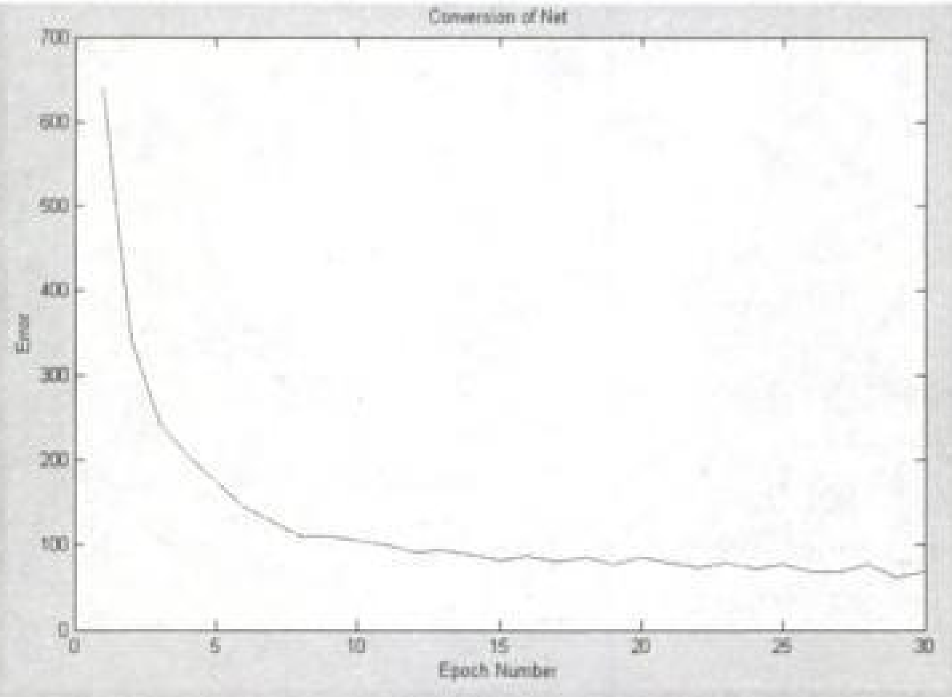
$$\Delta W_{ok} = \alpha\delta_k z_j$$
$$\Delta W_{11} = \alpha\delta_1 z_1 = 0.3 \times 0.0788 \times 0.8808 = 0.0208$$
$$\Delta W_{12} = \propto\delta_1 z_2 = 0.3 \times 0.0788 \times 0.9002 = 0.0212$$
$$\Delta W_{13} = \propto\delta_1 z_3 = 0.3 \times 0.0788 \times 0.646 = 0.0153$$
$$\Delta W_{11}(new) = w_{11}(old) + \Delta w_{11} = -1 + 0.0208 = 0.9792$$
$$\Delta W_{12}(new) = w_{12}(old) + \Delta w_{12} = 1 + 0.0212 = 1.0212$$
$$\Delta W_{13}(new) = w_{13}(old) + \Delta w_{13} = 2 + 0.0153 = 2.0153$$
$$\Delta W_0 = \alpha s_1 = 0.3 \times 0.0788 = 0.02364$$
$$w = [0.9792 \;\; 1.0212 \;\; 2.0153]$$

Thus the weights are calculated. The process can be continued upto any specified stopping condition.

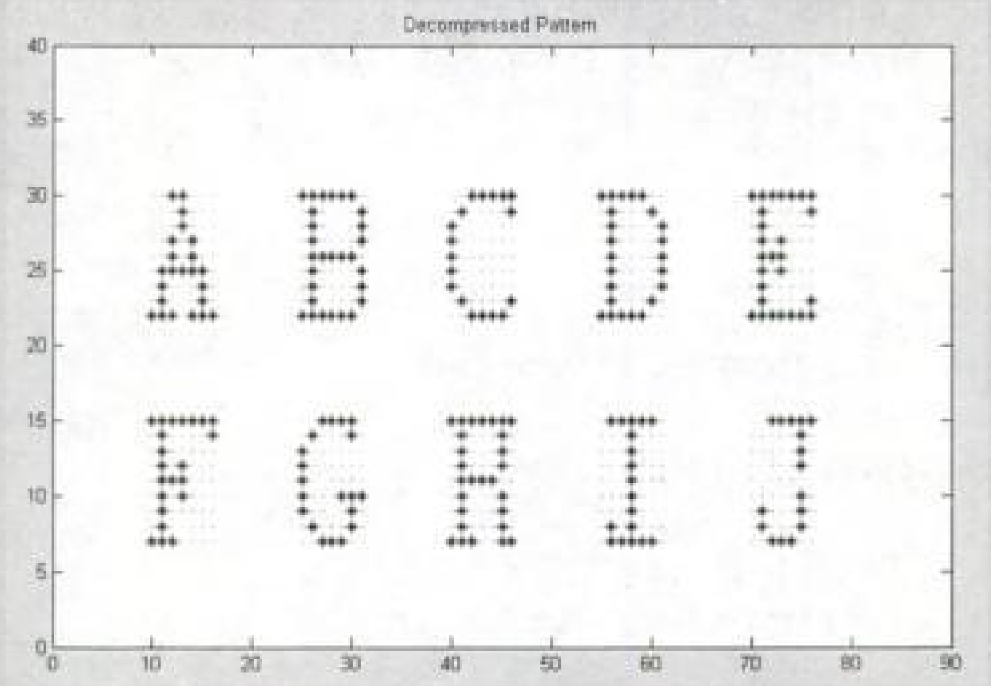
Example 8.2 Write a M-file for XOR function (binary input and output) with momentum factor using back propagation algorithm.

```
            %Feed forward
          for j=1:4
              zin(j)=b1(j);
               for i=1:2
                  zin(j)=zin(j)+x(i,I)*v(i,j);
            end
             z(j)=bipsig(zin(j));
        end
         yin=b2+z*w;
         y(I)=bipsig(yin);
         %Backpropagation of Error
         delk=(t(I)-y(I))*bipsig1(yin);
         delw=alpha*delk*z'+mf*(w-w1);
         delb2=alpha*delk;
         delinj=delk*w;
          for j=1:4
             delj(j,1)=delinj(j,1)*bipsig1(zin(j));
        end
          for j=1:4
               for i=1:2
                 delv(i,j)=alpha*delj(j,1)*x(i,I)+mf*(v(i,j)-v1(i,j));
                      end
        end
         delb1=alpha*delj;
          w1=w;
          v1=v;
         %Weight updation
         w=w+delw;
         b2=b2+delb2;
         v=v+delv;
         b1=b1+delb1';
         e=e+(t(I)-y(I))^2;
     end
     if e<0.005
         con=0;
     end
    epoch=epoch+1;
end
disp('BPN for XOR funtion with Bipolar Input and Output');
disp('Total Epoch Performed');
disp(epoch);
disp('Error');
disp(e);
disp('Final Weight matrix and bias');
v
b1
```

```
figure(1);
k=1;
for i=1:2
  for j=1:5
      charplot(x(k,:),10+(j-1)*15,30-(i-1)*15,9,7);
       k=k+1;
   end
end
title('Input Pattern for Compression');
axis([0 90 0 40]);
figure(2);
plot(x1,y1);
xlabel('Epoch Number');
ylabel('Error');
title('Conversion of Net');
%Output of Net after training
for I=1:10
   for j=1:h
      zin(j)=b1(j);
       for i=1:n
         zin(j)=zin(j)+x(I,i)*v(i,j);
       end
      z(j)=bipsig(zin(j));
   end
   for k=1:m
      yin(k)=b2(k);
       for j=1:h
         yin(k)=yin(k)+z(j)*w(j,k);
       end
      y(k)=bipsig(yin(k));
      ty(I,k)=y(k);
   end
end
for i=1:10
   for j=1:63
      if ty(i,j)>=0.8
          tx(i,j)=1;
      else if ty(i,j)<=-0.8
             tx(i,j)=-1;
           else
             tx(i,j)=0;
           end
       end
   end
```



Conversion of the Net



Decompressed Pattern after Training

Example 8.5 Write a MATLAB program for approximating a two 2-dimensional functions using back propagation in batch mode.

Solution Surf command used here gives the 3-dimensional view output. The MATLAB program for approximating two 2-dimensional functions is given as follows.

Program

```
clear;
clc;
p = 3 ; % Number of inputs (2) plus the bias input
L = 12; % Number of hidden signals (with bias)
m = 2 ; % Number of outputs
na = 16 ;  N = na^2; nn = 0:na-1;  % Number of training cases
% Generation of the training cases as coordinates of points from two 2-D surfaces
```

3. $\phi(r) = r^2$
4. $\phi(r) = r^3$
5. $\phi(r) = \exp(-r^2)$

It has been proved that the global basis functions may have slightly better interpolation properties than the local ones.

8.3.1 Architecture

The architecture of radial basis function network consists of three layers, the input, hidden and the output layers as shown in Fig. 8.4.

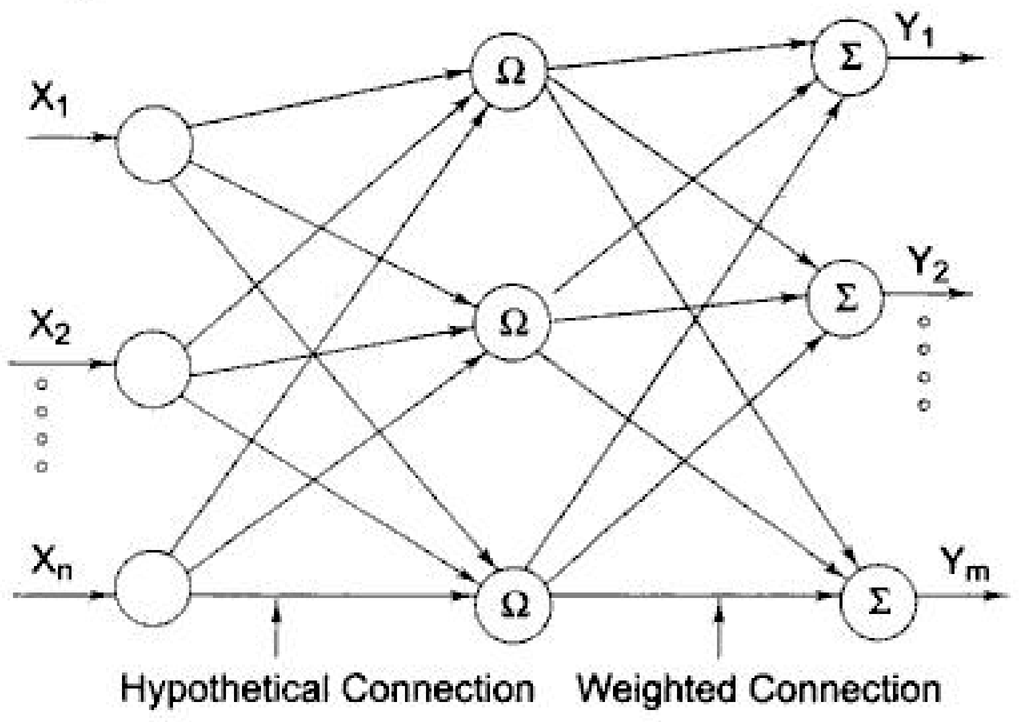


Fig. 8.4 *Architecture of Radial Basis Function Network*

The architecture of radial basis function network is a multilayer feed forward network as discussed in Section 2.7 of Chapter 2. There exists 'n' number of input neurons and 'm' number of output neurons with the hidden layer existing between the input and output layer. The interconnection between the input layer and hidden layer forms hypothetical connection and between the hidden and output layer forms weighted connections. The training algorithm is used for updation of weights in all the interconnections.

8.3.2 Training Algorithm for an RBFN with Fixed Centers

The training algorithm for the radial basis function network is given below. The important aspect of the radial basis function network is the usage of activation function for computing the output.

Activation Function

Radial basis function uses Gaussian activation function. The response of such function is non-negative for all value of x. The function is defined as

$$f(x) = \exp(-x^2)$$

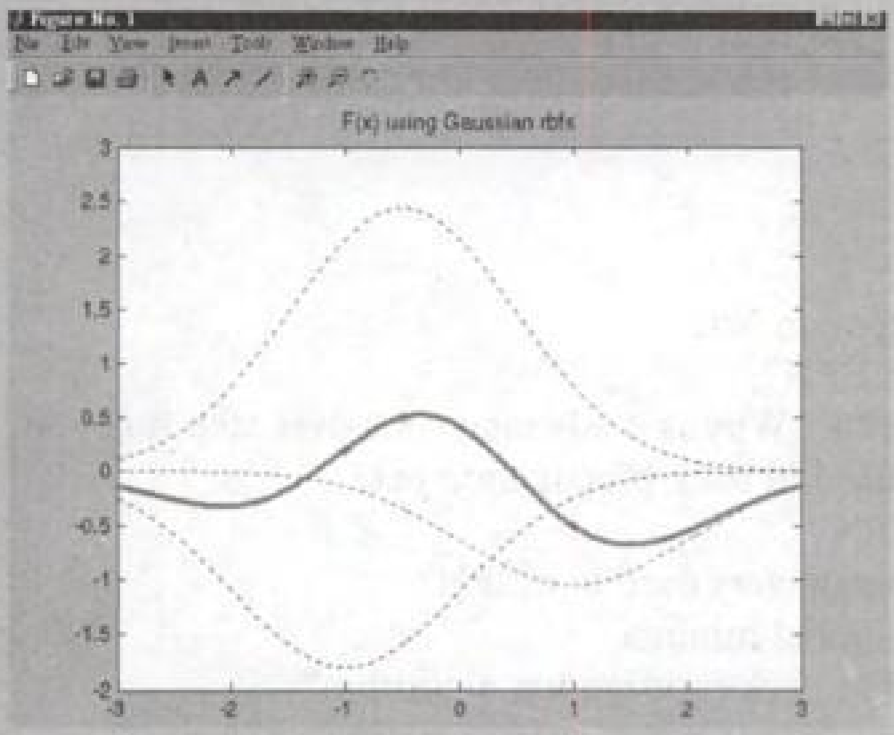
Its derivative is given by

$$f^1(x) = -2x\exp(-x^2) = -2xf(x)$$

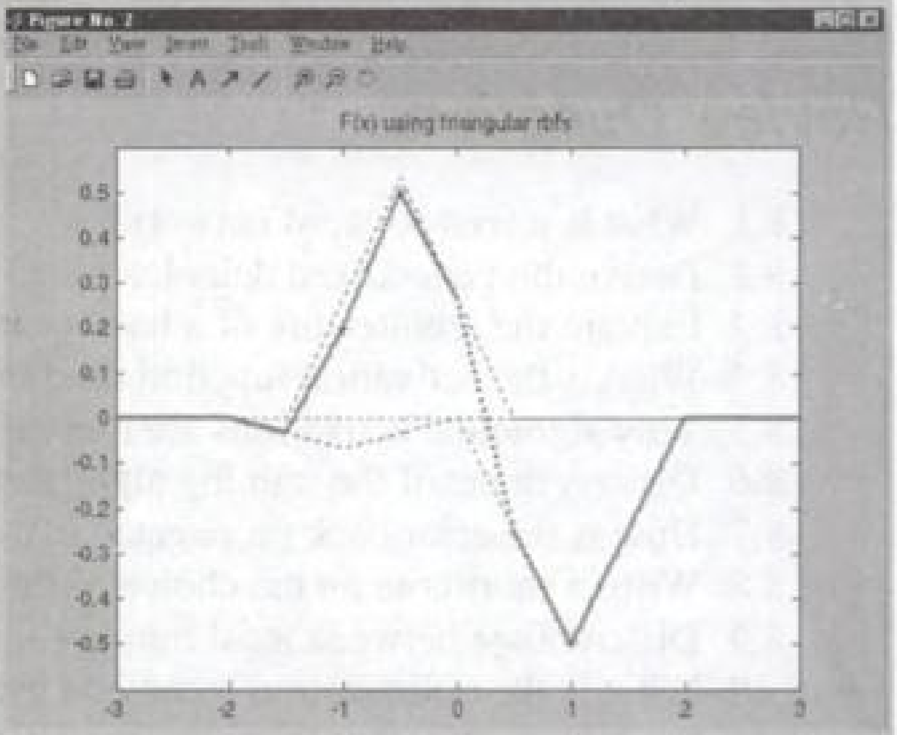
The radial basis function is different from the back propagation network in the Gaussian function it uses. The training algorithm for the network is given as follows:

```
   -0.5000
```

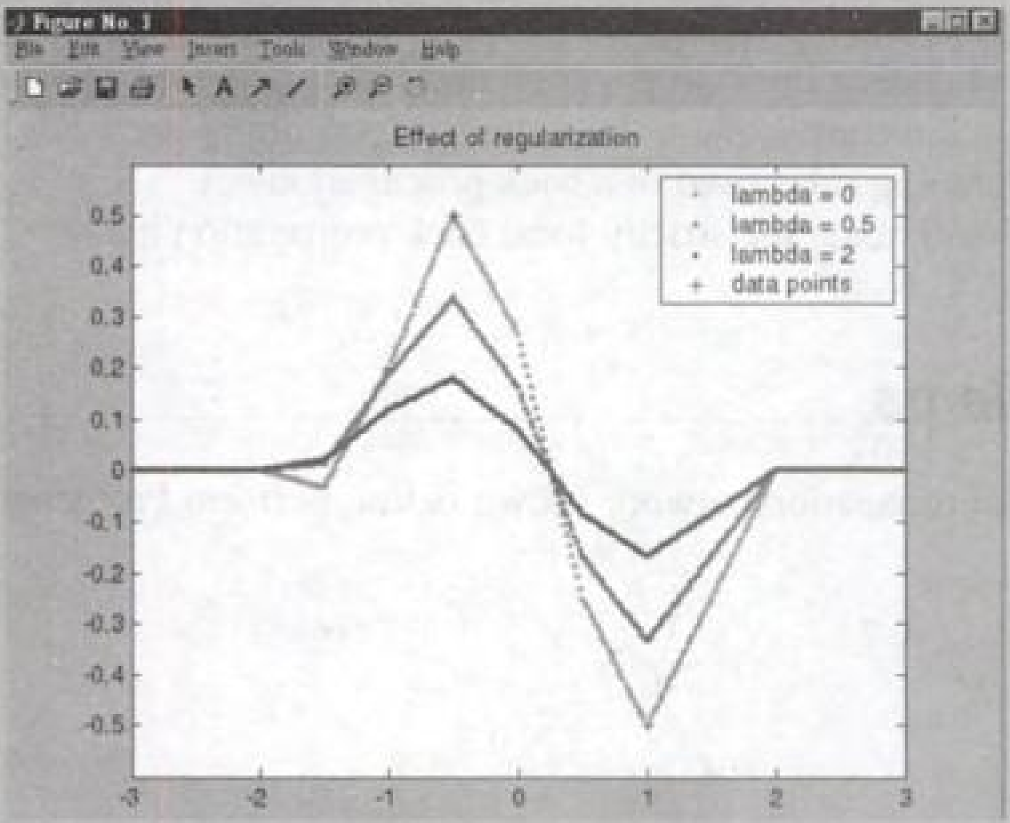
The output response curves are given as,



F(x) using Gaussian RBFs



F(x) using Triangular RBFs



Effect of Regularization

Summary

In this chapter we have discussed the feed forward networks. Application of these nets can be found in every field that uses neural nets for solving problems that involve mapping a given set of inputs to a specified set of target outputs. Despite its limitations, back propagation has dramatically expanded the range of problems to which artificial neural networks can be applied, and it has generated many successful

But only one of the several neurons has to respond. Hence additional structure can be included in the network so that the net is forced to make a decision as to which one unit will respond. The mechanism by which only one unit is chosen to respond is called Competition.

The mostly used competition among group of neurons is Winner-Takes-all. Here, only one neuron in the competing group will have a non-zero output signal when the competition is completed. The form of the learning depends on the purpose for which the net is being trained.

In a clustering net, there are as many input units as an input vector components. Since each output unit represents a cluster, the number of output units will limit the number of clusters that can be formed. The weight vector for an output unit in a clustering net is called exemplar or code-book vector for the input patterns, which the net has placed on that cluster. During training, the net determines the output unit that is the best match for the current input vector; the weight vector for the winner is then adjusted with respect to the net's learning algorithm.

The nets discussed in this chapter use Kohonen learning approach. In this learning, the units update their weights by forming a new weight vector that is a linear combination of the old weight vector and the current input vector. The unit whose weight vector is closest to the input vector is allowed to learn. All the competitive nets viz. Max net, Mexican hat, Learning vector quantization, Hamming net, Kohonen self organizing feature maps are discussed in this chapter.

9.2 Methods Used for Determining the Winner

There are two methods used for the determination of the winner unit.

Method 1

This method uses the squared Euclidean distance between the input vector and the weight vector, and chooses the unit whose weight vector has the smallest Euclidean distance from the input vector.

Method 2

This method uses the dot product of the input vector and the weight vector. The dot product of an input vector with a given weight vector is the net input to the corresponding cluster unit. The largest dot product corresponds to the smallest angle between the input and weight vectors if they are both of unit length.

9.3 Kohonen Self Organizing Feature Maps (SOM)

Kohonen worked in the development of the theory of competition, as a result the competitive processing elements are referred to as Kohonen unit. These self organizing maps can also be termed as topology-preserving maps.

In a topology-preserving map, units located physically next to each other will respond to classes of input vectors that are likewise located next to each other. Although, it is easy to visualize units located to each other in a two-dimensional array, it is not easy to determine which classes of vectors are next to each other in a high-dimensional space. Large dimensional input vectors are, in a sense, projected down

Step 4: Compute the squared Euclidean distance,

$$D(j) = \sum \left(w_{ij} - x_i\right)^2$$

$$D(1) = (0.2 - 0.3)^2 + (0.3 - 0.4)^2 = 0.02$$

$$D(2) = (0.6 - 0.3)^2 + (0.5 - 0.4)^2 = 0.10$$

$$D(3) = (0.4 - 0.3)^2 + (0.7 - 0.4)^2 = 0.10$$

$$D(4) = (0.9 - 0.3)^2 + (0.6 - 0.4)^2 = 0.40$$

$$D(5) = (0.2 - 0.3)^2 + (0.8 - 0.4)^2 = 0.17$$

Step 5: The input vector is closest to the output node 1, since D(1) is minimum, so J = 1.

(b)

Step 6: The weights on the wining unit are updated.

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha(x - w_{ij}(\text{old}))$$

$$w_{11}(\text{new}) = w_{11}(\text{old}) + 0.3(0.3 - 0.2)$$

$$= 0.2 + 0.3(0.3 - 0.2)$$

$$w_{11}(\text{new}) = 0.23$$

$$w_{21}(\text{new}) = w_{21}(\text{old}) + \alpha(x_2 - w_{21}(\text{old}))$$

$$= 0.3 + 0.3(0.4 - 0.3)$$

$$w_{21}(\text{new}) = 0.33$$

The new weights for untis are, $C_J = C_1$ is,

$$\begin{bmatrix} 0.23 & 0.6 & 0.4 & 0.9 & 0.2 \\ 0.33 & 0.5 & 0.7 & 0.6 & 0.8 \end{bmatrix}$$

(c) The new weights of C_{J-1} and C_{J+1},

Since J = 1, $C_{J-1} = C_{1-1} = C_0$, the weights of which cannot be calculated, since it does not exist.

$C_{J+1} = C_{1+1} = C_2$

Weight updation of unit 2,

$$w_{12(\text{new})} = w_{12(\text{old})} + \alpha[x_1 - w_{12},(old)]$$

$$= 0.6 + 0.3(0.3 - 0.6)$$

$$w_{12(\text{new})} = 0.51$$

$$w_{22(\text{new})} = w_{22(\text{old})} + \alpha[x_2 - w_{22},(\text{old})]$$

$$= 0.5 + 0.3(0.4 - 0.5)$$

$$w_{22(\text{new})} = 0.47$$

The new weights for units are,

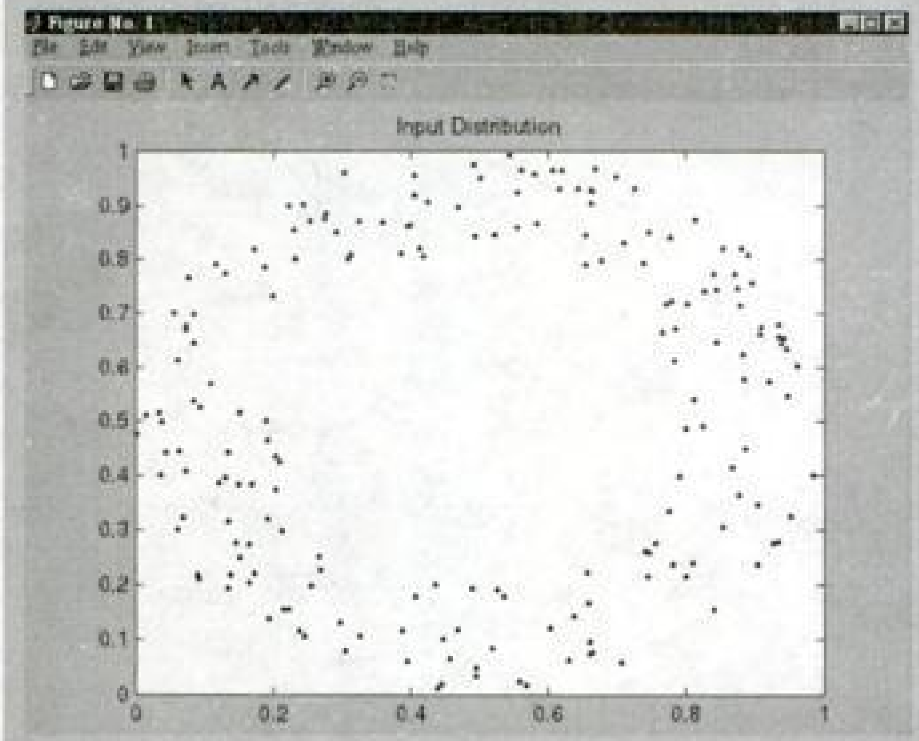
$$\begin{bmatrix} 0.23 & 0.51 & 0.4 & 0.9 & 0.2 \\ 0.33 & 0.47 & 0.7 & 0.6 & 0.8 \end{bmatrix}$$

```
% Next, the dimensionality of the output space, l, is selected.
% The output units ("neurons") can be arranged in a linear, i.e. 1-Dimensional way, or in a
rectangle, i.e., in a 2-D space.
el = menu('Select the dimensionality of the output domain:',...
           '1-dimensional output domain', ...
           '2-dimensional output domain');
   if isempty(el), el = 1; end
 m1 = 12 ; m2 = 18;  % m1  by  m2  array of output units
 if (el == 1), m1 = m1*m2 ; m2 = 1 : end
 m = m1*m2 ;
 fprintf('The output lattice is %d by %d\n', m1, m2)
 mOK = input('would you like to change it? Y/N [N]: ','s');
 if isempty(mOK), mOK = 'n' ;  end
 if (mOK == 'y') | (mOK == 'Y')
   m = 1 ;
   while ~((m1 > 1) & (m > 1) & (m < 4000))
     m1 = input('size of the output lattice: \n  m1 = ') ;
      if (el == 2)
       m2 = input('m2 = ') ;
      end
     m = m1*m2 ;
   end
 end
 fprintf('The output lattice is %d by %d\n', m1, m2)
 % The position matrix V
 if el == 1
   V = (1:m1)' ;
 else
   [v1 v2] = meshgrid(1:m1, 1:m2); V = [v1(:) v2(:)];
 end
 % Creating input patterns
 N = 20*m ;    % N is the number of input vectors
 X = rand(1, N)+j*rand(1, N) ;
 ix = 1:N;
  if (indom == 2),
   ix = find((imag(X)<=2*real(X))&(imag(X)<=2-2*real(X))) ;
   elseif (indom == 3),
     ix = find(abs(X-.5*(1+j))<= 0.5) ;
   elseif (indom == 4),
     ix = find((abs(X-.5*(1+j))<= 0.5) & (abs(X-.5*(1+j)) >= 0.3)) ;
   elseif (indom == 5),
     ix = find((imag(X)<(2/3)&imag(X)>(1/3))| ...
               (real(X)<(2/3)&real(X)>(1/3))) ;
   elseif (indom == 6),
     ix = find((2.5*real(X)-imag(X)>0 & 2.5*real(X)-imag(X)<0.5) | ...
            (2.5*real(X)+imag(X)>2 & 2.5*real(X)+imag(X)<2.5) | ...
```

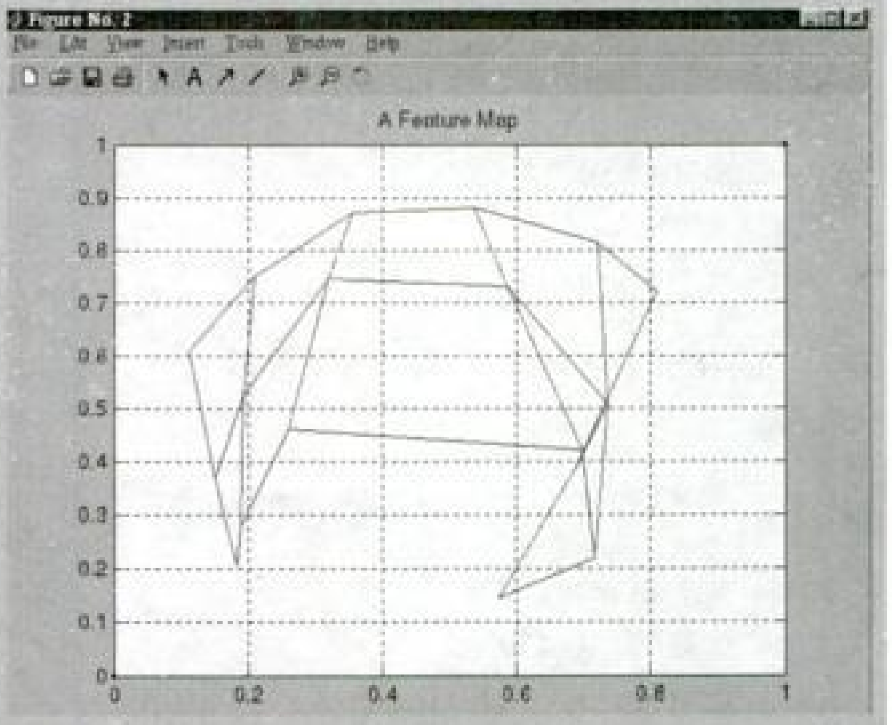
If the input domain is chosen as a **ring** and the dimensionality as 2, then we get,

```
Initialisation? Y/N [Y]: y
The output lattice is 12 by 18
would you like to change it? Y/N [N]: y
size of the output lattice:
 m1 = 6
 m2 = 3
The output lattice is 6 by 3
```

The response obtained is given by,



Input Distribution



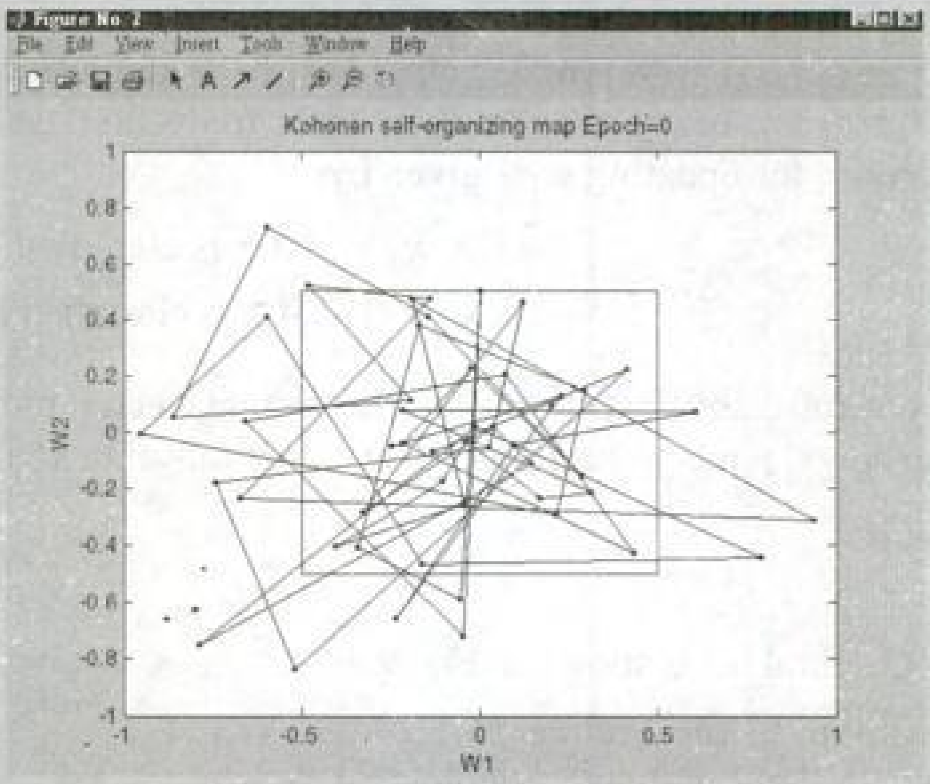
A Feature map

Example 9.5 Write a MATLAB program for a Kohonen self organizing feature map to cluster four vectors, (1 1 0 0), (0 0 0 1), (1 0 0 0) and (0 0 1 1) into two output cluster vectors i.e. m = 2. Take $\alpha(0) = 0.6$.

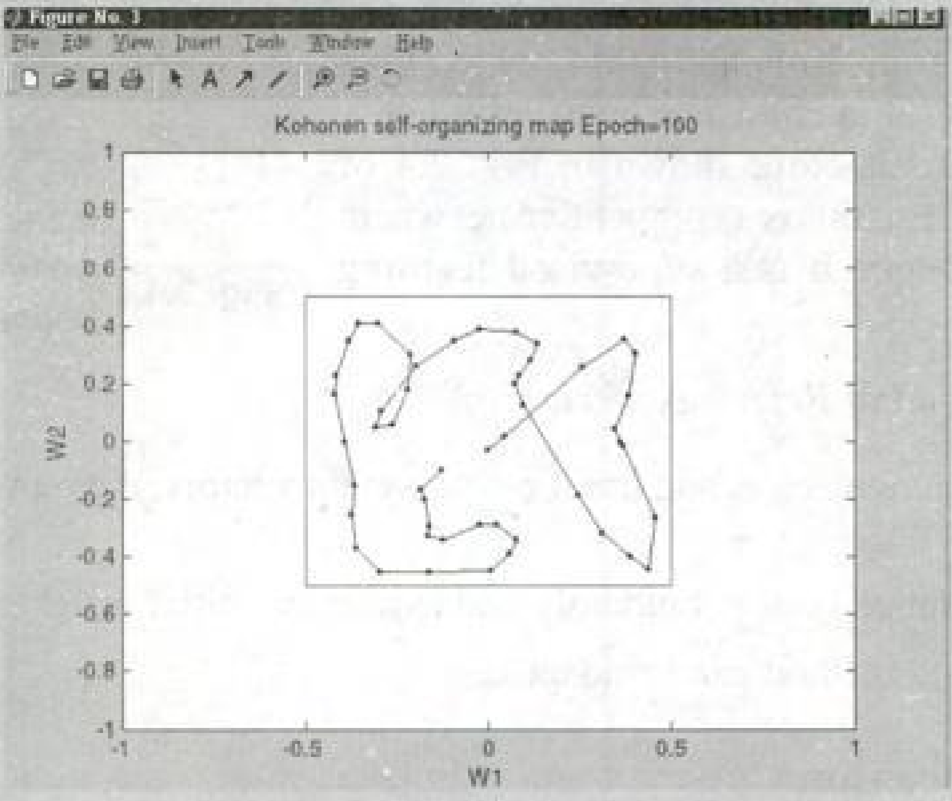
Solution The MATLAB program to cluster two vectors is given as follows.

Program

```
%Kohonen self organizing maps
clc;
clear;
x=[1 1 0 0;0 0 0 1;1 0 0 0;0 0 1 1];
alpha=0.6;
%initial weight matrix
w=rand(4,2);
disp('Initial weight matrix');
disp(w);
con=1;
epoch=0;
```



Self Organizing Map at Epoch 0



Self Organizing Map at 100th Epoch

9.4 Learning Vector Quantization (LVQ)

Kohonen has shown that it is possible to fine tune the class boundaries of an SOM in a "supervised" way and has developed several methods for doing this. They are all variations of learning vector quantization. Vector quantization is a standard statistical clustering technique, which seeks to divide the input space into areas that are assigned as "code book" vectors. In LVQ network, target values are available for the input training pattern and the learning is supervised. In training process, the output units are positioned to approximate the decision surfaces. After training, an LVQ net classifies an input vector by assigning it to the same class as the output unit that has its weight vector (reference or code book vector) closest to the input vector.

Step 5: Since T = 1 and $C_J = 1$, then. $T = C_J$

$$w_{1(new)} = w_{1(old)} + \alpha\,[x - w_1[old]]$$
$$= (1\ 0\ 1\ 0) + 0.1[(1\ 1\ 0\ 0) - (1\ 0\ 1\ 0)]$$
$$w_{1(new)} = (1\quad 0.1\quad 0.9\quad 0)$$

Step 3: For the input vector x = (1 0 0 1) with T = 2 perform Steps 4–5.

Step 4: Calculate J

$$D(j) = \sum_i \left(w_{ij} - x_i\right)^2$$
$$D(1) = (1-1)^2 + (0.1-0)^2 + (0.9-0)^2 + (0-1)^2 = 1.82$$
$$D(2) = (0-1)^2 + (0-0)^2 + (1-0)^2\,(1-1)^2 = 2$$
$$D(1) \text{ is minimum } J = 1 \Rightarrow C_J = 1.$$

Step 5: Since T = 2 and $C_J = 1$, $T \neq C_J$, the weight updation is,

$$w_{1(new)} = w_{1(old)} - \alpha[x - w_1[old]]$$
$$= (1\ \ 0.1\ \ 0.9\ \ 0) - 0.1\,[(1\ \ 0\ \ 0\ \ 1) - (1\ \ 0.1\ \ 0.9\ \ 0)]$$
$$w_{1(new)} = (1\quad 0.11\quad 0.99\quad -0.1)$$

Step 6: One epoch of training is completed. Reduce the learning rate

$$\alpha(t+1) = 0.5\,\alpha(t)$$
$$\alpha(1) = 0.5 \times 0.1$$
$$\alpha(1) = 0.05.$$

Step 7: Test the stopping condition. The required learning rate is not obtained. Perform the second epoch.

EPOCH-2

Step 1: Initialize weights $w_1 = (1\ \ 0.11\ \ 0.99\ \ -0.1)$ and $w_2 = (0\quad 0\quad 1\quad 1)$

Initializing the learning rate : $\alpha = 0.05$

Step 2: Begin the training process.

Step 3: For input vector x = (1 1 0 0) with T = 1 perform Steps 4–5.

Step 4: Calculate J.

$$D(j) = \sum_i \left(w_{ij} - x_i\right)^2$$
$$D(1) = (1-1)^2 + (0.11-1)^2 + (0.99-0)^2 + (-0.1-0)^2 = 1.7822$$
$$D(2) = (0-1)^2 + (0-1)^2 + (1-0)^2\ \ (1-0)^2 = 4$$
$$D(1) \text{ is minimum, } J = 1, \text{ hence } C_J = 1.$$

Step 5: Since T = 1 and $C_J = 1$, then $T = C_J$

$$w_{1(new)} = w_{1(old)} + \alpha[x - w_1(old)]$$
$$= (1\ \ 0.11\ \ 0.9\ \ -0.1) + 0.05[(1\ \ 1\ \ 0\ \ 0) - (1\ \ 0.11\ \ 0.99\ \ -0.1)]$$
$$w_{1(new)} = (1\quad 0.15\quad 0.94\quad -0.095)$$

```
            for j=1:2
             D(j)=0;
             for k=1:4
               D(j)=D(j)+(w(k,j)-x(i,k))^2;
              end
          end
         for j=1:2
            if D(j)==min(D)
                J=j;
              end
          end
         if J==t(i)
           w(:,J)=w(:,J)+alpha*(x(i,:)'-w(:,J));
          else
           w(:,J)=w(:,J)-alpha*(x(i,:)'-w(:,J));
          end
      end
     alpha=0.5*alpha;
     epoch=epoch+1;
     if epoch==100
         con=0;
      end
end
disp('Weight Matrix after 100 epochs');
disp(w);
```

```
Output
  Initial weight matrix
     1  0
     1  0
     0  0
     0  1
  Weight Matrix after 100 epochs
    1.0000    0
    0.2040  0.5615
       0        0.9584
       0        0.4385
```

9.5 Max Net

A specific competitive net that performs winner-takes-all competition is the Max net. Max net comes under the category of fixed-weight competitive net. In fixed-weight competitive nets, learning is not considered as an essential criteria. In Max net, a neural subnet is given which achieves the winner-takes-all competition.

Even if further interactions are made, the value of $a_{4(5)}$ remains the same, since all the other values $a_{1(5)}$, $a_{2(5)}$, $a_{3(5)}$ are equal to zero.

Thus the convergence has occurred.

$a_{1(1)} = 0$	$a_{2(1)} = 0.12$	$a_{3(1)} = 0.36$	$a_{4(1)} = 0.6$
$a_{1(2)} = 0$	$a_{2(2)} = 0$	$a_{3(2)} = 0.216$	$a_{4(2)} = 0.504$
$a_{1(3)} = 0$	$a_{2(3)} = 0$	$a_{3(3)} = 0.1152$	$a_{4(3)} = 0.4608$
$a_{1(4)} = 0$	$a_{2(4)} = 0$	$a_{3(4)} = 0.023$	$a_{4(4)} = 0.4378$
$a_{1(5)} = 0$	$a_{2(5)} = 0$	$a_{3(5)} = 0$	$a_{4(5)} = 0.4332$

9.6 Mexican Hat

A more general form of competition is the Mexican hat or On-center-off surround contrast enhancement. Each neuron is connected with excitatory links to a number of "cooperative neighbors", neurons that are in close proximity. Each neuron is also connected with inhibitory links to a number of "competitive neighbors". The excitatory links are positively weighted and the inhibitory links are negatively weighted. There may also be a number of neurons, further away still, to which the neuron is not connected. The neurons receive an external signal in addition to these interconnection signals. The pattern of interconnections is repeated for each neuron in the layer.

9.6.1 Architecture

The architecture for the Mexican Hat is shown in Fig. 9.6.

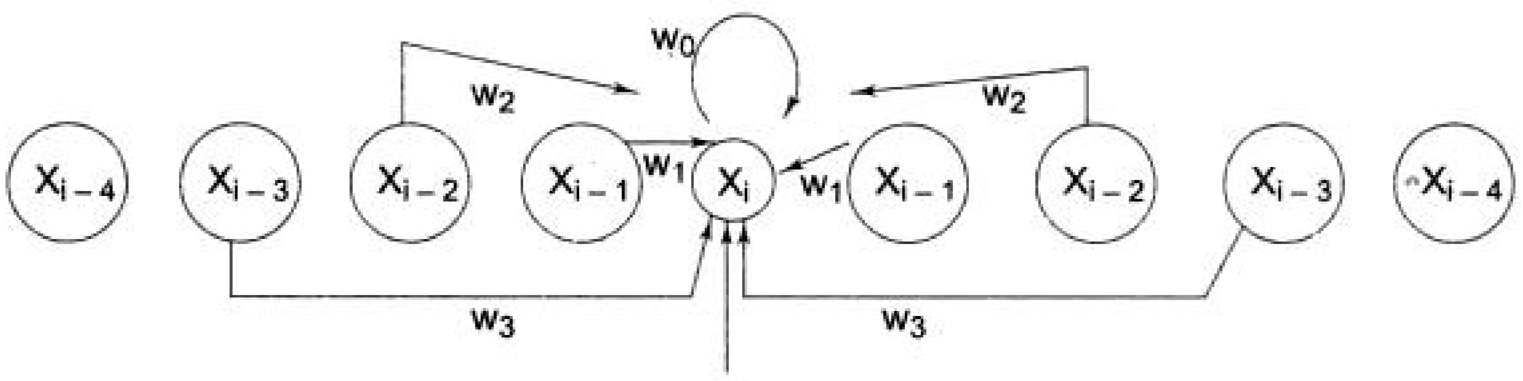


Fig. 9.6 *Mexican Hat Interconnections for Unit X_i*

It is seen from the architecture that the neurons are arranged in a linear order, with positive connection between unit X_i and neighboring units one or two positions or either side; negative connections are shown for units three positions on either side. The size of the region of cooperation (positive connection) and the region of competition (negative connection) may vary, as may the relative magnitudes of the positive and negative weights and the topology of the regions.

The interconnection for the Mexican hat net involves two symmetric regions around each individual neuron. The weights between a typical unit X_i and units X_{i+1}, X_{i+2}, X_{i-1} and X_{i-2}, are positive. These weights are w_1 and w_2. The weights between X_1 and units X_{i-3} and X_{i+3} are negative. The X_{i-4}, X_{i-4} are not connected to X_i.

Units within radius 2 have positive weights.

Step 6: The activations are stored in x-old.

x-old = (0.0, 0.704, 2.0, 2.0, 2.0, 0704, 0.0)

Step 7: Incrementing the creation counter.

$$t = t + 1$$

$$t = 2 + 1 \Rightarrow t = 3.$$

Step 8: The stopping condition has been reached since 't' exceeds t-max which equals 2. (iteration of contrast enhancement).

The activations for t = 0, 1, 2 are shown in Fig. 9.7.

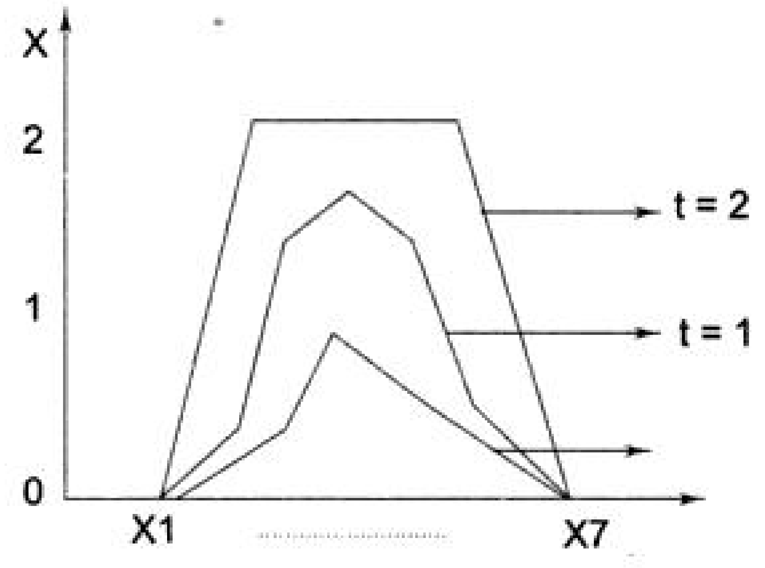


Fig. 9.7 *Results of Mexican Hat Problem Example 9.6*

9.7 Hamming Net

Lippmann worked on the hamming net during 1987. In Hamming net, max net is used as a subnet to find the unit with the largest net input. It is a maximum likelihood classifier net that determines which of several exemplar vectors is most similar to an input vector (n-tuple). The weight of the net in this case is determined by the exemplar vectors.

Hamming Distance

The hamming distance between two vectors is the number of components in which the vectors differ.

It can also be defined as the number of differences between two binary or bipolar vectors (x, y). It can be denoted as H(x, y).

The average hamming distance is given as,

$$= \frac{1}{n} H(x, y)$$

where n is the number of components in each vector.

In Hamming net, the measure of similarity between the input vector and the stored exemplar is minus the Hamming distance between the vectors.

Consider two bipolar vectors x and y,

If 'a' –	Number of components in which the vectors agree
and 'd' –	Number of components in which the vectors differ (hamming distance)
Then,	x. y = a – d

If 'n' is the number of components, then,

$$n = a + d \text{ (or) } d = n - a$$

As a result

$$x.\,y = a - d = a - (n - a)$$

$$x.\,y = 2a - n \quad \text{(or)} \quad 2a = x\,.\,y + n$$

For example, for the first pattern x = (1 1 –1 –1), y_{in1} = 2, which also gives the number of matching components in the input pattern and exemplar vector c(1) = (–1 1 –1 1). The match is found to occur in the 2nd and the 3rd component. If we check, all the net input patterns calculated for the respective input patterns, their match with the exemplar vectors can be found.

Summary

The competitive nets discussed in this chapter determine the winner by means of winner-takes-all. The learning rule used here is the Kohonen learning rule. The fixed weight competitive networks Max Net, Mexican Hat and Hamming net are included in this chapter. Supervised type of competitive net, Learning Vector Quantization is discussed along with its variants. The architecture, algorithm and other solved problems related to Kohonen self organizing feature maps have also been described in detail in this chapter.

Review Questions

9.1 How is competition performed using neural networks?

9.2 Write short note on the winner-takes-all competition.

9.3 What are the methods of determining the winner-takes-all?

9.4 What are the various classifications of competition based nets?

9.5 How is competition performed for supervised learning and unsupervised learning?

9.6 Define exemplar vector (code book vector).

9.7 State the Kohonen learning rule.

9.8 Discuss the dot product and Euclidean distance of determining the winner-takes-all in detail.

9.9 What are the various fixed weight nets?

9.10 How does max net act as a subnet?

9.11 Draw the architecture of the max net. How are the symmetrical weights assumed here?

9.12 What is the activation function used in a max net.

9.13 State the application algorithms of a max net.

9.14 Give a detail description of a Mexican hat net.

9.15 Explain in detail with architecture, the training algorithm of a Mexican hat net.

9.16 Differentiate between reinforcement and negative reinforcement signals.

9.17 How is hamming net used as a maximum likelihood classifier net?

9.18 How is max net used as a subnet in hamming net?

9.19 Define hamming distance.

9.20 Derive an expression involving bipolar vectors x and y, with the total no of components and no of components agreed.

9.21 Draw the architecture of a Hamming net.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

sense that it provides solution for those applications which cannot have larger iterations. As a result, CPN can be used for data compression, approximation of functions, pattern association, pattern completion and signal enhancement applications.

Counter propagation is a combination of two well-known algorithms: the self organizing map of Kohonen and the Grossberg out star. Together they have the properties which are absent in either of the one.

The counter propagation network functions as a look up table capable of generalization. The training process associates input vectors with corresponding output vectors. By constructing a look up table, a large number of data points are compressed to a more number of entries (in the Look Up Table (LUT)). The net will approximate a function; if the data given represents function values.

Counter propagation networks are trained in two stages:

Stage 1: The input vectors are clustered on the basis of Euclidean distances or by the dot product method.

Stage 2: The desired response is obtained by adopting the weights from the cluster units to the output units.

Counter propagation network is classified in two types. They are

1. Full counter propagation network
2. Forward only counter propagation network.

CPN is inferior to back propagation for most mapping network applications. Its advantages are that it is simple and forms a good statistical model of its input vector environment.

The CPN trains rapidly. If appropriately applied it can save large amounts of computing time. It is also useful for rapid prototyping of systems. The full CPN and forward only CPN are discussed in detail in the subsections that follow.

10.2 Full Counter Propagation Network (Full CPN)

The full CPN possess the generalization capability which allows it to produce a correct output even when it is given an input vector that is partially incomplete or partially incorrect. Full CPN can represent large number of vector pairs, x:y by constructing a look up table.

We know that the CPN operates in two stages. In full CPN, during first phase, the training vector pairs are used to form clusters. Clustering can be done either by dot product or by Euclidean distance. But if dot product method is adopted to compare vector pairs, normalization is a must. Hence Euclidean distance method is usally followed. During second phase, the weights are adjusted between the cluster units and the output units.

10.2.1 Architecture

The architecture of the full CPN is shown in Fig. 10.1.

The given vector pairs are x:y and the approximated vector pairs may be x* : y*.

Step 11: Set X input layer activations to vector x;

Set Y input layer activations to vector y;

Step 12: Find winning cluster unit using Euclidean distance

Step 13: Updating the weights for winning unit, the value of α and β in this phase are constant.

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \alpha(x_i - v_{ij}(\text{old})); \quad i = 1 \text{ to } n$$

$$w_{kj}(\text{new}) = w_{kj}(\text{old}) + \beta(y_k - w_{kj}(\text{old})); \; k = 1 \text{ to } m$$

Step 14: Update weights from unit z_j to the output layers

$$u_{jk}(\text{new}) = u_{jk}(\text{old}) + a(y_k - u_{jk}(\text{old})); \quad k = 1 \text{ to } m$$

$$t_{ji}(\text{new}) = t_{ji}(\text{old}) + b(x_i - t_{ji}(\text{old})); \quad i = 1 \text{ to } n$$

Step 15: Learning rates *a* and *b* are to be reduced.

Step 16: Test the stopping condition for phase 2 training.

The winning unit selection is done either by dot product or Euclidean distance.

The dot product is done by calculating the net input.

$$Z_{\text{-in}j} = \sum_i x_i u_{ij} + \sum_k y_k w_{kj}$$

The cluster unit with the largest net input is winner. Here the vectors should be normalized.
In Euclidean distance,

$$D_j = \sum_i \left(x_i - v_{ij}\right)^2 + \sum_k \left(y_k - w_{kj}\right)^2$$

The square of whose distance from the input vector is smallest is the winner.

In case of tie between the selections of the winning unit, the unit with the smallest in desk is selected.

The stopping condition may be the number of iteration or the reduction in the learning rate up to a certain level.

10.2.4 Application Procedure

In the training algorithm, if only one Kohonen neuron is activated for each input vector, this is called the Accretive mode. If a group of Kohonen neurons having the highest outputs is allowed to present its outputs to the Grossberg layer, this is called the Interpolative mode. This mode is capable of representing more complex mappings and can produce more accurate results.

The application procedure of the full CPN is

Step 1: Initialize weights

Step 2: For each input pair x : y, perform Steps 3–5

Step 3: Set x input layer activations to vector x;

Set y input layer activations to vector y;

Step 4: Find the cluster unit Z_J close to the input pair.

Step 5: Compute approximations to x and y:

$x_i^* = t_{Ji}$

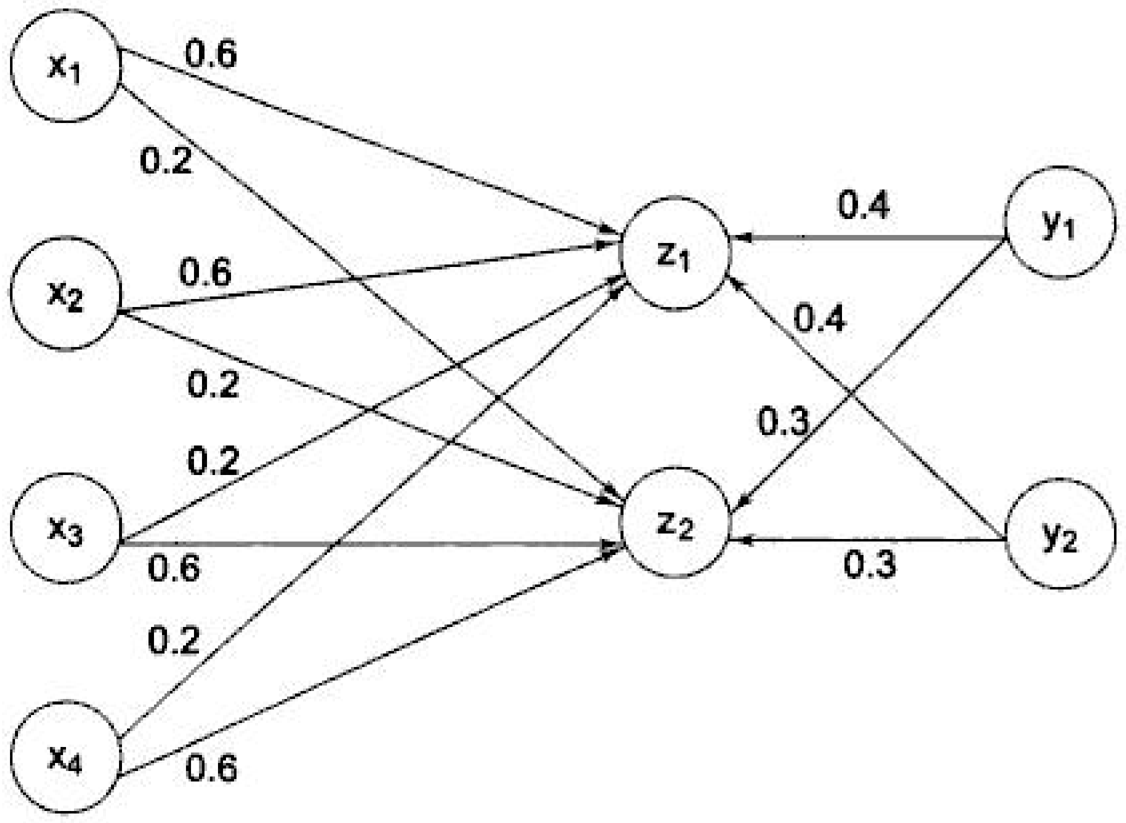
$y_k^* = u_{Jk}$

Thus the first iteration is performed. The updated weights are,

$$u = \begin{bmatrix} 0.2 & 0.51 \\ 0.4 & 0.65 \end{bmatrix} \quad w = \begin{bmatrix} 0.1 & 0.21 \\ 0.6 & 0.79 \end{bmatrix}$$

These weights may be further used for iterations, depending upon the reduction of learning rates and the stopping conditions.

Example 10.3 Write a MATLAB program for a full CPN net work given below.



Use input pair x = (0 1 1 0), y = (1 0). Perform first phase of training (one step only). Find the activation of the cluster layer units and update weights using a learning rate of 0.3.

Solution The counter propagation is most popular for its application in data compression. In the given problem, the network is trained only for one step and the output is given.

Program

```
%Full Counter Propagation Network for given input pair
 clc;
 clear;
 %set initial weights
 v=[0.6 0.2;0.6 0.2;0.2 0.6; 0.2 0.6];
 w=[0.4 0.3;0.4 0.3];
 x=[0 1 1 0];
 y=[1 0];
 alpha=0.3;
 for j=1:2
    D(j)=0;
    for i=1:4
       D(j)=D(j)+(x(i)-v(i,j))^2;
    end
    for k=1:2
      D(j)=D(j)+(y(k)-w(k,j))^2;
    end
```

Step 4: Calculate winning cluster unit

$$D_{(j)} = \sum_i \left(x_i - v_{ij}\right)^2 \ ; i = 1 \text{ to } 2.$$

$$D(1) = \sum_{i=1}^{2} \left(x_i - v_{i1}\right)^2$$

$$D(1) = (1 - 0.3)^2 + (0 - 0.1)^2 = 0.5$$

$$D(2) = (1 - 0.6)^2 + (0 - 0.5)^5 = 0.41$$

$$D(2) < D(1)$$

$$J = 2$$

Step 5: Updating the weights on the winner unit,

$$V_{in(new)} = v_{iJ}(\text{old}) + \alpha(x_i - v_{iJ}(\text{old})),\ i = 1 \text{ to } 2.$$

$$v_{12}(n) = v_{12(old)} + \alpha(x_1 - v_{12(0)})$$

$$v_{12}(n) = 0.6 + 0.5(1 - 0.6) = 0.8$$

$$v_{22}(n) = 0.5 + 0.5(0 - 0.5) = 0.25$$

The updated weights are,

$$\begin{bmatrix} 0.3 & 0.8 \\ 0.1 & 0.25 \end{bmatrix}$$

Step 6&7: Keep α-same constant value for using it in the second phase. If stopping condition is mentioned, check or move to next step.

Step 8: Begin second phase of the training.

Step 9: Present input vector pair x and y.

Step 10: Set the activations to,

x = (1 0) and y = (0 1)

Step 11: Calculate the winning cluster unit,

$$D_{(j)} = \sum_i \left(x_i - v_{ij}\right)^2$$

$$D(1) = (1 - 0.3)^2 + (0 - 0.1)^2 = 0.5$$

$$D(2) = (1 - 0.8)^2 + (0 - 0.25)^2 = 0.1025$$

$$D(2) < D(1)$$

$$J = 2$$

Step 12: Update the weights into cluster unit

$$v_{in(new)} = v_{iJ}(\text{old}) + \alpha(x_i - v_{iJ(old)})\ I = 1 \text{ to } 2.$$

$$v_{12(n)} = v_{12(o)} + \alpha(x_i - v_{12(old)})$$

$$v_{12(n)} = 0.8 + 0.5(1 - 0.8) = 0.9$$

$$v_{22(n)} = 0.25 + 0.5(0 - 0.25) = 0.125$$

Step 13: Update the weights from the cluster unit to the output unit

$$w_{Jk}(\text{new}) = w_{Jk}(\text{old}) + a(y_k - w_{Jk}(\text{old}));\ K = 1 \text{ to } 2.$$

What You Will Learn

- ART networks, their properties and algorithms.
- ART1 and ART2 networks and the differences between them.
- Phases in the ART classification process: recognition, comparison and search.
- Basic architecture of adaptive resonance neural networks.

CHAPTER 11 Adaptive Resonance Theory

11.1 Introduction

Adaptive Resonance Theory or ART refers to a class of self organizing neural architectures that cluster the pattern space and produce appropriate weight vector templates. Conventional artificial neural networks have failed to solve the stability–plasticity dilemma. A network remains open to new learning (remain plastic) without washing away previously learned codes. Too often, learning a new pattern erases or modifies previous training. If there is only a fixed set of training vectors, the network can be cycled through these repeatedly and may eventually learn all. In a real network, it will be exposed to a constantly

11.3.1 Architecture

The ART 1 network has computational units and supplemental units. It's architecture is shown in Fig. 11.1.

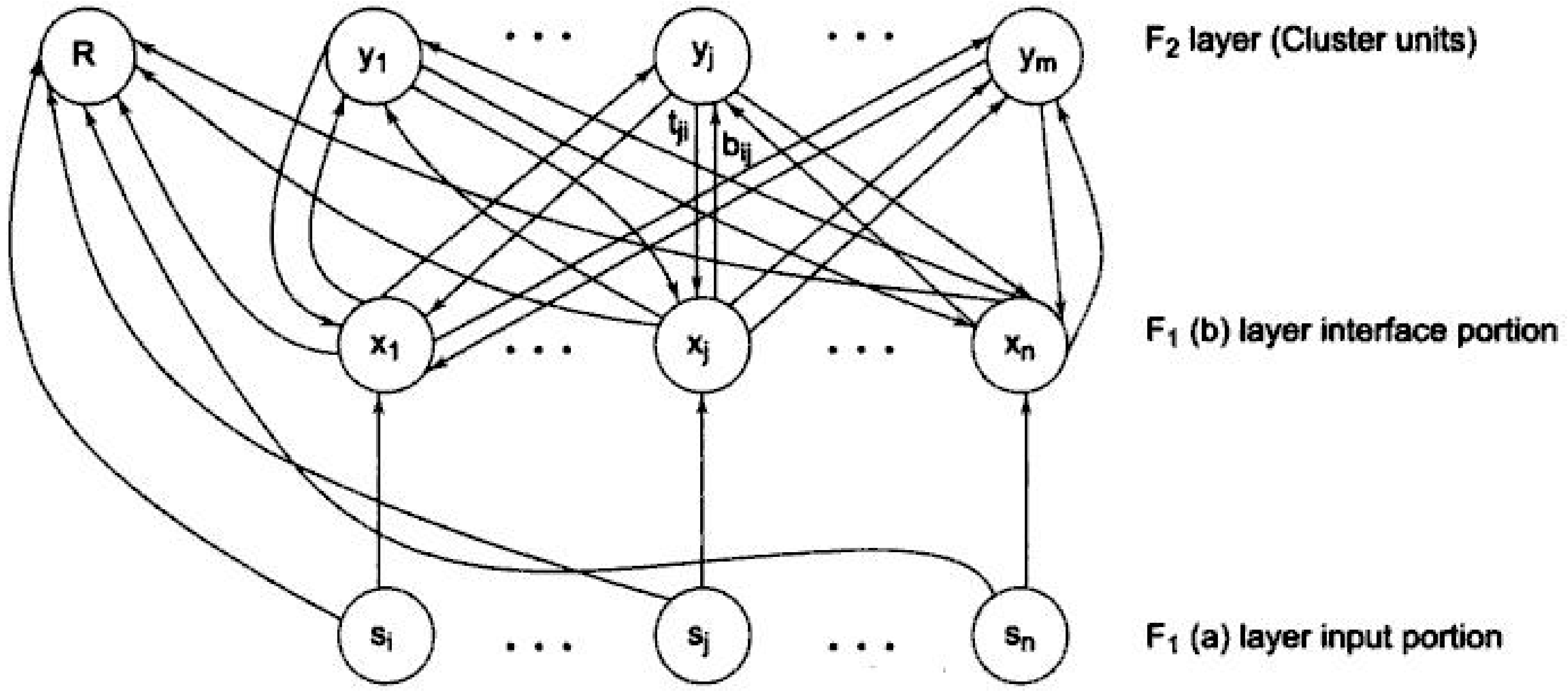


Fig 11.1 Architecture of ART 1

Computational Unit

The computational unit has F_1 and F_2 units and reset unit. The F_1 (a) is connected to F_1 (b) interface unit. Both input and interface units are connected to reset mechanism unit. By means of top-down and bottom-up weights, the interface layer units are connected to the cluster units and the reciprocity is also achieved.

Supplemental Units

The structure of supplemental unit is shown in Fig. 11.2.

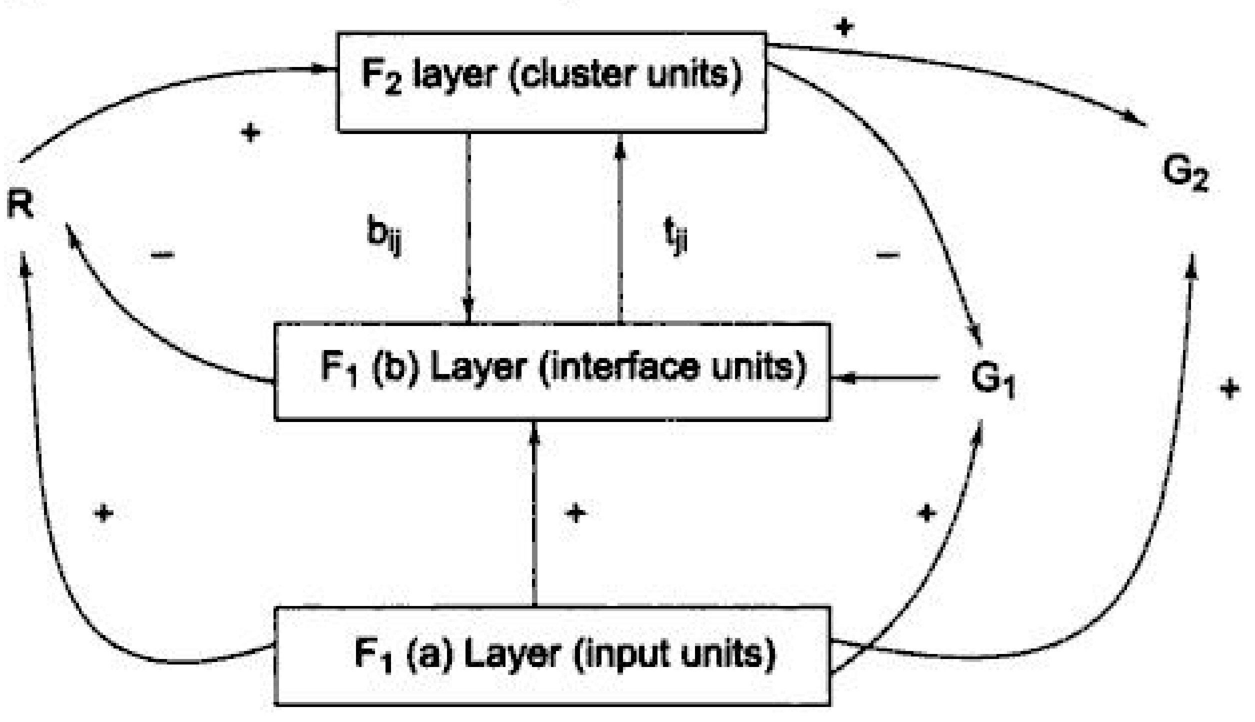


Fig 11.2 Structure of Supplemental Units

Solution (a) The vigilance parameter is 0.4

Step 1: Initialize parameters

$n = 9, m = 2, L = 2, \rho = 0.4$

$$b_{ij} = \begin{bmatrix} \frac{1}{2} & \frac{1}{5} \\ 0 & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{5} \\ 0 & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{5} \\ 0 & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{5} \\ 0 & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{5} \end{bmatrix} \qquad t_{ji} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Step 2: Begin training.

Step 3: For input vector (1 1 1 1 0 1 1 1 1) do Steps 4 to 13.

Step 4: Set activations of all F_2 units to zero.

Set activations of F_1 (a) units to input vector

$$s = (1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1)$$

Step 5 : Compute the norm of s.

$$\|s\| = \sum s_i$$

$$\|s\| = 8.$$

Step 6: Input signal from F_1 (a) to F_1(b), so,

$$x = (1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1)$$

Step 7: Calculate the net input $y_j = \sum_i x_i b_{ij}$

$$y_1 = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} = 2$$

$$y_2 = 8\left(\frac{1}{5}\right) = 1.6$$

$$y_1 > y_2$$

$$J = 1.$$

$$b_{ij(new)} = \begin{bmatrix} \frac{1}{2} & \frac{2}{9} \\ 0 & \frac{2}{9} \\ \frac{1}{2} & \frac{2}{9} \\ 0 & \frac{2}{9} \\ \frac{1}{2} & \frac{2}{9} \\ 0 & \frac{2}{9} \\ \frac{1}{2} & \frac{2}{9} \\ 0 & \frac{2}{9} \\ \frac{1}{2} & \frac{2}{9} \end{bmatrix}$$

Then new top down weight is.

$$t_{Ji(new)} = x_i$$

$$t_{Ji(new)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Step 14: Test for stopping condition.

Example 11.2 Consider an ART1 network with four input units and three cluster units. Determine the updation in weights when vectors (1 0 1 0) (1 0 0 1), (0 1 1 1) and (1 1 0 1) are input. Assume the vigilance parameter as 0.3.

Solution The algorithm is traced as follows.

Step 1: Initialize parameters

$$n = 4, m = 3, \rho = 0.3, L = 2, b_{ij}(0) = \frac{1}{1+n} = \frac{1}{1+4} = 0.2$$

$$t_{ij}(0) = 1. \text{ i.e. } b_{ij} = \begin{bmatrix} 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \end{bmatrix} \text{ and } t_{ij} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Step 2: Begin training.

Step 3: For the first input vector (1 0 1 0) do Steps 4 to 13

Step 4: Set activations of all F_2 units to zero
Set activations of $F_1(a)$ units to input vector.

$$s = (1\ 0\ 1\ 0)$$

Step 3: For the four the input vector (1 1 0 1) do Steps 4 to 13.

Step 4: Set activations of all F_2 units to zero

Set activations of F_1(a) units to input vector

$$S = (1\ 1\ 0\ 1)$$

Step 5: Compute $\|s\|$

$$\|s\| = 3$$

Step 6: Compute activation of F_1(b) layer

$$x = (1\ 1\ 0\ 1)$$

Step 7: Calculate net input

$$y_j = \sum_i x_i b_{ij}$$

$$y_1 = 1 \times 1 + 0 \times 1 + 0 \times 0 + 1 \times 0 \quad = 1$$

$$y_2 = 1 \times 0 + 1 \times 0.5 + 0 \times 0.5 + 1 \times 0.5 \quad = 1$$

$$y_3 = 1 \times 0.2 + 1 \times 0.2 + 0 \times 0.2 + 1 \times 0.2 = 0.6$$

Since y_1 and y_2 are equal, the unit with smallest index is chosen J = 1

Step 8: While reset is true do Steps 9–12.

Step 9: The winning unit is J = 1

Step 10: Recompute activation to F_1

$$\begin{aligned} x_i &= s_i t_{Ji} \\ &= (1\ 1\ 0\ 1)\,(1\ 0\ 0\ 0) \\ x_i &= (1\ 0\ 0\ 0) \end{aligned}$$

Step 11: Calculate norm of x.

$$\|x\| = 1$$

Step 12: Test for reset

$$\frac{\|x\|}{\|s\|} = \frac{1}{3} = 0.33 > 0.3$$

i.e.

$$\frac{\|x\|}{\|s\|} > \rho$$

So, proceed with Step 13.

Step 13: Updation of weights

$$b_{ij(new)} = \frac{Lx_i}{L - 1 + \|x\|}$$

For $x_i = 1$

```
                    break;
                end
            end
           if y(J)==-1
               con1=0;
           else
               for i=1:n
                 x(i)=s(I,i)*t(J,i);
               end
              nx=sum(x);
              if nx/ns <vp
                  y(J)=-1;
                  con1=1;
               else
                  con1=0;
               end
           end
       end
      cl(I)=J;
      for i=1:n
        b(i,J)=L*x(i)/(L-1+nx);
        t(J,i)=x(i);
      end
   end
   epoch=epoch+1;
   if epoch==epn
      con=0;
   end
end
for i=1:n
   for j=1:m
      if b(i,j)>0
         pb(i,j)=1;
      else
         pb(i,j)=-1;
      end
   end
end
pb=pb';
figure(2);
k=1;
for i=1:3
   for j=1:5
     charplot(pb(k,:),10+(j-1)*15,50-(i-1)*15,9,7);
      k=k+1;
   end
```

Step 1: Initialize parameters a, b, θ, c, d, e, α, p.

Step 2: Perform steps 3–13 up to specified number of epochs of training

Step 3: For each input vector 's', do Steps 4–12

Step 4: Update F_1 unit activations

$$u_i = 0, \quad x_i = \frac{s_i}{e + \|s\|}$$

$$w_i = s_i, \qquad q_i = 0$$

$$p_i = 0\,, \qquad v_i = f(x_i)$$

Update F_1 unit activations again.

$$x_i = \frac{v_i}{e + \|v\|}, \qquad w_i = s_i + au_i$$

$$p_i = u_i, \qquad x_i = \frac{w_i}{e + \|w\|}, \qquad q_i = \frac{p_i}{e + \|p\|}$$

$$v = f(x_i) + bf(q_i)$$

Step 5: Compute signals to F_2 units

$$y_j = \sum_i b_{ij} p_i$$

Step 6: While reset is true, perform Steps 7–8.

Step 7: For F_2 unit choose Y_J with largest signal

Step 8: Check for reset

$$u_i = \frac{v_i}{e + \|v\|}, \qquad p_i = u_i + d\, t_{Ji}$$

$$r_i = \frac{u_i + cp_i}{e + \|u\| + c\|p\|}$$

If $\|r\| < \rho - e$, then

$$Y_J = -1 \text{ (inhibit J)}$$

Since reset is true, go to Step 6

If $\|r\| > \rho - e$, then

$$w_i = s_i + au_i$$

$$v = f(x_i) + bf(q_i)$$

Reset is false, so go to Step 9.

Step 9: Perform Steps 10–12 up to specified number of learning iterations.

Step 10: Update weights for winning unit J.

$$x = \frac{w}{e + \|w\|} = \frac{(10.71,\ 0.69)}{10.732} = (0.998, 0.064)$$

$$q = \frac{p}{e + \|p\|} = \frac{(1.486,\ 0)}{0 + 1.486} = (1, 0)$$

$$v = f(x) + bf(q) = (10.998, 0)$$

Step 12: Test stopping conditions for weight updates.

Step 13: Test stopping condition for number of epochs.

Step 3: For input vector (0.69, 0.71) do Steps 4--12.

Step 4: Update F_1 unit activations

$$s = (0.69, 0.71)$$

$$u = \frac{v}{e + \|v\|} = 0$$

$$w = s + au = (0.69, 0.71)$$

$$p = u + dt_J = (0, 0)$$

$$x = \frac{w}{e + \|w\|} = \frac{(0.69,\ 0.71)}{0.99} = (0.697, 0.717)$$

$$q = \frac{p}{e + \|p\|} = (0, 0)$$

$$u = f(x) + bf(q)$$
$$= f(0.697, 0.717)$$
$$v = (0, 0.717) = (0, 0.72)$$

Update F_1 unit activations again

$$u = \frac{v}{e + \|v\|} = \frac{(10,\ 0.72)}{0.72} = (0, 1)$$

$$w = s + au = (0.69, 0.71) + 10(0, 1) = (0.69, 10.71)$$

$$p = u + dt_J = (0, 1)$$

$$x = \frac{w}{e + \|w\|} = \frac{(10.69,\ 10.71)}{0 + 10.732} = (0.064, 0.998)$$

$$v = f(x) + bf(q)$$
$$= f(0.064, 0.998) + 10f(0, 1)$$
$$= (0, 0.998) + 10(0, 1)$$
$$v = (10.998, 0)$$

Step 5: Compute signals to F_1 units

```
disp(m);
disp('Top Down Weight');
disp(tw);
disp('Bottom Up Weight');
disp(bw);
```

```
Output
  Number of inputs
    2
  Number Cluster Formed
    3
  Top Down Weight
        0      0
    4.2600     0
        0      0
  Bottom Up Weight
    7.0711   8.3188   7.0711
    7.0711   4.0588   7.0711
```

Summary

In this chapter, the adaptive resonance theory was described in detail. It solves the stability–plasticity problem and performs well. The architecture of the ART was designed to be biologically plausible. The mechanism of ART is intended to be consistent with that of the brain. The ART network can be implemented with parallel processors; all dot products in recognition layer can be performed simultaneously. The search in this case is found to be very rapid. The ART networks prevent the modification of the patterns that have been learned previously. Hence they maintain the plasticity required to learn new patterns. In the above sections ART1 and ART2 network has been discussed in detail.

Review Questions

11.1 What is the basic concept behind Adaptive Resonance Theory (ART)?

11.2 What is plasticity with reference to neural networks?

11.3 How is ART net designed for both stability and plasticity?

11.4 What is the learning trail?

11.5 Write short note on the basic architecture and operation of ART network.

11.6 What are the three states of ART network?

11.7 What are the two types of learnings in the ART net?

11.8 What are the two forms of ART network?

11.9 Explain the architecture of ART1 network. State in detail the computational and supplemental units.

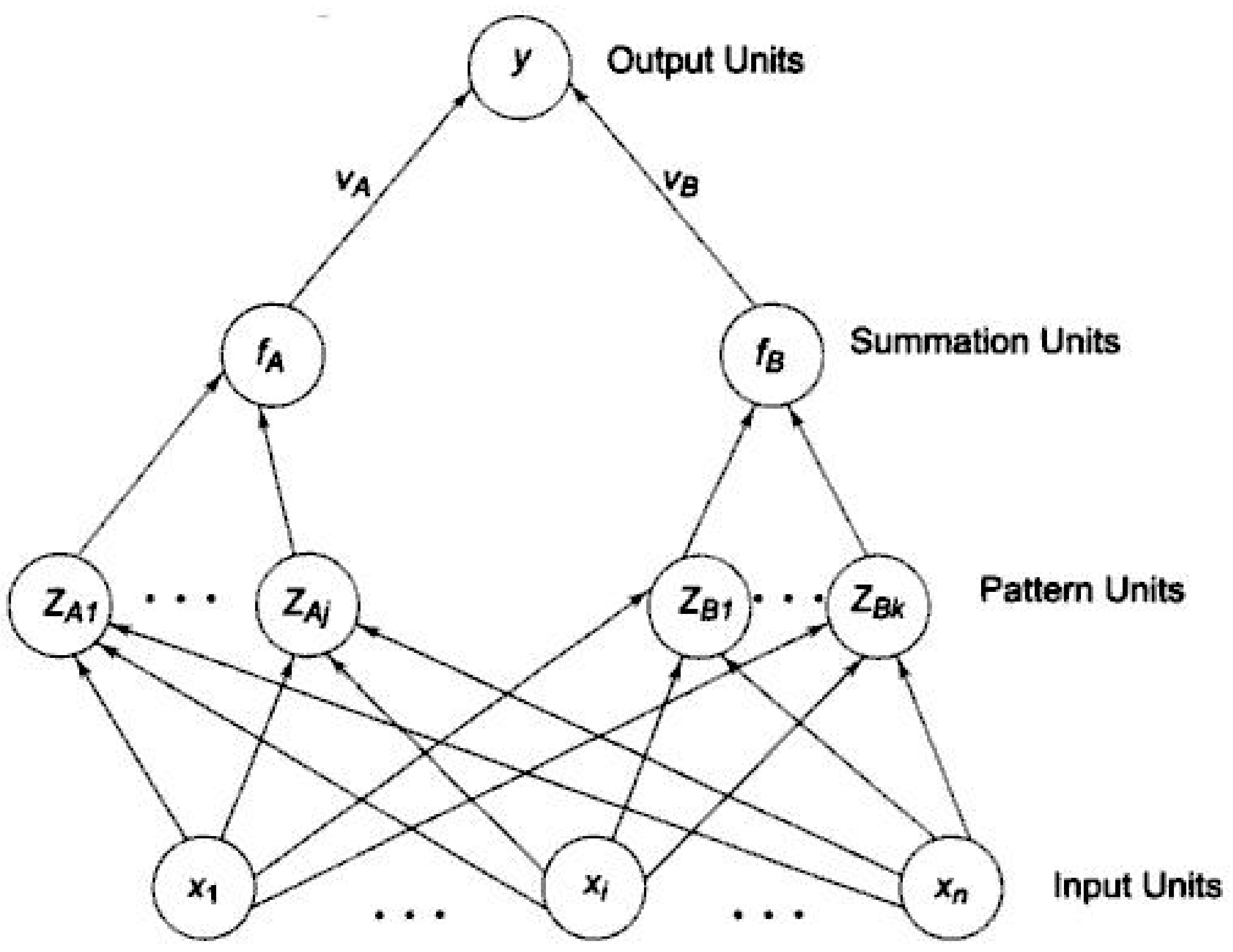


Fig. 12.1 Architecture of Probabilistic Neural Net

The architecture is made up of four types of units.

- Input units
- Pattern units – Class A and Class B.
- Summation units
- Output units

The weights between the summation unit and the output unit are,

$$V_A = 1$$

$$V_B = -\frac{P_B C_B m_A}{P_A C_A m_B}$$

12.2.2 Training Algorithm

The training algorithm for the probabilistic neural net is given as,

Step 1: For each training input pattern, $x(p)$, $p = 1,\ldots,$ P perform Steps 2–3

Step 2: Create pattern unit Z_p:

Weight vector for unit Z_p:

$$w_P = x(p)$$

(unit Z_p is either a Z_A unit or Z_B unit)

neuron gives output signals. Similarly in layer 1, there is an inhibitory neuron with the same connection region. The weights coming into inhibitory cells are not modified during training, their weights into any inhibitory neuron is equal to one. Hence, the output of the inhibitory cell is simply the weighted sum of its inputs, which in this case is the arithmetic mean of the excitatory output to which it connects. Hence,

$$\text{Inhibit} = \sum_i u_i \, \text{output}_i$$

where,

$$\sum_i u_i = 1$$

and

$$u_i = \text{inhibitory weight } i$$

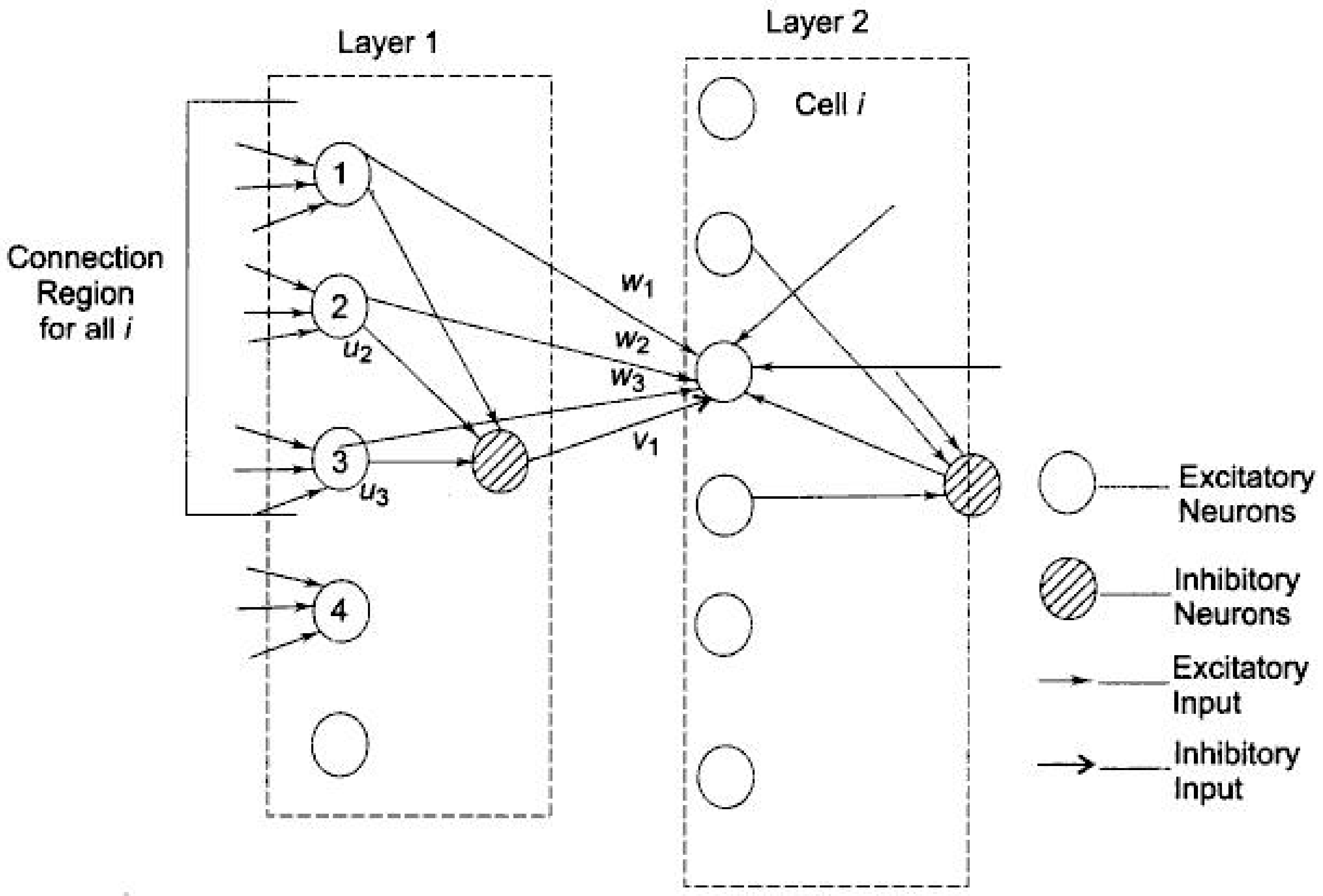


Fig. 12.5 *Cognitron Layers*

12.3.5 Training Problems

The weights associated with an excitatory neuron are adjusted only when it is firing more strongly than any of the neighboring cells in the competitive region. The change in one of its weights in the case is calculated as,

$$\delta w_i = b u_j x_j$$

where,

- u_j – The inhibitory weight coming from neuron j in layer 1 to the inhibitory neuron i.
- x_j – The output of neuron j in layer 1.
- w_i – Excitatory weight i.
- b – Learning rate coefficient.

The net finds the maximum by making each unit to change its state.

The change in consensus if unit X_i were to change its state is given by,

$$\Delta c(i) = [1 - 2x_i][w_{ij} + \sum w_{ij}x_j]$$

where x_i is the current state.

$$(1 - 2x_i) = \begin{cases} +1, & \text{if } x_i\text{–'on'} \\ -1, & \text{if } x_i\text{–'off'} \end{cases}$$

The probability of the net accepting a change in state for unit X_i is,

$$A(i, t) = \frac{1}{1 + \exp\left(\frac{-\Delta e(i)}{T}\right)}$$

To obtain the maximum consensus, the parameter T is to be reduced gradually.

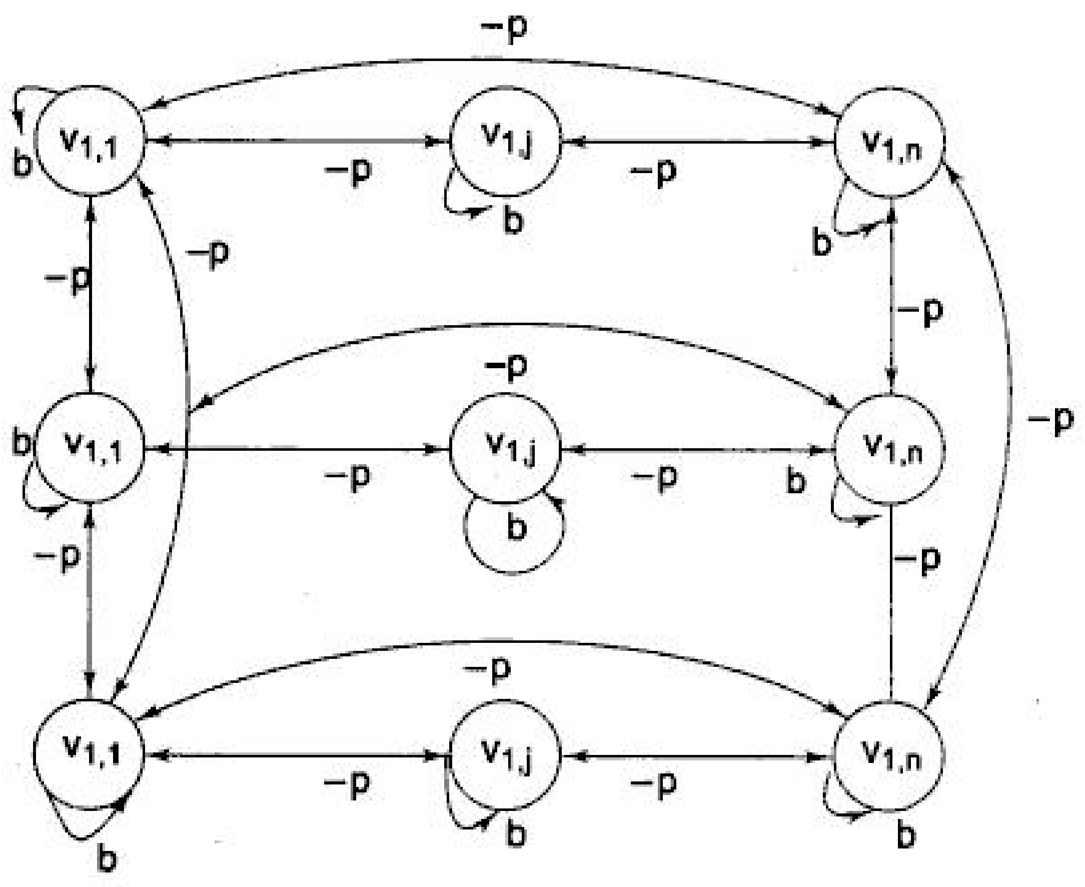


Fig. 12.6 Architecture of Boltzmann Machine

The weights $-p$ indicates the penalties for violating the condition that almost one unit be "on" in each row and column. The self-connection weights b indicates the incentives or bonus to a unit to turn "on", without making more than one unit to be "on" in a row or column. The net will function as desired if $p > b$.

12.5.2 Application Algorithm

The application algorithm of the Boltzmann machine is as follows: here, the weights between unit V_{ij} and $V_{I,J}$ is denoted by W(i, j : I, J)

12.9 Optical Neural Networks

Optical neural networks interconnect neurons with light beams. As a result no insulation is required between signal paths, the light rays can pass through between each other without interacting. The signal path can be made to travel in three dimensions. The density of the transmission path is limited only by the spacing of light sources, the effect of divergence and the spacing of the detectors. As a result all signal paths operate simultaneously, which results in a true data rate. The strengths of weights are stored in holograms with high density. These weights can be modified during operation to produce a fully adaptive system. There are two categories of this optical neural network. They are electro-optical matrix multipliers and Holographic correlators.

12.9.1 Electro-optical Matrix Multipliers

These nets provide a means for performing matrix multiplication in parallel. The network speed is limited only by the available electro-optical components; in this case the computational time is potentially in the nanosecond range. The Fig. 12.10 shows the electro-optical vector matrix multiplier.

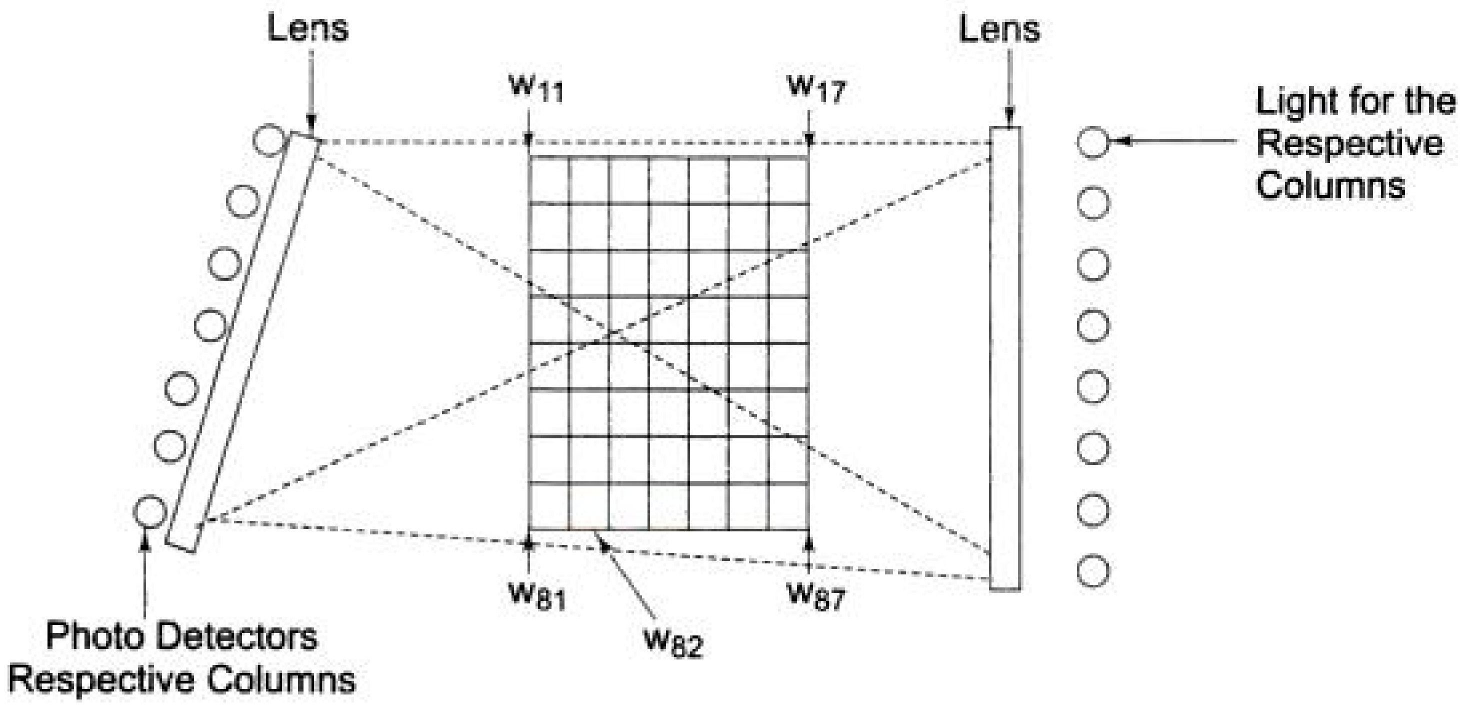

Lens
Lens
w_{11}
w_{17}
Light for the Respective Columns
w_{81}
w_{82}
w_{87}
Photo Detectors Respective Columns

Fig. 12.10 *Electro-optical Vector Matrix Multiplier*

The above shown system is capable of multiplying an 8-element input vector by a 8 × 7 matrix, which produces seven-element NET vector. There is a column of light sources that passes its rays through a lens, such that each light-illuminates a single row of weight shield. The weight shield is a photographic film in which the transmittance of each square is proportional to the weight. There is an another lens, which focuses the light from each column of the shield to a corresponding photo detector. The NET is calculated by,

$$\text{NET} = \sum_i w_{ik} x_i$$

where NET_k – NET output of neuron k.

w_{ik} – weight from neuron i to neuron k.

x_i – input vector component i.

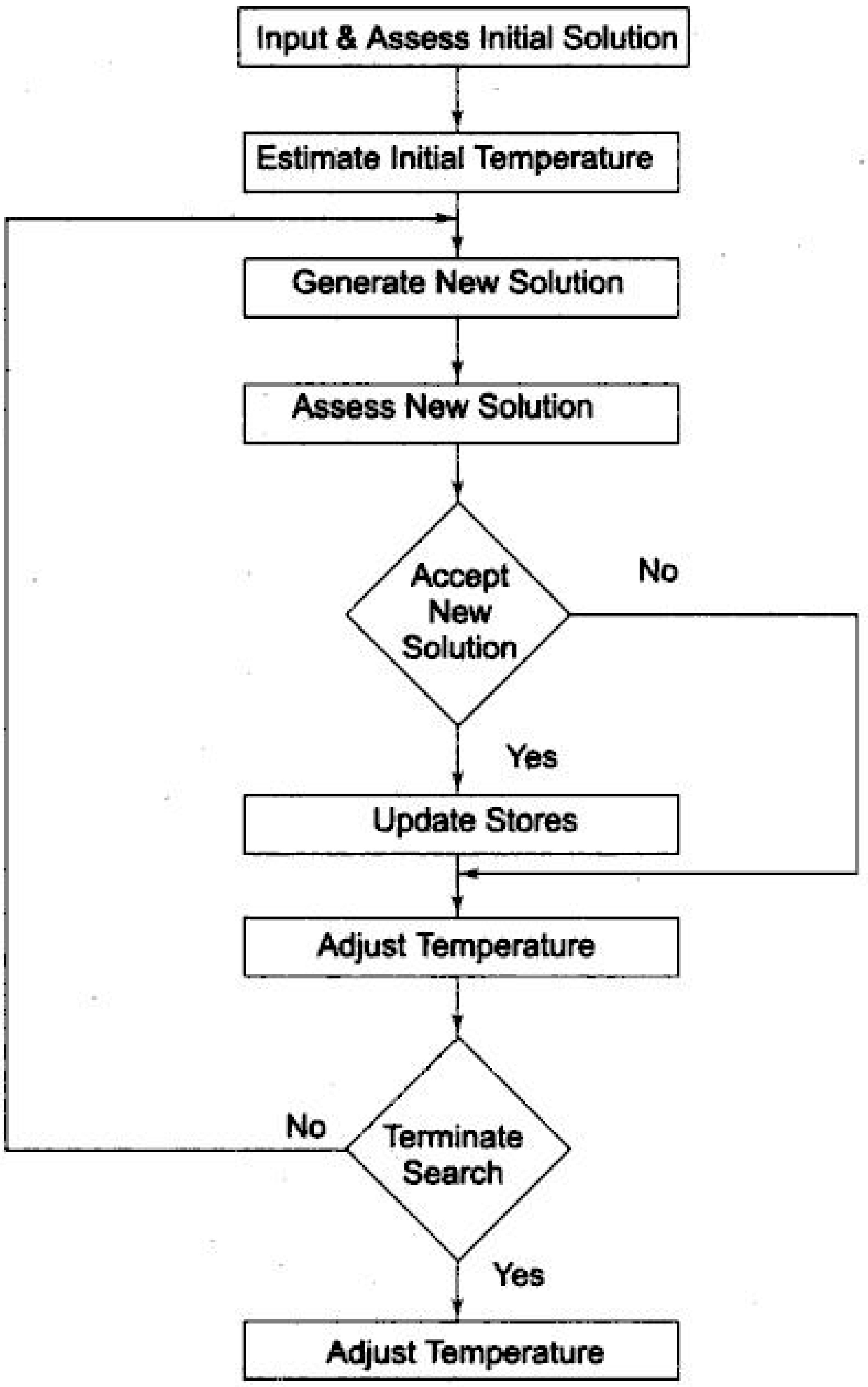


Fig. 12.12 *Structure of Simulated Annealing*

12.10.3 When to Use Simulated Annealing

Because of its highly iterative (thus computationally slow) nature, simulated annealing should be either completely avoided, combined with the gradient descent algorithm, or only used when speed is not of the essence. Yet, since simulated annealing can deal with highly dimensional minimization problems, or problems with many false minima, it should always be considered as an alternative when other methods fail.

Simulated annealing is a random-search technique, which exploits an analogy between the way in which a metal cools and freezes into minimum energy crystalline structure. It is a technique for combinatorial optimization problems, such as minimizing functions of many variables. Our model has

12.12.2 Long- and Short-term Memory in Spatio-temporal Connectionist Networks

Both conventional and spatio-temporal connectionist networks are equipped with memory in the form of connection weights, denoted by one or more matrices, depending on the number of layers of connections present in the network. These are typically updated after each training step and constitute a memory of all previous training. In the sense that this memory extends back past the current input pattern, all the way to the first training step, we shall refer to the weights as long-term memory. Once a connectionist network has been successfully trained, this long-term memory remains fixed during the operation of the network.

In addition to these weight matrices, some networks also use other trainable parameters. These may represent the connectivity scheme of the network, the types of transmission delays associated with the connections, or the initial activation values of the internal processing elements. These parameters are also part of the long-term memory. Let W denote an n-tuple representing all the adaptable parameters of a network. W includes one or more weight matrices, and may include connectivity scheme parameters, transmission delay parameters, and initial activation values depending on the type of network.

The distinguishing characteristic of STCNs is that they also include a form of short-term memory. It is this memory which allows these networks to deal with input and output patterns that vary across time and thus defines them as STCNs. Conventional connectionist networks compute the activation values of all nodes at time t based only on the input at time t. By contrast, in STCNs the activations of some nodes at time t are computed based on activations at time $t - 1$, or earlier. These activations serve as a short-term memory. We use the *state vector*, $\vec{s}(t-1)$, to represent the activations at time $t - 1$ of those nodes that are used to compute the activations of other nodes at time t, i.e. state nodes. Unlike the long-term memory, which remains static once training is completed, the short-term memory is continually recomputed with each new input vector (both during training and during operation). This is due to the fact that long-term memory is stored in connection weights (which are updated only during training) while short-term memory is represented by node activations (which are computed with each time-step even after training).

12.12.3 Output, Teaching, and Error

Finally, it is necessary to specify a representation for the response of the network to its stimuli. Like the traditional models, STCNs encode their response in the activations of a special set of units called "output units". Thus, we represent the output of a STCN by a vector, $\vec{y}(t)$. Most connectionist architectures learn by computing the difference between their response and a teacher-supplied desired (or ideal) response and adjusting their long-term memory accordingly. We denote the desired response by another vector, $\vec{\bar{y}}(t)$. The difference between the desired output vector and the actual output vector is the error vector $\vec{E}(t) = \vec{\bar{y}}(t) - \vec{y}(t)$, and the total network error, ε, is defined as one-half of the square of the magnitude of this vector:

$$\varepsilon = \sum_{t \geq 0} \left\{ \frac{1}{2} \left\| \vec{E}(t) \right\|^2 \right\}$$

The total error, ε, is a measure of the overall performance. It is this quantity that is minimized via gradient descent during training.

Compared with artificial neural networks (ANNs), support vector machine are faster, can be used with larger numbers of genes, are more interpretable, and are deterministic. They find an optimal separating hyperplane between data points of different classes in a (possibly) high dimensional space. The actual support vectors are the points that form the decision boundary between the classes. Given below in Fig. 12.17 is a simple example in 2D where a user is trying to separate data of two classes of samples (circles and squares):

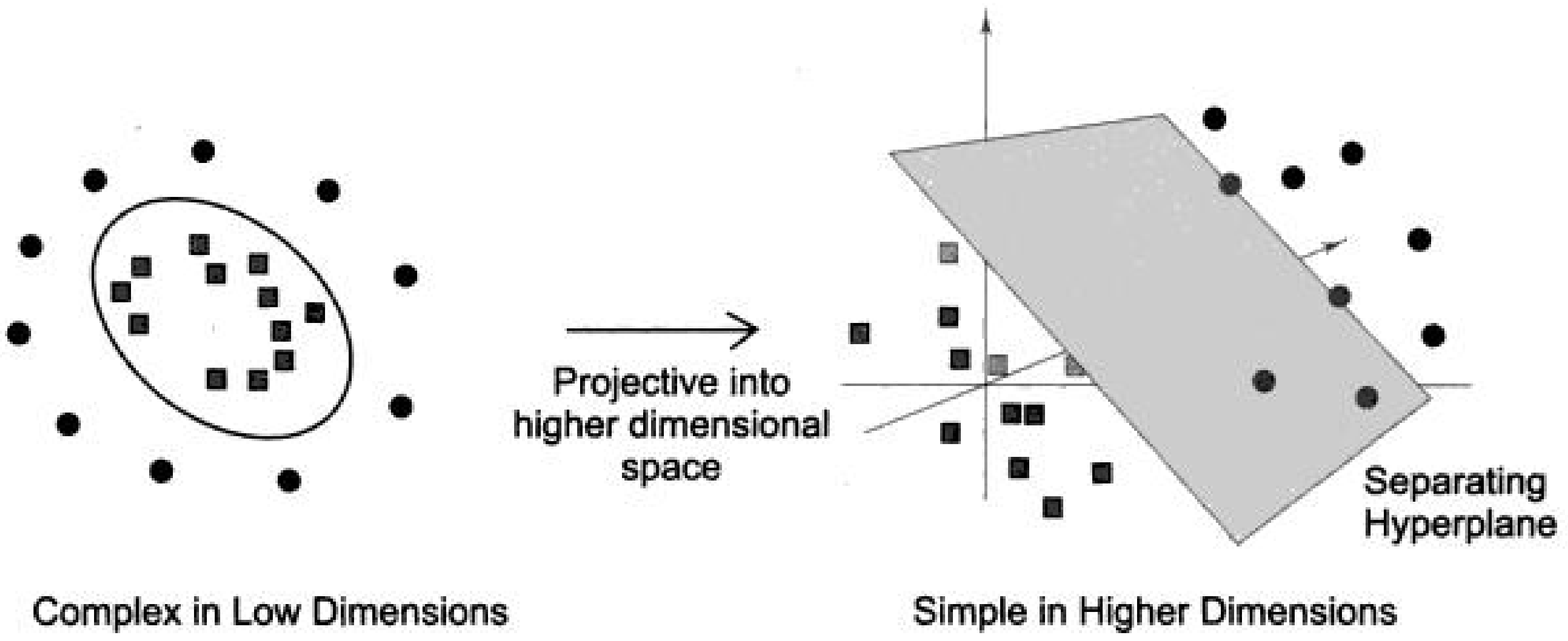

Projective into higher dimensional space
Separating Hyperplane
Complex in Low Dimensions
Simple in Higher Dimensions

Fig. 12.17 *SVM Classifier Classifying 2D Samples of Data*

One of the main advantages of SVMs is that they are maximal margin classifiers. For example, in the plots in Fig. 12.18, D is a better separator than A, B or C since it will be more likely to classify new samples that are close to the current decision boundary:

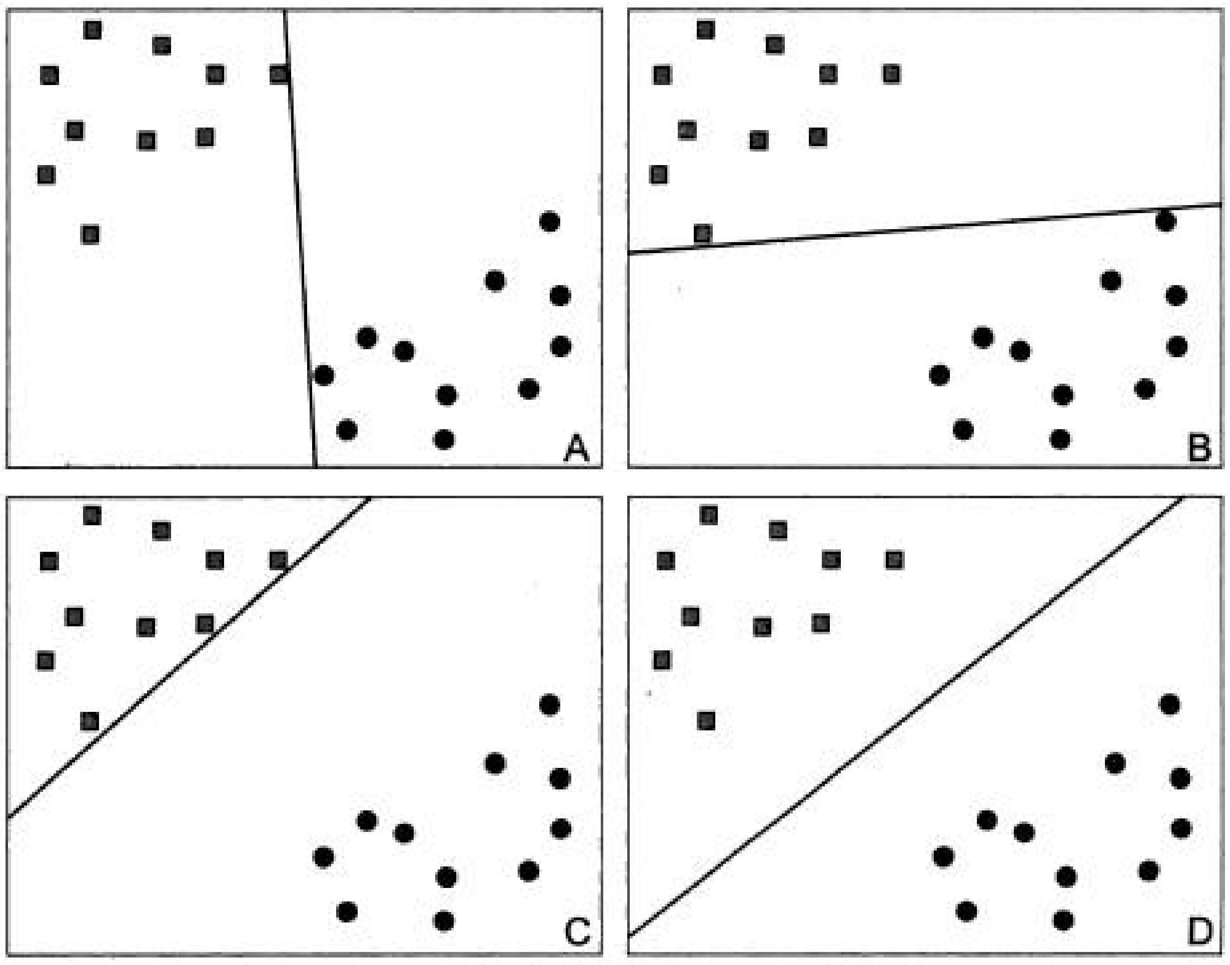


Fig. 12.18 *SVM Marginal Classifier*

12.15.1 Example of Dynamic Programming

Consider the machine replacement problem. A machine is supposed to be working effectively for N time periods. It can be operated in n deterioration levels and there is an operating cost $g(i)$ associated with each state level. We assume that,

$$g(1) \leq g(2) \leq \ldots \leq g(n)$$

The cost of replacing the machine is R and after replacement the machine is back to the best state, namely 1. During a period of operation, the state of a machine can be worse or stay unchanged. Let p_{ij} denote the probability of the state change from i to j. Apparently, $p_{ij} = 0$ if $j < i$: The problem is whether or when to replace the machine so as to minimize the operating cost. The only reasonable way to solve this problem is analyzed it backwards. At the last step, the decision is obviously "no replacement". At the step $N - 1$, the decision is, if the operating cost is greater than R, replace it, otherwise, keep it running.

Let the total cost of the optimal decision for the rest of N – k period is $C_i(k)$: we can make the optimal decision at step N – k – 1. That is the main idea of dynamic programming.

Dynamic programming offers a suite of algorithms for generating optimal control strategies. However, the overwhelming computational requirements associated with these algorithms render them inapplicable in practical situations. Due to a lack of other systematic approaches for dealing with such problems simplified problem specific analyses and heuristics have become the norm. Such analysis and heuristics often ignore much information that is important to effective decision making, leading to control policies that are far from optimal. The recent emergence of neuro-dynamic programming puts forth an exciting new possibility. New and highly promising approaches for addressing complex stochastic control problems have been developed. These approaches focus on approximating solutions that would be generated by dynamic programming, except in a computationally feasible manner. Neuro-dynamic programming can be done based on self-organized patterns.

The main idea in neuro-dynamic programming is to approximate the mapping $J^*: S \rightarrow \Re$ using an approximation architecture. An approximation architecture can be thought of as a function j_i: $S \times \Re^k \rightarrow \Re$ NDP algorithms try to find a parameter vector $r \in \Re^k$ such that the function J_i closely approximates J^*.

In general, choosing an appropriate approximation architecture is a problem dependent task. Generally, we design approximation architectures involving two stages: (1) a feature extractor and (2) a function approximator. The feature-based approximation architecture is shown in Fig. 12.22.

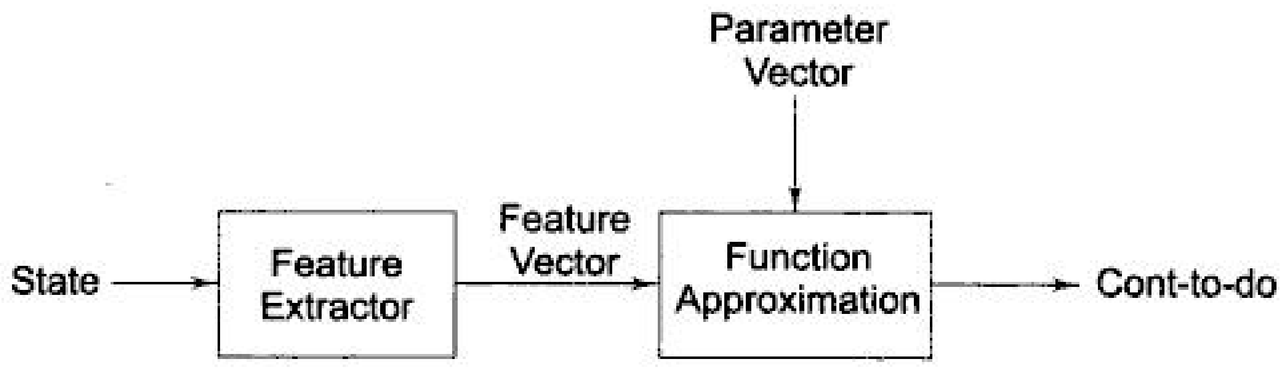


Fig. 12.22 | A Feature Based Approximation Architecture

The feature extractor uses the post-decision state y_t to compute a feature vector z_t. The components of z_t are values that we thought were natural for capturing key information concerning states of the application

13.1 Applications of Neural Networks in Arts

Traditionally, computer applications in the arts have been pretty much like computer applications in any other field: you would like the computer to create a certain sort of output, so you construct an algorithm for it to follow step by step which will result in that output. This is a fine approach when you know what you are going for and where the production process should be headed at each step, i.e. when you can precisely specify the rules involved in generating the output. Programming a computer to invert a matrix or control an assembly line drill-punch is certainly like this, and the production of a variety of types of art are amenable to this approach as well. But there are also a wide range of instances in which artists may not want to, or be able to, specify a set of rules by which our next computer artwork is to be created. Sometimes it may just be too difficult or time-intensive to list the intricacies of a particular creative process in a fashion definite enough for a computer to follow; and other times, we may have no idea how to even go about constructing an appropriate set of rules (what are the "rules" to follow in creating a Rodin sculpture?). At times like these, a new approach is needed.

The rapidly expanding field of neural networks provides a new approach to computer art applications. Rather than operate in the traditional style of preprogrammed rule-following systems, neural networks have the power to learn to produce specific types of outputs from specific inputs, based on examples they are taught. Thus, instead of having to specify how to create a certain artwork, the artist can instead teach a network examples of the desired output, and have the network generate new instances in that style. We need not specify the steps involved in creating a Rodin sculpture—we can just collect instances of the sorts of sculpture we'd like to get and use those to train a network. This sort of thing is the promise of neural network applications in the arts, stated in a very provocative and exaggerated manner. But we are beginning to realize this promise in limited domains, and the space for further applications and explorations of these techniques is wide open. In this section, we will briefly describe neural networks, some of what they can and can't do, some of what they have done already in the arts, primarily music, and some of what they can be applied to in other areas.

13.1.1 Neural Networks

The standard von Neumann style computer programming paradigm is based on the concept of a single powerful central processing unit that performs a sequence of operations on the contents of an external memory. The neural network computing paradigm, in contrast, is based largely on the notion of "brain-style computation" (Rumelhart, 1990), in which a large number of simple processing units act simultaneously on a distributed pattern of data—hence the common name, parallel distributed processing (Rumelhart and McClelland, 1986). The analogy here is to the type of computation done by the vast numbers of simple neurons in the brains of living creatures. There is a vast assortment of mathematical formulations of different neural network types, but for our purposes we can think about a simple case in which a network consists just of a set of simple processing units, connected by a set of weighted links. The currency of the network is numerical values—the units all take in numerical values passed to them on the links from other units, and they in turn produce a single numeric output value which they pass on their output links to still other units for further processing. A subset of the units will typically have a special status as input units, which take some pattern of "information" (represented as a set of numerical values) from the external world into the network for processing. Similarly, there is a set of output units, whose final processed values are taken as the end product of the network's computation. Along the path of links in between these two sets can be a collection of "hidden" units, which perform the network's

There is some debate over whether the neural network approach to art, and in particular music, is an appropriate one. Some argue that the reason computers have been so valuable in art is precisely that they've allowed us to imagine processes by which we'd like to create artworks, specify a very precise set of rules to carry out that process, and then implement those rules to fulfill our imagination.

Neural networks, they argue, have eliminated that ability, and chained us back down to only those art forms and styles that we can collect examples of. But this argument, of course, is invalid—if we can create an algorithm to generate new artistic styles in the traditional manner, then we can use those computer-generated works as further examples on which to train our networks. Thus networks need not constrain us at all, but rather add yet another tool into the artist's toolbox. And it is ultimately the products of those tools that we will judge artistically, not the tools themselves.

13.2 Applications of Neural Networks in Bioinformatics

Over the past fifty years, modern science has been trying to unravel the mystery behind "Human Intelligence" and how to simulate it in artificial conditions. Since a modern crane can lift weights with magnitudes over thousand times what a human hand can lift then, theoretically, it is possible to simulate human intelligence in machines in those similar orders. A major approach to solving this challenge of simulating intelligence has been the development of the Neural Network. The fundamental stimulus behind the notion of the neural network has been the human brain. Computational scientists and psychologists have collaborated to try to mimic the biological dynamics of the human brain on a logical and artificial paradigm, such as a computer. The basic goal was to break the logical reasoning process into a scattered network where each 'network unit' (analogous to a biological neuron in the brain) would inter-communicate and make a decision (also known as output) based on its analysis of the problem to be solved (also known as input).

With the advent of the human genome project, the area of Bioinformatics, especially protein sequencing, has become a major target for neural networks. The following section provides a view of the application of neural networks in solving the Protein Folding problem in the field of protein engineering.

13.2.1 A Bioinformatics Application for Neural Networks

Due to the advancement in genetic engineering especially the advent of the human genome project, the pressure for molecular computation and sequencing technologies have been immensely increased. Since the human genome project is aiming to map and sequence the entire human genome, it is expected that it will generate sequence data that fills a book with a million pages. In other words it is expected to produce data beyond our current abilities to sequence and interpret, given our present computing ways. There are many sequencing problems in the area of Bioinformatics for machines to solve and rationalize. A problem, many scientists believe to be the most significant problem remaining in genetic engineering is *Protein Folding*. Some refer to this problem as breaking the second half of the genetic code. Here we shall analyze the impacts of the protein folding problem and how neural networks are impacting its development.

Protein folding refers to the problem of predicting a protein's three-dimensional structure from a one-dimensional amino-acid sequence. So far, neural networks have shown a lot of promise and initial experimental success towards the protein folding problem. The protein folding problem is a key initiative in the fields of protein engineering and 'rational' drug design. Traditionally, the protein structure has been predicted using methods such as X-ray crystallography, which involves manual and experimental

rules generated by SIG* were described by 4 or 5 attributes resembling more closely the decisions made by medical experts. In order to test the rule generation algorithm SIG*, another data set in the medical domain was chosen. We used a data set with patients having a blood disease, anaemia. Here no classifications were known *apriori*. In medical literature values/ranges for all attributes can be found. Deviations of a blood value were indicators for a diagnosis of an anaemia disease. Rules were extracted with SIG* out of the classifications generated by the U-matrix method. These rules showed the same results as the rules in the medical book. But additional rules were also found and could be verified by medical experts.

The advantage of our system lies in the detection of structures in data without a previous classification. Here we got some results on environmental and industrial applications. In the later case we used the SOFM for monitoring and knowledge acquisition in chemical processes. Industrial processes are often hard to control because of the difficulty to observe or due to their non-linear dynamics. With the help of the learned SOFM combined with an UMM it is possible to distinguish different regions (classes) on the map which correspond to different good or bad process states. The actual process state appears as one mark on the map and the continuous flow of measured process states describes a path on the map, which gives the controller an insight whether the process stays in a normal region or changes to a critical region. In addition, with the help of the rule extraction mechanisms, descriptions of the different classes can be derived to be used either to judge the quality of the SOFM or to integrate them into an expert system. Using Kohonen's Feature Map to monitor a chemical process, as seen before, suggests its suitability to represent dynamical processes and give short-term predictions. In a further application, a forecasting of avalanches, especially during the winter, was made using snow, weather and snow cover data as input data to evaluate and classify the degree of danger (1–5).

A hybrid expert system was realised with the following objectives: assessment of degree of avalanche danger using daily measurements, ability of using incomplete and inconsistent data, explanation of results, better overall performance over existing tools (also in critical situations). It integrated a learned Kohonen feature map and a rule base to forecast new input data. For some classes (1,4,5) direct rules with a good performance could be generated by the rule generation algorithm SIG*. The other two classes 2 and 3 could not be distinguished from each other only by using the SIG* algorithm. Hence U-matrices were generated for both the "difficult" subclasses. Structure rules were the result of the use of SIG* for rule extraction of the sub-subclasses.

The hybrid system also integrated rules from other knowledge acquisition tools. The final prognosis of an avalanche was then achieved by a combination of both systems, the Rule Base and the Kohonen Network. Four cases could be distinguished: (a) no diagnosis was possible using the rule base, so the final diagnosis corresponded to the diagnosis of the Kohonen feature map, (b) both methods found the same diagnosis, (c) several diagnosis were detected by the expert system, so that the Kohonen Network diagnosis corresponding to one of the expert system diagnosis was used for the final diagnosis and (d) a conflict situation where different results were found by the subsystems. Also cases with unknown input values can be classified by the Kohonen Network, the unknown parameters can be filled later and then diagnosed.

In this sense the "network" completes the classification task. Other methods were also used to predict avalanche danger. The deterministic method models the physical processes of the snow cover. A statistical approach uses daily data over several years and is able to detect avalanches of the ten nearest neighboring days. Both the approaches were integrated into a deterministic statistic model achieving 50% of correct diagnosis. Other approaches, like expert systems get a performance of 60% and 70%. The performance of a hybrid expert system will be at least better than 74% (the performance of the rule based system).

constant and continuous training schedule in order to keep the most recent activity of the S&P 500 in its training data.

Another example of neural network application in the stock market involves Daiwa Securities Company. This company utilizes neural network technology to learn and recognize stock price patterns for use in price forecasting. Currently, the neural network systems used by Daiwa Securities Company has been trained to analyze 1,134 company's stock price patterns. This information is then used to forecast future stock price fluctuations for these particular companies.

In addition, neural network technology has also been applied to forecasting activities for entire financial markets and financial indexes. O'Sullivan Brothers Investments. Ltd. in Connecticut has utilized neural network software to follow over twenty different financial markets and generate reports regarding the behavior of any of the studied markets. More specifically, this particular neural network system will generate information such as the probability of a certain price occurring in the next time period, as well as the most optimal target long and short-term prices. Still another product of this particular neural network can notify the user as to what level must be reached for the overall market to experience an increase. Similarly, in the futures market, predictions are also being made with neural network systems. Neural network technology has been applied in predicting corn futures. In such applications, the study concluded that when compared to traditional forecasting models used in this area, the neural network produced between 18 and 40 percent fewer errors.

Neural networks have also been used in determining bond ratings. Traditionally, bond ratings have been chosen as a result of statistical regression analysis. By utilizing neural networks to determine a bond's rating, additional important factors affecting the potential default risk, can be accounted for. Unlike statistical regression analysis, the less obvious variables affecting default risk can be taken into account. Studies show this method of determining bond ratings has provided between 95% and 100% accuracy, while statistical regression analysis has been 85% accurate. In addition to bond ratings, bond prices can also be predicted with neural networks. Currently, G.R. Pugh & Company of New Jersey utilizes neural networks to predict bond prices of 115 public utility companies. The company relies heavily on this technique when advising clients of potential investment opportunities.

Bank loan decisions are another area in which neural networks are proving useful. Because the decision to make or deny a loan is very subjective or non-linear in nature, the use of neural networks resulted in a significant improvement in this decision-making process. By utilizing a neural network, the banker can rely on the network to "recognize certain similarities and patterns" and make a more accurate prediction with regard to potential default on a loan. Corporations today are finding ways to utilize neural network technology to improve overall business operations including the computer capabilities within the company. Computer Associates, for example, designed a neural network system called Neugents that predict the failure of their in-house computer servers. As described by Charles Wang of Computer Associates: "Neugents is unlike other "rule-based" systems which must be fed knowledge of analyze in that it derives the rules of analysis by itself. It takes all of the raw chaotic data and it says, OK, let me take a good look at what causes and what correlates. The beauty is that when you start to look at causation you can start to predict."

The ability to forecast server downtime has been advantageous to companies such as Computer Associates because such predictions make it possible for the company to fix any potential network problem prior to complete computer network failure. Unlike the area of finance, there are currently few available and reliable studies that have been completed in the area of forecasting macroeconomic variables with neural networks. However, there appears to be a growing interest in using neural network systems to forecast macroeconomic variables.

- **protection** against malicious tampering
- **interoperability** with other network management and security tools
- **comprehensiveness**, to expand the concept of intrusion detection such as blocking Java applets or Active-X controls, monitoring e-mail content, blocking specific URLs
- **event management**, such as managing and reporting event trace, updating attack database
- **active response** when an attack occurs, such as firewall or router reconfiguration
- **support** for the product

Another recent market survey of commercially available intrusion detection tools is available in Infosecurity magazine 2001. We present here examples of IDS tools, classified according to the three models: host-based, network-based and vulnerability assessment tools.

Host-based IDS Tools

Host-based IDS systems detect attacks for an individual system, using system logs and operating system audit trials. Examples of well known host-based commercial tools are, Cybercop from Network Associates (NAI) (http://www.pgp.com), Kane Security Monitor (KSM) from RSA Security (http://www.rsasecuriy.com). Tripwire (http://www.tripwire.org) is a specific tool to detect changes of administrative or user files on one server.

Network-based IDS Tools

Network-based IDS systems detect attacks by capturing and analyzing network packets, from sensors placed at various points in a network. Examples of well known network-based commercial tools are: RealSecure from Internet Security Scanner (ISS) (http://www.iss.net), Cisco Secure IDS or NetRanger from Cisco Systems (ex Wheel Group Corporation), Centrax from CyberSafe corporation, and Network Flight Recorder (NFR). A popular and freely-available Network-based IDS is Snort, a lightweight IDS (http://www.snort.org).

The main difficulty for network-based IDS is to process in real-time all packets for a large network; specific hardware solutions may be employed. Another problem is segmentation of networks by switches which involve difficulties in capturing traffic for a global network.

Vulnerability-assessment Tools

Vulnerability-assessment tools are security scanners used to detect known vulnerabilities on specific Operating System's configuration. Examples of well-known vulnerability-assessment tools are: CyberCop Scanner from PGP Security (a Network Associates Division) and SecureScan NX from Networks Vigilance (formally known as NV e-secure). A freely-available vulnerability-assessment tool is Nessus, a Linux-based vulnerability scanner (http://www.nessus.org) written by R. Deraison.

Performances for Commercial Tools

The majority of tools available today refer to the misuse detection model, meaning that administrators need to regularly update vulnerabilities database. Then, all these tools are vulnerable to new signatures of attacks. Tools are also very sensitive to false attacks, corresponding to normal network traffic. Major commercial IDS do not handle fragmentation /re-assembly of IP packets. For large networks, it would be necessary to store gigabytes of event data every day, to treat them off line.

applications. For example in the pattern recognition field, sub-topics include image processing and security applications, fingerprint recognition, signature verification, secure entry systems and intelligent alarms. It can be seen that these do not operate on the communications channel or control a network, but are added-value services to provide superior user interfaces or additional security. Some other examples of existing neural network pattern-recognition applications in telecommunications are shown in Table 13.2.

Table 13.2 | *High Level Applications*

Data Interpretation
Table Structure Interpretation & Text Recognition for Conversion of Telephone Company Tabular Drawings
Arabic Character Recognition System
Rel-Time Identification of Language from Raw Speech Waveforms
Temporal Difference Learning Applied to Continuous Speech Recognition
Keyword Search in Handwittern Documents
Face Recognition
Self-organizing Finite State Vector Quantization for Image Coding

Some specific current and past telecommunications applications for artificial neural networks are listed in Table 13.3.

Table 13.3 | *Selected Artificial Neural Network Applications in Communications*

Equalisers
Programmable VLSI Neural Network Processors for Equalization of Digital Communication Channels
Adaptive Equalization
Channel Equalization by Distribution Learning
Equalisation of Rapidly Time-Varying Channels Using an Efficient RBF Neural Network
Equalization and Fast Adaptive Signal Recovery in Very Heavy Tailed Noise
Neural Receiver Structures Based on Self-Organizing Maps in Nonlinear Multipath Channels
Network Design, Management, Routing and Control
Adaptive Routing in Very Large Communication Networks
A Distributed Reinforcement Learning Scheme for Network Routing
A Learning Model for Adaptive Network Routing
Optimal Traffic Routing Using the Self-organization Priniciple
Hopfield Optimization Techniques Applied to Routing in Computer Networks
Scheduling Problems in Radio Networks Using Hopfield Networks
New Q-Routing Approaches to Adaptive Traffic Control
Dynamic Routing in ATM Networks with Effective Bandwidth Estimation
Intelligent Capacity Evaluation/Planning with Clustering Algorithms
Neural Networks for Network Topological Design
Location Prediction in Mobile Networks

(*Contd.*)

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

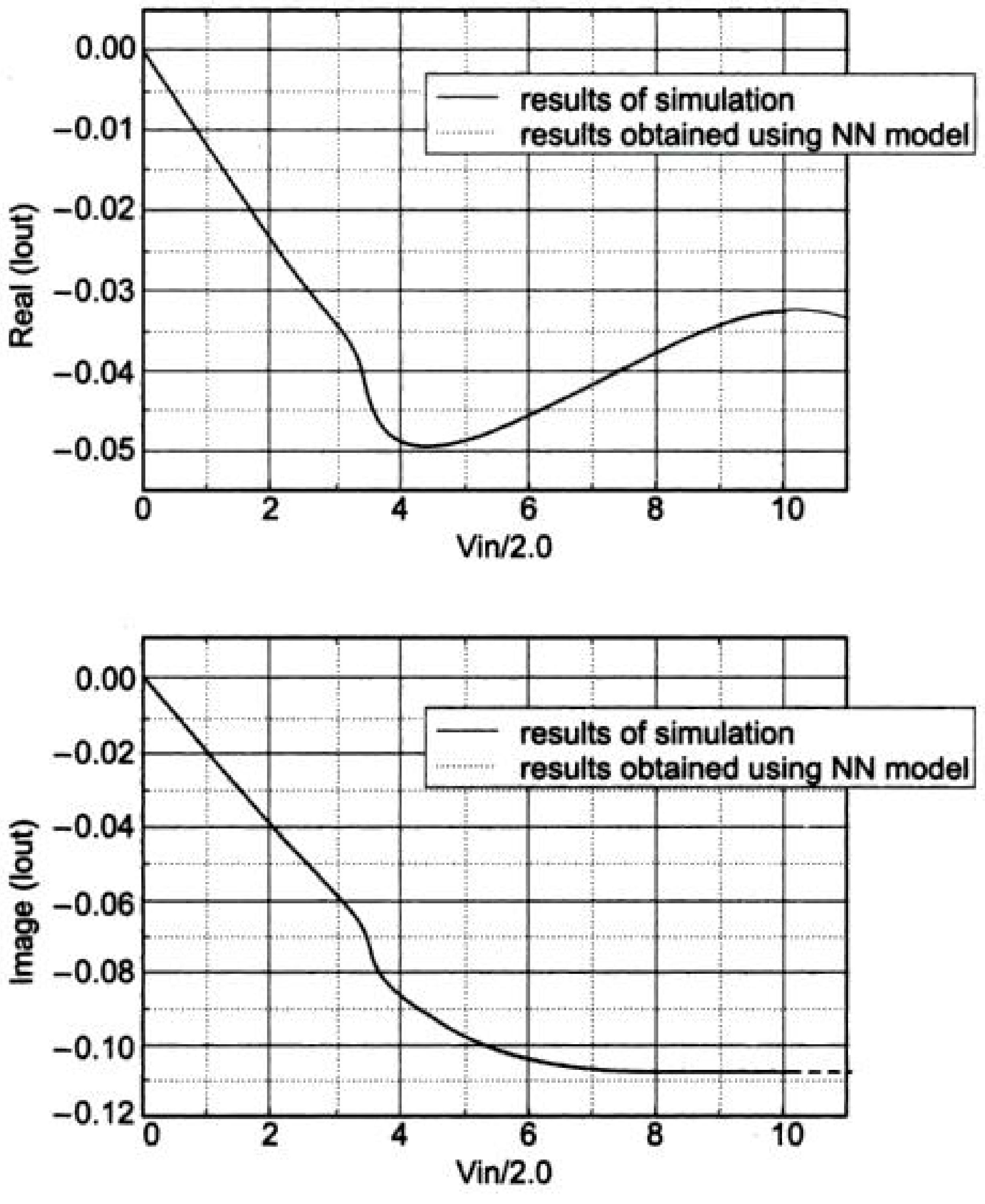


Fig. 13.8 *Circuit Simulation Results*

To verify the model efficiency, we have applied an envelope transient analysis to the amplifier circuit as well as to its NN model. We have used an input QPSK signal and have compared the spectra of the amplified output signals. The Envelope Transient simulator of Xpedion's GoldenGate™ has performed the results of the calculations. The use of the model reduces the CPU time of the simulation by 50x to more than 100x, while maintaining the accuracy of simulation results. Spectral re-growth, adjacent channel interference and other characteristics also were evaluated by using the amplifier model to within an accuracy level of 3%.

13.8.6 Modeling the Passive Elements

During the course of an RF circuit design, it is desirable to rapidly derive the s-parameters of the circuit elements as functions of frequency and the physical parameters of the structure. These functions may have a very complex nature, and may contain many minima and maxima. Polynomial approximations or spline-interpolation schemes can become very unwieldy even when the number of parameters is small, and it is highly desirable to find alternate approaches to make this problem more manageable. One such approach is to replace these elements by their equivalent circuits and to use well-developed analytical

Multilayer Perceptron

Backpropagation with momentum term and spot elimination was used to train a three layer feed forward neural network. Networks with various configurations were trained multiple times on this recognition problem using the training set to find the network that achieved the lowest mean square error on the test set. The parameters that were varied in the networks were number of hidden layer neurons, learning rate and momentum term. The lowest mean square error achieved was 0.0955 with 8 hidden neurons, a learning rate of 0.4 and a momentum term of 0.25. Classification accuracy achieved in the test set is shown in Table 13.4, where accuracy is defined as the number of accurate predictions divided by the total number of samples in the test set. The result of classification using MLP on a test environment is shown in Fig. 13.15(a).

Table 13.4 *Accuracy Derived on the Test of MLP and LVQ*

	MLP	LVQ
concave	90%	90%
convex	60%	100%
others	95%	88%

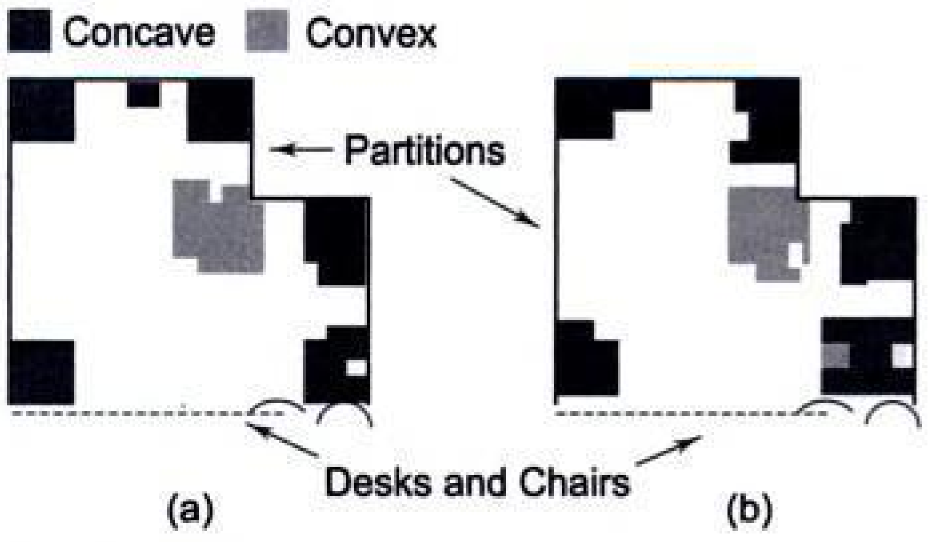


Fig. 13.15 *Classification with (a) MLP (b) LVQ*

Learning Vector Quantisation

According to Kohonen, all the different LVQ variants should yield similar accuracy. LVQ was picked for this problem because of its fast training time. Networks with different numbers of neurons, or codebook vectors, distributed among the three classes were trained for 40 epochs. A network with 30 neurons yielded good results. The accuracy achieved is shown in Table 13.4. The result of classification using LVQ on the test environment is shown in Fig. 13.15(b).

Supervised neural networks were chosen for this application because the landmark types to be recognized were predefined. The neural networks were thus trained to generate rules statistically to classify sonar range data. This is different from existing use of neural networks in landmark-based world models, where unsupervised neural networks partition environments into separate regions according to similarity of input sensory data. It can be seen from Table 13.4 that the two neural networks gave different accuracy rates for the three categories to be classified. Despite the difference in accuracies, the

Conclusion

The computational results have shown that the proposed method can compress the multi-spectral image data. The compressibility of this method and the image distortion from quantizing process depends on the number of clusters used in multi-layered perceptron neural network process. Neural network image compression technique is a lossy technique. So the reconstructed image has some distortion and noisy. These results show that the Windows-based neural network algorithm appears to give extremely good performance for all digital image compression, and may be very useful in practical implement the other type of images such as Medical (MRI or CT) and SAR images and digital picture archiving and communications systems. It should be noted that the above-mentioned PSNR and LSMSE can be varied and the selected areas can be adjusted to other types of images. In addition; the images were restored using a Windows-based neural network image compression and restoration method appear clearer and more visually pleasing despite a slight decrease in SNR when compared to the images produced by the other algorithms. But it has not high compression ratio, if it compares with wavelet-based compression techniques.

13.10.2 Application of Artificial Neural Networks for Real Time Data Compression

In today's world of global communications, people seek information almost as soon as it is made available. A classic example of this is news reporting where broadcasting companies compete with each other to deliver higher qualities of broadcast/telecast. This is now increasingly possible due to the development of higher data rates in communication as well as newer technologies for delivering us that information instantaneously. One such area of communication is the Internet where the number of users double each year. This puts increasing pressure on the communications bandwidth available. One way of mitigating such effects is to adopt a method of compression that works in real time.

Data compression techniques often involve compact representations by identifying and using structures inherent within the data itself. These structures usually have some degree of variance between themselves. An Artificial Neural Network (ANN) approach helps simplify the recognition of these complex variances and subsequently the classification of these data. Real time data compression involves the identification of the Probability Distribution Function (PDFs) or the density of each of the features for each file containing such data. By using this ANN approach for data classification, one can then encode a Huffman data compression engine. If this scheme could be employed on an Internet Service Provider (ISP) server, then at the user end a decoder (a Huffman decoder engine) could retrieve the original values. The objective of this section is to demonstrate a scheme for compressing data based on the feature content of the data present within html files, which make up the bulk of all Internet related formats. It is based on some experimental work involving Self-Organizing Maps and a Huffman encoder/decoder engine.

Existing Compression Technologies

Existing compression technologies are usually based around the concept of the inherent 'source entropy' or information content of the file as separate from the memory needed to store the original (uncompressed file). For example, a graphics file containing nothing but 'white space' may consume many megabytes of storage in an uncompressed form but, using a trivial compression scheme such as run-length encoding, the file may be compressed to a few bytes by noting the number of white pixels and their binary value.

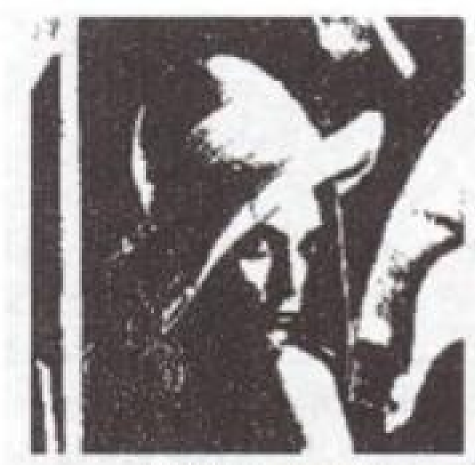
Looked at from a more general perspective, it is easy to demonstrate that the more frequently a particular piece of information occurs in a file, fewer bytes are needed to represent it. This has been recognised for a long time; for example Samuel Morse, in devising the Morse Code, ensured that letters of the alphabet that occur more frequently in written text (a, e, i, o, u) can be represented by fewer

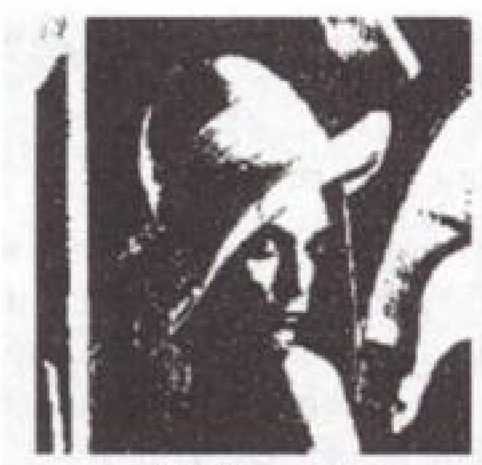
in Fig. 13.26–13.28. As with the DSM, the best compression ratio (64:20 or 3:1) was achieved with an ANN containing 20 hidden units.

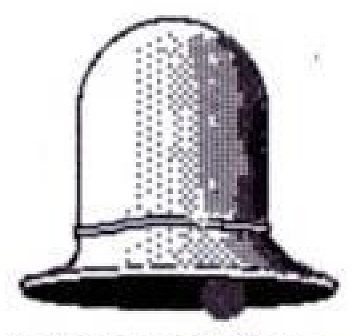
Table 13.7 | *Performance of EBP Algorithm with a Variety of Images*

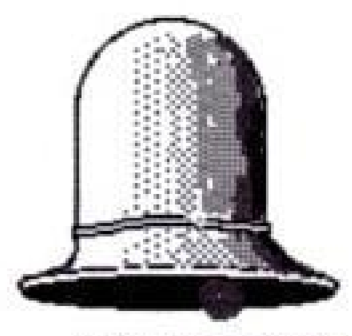
Image	*Hidden Unit*	*No. of Iterations*	*Total Training Pairs*	*Reduced Training Pairs*	*Time Required to Send Original Image (sees)*	*Time Required to send Compressed Image (sees)*
Lena	20	200	1024	404	0.3273	0.1557
Bell	20	200	288	115	0.03953	0.01938
Camera	20	200	459	210	0.02232	0.01070
Church	20	200	437	121	0.02219	0.01091

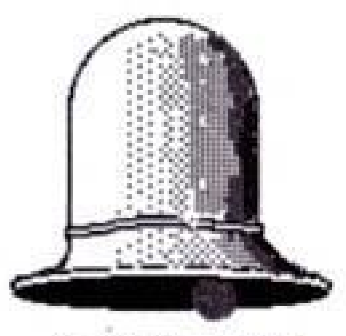
(a) Original Image


(b) Using DSM

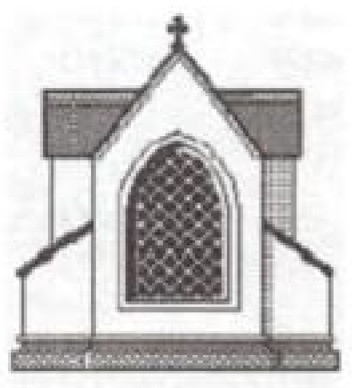

(c) Using EBP


(a) Original Figure


(b) Using DSM


(c) Using EBP

(a) Original Figure


(b) Using DSM

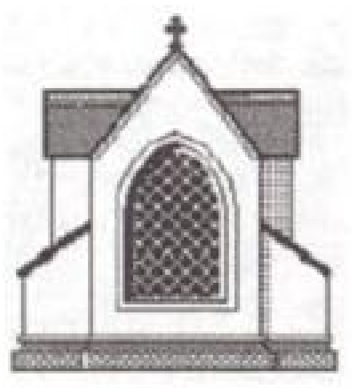

(c) Using EBP

Fig. 13.26–28 | *In All Cases (a) Original Figures (b) Using DSM (c) Using EBP*

Stage Two: Arbitration and Merging Overlapping Detections The examples in Fig. 13.31 showed that just one network couldn't eliminate all false detections. To reduce the number of false positives, we apply two networks, and use arbitration to produce the final decision. Each network is trained in a similar manner, with random initial weights, random initial non-face images, and random permutations of the order of presentation of the scenery images. The detection and false positive rates of the individual networks are quite close. However, because of different training conditions and because of self-selection of negative training examples, the networks will have different biases and will make different errors.

We used very simple arbitration strategies here. Each detection by a filter at a particular position and scale is recorded in an image pyramid. One way to combine two such pyramids is by logical ANDing. This strategy signals detection only if both networks detect a face at precisely the same scale and position. This ensures that, if a particular false detection is made by only one network, the combined output will not have that error. The disadvantage is that if an actual face is detected by only one network, it will be lost in the combination. Similar heuristics, such as logical ORing the outputs, were also tried.

Further heuristics (applied either before or after the arbitration step) can be used to improve the performance of the system. Note that in Fig. 13.31, most faces are detected at multiple nearby positions or scales, while false detections often occur at single locations. At each location in an image pyramid representing detections, the number of detections within a specified neighborhood of that location can be counted. If the number is above a threshold, then that location is classified as a face. These detections are then collapsed down to a single point, located at their centroid. When this is done before arbitration, the centroid locations rather than the actual outputs from the networks are ANDed together.

If we further assume that a position is correctly identified as a face, then all other detections, which overlap it, are likely to be errors, and can therefore be eliminated. There are relatively few cases in which this heuristic fails; however, one such case is illustrated in the left two faces in Fig. 13.31B, in which one face partially occludes another. Together, the steps of combining multiple detections and eliminating overlapping detections will be referred to as *merging detections*.

Conclusions

This algorithm can detect up to 92.9% of faces in a set of test images with an acceptable number of false positives. The system can be made more conservative by varying the arbitration heuristics or thresholds. Currently, the system does not use temporal coherence to focus attention on particular portions of the image. In motion sequences, the location of a face in one frame is a strong predictor of the location of a face in next frame. Standard tracking methods can be applied to focus the detector's attention. The system's accuracy might be improved with more positive examples for training, by using separate networks to recognize different head orientations, or by applying more sophisticated image preprocessing and normalization techniques.

13.10.6 Rotation Invariant Neural Network-based Face Detection

In this section, we present a neural network-based face detection system. Unlike similar systems, which are limited to detecting upright, frontal faces, this system detects faces at any degree of rotation in the image plane. The system employs multiple networks; a "router" network first processes each input window to determine its orientation and then uses this information to prepare the window for one or more "detector" networks. We present the training methods for both types of networks. We also perform sensitivity analysis on the networks, and present empirical results on a large test set. Finally, we present preliminary results for detecting faces rotated out of the image plane, such as profiles and semi-profiles.

Furthermore, Table 13.8 also shows that a hybrid approach is used in modeling the stock performance, while individual NN algorithms are used in other problem domains. The next characteristic of NN methodology that should be compared with the problem domains is NN's learning function. It is found that majority of applications use the sigmoid transfer function, with changeable learning parameters that are optimized in the experiments. An important trend in the applications is combining two or more NNs into a single NN system, or incorporating other artificial intelligence methods into a NN system, such as expert systems, genetic algorithms, natural language processing. The number of Kohonen's, Hopfiled's, and other algorithms is relatively small in the stock market NN applications. This could be caused by the convenience of the NN algorithms for classification rather than prediction.

NN Methodology in Relation to Data Model

It is to be noted that almost all applications of NN in stock markets are based on a different data model. In order to see if there are similarities among various data models that are used with certain NN architectures, it was necessary to observe the NN algorithms, structure and learning functions in relation to data models. Because the design of a data model for a NN is determined mostly by the choice of input and output variables, four characteristics of data model are observed: the number of input variables, the names of input variables, the number of output variables, and the names of output variables. The comparison is shown in Table 13.9.

Table 13.9 *NN Algorithms in Relation to Data Models*

		DATA MODEL			
NN Application	*NN Algorithm*	*Number of Input Variables*	*Input Variables*	*Number of Output Variables*	*Output Variables*
Predicting stock performance	Backpropagation	9	– recurring themes in president's letter to stock-holders (qualitative data)	1	stock performance (posible values:) – well – poor
Recommendation for trading	several NN + set of rules	3	– open price of S&P 500 stock index – low price of S&P 500 stock index – close price of S&P 500 stock index	1	recommendation (possible values – long – short
Classsification of stocks	Boltzmann machine	88	– 14 company financial ratios, – 14 relative ratios of current-to mean financial ratios. – 20 features of relative performance of 5 financial ratios to respective industry	1	stock return (possible values:) – positive – neutral

Contd.

characteristics. Our goal is to maintain constant torque at variable speed known as vector control of induction motor. In vector control induction motor, the concept of Field Oriented Control (FOC) has enabled to vectorize the three phase currents to two orthogonal axes – the direct axis and the quadrature axis. Varying the current in these two axes independently helps to control torque and flux separately, where speed is inversely proportional to flux. The neural network has been applied to estimate torque, flux and flux angle of the motor at any instant of time comparing the torque and speed with desired set values generates the error signal which is used to compensate the error. The overall block diagram of the neural controller of induction of the motor is shown in Fig. 13.44. The diagram shows the feedback signals from the current transformers (CT) and the potential transformers being fed to a neural networks to predict the torque, flux and flux angle.The ANN controller compares the actual values with the set values and accordingly controls the firing of the power electronics devices, which can be a thyristor or power transistor or any other suitable devices.

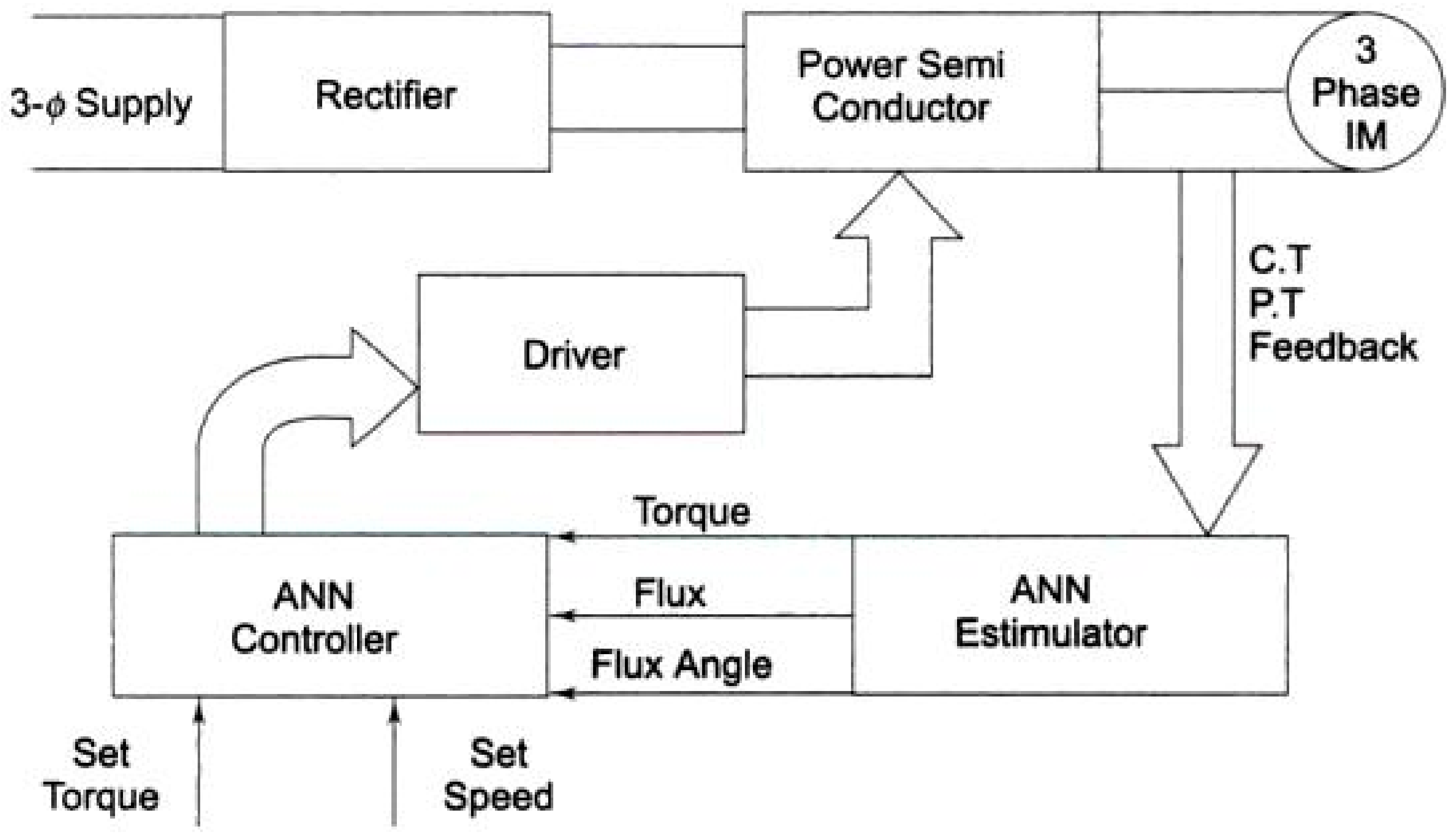


Fig. 13.44 *Neural Network Controller for Induction Motor*

The ANN estimator can be trained as shown in the Fig. 13.45. The digital signal processor estimator or any other suitable proved reference model can be used to train a neural network for the desired estimation. Similarly a reference model can be used to train a neural controller to decide the firing angle to minimize the error and operate the induction motor with constant torque at variable speed.

The above circuit is used to train the neural network. The BPN circuit is used to train the neural network for controllers and also for the neural network estimator to estimate the torque, flux and flux angle.

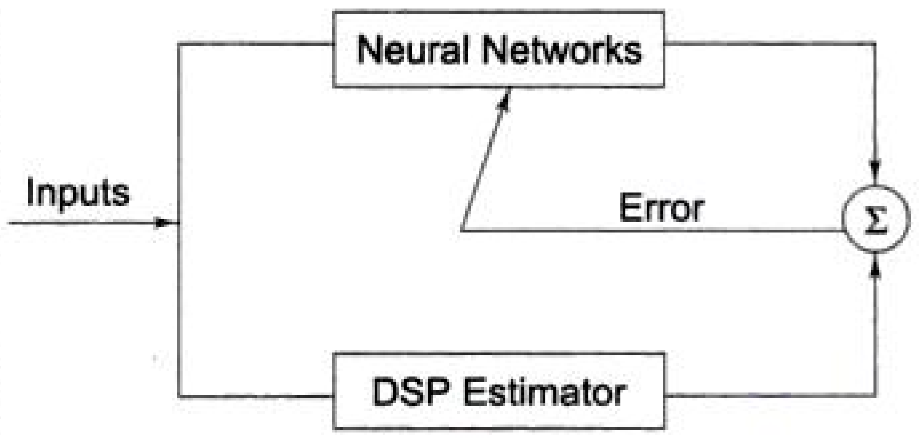


Fig. 13.45 *ANN Estimator*

require a large number of parameters to represent the character, which then results in difficulty in establishing the rules for recognition. In other words the MLPs become difficult to train. Moreover, the greater the size of the network, the greater is the computation time. This can greatly restrict their practical use. So, it is necessary to perform efficient features extraction on the one hand, and to optimize the layout of the artificial neural network on the other hand.

Data Collection

The first stage in any pattern recognition system is data collection. Before a pattern is made up of a set of measurements, these measurements need to be performed using some technical equipment and converted to numerical form. In the case of character recognition, such equipment includes video cameras and scanners.

The input data, whatever its form, is sampled at fixed intervals in time and digitalized to be presented with a preset number of bits per measurement. In any case, the data collection devices should record the objects with the highest fidelity available.

Additional noise will be disadvantageous to successful operation of the system. The data collection phase should also be designed in such a manner that the system will be robust to variations in operation of individual signal measurement devices.

Registration

In the registration of data, rudimentary model fitting is performed. The internal coordinates of the registration system are somehow fixed to the actual data acquired. Some *a priori* knowledge about the world surrounding the system is utilized in designing this stage.

This external information mainly answers questions such as: How has the data been produced? Where or when does the sensible input begin and end?

The registration process thus defines the framework in which the system operates so that it knows what to expect as valid input.

Preprocessing

The next stage of the process is data preprocessing. The goal of preprocessing data is to simplify pattern recognition problem without throwing away any important information.

Real world input data always contains some amount of noise. One of the primary reasons for preprocessing is to reduce noise and inconsistent data. Noisy data can obscure the underlying signal and cause confusion, especially if the key input variable is noisy. Preprocessing can often reduce noise and enhance the signal.

The term noise is to be understood broadly: Anything that hinders a pattern recognition system in fulfilling its commission may be regarded as noise no matter how inherent this noise is in the nature of the data.

Some desirable properties of the data may also be enhanced with preprocessing before the data is fed further in the recognition system. Preprocessing is normally accomplished by some simple filtering method on the data.

Segmentation

Two different algorithms–bottom up and top down–have been developed to insure an efficient segmentation of the characters.

The bottom-up or analytic approach, very simple but very fast, is systematically applied and is sufficient to perform the segmentation of the characters in most practical cases. It builds words out of the recognition

From the block diagram of idealized pattern recognition, by making the successive blocks to interact, the overall performance of the system can be considerably enhanced. The system of course becomes complicated, but generally there is no other way to increase the classification accuracy.

Three possible routes for the backward links are drawn in the Fig. 13.60 shown below with dashed arrows and labeled (a), (b) and (c).

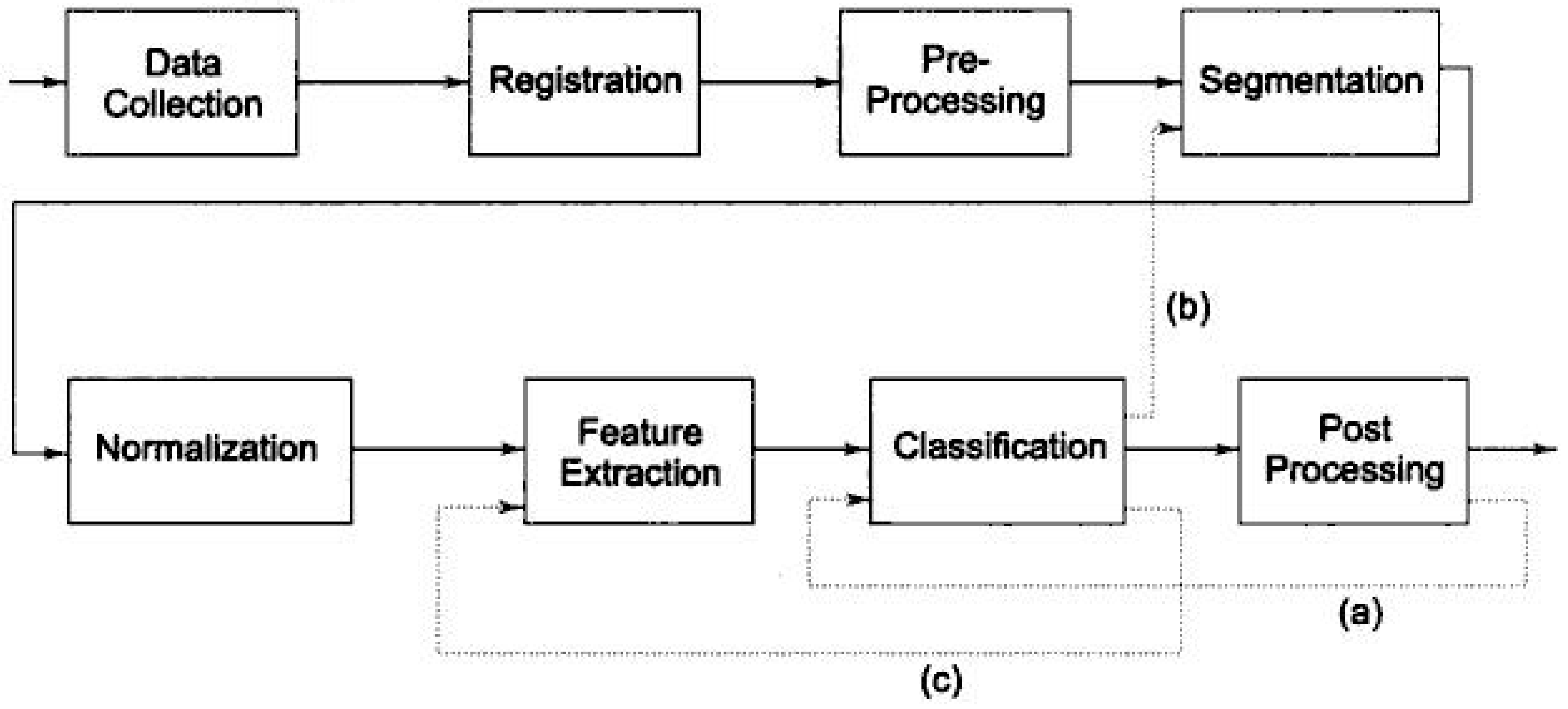


Fig. 13.60 *Block Diagram of Pattern Recognition Process*

The motivations behind these three configurations are:

(a) Information is fed back from post-processing to classification. When the post-processor detects an impossible or highly importable combination of output from the classifier, it notifies the classifier. Either the post-processor itself is able to correct the fault, or it asks the classifier for a new trial. In either case, the classifier ought to be able to reverse its behavior and to not produce similar errors in the future.

(b) The classifier revises the segmentation phase. In this case, the classifier or the postprocessor has detected one or more successive patterns that are hard to classify. There might be an indication of malformed segmentation, which should be located and corrected. This scheme can also be viewed as a segmentation algorithm probing the succeeding stages with tentative segments. It is then left for the classifier to select the most probable combination.

This view can also accommodate the possibility that segmentation is performed after classification. In this scheme, the data first flows unmodified in the first pass through the segmentation block. When classification has taken place, the data is fed back to the segmenter and the actual segmentation is performed.

(c) The correctness of the classifications is used to revise the feature extractor. This kind of operation is mostly possible only during the training phase & generally necessitates the redesign of the classifier. This kind of scheme may be called "error- corrective feature extraction".

Demonstration The proposed demonstration shown in Fig. 13.61 is an omni-scriptor recognition system that is able to classify the 10 handwritten digits, as well as 8 operators of a basic calculator: + – * / ( ) , =.

For determination of activation function based on op-amp circuit, a study on transfer characteristics of operational amplifier is essential. The voltage-transfer characteristics of the open loop high gain operational amplifier is shown in Fig. 13.64.

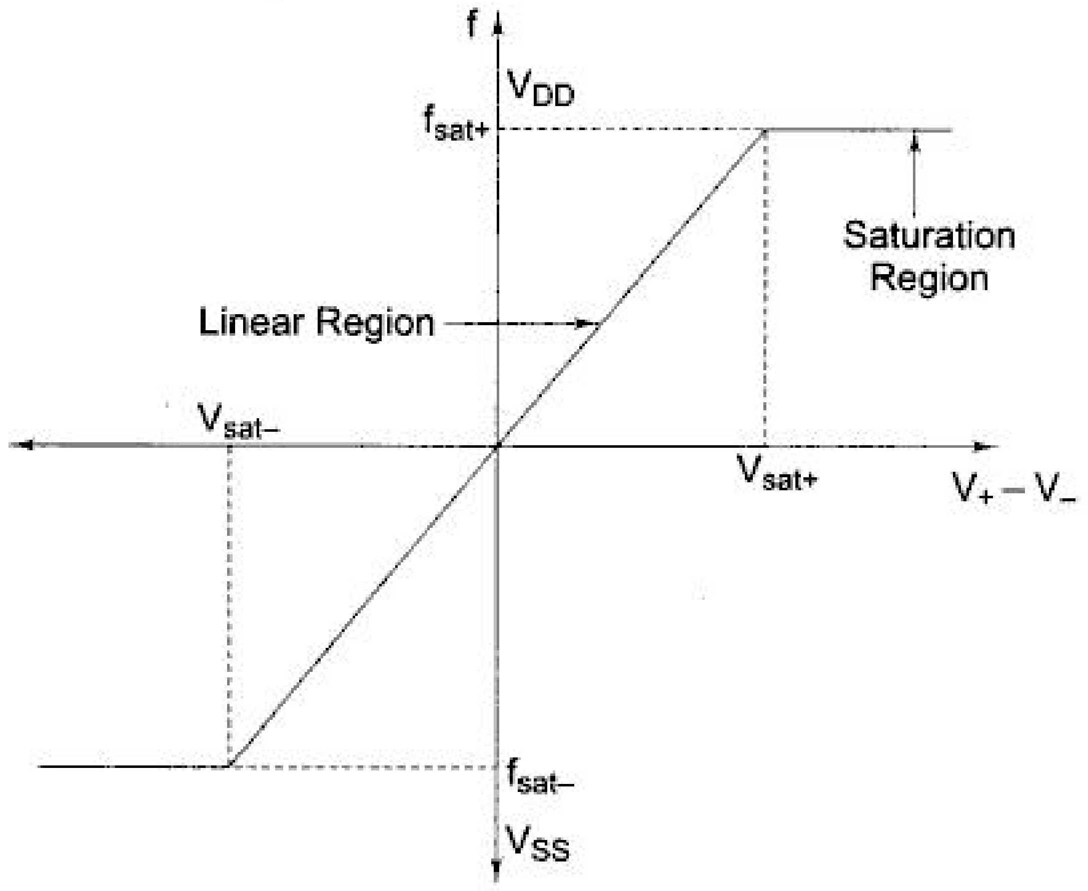


Fig. 13.64 *Transfer Characteristics of Operational Amplifier*

In the linear region, the open loop operational amplifier performs as a constant gain amplifier of high gain. In this region, the open loop gain of the operational amplifier corresponds to the slope of the steep part of the characteristics shown in Fig. 13.64 above. The output voltage 'f'of the operational amplifier ranges between f_{sat+} and f_{sat-}. This result based on the saturation of transistors within the device. f_{sat+} and f_{sat-} are of equal in magnitude but of opposite in sign. Their values are 1V below the operational amplifier supply voltage V_{DD} and 1V above V_{SS}. Due to this, the operational amplifier operates at a very narrow input voltages in the range from V_{sat+} and V_{sat-}. The differential input voltage V_+V_- remains in the range of V_{sat+} and V_{sat-}, only if the operational amplifier is in its linear region of operation. The operational amplifier operating with the closed negative feedback loop remains in the linear region. The negative feedback loop is closed when the resistance R_s connects the output with an inverting input of the operational amplifier.

Derivation of Activation Function based on Operational Amplifier Characteristics

The Kirchoff's Current Law being applied to the circuit shown in Fig. 13.62, we get,

$$-i_r = i_1 + i_2 + \ldots + i_n \tag{2}$$

which can also be given as,

$$(X_1 \_ V_-)\, G_1 + (X_2 \_ V_-)G_2 + \ldots + (X_n \_ V_-)G_n = (V_- - f)\frac{1}{R_r} \tag{3}$$

with virtual ground property, we get

$$X_1G_1 + X_2G_2 + \ldots + X_n G_n = \frac{-f}{R_s} \tag{4}$$

The o/p voltage for the linear range of operation can be given by,

may not be the case in real life situations. Although accurate prediction cannot be achieved only based on the temporal contextual information, if the output of the PNN produces the same classification result as the predictor, then a much higher confidence can be assured. The initial classification result of the PNN and the output of the predictor are then compared. If the classification labels for a block are the same for both the PNN and the predictor, then that block is classified with a high level of confidence. All such blocks form a subset, which is referred to as "pseudotruth."

On the other hand, all of the blocks for which the PNN and the predictor provide different class labels form the subset. These subsets are used for the PNN updating even though the learning mechanisms are different. The updating process of the PNN is a type of online training. The goal of updating is to reestimate the parameters of the PNN so that it can more accurately represent the distribution of the temporally changed feature space. There are basically two requirements for the updating process. First, the updating process must be stable i.e. the updated PNN must maintain good classification ability for those previously established categories. Second, the updating must be flexible to accommodate temporal changes of the data and new class generation. Since the subset contains labeled data with relatively high level of confidence, a supervised learning is used to finetune the parameters of corresponding classes. This ensures the stability of the established classes. However, as far as the subset is concerned, since class labels are unknown an unsupervised learning is used to account for feature changes and provide flexibility needed in these situations. The Gaussian mixture models are used to represent the distribution of different classes. For both supervised and unsupervised learning paradigms maximum likelihood (ML) estimation is adopted to estimate the parameters of the Gaussian components. The expectation maximization (EM) method is then used to implement the ML estimation more efficiently. After the temporal updating, the updated PNN is used to reclassify the data again and generate the final result for frame. This process is repeated whenever a new frame arrives.

14.2 Application of Knowledge-based Cascade-correlation to Vowel Recognition

Neural network algorithms are usually limited in their ability to use prior knowledge automatically. A recent algorithm, knowledge-based cascade-correlation (KBCC), extends cascade-correlation by evaluating and recruiting previously learned networks in its architecture. In this section, we describe KBCC and illustrate its performance on the problem of recognizing vowels.

Neural network algorithms rarely allow prior knowledge to be incorporated into their learning. Most start from scratch and those that do use prior knowledge, require that knowledge to have a specific form, such as having the same architecture, being a symbolic domain theory, or being given as hints. However, prior knowledge can often take the form of some existing classifier or function approximator and no algorithm is flexible enough to permit the integration of such a wide variety of knowledge.

It is clear that humans do not learn from scratch, but make extensive use of their knowledge in learning. Use of prior knowledge in learning can ease and speed learning and lead to better generalization as well as interference effects. The current difficulty in using prior knowledge is arguably the major limitation in neural network modeling of human learning and cognition. In this section, we describe and test a neural learning algorithm that implements a general mechanism of knowledge reuse.

Knowledge-based cascade-correlation (KBCC) is a fundamental extension of cascade-correlation (CC), a constructive learning algorithm that has been successfully used in many real applications and in simulations of cognitive development. CC builds its own network topology by adding new hidden units

A simple way to use RBMrate for face recognition is to train a single model on the training set, and identify a face by finding the gallery image that produces a hidden activity vector that is most similar to the one produced by the face. This is how eigen faces are used for recognition, but it does not work well because it does not take into account the fact that some variations across faces are important for recognition, while some variations are not. To correct this, we instead trained an RBM rate model on pairs of different images of the same individual, and then we used this model of pairs to decide which gallery image is best paired with the test image. To account for the fact that the model likes some individual face images more than others, we define the fit between two faces f1 and f2 as G(f1; f2)+ G(f2; f1) – G(f1; f1) – G(f2; f2) where G(v1; v2) is the goodness score of the image pair v1; v2 under the model. The goodness score is the negative free energy which is an additive function of the total input received by each hidden unit. Each hidden unit has a set of weights going to each image in the pair, and weight-sharing is not used, hence G(v1; v2) = G(v2; v1). However, to preserve symmetry, each pair of images of the same individual v1; v2 in the training set has a reversed pair v2; v1 in the set. We trained the model with 100 hidden units on 1000 image pairs (500 distinct pairs) for 2000 iterations in batches of 100, with a learning rate of 2.5×10^{-6} for the weights, a learning rate of 5×10^{-6} for the biases, and a momentum of 0.95.

One advantage of eigenfaces over correlation is that once the test image has been converted into a vector of eigenface activations, comparisons of test and gallery images can be made in the low-dimensional space of eigenface activations rather than the high-dimensional space of pixel intensities. The same applies to our face-pair network. The total input to each hidden unit from each gallery image can be precomputed and stored, while the total input from a test image it only needs to be computed once for comparisons with all gallery images.

The FERET Database

Our version of the FERET database contained 1002 frontal face images of 429 individuals taken over a period of a few years under varying lighting conditions. Of these images, 818 are used as both the gallery and the training set and the remaining 184 are divided into four, disjoint test sets:

The **Δexpression** test set contains 110 images of different individuals. These individuals all have another image in the training set that was taken with the same lighting conditions at the same time but with a different expression. The training set also includes a further 244 pairs of images that differ only in expression. The test sets are shown in Fig. 14.7.

The **Δdays** test set contains 40 images that come from 20 individuals. Each of these individuals has two images from the same session in the training set and two images taken in a session 4 days later or earlier in the test set. A further 28 individuals were photographed 4 days apart and all 112 of these images are in the training set.

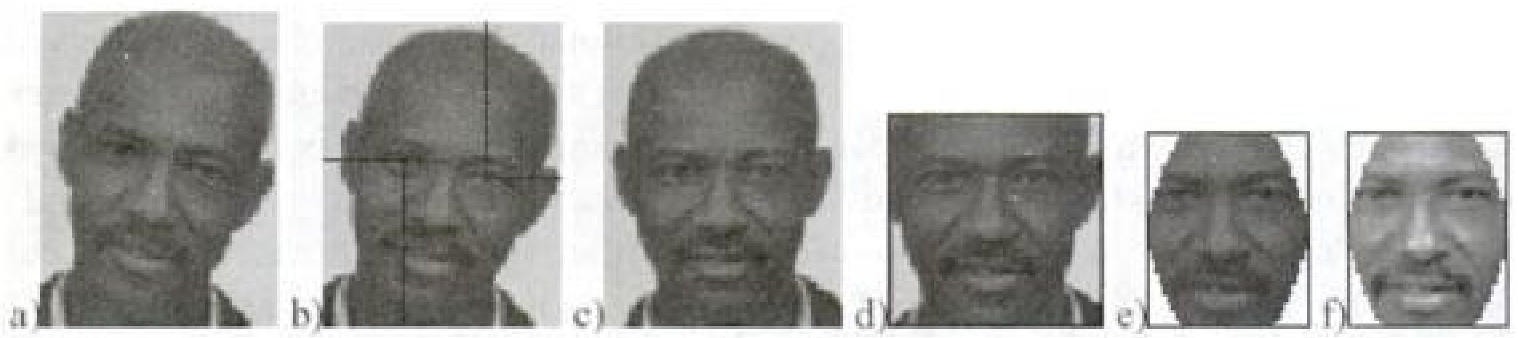


Fig. 14.7 *Test Sets Images are Normalized in five stages: (a) Original Image, (b) Locate Centers of Eyes by Hand, (c) Rotate Image, (d) Crop Image and Subsample at 56 × 56 pixels, (e) Mask out all of the background and Some of the Face, Leaving 1768 Pixels in an Oval Shape (f) Equalize the Intensity Histogram.*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Table 15.24 *Test Results for Analog Data*

Sl.No.	*Input Data*		*Simulation Time (min)*	*Efficiency (%)*
	Training Data	*Testing Data*		
1	20%	80%	4.05	73.2395
2	40%	60%	14.9	78.6471
3	60%	40%	31.92	81.5441
4	80%	20%	56.65	88.1474

15.6.10 Conclusion

Simulation of the back propagation network through this project has achieved the objective of data compression using the given database, according to a high level of accuracy based on the given dataset. Thus for a supervised input pattern, the output is obtained with a good level of accuracy.

Further development in the simulation can be obtained by increasing the level of accuracy by training more patterns and by increasing the number of hidden layers. The project is simulated for the space shuttle dataset. Using back propagation network learning and testing algorithm, about 91% of accuracy is obtained. Hence it can be concluded that backpropagation network is best suited for data compression applications.

15.7 System Identification using CMAC

Artificial neural networks offer the advantage of performance improvement through learning using parallel and distributed processing. These networks are implemented using massive connections among processing units with variable strengths, and they are attractive of applications in System Identification and control. This project uses a neural network called CMAC, which stands for Cerebellar Model Arithmetic Computer. The CMAC net is applied for system identification. This network is an alternative to backpropagated multi-layer networks in real time applications. CMAC has properties of generalization, rapid algorithmic computation based on LMS training, functional representation, output superposition all of which are discussed in detail. CMAC network has hundreds of thousands of adjustable weights that can be trained to approximate nonlinearities which are not explicitly written out or even known. CMAC can learn nonlinear relationships from a very broad category of functions. CMAC is an associative memory which has a built-in generalization. The architecture of CMAC is similar to that of cerebellum (a part of brain).

System Identification problem can be viewed as mapping between the inputs and outputs of the identification block. The identification block takes the current input and output variables of the process as its inputs and gives the estimates of the system parameters. The use of CMAC network in realizing the required mapping is explained. Its advantages in system identification of both linear and nonlinear systems are observed. The network is simulated using MATLAB software.

The simulated network is implemented to estimate the damping coefficient of a nonlinear pendulum subjected to variable driving force and viscous friction. The network is simulated for different learning rates and different number of training data points. It is shown that CMAC network can be applied for dynamic systems too. The network converges smoothly if the learning rate is moderate. The network's performance is superior when the training data points are uniformly distributed over the entire input range rather than random points.

Example of Fuzzification: Consider an air conditioning system that samples the current temperature and moisture levels to determine the optimal circulation level. In this case, the inputs consist of the current temperature and moisture level. The fuzzy system outputs the optimal air circulation level: "none", "low", or "high". The following fuzzy rules are used:

1. If the room is hot, circulate the air alot.
2. If the room is cool, do not circulate the air.
3. If the room is cool and moist, circulate the air slightly.

Finally, a knowledge engineer must determine two membership functions: one that maps temperatures to fuzzy values and one that maps moisture measurements to fuzzy values.

Inference

As inputs are received by the system, inference evaluates all IF THEN rules and determines their truth values. If a given input does not precisely correspond to an IF THEN rule, then partial matching of the input data is used to interpolate an answer. Several methods of answer interpolation exist, but lie beyond the scope of this introduction.

Example of Inference: Suppose the air conditioning system has measured temperature and moisture levels and mapped them to the fuzzy values of 0.7 and 0.1 respectively. The system now needs to infer the truth of each fuzzy rule presented above. A myriad of different inference techniques exist, but this example uses the simplest method called MAX-MIN. Basically, this method sets the fuzzy value of the THEN clause (or conclusion) to the fuzzy value of the IF clause. Thus, the method infers fuzzy values of 0.7, 0.1, and 0.1 for rules 1, 2, and 3 respectively.

Composition

Combines all fuzzy conclusions obtained by inference into a single conclusion. Different fuzzy rules might have different conclusions, so it is necessary to consider all rules. There are a number of composition methods available, but they lie beyond the scope of this introduction.

Example of Composition: Each inference conclusion about the air conditioning system suggests a different action; rule 1 suggests a "high" circulation level, rule 2 suggests turning off air circulation, and rule 3 suggests a "low" circulation level. A variety of techniques can be applied to determine the most appropriate conclusion, but this example uses MAX-MIN method of selection since it is simple. In this case, the method uses the maximum fuzzy value of the inference conclusions as the final conclusion. In particular, composition selects a fuzzy value of 0.7 since this was the highest fuzzy value associated with the inference conclusions.

Defuzzification

Converts the fuzzy value obtained from composition into a "crisp" value. This process is often complex since the resulting fuzzy set might not translate directly into a crisp value. Defuzzification is necessary, since controllers of physical systems require discrete signals. There are many defuzzification methods, but they lie beyond the scope on this introduction.

Example of Defuzzification: Recall that the air conditioning composition logic output a fuzzy value of 0.7. Unfortunately, this imprecise value is not directly useful to the air conditioner; it needs to know whether it should set its air circulation level to "none", "low", or "high". Thus, the defuzzification process must convert the fuzzy output of 0.7 into one of the air circulation levels. Many techniques exist to do this but, in this example, it is clear that a fuzzy output of 0.7 indicates that the circulation should be set to "high".

Using fuzzy operators and connectives in the mathematical operations of an artificial neuron can develop fuzzy neural networks. Irrespective of what type of fuzzy neural networks are used, it is envisaged that they offer the following features:

1. Easy to implement natural language so that the structure of knowledge base is very clear and efficient.
2. Any changes in the task and environment can be very easily taken care of by adapting the weights of the neural network, and
3. Since a fuzzy system is one kind of interpolation, drastic reduction of data and software/hardware and overheads can be achieved.

Hence more research endeavors are necessary to evaluate the efficacy of both the types of fuzzy neural networks.

16.8.2 Neuro Fuzzy Hybrids

Presently, the neuro-fuzzy approach is becoming one of the major areas of interest because it gets the benefits of neural networks as well as of fuzzy logic systems and it removes the individual disadvantages by combining them on the common features. Different architectures of neuro-fuzzy system have been investigated by number of researchers such as Lin (1994), Medsker (1995) and Jana (1996). These architectures have been applied in many applications especially in the process control.

Neural networks and fuzzy logic have some common features such as distributed representation of knowledge, model-free estimation, ability to handle data with uncertainty and imprecision etc. Fuzzy logic has tolerance for imprecision of data, while neural networks have tolerance for noisy data.

The most widely researched of all the hybrid systems at the present time, are a combination of neural networks and fuzzy logic. Fuzzy logic provides a structure within which the learning ability of neural networks is employed. In this field there are a number of possible uses. Firstly, neural networks can be used to generate the membership functions for a fuzzy system and to tune them; a schematic for this is shown in Fig. 16.4.

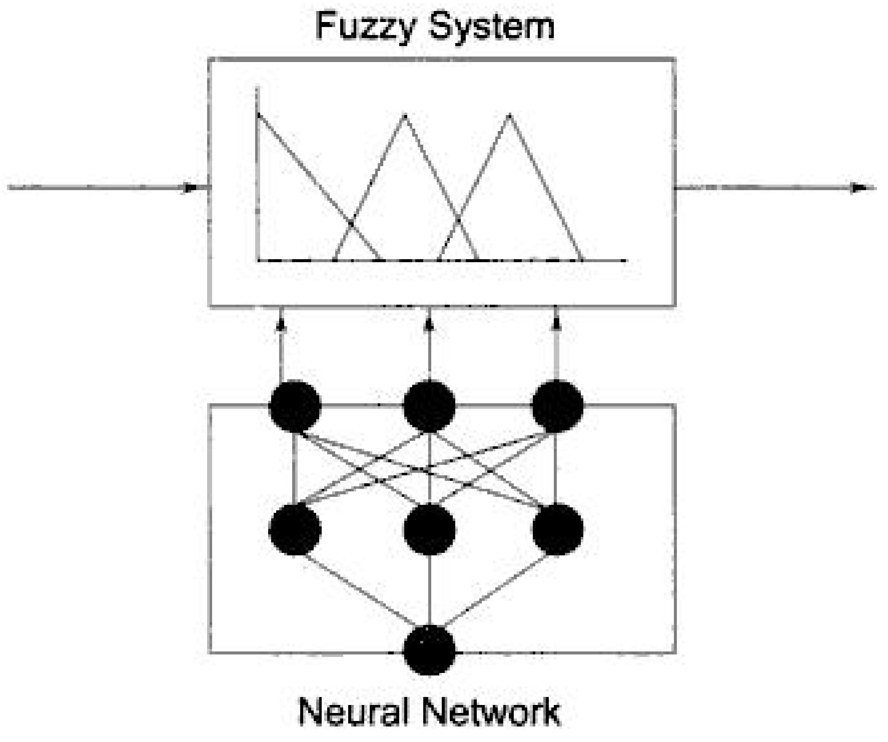


Fig. 16.4 *Neuro-fuzzy System for Tuning a Membership Function*

```
>> plot(P,T,'g–',P,Y,'g–.');
>> Y = sim(net,P);
>> plot(P,T,'g-',P,Y,'g–.');
```

A.1.1 Prior to Training

Here, the inputs are passed through the neural network prior to training the network to recognize the targets. Note the neural network does not match the desired targets very well. Refer Fig. A.1.

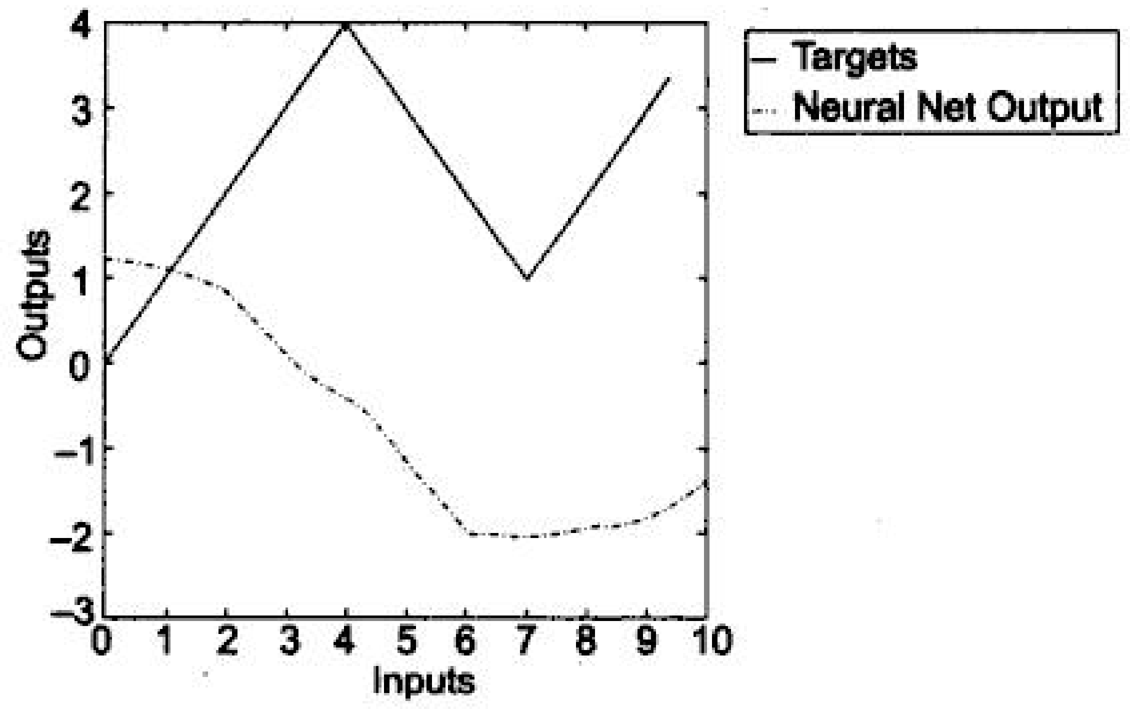


Fig. A.1 *Before Training*

A.1.2 Training Error

We pass the full set of input samples through the neural network to compute the least squared error function we will use in the back propagation of errors step. Each such pass is called an **epoch**. Refer Fig. A.2.

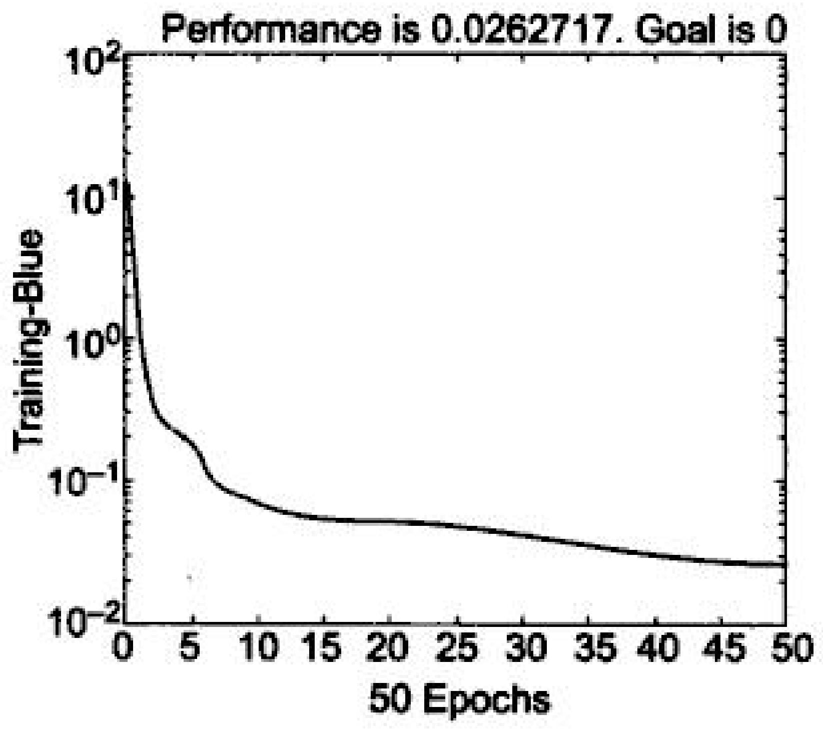


Fig. A.2 *Training Results*

A.3 Saving Neural Objects in MATLAB

Here is sample MATLAB session code that shows you how to save a trained neural object to a disk file and then load it back into a session.

```
To get started, select "MATLAB Help" from the Help menu.
% set paths
>> path(path,'/local/petersj/BioInfo/NeuralNet');
>> cd /local/petersj/BioInfo/NeuralNet
% How can we save an object we create during a MATLAB session?
>> help save
SAVE Save workspace variables to disk.
SAVE FILENAME saves all workspace variables to the binary "MAT-file" named FILENAME.mat.
The data may be retrieved with LOAD. If FILENAME has no extension, .mat is assumed.
SAVE, by itself, creates the binary "MAT-file" named 'matlab.mat'. It is an error if 'matlab.mat' is not
writable.
SAVE FILENAME X  saves only X.
SAVE FILENAME X Y Z  saves X, Y, and Z. The wildcard '*' can be used to save only those variables
that match a pattern.
ASCII Options:
SAVE ... -ASCII  uses 8-digit ASCII form instead of binary regardless
            of file extension.
SAVE ... -ASCII -DOUBLE  uses 16-digit ASCII form.
SAVE ... -ASCII -TABS  delimits with tabs.
SAVE ... -ASCII -DOUBLE -TABS  16-digit, tab delimited.
MAT Options:
SAVE ... -MAT    saves in MAT format regardless of extension.
SAVE ... -V4    saves a MAT-file that MATLAB 4 can LOAD.
SAVE ... -APPEND adds the variables to an existing file (MAT-file only).
  When using the -V4 option, variables that are incompatible with MATLAB 4 are not saved to the
MAT-file. For example, ND arrays, structs, cells, etc. cannot be saved to a MATLAB 4 MAT-file. Also,
variables with names that are longer than 19 characters cannot be saved to a MATLAB 4 MAT-file.
  Use the functional form of SAVE, such as SAVE('filename','var1','var2'), when the filename or
variable names are stored in strings.
See also LOAD, DIARY, FWRITE, FPRINTF, UISAVE, FILEFORMATS.
Let's try this out. We'll create a simple nn object example
>> x = linspace(0,2*pi,40);
>> y = sin(x)+ 5*cos(3*x);
>> net = newff([0,6.28],[1 7 7 1],{'tansig','tansig','tansig','purelin'});
% Let's see how the net does before training
>> Y = sim(net,x);
```

Computer Engineering Series

INTRODUCTION TO NEURAL NETWORKS USING MATLAB 6.0

This book is designed for the first course on Neural Networks. Integration of MATLAB throughout the book is its unique feature. Beginners to the subject will find the explanations easy to comprehend.

SALIENT FEATURES

- Application of Neural Networks to areas like Bioinformatics, Robotics, Communication, Image Processing, Pattern Recognition, Controls, Security, Healthcare, Business, etc.
- Simulated results obtained for the neural computing techniques using MATLAB.
- A chapter on Special Networks.
- Projects related to pattern classification, system identification using different networks, all with MATLAB programs.
- Over 325 illustrations.

ADVANCE PRAISE FOR THE BOOK

"This is a comprehensive book with vast coverage. Applications include a wide range of topics of modern interest. ....I strongly recommend the book for its content, coverage and insightful presentation style."
- Dr. Amit Konar, Dept. of Electronics and Telecom Engineering, Jadavpur University

"This is a unique book on Neural Networks with MATLAB."
- Dr. D. M. Vinod Kumar, Dept. of Electrical Engineering, NIT Warangal

visit us at www.tatamcgrawhill.com

ISBN-13: 978-0-07-059112-7
ISBN-10: 0-07-059112-1

9 780070 591127

Tata McGraw-Hill