

**Eastern
Economy
Edition**

Neural Networks, Fuzzy Logic and Genetic Algorithms

Synthesis and Applications



**S. Rajasekaran
G.A. Vijayalakshmi Pai**



Neural Networks, Fuzzy Logic, and Genetic Algorithms Synthesis and Applications

S. RAJASEKARAN

*Professor of Infrastructural Engineering
PSG College of Technology
Coimbatore*

G.A. VIJAYALAKSHMI PAI

*Senior Lecturer in Computer Applications
PSG College of Technology
Coimbatore*

PHI Learning Private Limited

Delhi-110092
2012



**NEURAL NETWORKS, FUZZY LOGIC,
AND GENETIC ALGORITHMS: Synthesis
and Applications**

**Neural Networks, Fuzzy Logic,
and Genetic Algorithms
Synthesis and Applications**

S. RAJASEKARAN

Professor of Infrastructural Engineering

PSG College of Technology

Coimbatore

G.A. VIJAYALAKSHMI PAI

Senior Lecturer in Computer Applications

PSG College of Technology

Coimbatore

Delhi-110092

2012

.

Copyright

NEURAL NETWORKS, FUZZY LOGIC, AND GENETIC ALGORITHMS:

Synthesis and Applications (with CD-ROM)

S. Rajasekaran and G.A. Vijayalakshmi Pai

© 2003 by PHI Learning Private Limited, New Delhi. All rights reserved. No part of this book may be reproduced in any form, by mimeograph or any other means, without permission in writing from the publisher.

ISBN-978-81-203-2186-1

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

**Fifteenth
Printing
July, 2011**

.....

Published by Asoke K. Ghosh, PHI Learning Private Limited, Rimjhim House, 111, Patparganj Industrial Estate, Delhi-110092 and Printed by Rajkamal Electric Press, Plot No. 2, Phase IV, HSIDC, Kundli-131028, Sonepat, Haryana.

To the major candles in my life

My Parents

Dr. R. Sundaramoorthy and S. Indumathi

—S. Rajasekaran

To the memory of

My Father

Prof. G.A. Krishna Pai

—G.A. Vijayalakshmi Pai

Table of Contents

Preface

ORGANIZATION

1. Introduction to Artificial Intelligence Systems

1.1 Neural Networks

1.2 Fuzzy Logic

1.3 Genetic Algorithms

1.4 Structure of This Book

SUMMARY

REFERENCES

Part 1: Neural Networks

2. Fundamentals of Neural Networks

2.1 Basic Concepts of Neural Networks

2.2 Human Brain

2.3 Model of an Artificial Neuron

2.4 Neural Network Architectures

2.4.1 Single Layer Feedforward Network

2.4.2 Multilayer Feedforward Network

2.4.3 Recurrent Networks

2.5 Characteristics of Neural Networks

2.6 Learning Methods

2.7 Taxonomy of Neural Network architectures

2.8 HISTORY OF NEURAL NETWORK RESEARCH

2.9 Early Neural Network Architectures

2.9.1 Rosenblatt's Perceptron

XOR Problem

Algorithm 2.1

2.9.2 ADALINE Network

2.9.3 MADALINE Network

2.10 Some Application Domains

SUMMARY

PROGRAMMING ASSIGNMENT

SUGGESTED FURTHER READING

REFERENCES

3. Backpropagation Networks

3.1 ARCHITECTURE OF A BACKPROPAGATION NETWORK

3.1.1 The Perceptron Model

3.1.2 The Solution

3.1.3 Single Layer Artificial Neural Network

3.1.4 Model for Multilayer Perceptron

3.2 BACKPROPAGATION LEARNING

3.2.1 Input Layer Computation

3.2.2 Hidden Layer Computation

3.2.3 Output Layer Computation

3.2.4 Calculation of Error

3.2.5 Training of Neural Network

3.2.6 Method of Steepest Descent

3.2.7 Effect of Learning Rate ‘ η ’

3.2.8 Adding a Momentum Term

3.2.9 Backpropagation Algorithm

Algorithm 3.1 (Backpropagation Learning Algorithm).

3.3 ILLUSTRATION

3.4 APPLICATIONS

3.4.1 Design of Journal Bearing

3.4.2 Classification of Soil

3.4.3 Hot Extrusion of Steel

3.5 EFFECT OF TUNING PARAMETERS OF THE

BACKPROPAGATION NEURAL NETWORK

3.6 SELECTION OF VARIOUS PARAMETERS IN BPN

3.6.1 Number of Hidden Nodes

3.6.2 Momentum Coefficient α

3.6.3 Sigmoidal Gain λ

3.6.4 Local Minima

3.6.5 Learning Coefficient η

3.7 VARIATIONS OF STANDARD BACKPROPAGATION

ALGORITHM

3.7.1 Decremental Iteration Procedure

3.7.2 Adaptive Backpropagation (Accelerated Learning)

3.7.3 Genetic Algorithm Based Backpropagation

3.7.4 Quick Prop Training

3.7.5 Augmented BP Networks

3.7.6 Sequential Learning Approach for Single Hidden Layer

Neural Networks

3.8 RESEARCH DIRECTIONS

3.8.1 New Topologies

3.8.2 Better Learning Algorithms

3.8.3 Better Training Strategies

3.8.4 Hardware Implementation

3.8.5 Conscious Networks

SUMMARY

PROGRAMMING ASSIGNMENT

REFERENCES

4. Associative Memory

4.1 AutoCorrelators

4.2 HeteroCorrelators: Kosko's Discrete BAM

4.2.1 Addition and Deletion of Pattern Pairs

4.2.2 Energy Function for BAM

4.3 WANG ET AL.'S MULTIPLE TRAINING ENCODING

STRATEGY

Algorithm 4.1 (Wang et al.'s Multiple Training Encoding Strategy).

4.4 EXPONENTIAL BAM

4.4.1 Evolution Equations

4.5 Associative Memory for real-coded pattern pairs

4.5.1 Input Normalization

4.5.2 Evolution Equations

Algorithm 4.2 (Simplified Bi-directional Associative Memory).

4.6 Applications

4.6.1 Recognition of Characters

4.6.2 Fabric Defect Identification

4.7 RECENT TRENDS

SUMMARY

PROGRAMMING ASSIGNMENT

REFERENCES

5. Adaptive Resonance Theory

5.1 INTRODUCTION

5.1.1 Cluster Structure

5.1.2 Vector Quantization

FOR THRESHOLD DISTANCE OF 2

FOR THRESHOLD DISTANCE OF 4.5

5.1.3 Classical ART Networks

5.1.4 Simplified ART Architecture

5.2 ART1

5.2.1 Architecture of ART1

5.2.2 Special Features of ART1 Models

5.2.3 ART1 Algorithm

Algorithm 5.1 (Art1 Algorithm)

5.2.4 Illustration

5.3 ART2

5.3.1 Architecture of ART2

5.3.2 ART2 Algorithm

Algorithm 5.2 (ART2 Algorithm)

5.3.3 Illustration

5.4 APPLICATIONS

5.4.1 Character Recognition Using ART1

5.4.2 Classification of Soil (Rajasekaran et al., 2001)

5.4.3 Prediction of Load from Yield Patterns of Elastic-Plastic

Clamped Square Plate

Output of the Example 5.4

5.4.4 Chinese Character Recognition—Some Remarks

5.5 Sensitiveness of Ordering of Data

SUMMARY

PROGRAMMING ASSIGNMENT

SUGGESTED FURTHER READING

REFERENCES

Part 2: FUZZY LOGIC

6. Fuzzy Set Theory

6.1 FUZZY VERSUS CRISP

6.2 CRISP SETS

6.2.1 Operations on Crisp Sets

6.2.2 Properties of Crisp Sets

6.2.3 Partition and Covering

6.3 FUZZY SETS

6.3.1 Membership Function

6.3.2 Basic Fuzzy Set Operations

6.3.3 Properties of Fuzzy Sets

6.4 CRISP RELATIONS

6.4.1 Cartesian Product

6.4.2 Other Crisp Relations

6.4.3 Operations on Relations

6.5 FUZZY RELATIONS

6.5.1 Fuzzy Cartesian Product

6.5.2 Operations on Fuzzy Relations

SUMMARY

PROGRAMMING ASSIGNMENT

SUGGESTED FURTHER READING

REFERENCE

7. Fuzzy Systems

7.1 CRISP LOGIC

7.1.1 Laws of Propositional Logic

7.1.2 Inference in Propositional Logic

7.2 PREDICATE LOGIC

7.2.1 Interpretations of Predicate Logic Formula

7.2.2 Inference in Predicate Logic

7.3 Fuzzy Logic

7.3.1 Fuzzy Quantifiers

7.3.2 Fuzzy Inference

7.4 FUZZY RULE BASED SYSTEM

7.5 Defuzzification

7.6 Applications

7.6.1 Greg Viot's Fuzzy Cruise Controller

7.6.2 Air Conditioner Controller

SUMMARY

PROGRAMMING ASSIGNMENT

SUGGESTED FURTHER READING

REFERENCES

Part 3: GENETIC ALGORITHMS

8. Fundamentals of Genetic Algorithms

8.1 GENETIC ALGORITHMS: HISTORY

8.2 BASIC CONCEPTS

8.2.1 Biological Background

8.3 CREATION OF OFFSPRINGS

8.3.1 Search Space

8.4 WORKING PRINCIPLE

8.5 ENCODING

8.5.1 Binary Encoding

8.5.2 Octal Encoding (0 to 7)

8.5.3 Hexadecimal Encoding (0123456789ABCDEF).

8.5.4 Permutation Encoding

8.5.5 Value Encoding

8.5.6 Tree Encoding

8.6 FITNESS FUNCTION

8.7 REPRODUCTION

8.7.1 Roulette-wheel Selection

8.7.2 Boltzmann Selection

8.7.3 Tournament Selection

8.7.4 Rank Selection

8.7.5 Steady-state Selection

8.7.6 Elitism

8.7.7 Generation Gap and Steady-state Replacement

SUMMARY

PROGRAMMING ASSIGNMENT

REFERENCES

9. Genetic Modelling

9.1 INHERITANCE OPERATORS

9.2 CROSS OVER

9.2.1 Single-site Cross Over

9.2.2 Two-point Cross Over

9.2.3 Multi-point Cross Over

9.2.4 Uniform Cross Over

9.2.5 Matrix Cross Over (Two-dimensional Cross Over)

9.2.6 Cross Over Rate

9.3 INVERSION AND DELETION

9.3.1 Inversion

9.3.2 Deletion and Duplication

9.3.3 Deletion and Regeneration

9.3.4 Segregation

9.3.5 Cross Over and Inversion

9.4 MUTATION OPERATOR

9.4.1 Mutation

9.4.2 Mutation Rate Pm

9.5 BIT-WISE OPERATORS

9.5.1 One's Complement Operator

9.5.2 Logical Bit-wise Operators

9.5.3 Shift Operators

9.6 BIT-WISE OPERATORS USED IN GA

9.7 GENERATIONAL CYCLE

9.8 CONVERGENCE OF GENETIC ALGORITHM

9.9 APPLICATIONS

9.9.1 Composite Laminates

9.9.2 Constrained Optimization

9.10 MULTI-LEVEL OPTIMIZATION

9.11 REAL LIFE PROBLEM

9.12 DIFFERENCES AND SIMILARITIES BETWEEN GA AND

OTHER TRADITIONAL METHODS

9.13 ADVANCES IN GA

SUMMARY

PROGRAMMING ASSIGNMENT

SUGGESTED FURTHER READING

SOME USEFUL WEBSITES

REFERENCES

Part 4: HYBRID SYSTEMS

10. Integration of Neural Networks, Fuzzy Logic, and Genetic Algorithms

10.1 HYBRID SYSTEMS

10.1.1 Sequential Hybrid Systems

10.1.2 Auxiliary Hybrid Systems

10.1.3 Embedded Hybrid Systems

10.2 NEURAL NETWORKS, FUZZY LOGIC, AND GENETIC

10.2.1 Neuro-Fuzzy Hybrids

10.2.2 Neuro-Genetic Hybrids

10.2.3 Fuzzy-Genetic Hybrids

10.3 PREVIEW OF THE HYBRID SYSTEMS TO BE

DISCUSSED

10.3.1 Genetic Algorithm based Backpropagation Network

10.3.2 Fuzzy-Backpropagation Network

10.3.3 Simplified Fuzzy ARTMAP

10.3.4 Fuzzy Associative Memories

10.3.5 Fuzzy Logic Controlled Genetic Algorithms

SUMMARY

REFERENCES

11. Genetic Algorithm Based Backpropagation Networks

11.1 GA BASED WEIGHT DETERMINATION

11.1.1 Coding

11.1.2 Weight Extraction

11.1.3 Fitness Function

11.1.4 Reproduction

11.1.5 Convergence

11.2 APPLICATIONS

11.2.1 K-factor Determination in Columns

11.2.2 Electrical Load Forecasting

SUMMARY

PROGRAMMING ASSIGNMENT

SUGGESTED FURTHER READING

REFERENCES

12. Fuzzy Backpropagation Networks

12.1 LR-TYPE FUZZY NUMBERS

12.1.1 Operations on LR-type Fuzzy Numbers

12.2 FUZZY NEURON

12.3 FUZZY BP ARCHITECTURE

12.4 LEARNING IN FUZZY BP

12.5 INFERENCE BY FUZZY BP

Algorithm 12.2

12.6 APPLICATIONS

12.6.1 Knowledge Base Evaluation

12.6.2 Earthquake Damage Evaluation

SUMMARY

PROGRAMMING ASSIGNMENT

REFERENCES

13. Simplified Fuzzy ARTMAP

13.1 FUZZY ARTMAP: A BRIEF INTRODUCTION

13.2 SIMPLIFIED FUZZY ARTMAP

13.2.1 Input Normalization

13.2.2 Output Node Activation

13.3 WORKING OF SIMPLIFIED FUZZY ARTMAP

13.4 Application: Image Recognition

13.4.1 Feature Extraction—Moment Based Invariants

13.4.2 Computation of Invariants

13.4.3 Structure of the Simplified Fuzzy ARTMAP based

13.4.4 Experimental Study

13.5 RECENT TRENDS

SUMMARY

PROGRAMMING ASSIGNMENT

REFERENCES

14. Fuzzy Associative Memories

14.1 FAM—AN INTRODUCTION

14.2 SINGLE ASSOCIATION FAM

14.2.1 Graphical Method of Inference

14.2.2 Correlation Matrix Encoding

14.3 Fuzzy Hebb FAMs

14.4 FAM INVOLVING A RULE BASE

14.5 FAM RULES WITH MULTIPLE

ANTECEDENTS/CONSEQUENTS

14.5.1 Decomposition Rules

14.6 APPLICATIONS

14.6.1 Balancing an Inverted Pendulum

14.6.2 Fuzzy Truck Backer-upper System

SUMMARY

PROGRAMMING ASSIGNMENT

SUGGESTED FURTHER READING

REFERENCES

15. Fuzzy Logic Controlled Genetic Algorithms

15.1 SOFT COMPUTING TOOLS

15.1.1 Fuzzy Logic as a Soft Computing Tool

15.1.2 Genetic Algorithm as a Soft Computing Tool

15.2 PROBLEM DESCRIPTION OF OPTIMUM DESIGN

15.3 FUZZY CONSTRAINTS

15.4 ILLUSTRATIONS

15.4.1 Optimization of the Weight of A Beam

15.4.2 Optimal Mix Design for High Performance Concrete

15.5 GA IN FUZZY LOGIC CONTROLLER DESIGN

15.6 FUZZY LOGIC CONTROLLER

15.6.1 Components of Fuzzy Logic Controller (FLC)

15.6.2 Fuzzy IF-THEN Rules

15.7 FLC-GA BASED STRUCTURAL OPTIMIZATION

15.8 APPLICATIONS

15.8.1 Optimum Truss

15.8.2 112 Bar Dome Space Truss

SUMMARY

PROGRAMMING ASSIGNMENT

SUGGESTED FURTHER READING

REFERENCES

Index

Preface

Soft Computing refers to a consortium of computational methodologies.

Some of its principal components include *Fuzzy Logic* (FL), *Neural Networks* (NN), and *Genetic Algorithms* (GA), all having their roots in Artificial Intelligence (AI).

In today's highly integrated world, when solutions to problems are cross-disciplinary in nature, soft computing promises to become a powerful means for obtaining solutions to problems quickly, yet accurately and acceptably.

Also, a combination of one or more of the methodologies mentioned—termed *hybrid systems*—has resulted in the emergence of a new class of systems such as neuro-fuzzy, fuzzy-genetic, and neuro-genetic systems. Their healthy integration has resulted in extending the capabilities of the technologies to more effective and efficient problem-solving methodologies used in the design of intelligent systems.

Considering the plethora of findings and developments that have taken place during the past few years, it would be a herculean task to present before the reader the entire gamut of information in the field of intelligent systems.

It was therefore ours objective to keep the presentation ‘narrow and intensive’ rather than ‘wide and extensive’. This approach is meant to lead a motivated novice slowly but surely in a chartered area rather than allowing him/her to feel lost in the labyrinth of voluminous information. Our endeavour therefore has been to put emphasis on learning the design, implementation, and application of soft computing methodologies through a *selective* set of systems, thereby conveying the tricks of the trade to the reader.

In fact the purpose is considered served, if this book could kindle amongst the readers not just an understanding of the subject but a sustaining interest and a desire to contribute. It therefore discusses every architecture and concept in detail with applications and examples to illustrate the same.

Algorithms have been presented in pseudo-code and wherever possible implementation details have been elaborately presented. The companion CD-ROM contains several programs that facilitate learning and reinforcing the textual concepts. Numerous simulations and examples are also presented to give students a hands-on experience in problem solving.

ORGANIZATION

Chapter 1

Introduction to Artificial Intelligence

Systems

‘*Artificial Intelligence* (AI) is an area of computer science concerned with *designing intelligent computer systems*’ that is, systems that exhibit the characteristics we associate with intelligence in human behaviour’ (Avron Barr and Feigenbaum, 1981). ‘AI is a branch of computer science that is concerned with the automation of intelligent behaviour’ (Luger and Stubblefield, 1993).

However, the term *intelligence* is not very well defined and therefore has been less understood. Consequently, tasks associated with intelligence such as learning, intuition, creativity, and inference all seem to have been partially understood.

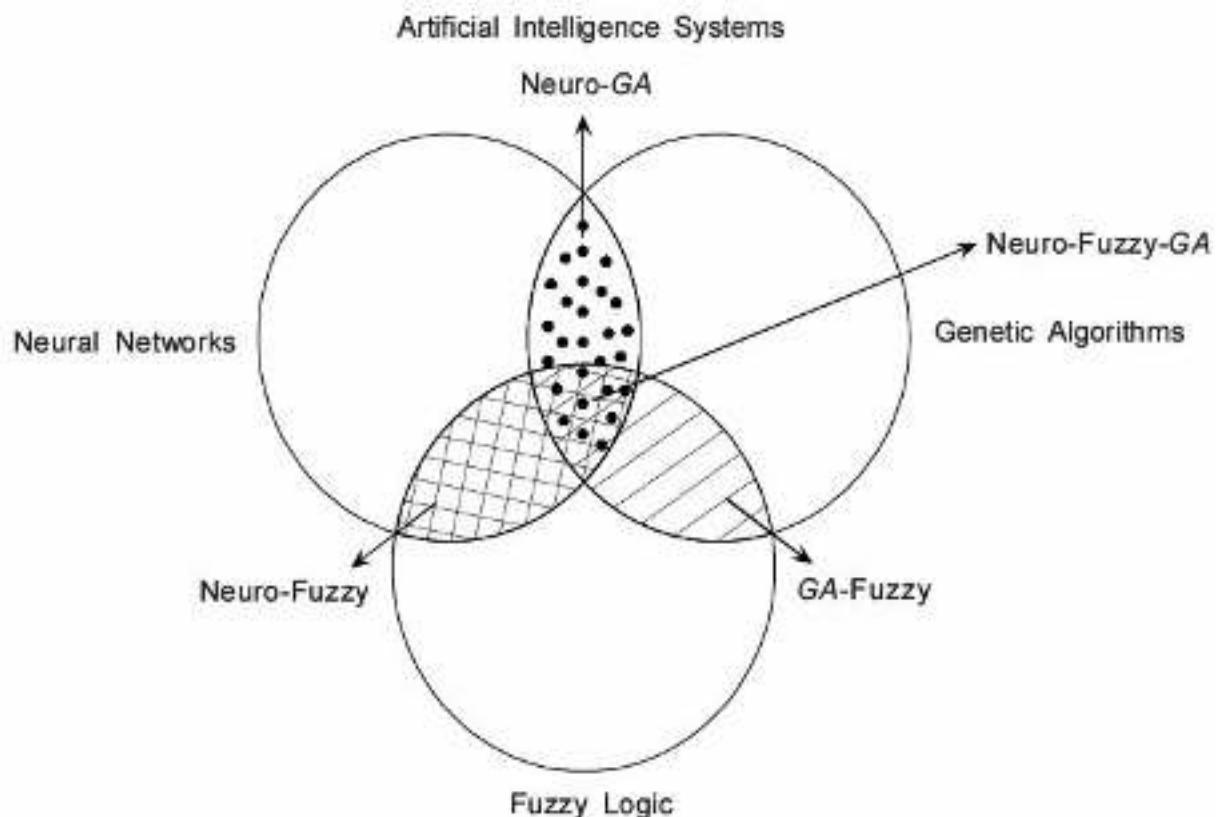
AI in its quest to comprehend, model and implement theories of intelligence, in other words, in its quest to design intelligent systems, has not just registered modest success in developing techniques and methods for intelligent problem solving, but in its relentless pursuit, has fanned out to encompass a number of technologies in its fold. Some of the technologies

include but are not limited to *expert systems*, *neural networks*, *fuzzy logic*, *cellular automata*, and *probabilistic reasoning*. Of these technologies, neural networks, fuzzy logic, and probabilistic reasoning

are predominantly known as *soft computing*. The term ‘soft computing’ was introduced by

Lotfi A. Zadeh of the University of California, Berkley, U.S.A. Probabilistic reasoning subsumes *genetic algorithms*, *chaos*, and parts of *learning theory*.

According to Zadeh, *soft computing differs from hard computing (conventional computing) in its tolerance to imprecision, uncertainty and partial truth*. In effect, the role model is the human mind. Hard computing methods are predominantly based on mathematical approaches and therefore demand a high degree of precision and accuracy in their requirements. But in most engineering problems, the input parameters cannot be determined with a high degree of precision and therefore, the best estimates of the parameters are used for obtaining solution to problems. This has restricted the use of



mathematical approaches for the solution of inverse problems when compared to forward problems.

On the other hand, soft computing techniques, which have drawn their inherent characteristics from biological systems, present effective methods for the solution of even difficult inverse problems. The guiding principle of soft computing is *exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low cost solution... .*

Also, ... *employment of soft computing for the solution of machine learning problems has led to high MIQ (Machine Intelligence Quotient).*

Hybrid intelligence systems deal with the synergistic integration of two or more of the technologies. The combined use of technologies has resulted in effective problem solving in comparison with each technology used individually and exclusively.

In this book, we focus on three technologies, namely *Neural Networks* (NN), *Fuzzy Logic* (FL) and *Genetic Algorithms* (GA) and their hybrid combinations. As illustrated in Fig. 1.1, each of these technologies individually and in combination can be employed to solve problems. The combinations include *neuro-fuzzy*, *GA-fuzzy*, *neuro-GA*, and *neuro-fuzzy-GA* technologies.

Fig. 1.1 Integration of neural networks, fuzzy logic, and genetic algorithm technologies.

We now briefly introduce the three technologies—NN, FL, and GA, viewing them in isolation. Chapter 10 discusses the promises and problems of the integration of these technologies into *hybrid systems*.

1.1 NEURAL NETWORKS

Neural networks are simplified models of the biological nervous system and therefore have drawn their motivation from the kind of computing performed by a human brain.

An NN, in general, is a highly interconnected network of a large number of processing elements called *neurons* in an architecture inspired by the *brain*.

An NN can be massively parallel and therefore is said to exhibit *parallel distributed processing*.

Neural networks exhibit characteristics such as mapping capabilities or pattern association, generalization, robustness, fault tolerance, and parallel and high speed information processing.

Neural networks learn by examples. They can therefore be *trained* with known examples of a problem to ‘acquire’ knowledge about it. Once appropriately trained, the network can be put to effective use in solving ‘unknown’ or ‘untrained’ instances of the problem.

Neural networks adopt various *learning mechanisms* of which *supervised learning* and *unsupervised learning* methods have turned out to be very popular. In supervised learning, a ‘teacher’ is assumed to be present during the learning process, i.e. the network aims to minimize the error between the target (desired) output presented by the ‘teacher’ and the computed output, to achieve better performance. However, in unsupervised learning, there is no teacher present to hand over the desired output and the network therefore tries to learn by itself, organizing the input instances of the problem.

Though NN architectures have been broadly classified as *single layer feedforward networks*, *multilayer feedforward networks*, and *recurrent networks*, over the years several other NN architectures have evolved. Some of the well-known NN systems include *backpropagation network*, *perceptron*, *ADALINE (Adaptive Linear Element)*, *associative memory*, *Boltzmann machine*, *adaptive resonance theory*, *self-organizing feature map*, and *Hopfield network*.

Neural networks have been successfully applied to problems in the fields of pattern recognition, image processing, data compression, forecasting, and optimization to quote a few.

1.2 FUZZY LOGIC

Fuzzy set theory proposed in 1965 by Lotfi A. Zadeh (1965) is a generalization of *classical set theory*. Fuzzy Logic representations founded on Fuzzy set theory try to capture the way humans represent and reason with real-world knowledge in the face of *uncertainty*. Uncertainty could arise due to generality, vagueness, ambiguity, chance, or incomplete knowledge.

A fuzzy set can be defined mathematically by assigning to each possible individual in the *universe of discourse*, a value representing its *grade of membership* in the fuzzy set. This grade corresponds to the degree to which that individual is similar or compatible with the concept represented by the fuzzy set. In other words, fuzzy sets support a flexible sense of membership of elements to a set.

In classical set theory, an element either belongs to or does not belong to a set and hence, such sets are termed *crisp sets*. But in a fuzzy set, many degrees of membership (between 0 and 1) are allowed. Thus, a *membership function* $\mu_A(x)$ is associated with a fuzzy set A such that the function maps every element of the universe of discourse X to the interval [0, 1].

For example, for a set of students in a class (the universe of discourse), the fuzzy set “tall” (fuzzy set A) has as its members students who are tall with a degree of membership equal to 1 ($\mu_A(x) = 1$), students who are of medium height with a degree of membership equal to 0.75 ($\mu_A(x) = 0.75$) and those who are dwarfs with a degree of membership equal to 0 ($\mu_A(x) = 0$), to cite a few cases. In this way, every student of the class could be graded to hold membership values between 0 and 1 in the fuzzy set A, depending on their height.

The capability of fuzzy sets to express gradual transitions from membership ($0 < \mu_A(x) \leq 1$) to non-membership ($\mu_A(x) = 0$) and vice versa has a broad utility. It not only provides for a meaningful and powerful representation of measurement of uncertainties, but also provides for a meaningful representation of vague concepts expressed in *natural language*.

Operations such as *union*, *intersection*, *subsethood*, *product*, *equality*, *difference*, and *disjunction* are also defined on fuzzy sets. *Fuzzy relations* associate crisp sets to varying degree of membership and support operations such as union, intersection, subsethood, and composition of relations.

Just as crisp set theory has influenced symbolic logic, fuzzy set theory has given rise to *fuzzy logic*. While in symbolic logic, truth values *True* or *False* alone are accorded to propositions, in fuzzy logic multivalued truth values such as *true*, *absolutely true*, *fairly true*, *false*, *absolutely false*, *partly false*, and so forth are supported. *Fuzzy inference rules* (which are computational procedures used for evaluating linguistic descriptions) and *fuzzy rule based systems* (which are a set of fuzzy IF-THEN rules) have found wide applications in real-world problems.

Fuzzy logic has found extensive patronage in consumer products especially promoted by the Japanese companies and have found wide use in *control systems*, *pattern recognition applications*, and *decision making*, to name a few.

1.3 GENETIC ALGORITHMS

Genetic Algorithms initiated and developed in the early 1970s by John Holland (1973; 1975) are *unorthodox search and optimization algorithms*, which mimic some of the processes of natural evolution. GAs perform directed random searches through a given set of alternatives with the aim of finding the best alternative with respect to the given criteria of goodness.

These criteria are required to be expressed in terms of an objective function which is usually referred to as a *fitness function*.

Fitness is defined as a figure of merit, which is to be either maximized or minimized. It is further required that the alternatives be coded in some specific finite length which consists of symbols from some finite alphabet.

These strings are called *chromosomes* and the symbols that form the chromosomes are known as *genes*. In the case of binary alphabet (0, 1) the

chromosomes are binary strings and in the case of real alphabet (0–9) the chromosomes are decimal strings.

Starting with an initial population of chromosomes, one or more of the *genetic inheritance operators* are applied to generate *offspring* that competes for survival to make up the next generation of population. The genetic inheritance operators are *reproduction, cross over, mutation, inversion, dominance, deletion, duplication, translocation, segregation, speciation, migration, sharing, and mating*.

However, for most common applications, reproduction, mating (cross over), and mutation are chosen as the genetic inheritance operators.

Successive generations of chromosomes improve in quality provided that the criteria used for survival is appropriate. This process is referred to as *Darwinian natural selection or survival of the fittest*.

Reproduction which is usually the first operator applied on a population selects good chromosomes in a population to form the *mating pool*. A number of reproduction operators exist in the literature (Goldberg and Deb, 1991). *Cross over* is the next operator applied. Here too, a number of cross over operators have been defined (Spears and De Jong, 1990). But in almost all cross over operators, two strings are picked from the mating pool at random and some segments of the strings are exchanged between the strings.

Single point cross over, two point cross over, matrix cross over are some of the commonly used cross over operators. It is intuitive from the construction that good substrings from either parent can be combined to form better offspring strings.

Mutation operator when compared to cross over is used sparingly. The operator changes a

1 to a 0 and vice versa with a small probability P_m . The need for the operator is to keep the diversity of the population.

Though most GA simulations are performed by using a binary coding of the problem parameters, real coding of the parameters has also been propounded and applied (Rawlins, 1990). GAs have been theoretically and empirically proven to provide robust search in complex space and have found wide applicability in scientific and engineering areas including *function optimization, machine learning, scheduling*, and others (Davis L., 1991; Buckles and Petry, 1992).

1.4 STRUCTURE OF THIS BOOK

This book is divided into four parts.

PART I, entitled “Neural Networks”, introduces the fundamentals of neural networks and explores three major architectures, namely:

Backpropagation Network—a multilayer feedforward network which exhibits gradient descent learning

Associative Memory—a single layer feedforward or recurrent network architecture with Hebbian learning

Adaptive Resonance Theory Networks—a recurrent architecture with competitive learning

Various applications of the above mentioned NN architectures have also been detailed. The chapters included are:

Chapter 2: Fundamentals of Neural Networks

Chapter 3: Backpropagation Networks

Chapter 4: Associative Memory

Chapter 5: Adaptive Resonance Theory

PART II, entitled “Fuzzy Logic”, discusses the basic concepts of the fuzzy set theory as much as is essential to understand the hybrid architectures discussed in the book. Also, Fuzzy Logic and Fuzzy Inference have been

discussed in brief. Crisp set theory, a ‘predecessor’ to fuzzy set theory has also been detailed wherever appropriate.

The chapters included are:

Chapter 6: Fuzzy Set theory

Chapter 7: Fuzzy Systems

PART III, entitled “Genetic Algorithms”, elaborates the basic concepts of GA, the genetic inheritance operators, the performance of the algorithm, and applications.

The chapters included are:

Chapter 8: Fundamentals of Genetic Algorithms

Chapter 9: Genetic Modeling

PART IV, entitled “Hybrid Systems”, discusses the integration of the three technologies, namely NN, FL, and GA. As an illustration, the following five hybrid architectures are discussed:

Genetic Algorithm based

Backpropagation Networks — illustrating a neuro-genetic hybrid system

Fuzzy Backpropagation Networks

Simplified Fuzzy ARTMAP

Fuzzy Associative Memories — illustrating neuro-fuzzy hybrid systems

Fuzzy Logic Controlled Genetic — illustrating a fuzzy-genetic hybrid system.

Algorithms

The neural network system playing host in each of the three neuro-fuzzy hybrids is representative of the three different classes of NN architectures, namely single layer feedforward, multilayer feedforward, and recurrent networks. The applications of the hybrid systems to various problems have also been elaborated.

The chapters included are:

Chapter 10: Integration of Neural Networks, Fuzzy Logic, and Genetic Algorithms

Chapter 11: Genetic Algorithm based Backpropagation Networks

Chapter 12: Fuzzy Backpropagation Networks

Chapter 13: Simplified Fuzzy ARTMAP

Chapter 14: Fuzzy Associative Memories

Chapter 15: Fuzzy Logic Controlled Genetic Algorithms

SUMMARY

Artificial Intelligence is a branch of computer science concerned with the design of intelligent computer systems. In pursuit of this goal, AI spans a broad spectrum of areas of which Neutral Networks, Fuzzy Logic and Genetic Algorithms have been chosen as the subject of discussion in this book.

Neural Networks are massively parallel, highly interconnected networks of processing elements called *neurons*.

Fuzzy Logic is an excellent mathematical tool to model uncertainty in systems.

Genetic Algorithms are unorthodox search and optimization algorithms inspired by the biological evolution process.

NN, FL and GA technologies have been individually and integratedly applied to solve various problems in the real world.

REFERENCES

Barr, A. and E. Feigenbaum (Eds.) (1981), *Handbook of Artificial Intelligence*, Vols. I, II,

Los Altos, CA, William Kaufmann.

Buckles B.P. and F.E. Petry (Eds.) (1992), *Genetic Algorithms*, IEEE Computer Society Press.

Davis, L. (1991), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, N.Y.

George, F. Luger and William A. Stubblefield (1993), *Artificial Intelligence Structures and Strategies for Complex Problem-solving*, The Benjamin Cummings Publishing Company, Inc.

Goldberg, D.E. and K. Deb (1991), A Comparison of Selection Schemes Used in Genetic Algorithms, in *Foundations of Genetic Algorithms*, Rawlins, G.J.E. (Ed.), San Mateo, California, Morgan Kaufmann Publishers, pp. 69–73.

Holland, J.L. (1973), Genetic Algorithms and their Optimal Allocation of Trials, *SIAM Journal of Computing*, Vol. 2, No. 2, pp. 88–105.

Holland, J.L. (1975), *Adaptation of Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor.

Rawlins, G.J.E. (1990), *Foundations of Genetic Algorithms*, San Mateo, California, Morgan Kaufmann Publishers, 1990.

Spears, W.M. and K.A. De Jong (1990), An Analysis of Multipoint Crossover, in *Foundations of Genetic Algorithms*, Rawlins G.J.E. (Ed.), San Mateo, California, Morgan Kaufmann Publishers, pp. 310–315.

Zadeh, L.A. (1965), *Fuzzy Sets, Information and Control*, Vol. 8, pp. 338–353.

PART 1

NEURAL NETWORKS

- Fundamentals of Neural Networks
- Backpropagation Networks
- Associative Memory
- Adaptive Resonance Theory

Chapter 2

Fundamentals of Neural Networks

In this chapter, we introduce the fundamental concepts of Neural Networks (NN). The biological neuron system, which has been the chief source of inspiration in much of the research work in Neural Networks and the model of an artificial neuron, are first elaborated. Neural Network architectures, their characteristics and learning methods are next discussed. A brief history of Neural Network research and some of the early NN architectures are presented. Finally, some of the application domains where NN architectures have made an impact are listed.

2.1 BASIC CONCEPTS OF NEURAL NETWORKS

Neural Networks, which are simplified models of the biological neuron system, is a massively parallel distributed processing system made up of

highly interconnected neural computing elements that have the ability to learn and thereby acquire knowledge and make it available for use.

Various learning mechanisms exist to enable the NN acquire knowledge.

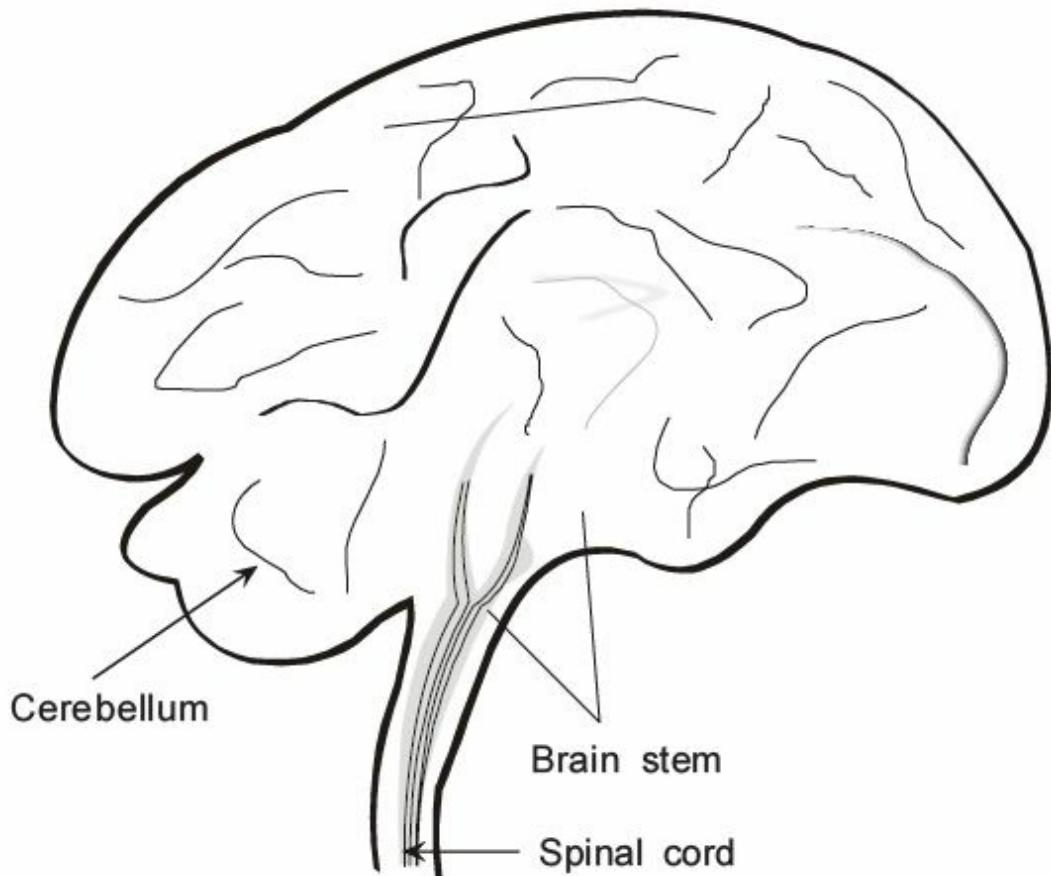
NN architectures have been classified into various types based on their learning mechanisms and other features. Some classes of NN refer to this learning process as *training* and the ability to solve a problem using the knowledge acquired as *inference*.

NNs are simplified imitations of the central nervous system, and obviously therefore, have been motivated by the kind of computing performed by the human brain. The structural constituents of a human brain termed *neurons* are the entities, which perform computations such as cognition, logical inference, pattern recognition and so on. Hence the technology, which has been built on a simplified imitation of computing by neurons of a brain, has been termed *Artificial Neural Systems* (ANS) technology or *Artificial Neural Networks* (ANN) or simply *Neural Networks*. In the literature, this technology is also referred to as *Connectionist Networks*, *Neuro-Computers*, *Parallel Distributed Processors* etc. Also neurons are referred to as *neurodes*, *Processing Elements (PEs)*, and *nodes*. In this book, we shall use the terms *Neural Networks* or *Artificial Neural Networks* and *neurons*.

A human brain develops with time and this, in common parlance is known as *experience*. Technically, this involves the ‘development’ of neurons to adapt themselves to their surrounding environment, thus, rendering the brain *plastic* in its information processing capability. On similar lines, the property of plasticity is also discussed with respect to NN architectures. Further, we are also interested in the *stability* of an NN system; i.e. the adaptive capability of an NN in the face of changing environments. Thus, the *stability*

-plasticity issue is of great importance to NN architectures. This is so since NN systems essentially being learning systems need to preserve the information previously learnt but at the same time, need to be receptive to learning new information. The NN needs to remain ‘plastic’ to significant or useful information but remain ‘stable’ when presented with irrelevant

information. This is known as *stability–plasticity dilemma* (Carpenter and Grossberg, 1987, 1988).



2.2 HUMAN BRAIN

The human brain is one of the most complicated things which, on the whole, has been poorly understood. However, the concept of neurons as the fundamental constituent of the brain, attributed to Ramón Y. Cajál (1911), has made the study of its functioning comparatively easier. Figure 2.1

illustrates the physical structure of the human brain.

Fig. 2.1 Physical structure of the human brain—cross-sectional view.

Brain contains about 10^{10} basic units called *neurons*. Each neuron in turn, is connected to about 10^4 other neurons. A neuron is a small cell that receives electro-chemical signals from its various sources and in turn responds by

transmitting electrical impulses to other neurons. An average brain weighs about 1.5 kg and an average neuron has a weight of 1.5×10^{-9} gms. While some of the neurons perform input and output operations (referred to as *afferent* and *efferent* cells respectively), the remaining form a part of an interconnected network of neurons which are responsible for signal transformation and storage of information. However, despite their different activities, all neurons share common characteristics.

A neuron is composed of a nucleus—a cell body known as *soma* (refer Fig. 2.2). Attached to the soma are long irregularly shaped filaments called *dendrites*. The dendrites behave as input channels, (i.e.) all inputs from other neurons arrive through the dendrites. Dendrites look like branches of a tree during winter. Another type of link attached to the soma is the *Axon*. Unlike the Dendritic links, the axon is electrically active and serves as an output channel. Axons, which mainly appear on output cells are non-linear threshold devices which produce a voltage pulse called *Action Potential* or *Spike* that lasts for about a millisecond. If the cumulative inputs received by the soma raise the internal electric potential of the cell known as *Membrane Potential*, then the neuron ‘fires’ by propagating the action potential down the axon to excite or inhibit other neurons. The axon terminates in a specialised contact called *synapse* or *synaptic junction* that connects the axon with the dendritic links of another neuron. The synaptic junction, which is a very minute gap at the end of the dendritic link contains a neuro-transmitter fluid. It is this fluid which is responsible for accelerating or retarding the electric charges to the soma. Each dendritic link can have many synapses acting on it thus bringing about massive interconnectivity. In general, a single neuron can have many synaptic inputs and synaptic outputs. The size of the synapses are believed to be related to learning. Thus, synapses with larger area are thought to be excitatory while those with smaller area are believed to be inhibitory. Again, it is the increased neuronal activity which is thought to be responsible for learning and memory. Infact, this was what motivated Donald Hebb (1949) to suggest “when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A’s efficiency as

one of the cells firing B is increased.” These observations branded as *Hebbian learning* has influenced many learning models in NN over the years.

Infact, the neuronal activity can be quite complex but viewing the activity as a simple summation of the inputs they receive, has turned out to be a reasonable approximation.

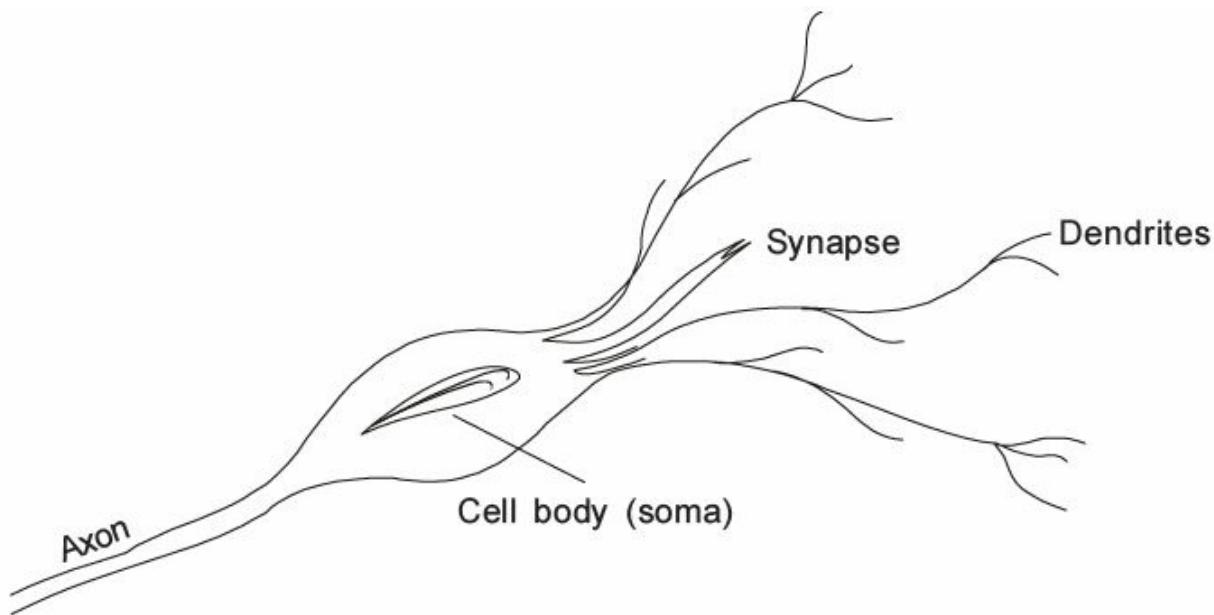
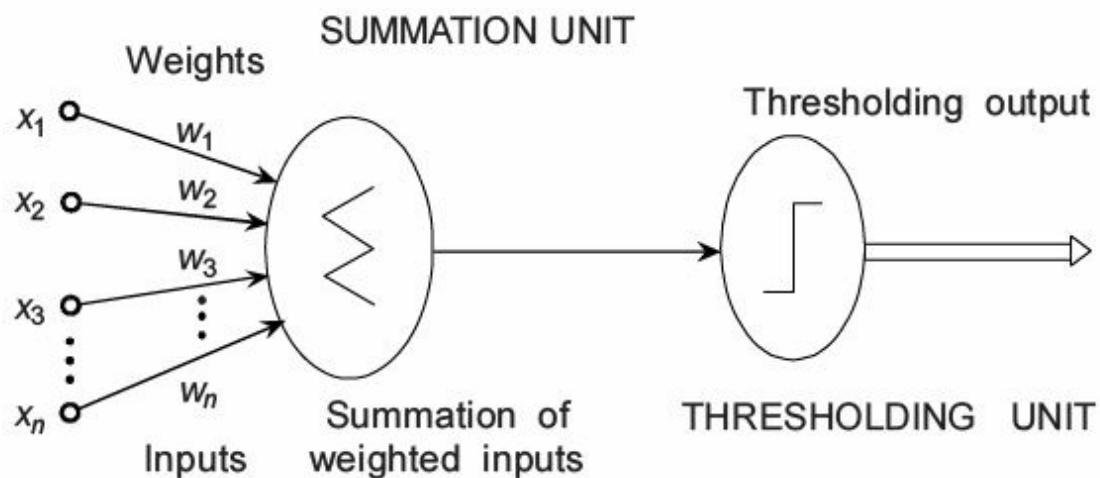


Fig. 2.2 Structure of a neuron.



2.3 MODEL OF AN ARTIFICIAL NEURON

As mentioned earlier, the human brain no doubt is a highly complex structure viewed as a massive, highly interconnected network of simple processing elements called neurons. However, the behaviour of a neuron can be captured by a simple model as shown in Fig. 2.3. Every component of the model bears a direct analogy to the actual constituents of a biological neuron and hence is termed as *artificial neuron*. It is this model which forms the basis of Artificial Neural Networks.

Fig. 2.3 Simple model of an artificial neuron.

Here, $x_1, x_2, x_3, \dots, x_n$ are the n inputs to the artificial neuron. w_1, w_2, \dots, w_n are the weights attached to the input links.

Recollect that a biological neuron receives all inputs through the dendrites, sums them and produces an output if the sum is greater than a threshold value. The input signals are passed on to the cell body through the synapse which may accelerate or retard an arriving signal.

It is this acceleration or retardation of the input signals that is modelled by the *weights*. An effective synapse which transmits a stronger signal will have a correspondingly larger weight while a weak synapse will have smaller weights. Thus, weights here are multiplicative factors of the inputs to account for the strength of the synapse. Hence, the total input I received by the soma of the artificial neuron is

$$\begin{aligned}
 I &= w_1x_1 + w_2x_2 + \dots + w_nx_n \\
 &= \sum_{i=1}^n w_i x_i
 \end{aligned} \tag{2.1}$$

To generate the final output y , the sum is passed on to a non-linear filter ϕ called *Activation function*, or *Transfer function*, or *Squash function* which releases the output.

i.e.

$$y = \phi(I) \tag{2.2}$$

A very commonly used Activation function is the *Thresholding function*. In this, the sum is compared with a threshold value θ . If the value of I is greater than θ , then the output is 1 else it is 0.

i.e.

$$y = \phi\left(\sum_{i=1}^n w_i x_i - \theta\right) \tag{2.3}$$

where, ϕ is the *step function* known as *Heaviside function* and is such that

$$\phi(I) = \begin{cases} 1, & I > 0 \\ 0, & I \leq 0 \end{cases} \tag{2.4}$$

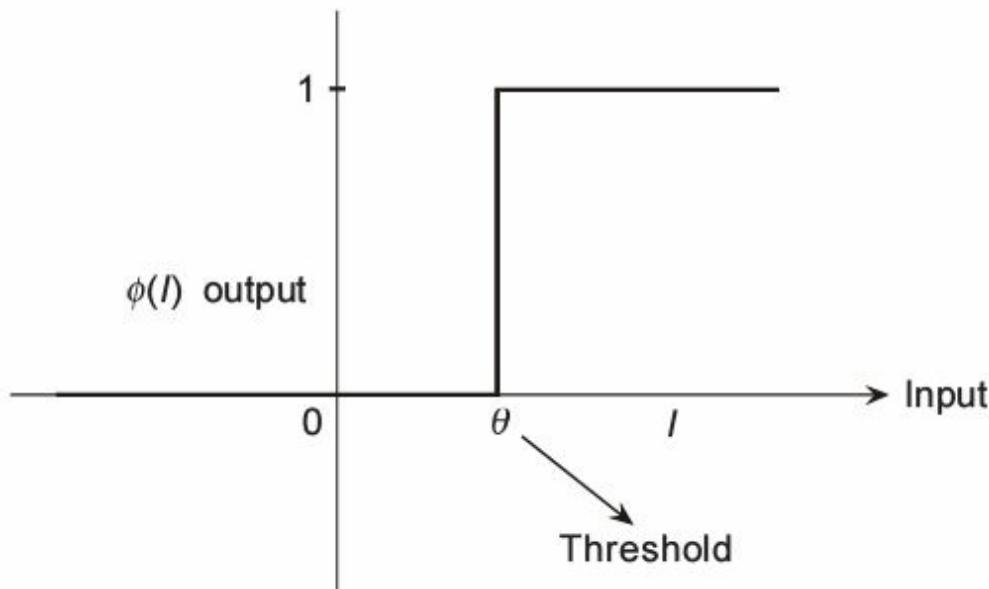


Figure 2.4 illustrates the Thresholding function. This is convenient in the sense that the output signal is either 1 or 0 resulting in the neuron being on or off.

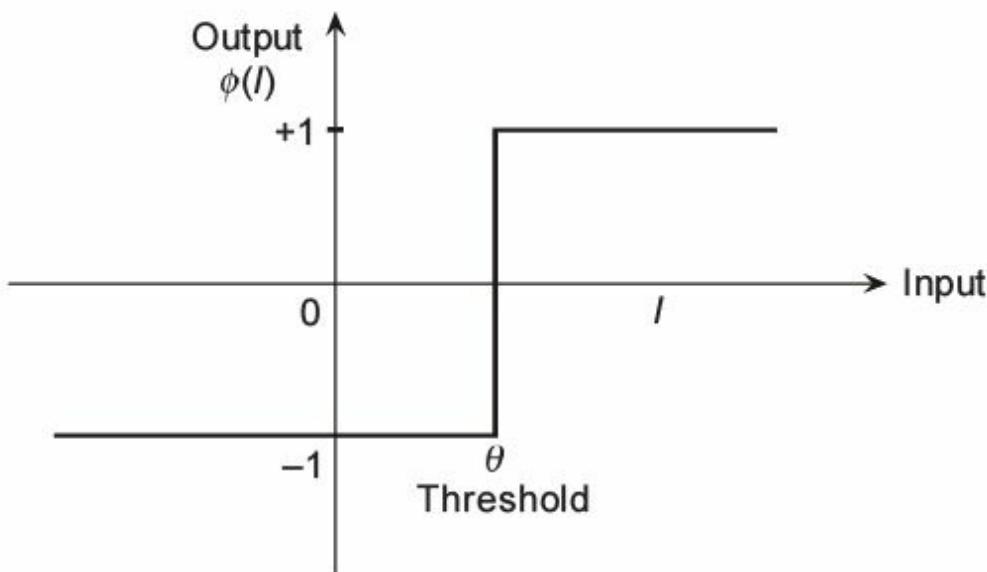
Fig. 2.4 Thresholding function.

The other choices for Activation function besides Thresholding function are as given below:

Signum function

Also known as the Quantizer function, the function ϕ is defined as

$$\phi(I) = \begin{cases} +1, & I > \theta \\ -1, & I \leq \theta \end{cases} \quad (2.5)$$



$$\phi(I) = \frac{1}{1 + e^{-\alpha I}} \quad (2.6)$$

Figure 2.5 illustrates the Signum function.

Fig. 2.5 Signum function.

Sigmoidal function

This function is a continuous function that varies gradually between the asymptotic values 0 and 1 or -1 and +1 and is given by

where, α is the slope parameter, which adjusts the abruptness of the function as it changes between the two asymptotic values. Sigmoidal functions are

differentiable, which is an important feature of NN theory. Figure 2.6 illustrates the sigmoidal function.

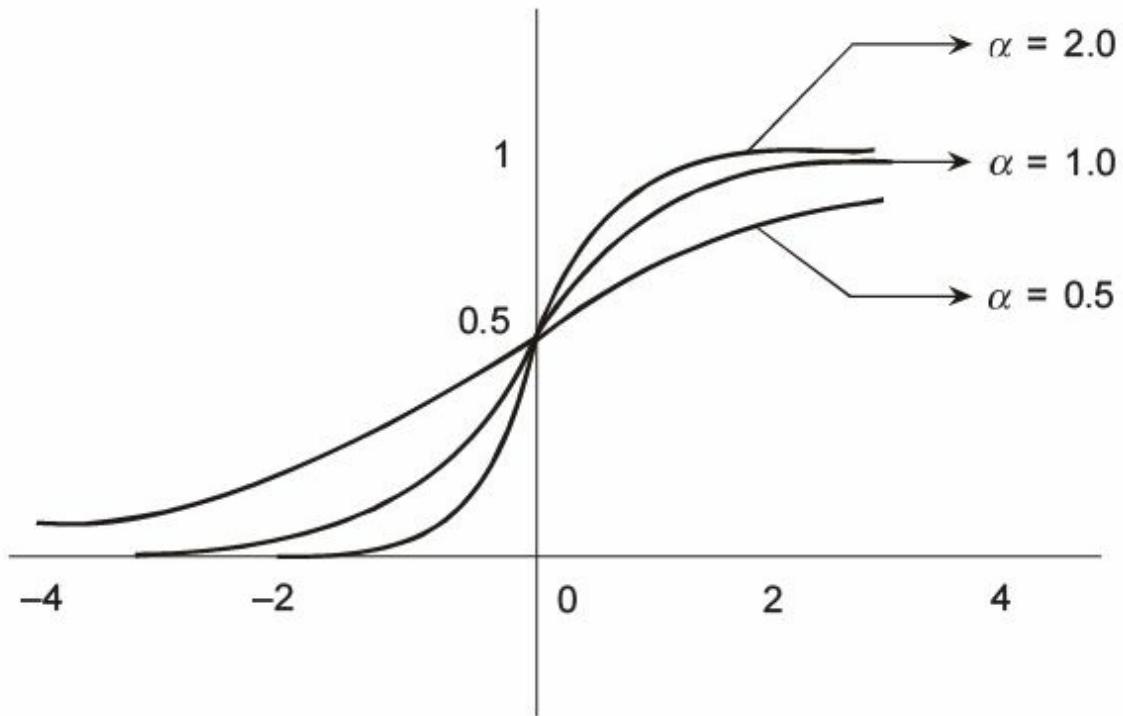


Fig. 2.6 Sigmoidal function.

Hyperbolic tangent function

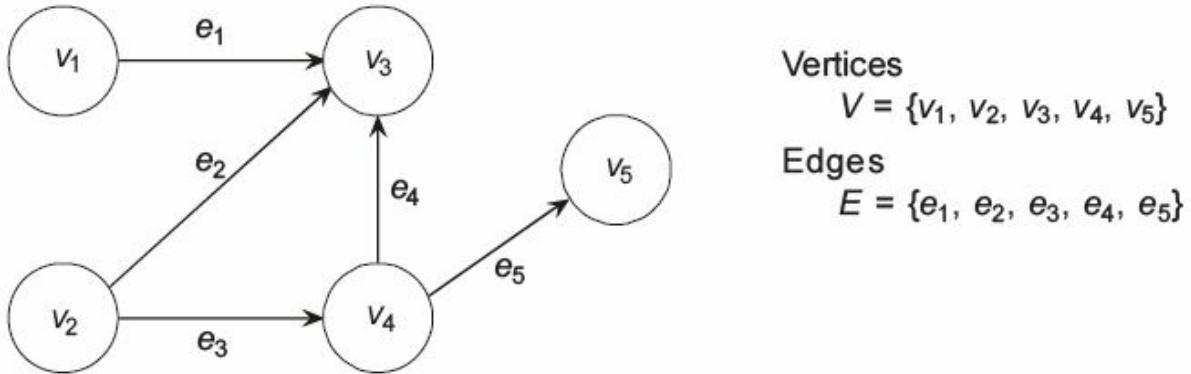
The function is given by

$$\varphi(I) = \tanh(I) \quad (2.7)$$

and can produce negative output values.

The first mathematical model of a biological neuron was presented by McCulloch and Pitts (1943). The model, known as McCulloch-Pitts model does not exhibit any learning but just serves as a basic building block which has inspired further significant work in NN research. The model makes use of a *bias* term whose weight is w_0 but with a fixed input of $x_0 = 1$. This is besides the other inputs x_i and weights w_i . The bias is an external parameter for the artificial neuron but serves the purpose of increasing or decreasing the

net input of the activation function depending on whether it is positive or negative. Other modelling schemes for the neuron have also been proposed (Macgregor, 1987).



2.4 NEURAL NETWORK ARCHITECTURES

An Artificial Neural Network is defined as a data processing system consisting of a large number of simple highly interconnected processing elements (artificial neurons) in an architecture inspired by the structure of the cerebral cortex of the brain (Tsoukalas and Uhrig, 1997). Generally, an ANN

structure can be represented using a *directed graph*. A *graph* G is an ordered 2-tuple (V, E) consisting of a set V of *vertices* and a set E of *edges*. When each edge is assigned an orientation, the graph is directed and is called a *directed graph* or a *digraph*. Figure 2.7 illustrates a digraph. Digraphs assume significance in Neural Network theory since signals in NN systems are restricted to flow in specific directions.

The vertices of the graph may represent neurons (input/output) and the edges, the synaptic links. The edges are labelled by the weights attached to the synaptic links.

Fig. 2.7 An example digraph.

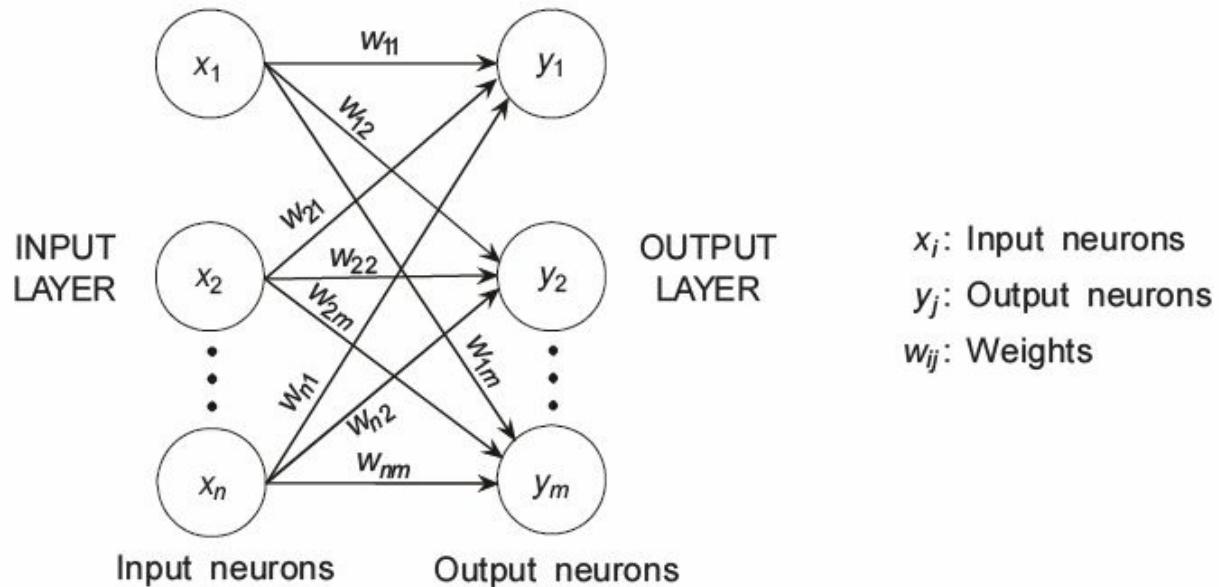
There are several classes of NN, classified according to their learning mechanisms. However, we identify three fundamentally different classes of

Networks. All the three classes employ the digraph structure for their representation.

2.4.1 Single Layer Feedforward Network

This type of network comprises of two layers, namely the *input layer* and the *output layer*. The *input layer neurons* receive the input signals and the *output layer neurons* receive the output signals. The synaptic links carrying the weights connect every input neuron to the output neuron but not vice-versa.

Such a network is said to be *feedforward* in type or acyclic in nature. Despite



the two layers, the network is termed single layer since it is the output layer, alone which performs computation. The input layer merely transmits the signals to the output layer. Hence, the name *single layer feedforward network*. Figure 2.8 illustrates an example network.

Fig. 2.8 Single layer feedforward network.

2.4.2 Multilayer Feedforward Network

This network, as its name indicates is made up of multiple layers. Thus, architectures of this class besides possessing an input and an output layer

also have one or more intermediary layers called *hidden layers*. The computational units of the hidden layer are known as the *hidden neurons* or *hidden units*. The hidden layer aids in performing useful intermediary computations before directing the input to the output layer. The input layer neurons are linked to the hidden layer neurons and the weights on these links are referred to as *input-hidden layer weights*. Again, the hidden layer neurons are linked to the output layer neurons and the corresponding weights are referred to as *hidden-output layer weights*. A multilayer feedforward network with l input neurons, m_1 neurons in the first hidden layer, m_2 neurons in the second hidden layer and n output neurons in the output layer is written as $l - m_1 - m_2 - n$.

Figure 2.9 illustrates a multilayer feedforward network with a configuration $l - m - n$.

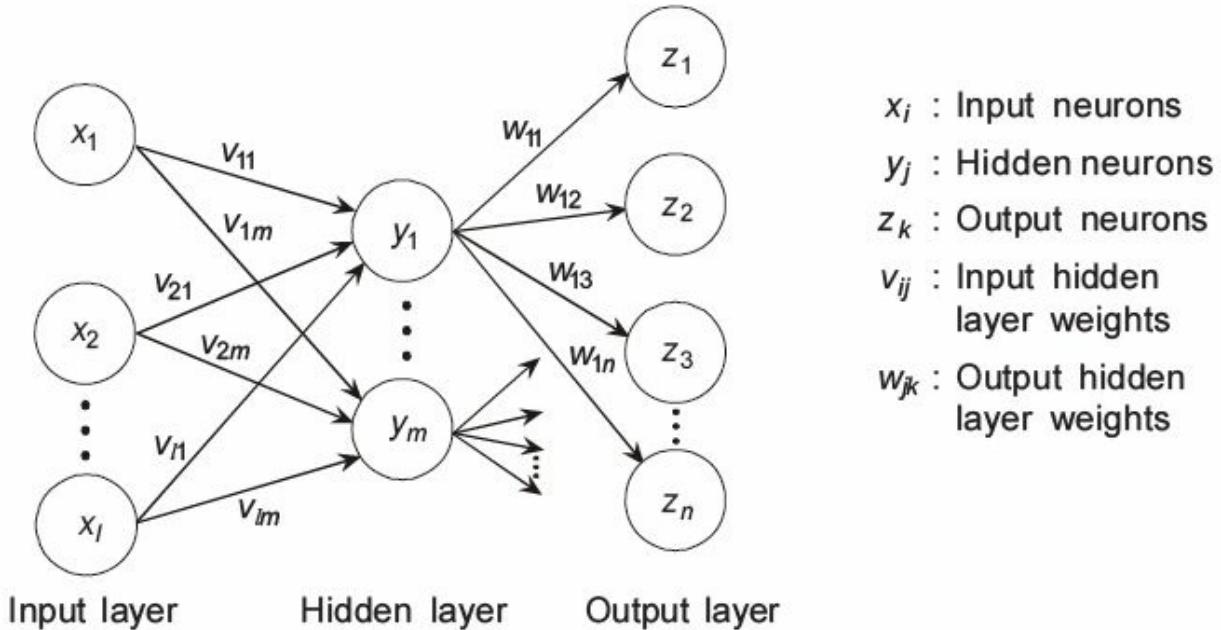


Fig. 2.9 A multilayer feedforward network ($l - m - n$ configuration).

2.4.3 Recurrent Networks

These networks differ from feedforward network architectures in the sense that there is atleast one feedback loop. Thus, in these networks, for example,

there could exist one layer with feedback connections as shown in Fig. 2.10.

There could also be neurons with self-feedback links, i.e. the output of a neuron is fed back into itself as input.

2.5 CHARACTERISTICS OF NEURAL NETWORKS

- (i) The NNs exhibit mapping capabilities, that is, they can map input patterns to their associated output patterns.
- (ii) The NNs learn by examples. Thus, NN architectures can be ‘trained’ with known examples of a problem before they are tested for their ‘inference’ capability on unknown instances of the problem. They can, therefore, identify new objects previously untrained.
- (iii) The NNs possess the capability to generalize. Thus, they can predict new outcomes from past trends.
- (iv) The NNs are robust systems and are fault tolerant. They can, therefore, recall full patterns from incomplete, partial or noisy patterns.
- (v) The NNs can process information in parallel, at high speed, and in a distributed manner.

$$W = \sum_{i=1}^n X_i Y_i^T \quad (2.8)$$

2.6 LEARNING METHODS

Learning methods in Neural Networks can be broadly classified into three basic types: *supervised*, *unsupervised*, and *reinforced*.

Supervised learning

In this, every input pattern that is used to train the network is associated with an output pattern, which is the target or the desired pattern. A teacher is

assumed to be present during the learning process, when a comparison is made between the network's computed output and the correct expected output, to determine the error. The error can then be used to change network parameters, which result in an improvement in performance.

Unsupervised learning

In this learning method, the target output is not presented to the network. It is as if there is no teacher to present the desired patterns and hence, the system learns of its own by discovering and adapting to structural features in the input patterns.

Reinforced learning

In this method, a teacher though available, does not present the expected answer but only indicates if the computed output is correct or incorrect. The information provided helps the network in its learning process. A reward is given for a correct answer computed and a penalty for a wrong answer. But, reinforced learning is not one of the popular forms of learning.

Supervised and unsupervised learning methods, which are most popular forms of learning, have found expression through various rules. Some of the widely used rules have been presented below:

Hebbian learning

This rule was proposed by Hebb (1949) and is based on correlative weight adjustment. This is the oldest learning mechanism inspired by biology.

In this, the input–output pattern pairs (X_i, Y_i) are associated by the weight matrix W , known as the correlation matrix. It is computed as

$$\eta \frac{\partial E}{\partial W_{ij}}$$

Here, $Y T$

i is the transpose of the associated output vector Y_i . Numerous variants of the rule have been proposed (Anderson, 1983; Kosko, 1985; Lippman, 1987; Linsker, 1988).

Gradient descent learning

This is based on the minimization of error E defined in terms of weights and the activation function of the network. Also, it is required that the activation function employed by the network is differentiable, as the weight update is dependent on the gradient of the error E .

Thus, if ΔW_{ij} is the weight update of the link connecting the i th and j th neuron of the two neighbouring layers, then ΔW_{ij} is defined as $\Delta W_{ij} =$

(2.9)

where, η is the learning rate parameter and $\partial E / \partial W_{ij}$ is the error gradient with reference to the weight W_{ij} .

The Widrow and Hoff's Delta rule and Backpropagation learning rule are all examples of this type of learning mechanism.

Competitive learning

In this method, those neurons which respond strongly to input stimuli have their weights updated. When an input pattern is presented, all neurons in the layer compete and the winning neuron undergoes weight adjustment. Hence, it is a “winner-takes-all” strategy.

Stochastic learning

In this method, weights are adjusted in a probabilistic fashion. An example is evident in simulated annealing—the learning mechanism employed by Boltzmann and Cauchy machines, which are a kind of NN systems.

Figure 2.11 illustrates the classification of learning algorithms.

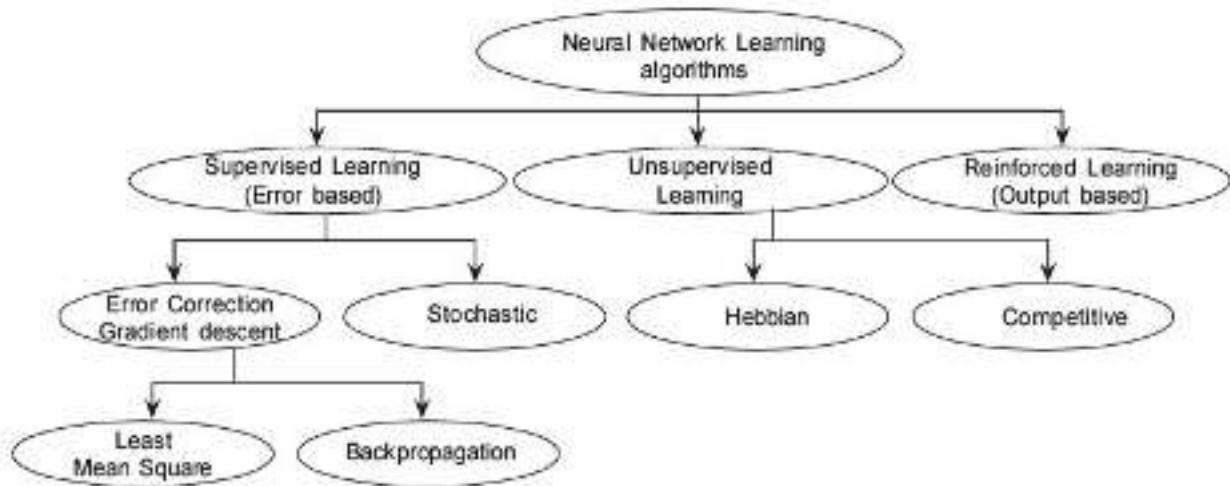


Fig. 2.11 Classification of learning algorithms.

2.7

TAXONOMY

OF

NEURAL

NETWORK

ARCHITECTURES

Over the years, several NN systems have evolved. The following are some of the systems that have acquired significance.

ADALINE (Adaptive Linear Neural Element)

ART (Adaptive Resonance Theory)

AM (Associative Memory)

BAM (Bidirectional Associative Memory)

Boltzmann Machine

BSB (Brain-State-in-a-Box)
CCN (Cascade Correlation)
Cauchy Machine
CPN (Counter Propagation Network)
Hamming Network
Hopfield Network
LVQ (Learning Vector Quantization)
MADALINE (Many ADALINE)
MLFF (Multilayer Feedforward Network)
Neocognitron
Perceptron
RBF (Radial Basis Function)
RNN (Recurrent Neural Network)
SOFM (Self-organizing Feature Map)

Table 2.1 shows the classification of the NN systems listed above, according to their learning methods and architectural types.

Table 2.1 The classification of some NN systems with respect to learning methods and architecture type **LEARNING METHOD**

Gradient

Hebbian

Competitive

Stochastic

descent

ADALINE

Single-layer

AM

LVQ

feedforward

Hopfield

Hopfield

SOFM

—

Perceptron

CCN

TYPE OF

Multilayer

MLFF

Neocognitron

—

—

ARCHITECTURE

feedforward

RBF

Boltzmann

Recurrent

BAM

machine

neural

RNN

BSB

ART

Cauchy

network

Hopfield

machine

2.8 HISTORY OF NEURAL NETWORK RESEARCH

The pioneering work of McCulloch and Pitts (1943) was the foundation stone for the growth of NN architectures. In their paper, McCulloch and Pitts suggested the unification of neuro- physiology with mathematical logic, which paved way for some significant results in NN research. Infact, the McCulloch-Pitts model even influenced Von Neumann to try new design technology in the construction of EDVAC (Electronic Discrete Variable Automatic Computer).

The next significant development arose out of Hebb's book '*The organization of behaviour*'. In this, Hebb proposed a learning rule derived from a model based on synaptic connections between nerve cells responsible for biological associative memory.

The Hebbian rule was later refined by Rosenblatt in 1958, in the Perceptron model (Rosenblatt, 1958). However, a critical assessment of the Perceptron model by Minsky in 1969 (Minsky and Papert, 1969) stalled further research in NN. It was much later in the 1980s that there was a resurgence of interest in NN and many major contributions in the theory and application of NN

were made.

The only important contribution made in the 1970's was the Self Organizing Map Architecture based on Competitive learning (Will Shaw and Von der Malsburg, 1976). Some of the well known architectures which turned out to be milestones in NN research have been listed in Table 2.2.

Table 2.2 Some milestones in NN research

Year

Name of the

Developer

(development

Remarks

neural network

and growth)

Adaptive

•

The networks employ a new principle of self organization called Resonance Carpenter,

1980 and

Adaptive Resonance Theory based on Competitive learning. The Theory (ART)

Gross-berg

onwards

general complexity of the network structures is a limitation.

and others

networks

•

Rumelhart,

1985

The Backpropagation learning rule is applicable on any

Backpropagation

• Hinton,

.

feedforward network architecture. Slow rate of convergence and networks Williams

1974

local minima problem are its weaknesses.

- Werbos

1985

- Parker

Bidirectional

•

Bart

These are two-layer recurrent, hetero associative networks that 1988

Associative

Kosko

can store pattern pairs and retrieve them. They behave as content addressable memories.

Memory (BAM)

- Hinton,

1983, 1985

Boltzmann and

Sejnowski

.

These are stochastic networks whose states are governed by the Cauchy

- Szu H.,

Boltzmann

distribution/Cauchy

distribution.

The

heavy

1986

machines

computational load is a drawback.

•

E.

1987

Hartley

Brain-state-in-a-

•

James

A recurrent auto associative network which makes use of 1977

box

Anderson

Hebbian/Gradient descent learning.

The network belongs to the category of self-organization networks Counter

•

Robert

and functions as statistically optimal self-programming look up propagation
Hecht

1987

table. The weight adjustments between the layers follow
tion network

Nielsen

Kohonen's unsupervised learning rule and Grossberg's supervised learning
rule.

Hopfield

•

John

Single layer recurrent network which makes use of Hebbian 1982
network

Hopfield

learning or Gradient Descent learning.

It is created by a combination of ADALINE networks spread MADALINE

• Bernard

1960

across multiple layers with adjustable weights. The network network

- Widrow

1988

employs a supervised learning rule called MADALINE adaptation Rule (MR) based on ‘minimal disturbance principle’.

A hybrid hierarchical multilayer feedback/forward network which

- Kunihiko

Neocognitron

1982

closely models a human vision system, the network employs either Fukushima

supervised or unsupervised learning rules.

A single layer or multilayer feedforward network best understood

•

Frank

Perceptron

1958

and extensively studied. However, the network is able to obtain Rosenblatt weights, only for linearly separable tasks.

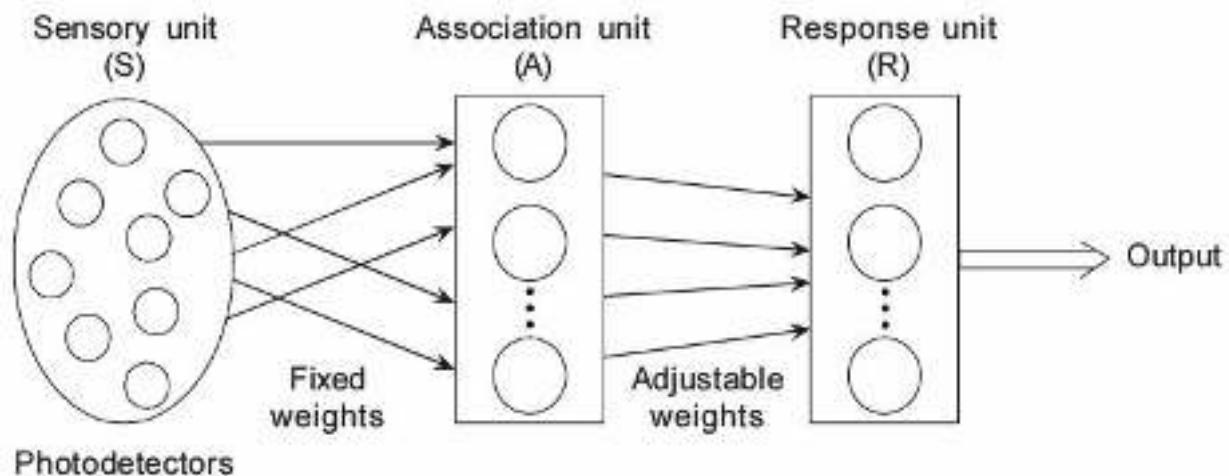
Self-organizing

The network is a simplified model of the feature-to-localized-Feature Map

- Kohonen

1982

region mapping of a brain. It is a self-organizing network networks employing competitive learning.



2.9 EARLY NEURAL NETWORK ARCHITECTURES

2.9.1 Rosenblatt's Perceptron

The perceptron is a computational model of the retina of the eye and hence, is named ‘perceptron’. The network comprises three units, the *Sensory unit S*, *Association unit A*, and *Response unit R* (refer Fig. 2.12).

Fig. 2.12 Rosenblatt's original perceptron model.

The *S* unit comprising 400 photodetectors receives input images and provides a 0/1 electric signal as output. If the input signals exceed a threshold, then the photodetector outputs 1 else 0. The photodetectors are randomly connected to the Association unit *A*. The *A* unit comprises *feature demons* or *predicates*. The predicates examine the output of the *S* unit for specific features of the image. The third unit *R* comprises *pattern recognizers* or *perceptrons*, which receives the results of the predicate, also in binary form. While the weights of the *S* and *A* units are fixed, those of *R* are adjustable.

The output of the R unit could be such that if the weighted sum of its inputs is less than or equal to 0, then the output is 0 otherwise it is the weighted sum itself. It could also be determined by a step function with binary values (0/1) or bipolar values (-1/1). Thus, in the case of a step function yielding 0/1

output values, it is defined as

$$y_j = f(\text{net}_j) = \begin{cases} 1, & \text{if } \text{net}_j > 0 \\ 0, & \text{otherwise} \end{cases}$$

where

$$\text{net}_j = \sum_{i=1}^n x_i w_{ij} \quad (2.10)$$

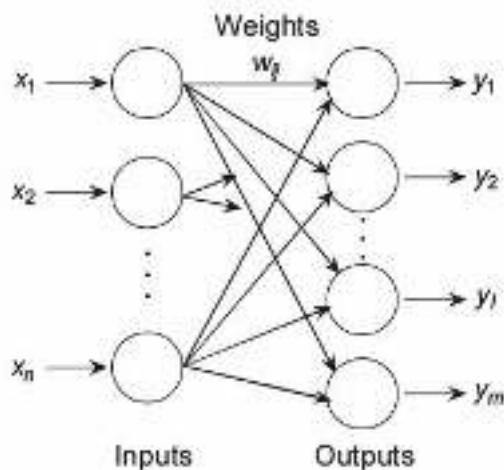


Fig. 2.13 A simple perceptron model.

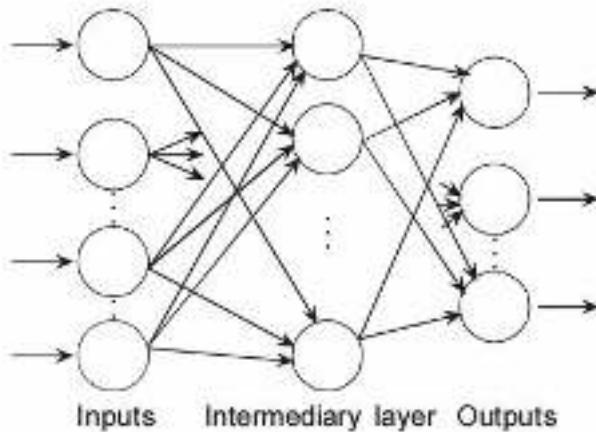


Fig. 2.14 A multilayer feedforward perceptron model.

- If the output is correct then no adjustment of weights is done.

i.e.

$$W_{ij}^{(k+1)} = W_{ij}^{(k)} \quad (2.11)$$

- If the output is 1 but should have been 0 then the weights are decreased on the active input links.

i.e.

$$W_{ij}^{(k+1)} = W_{ij}^{(k)} - \alpha \cdot x_i \quad (2.12)$$

- If the output is 0 but should have been 1 then the weights are increased on the active input links.

i.e.

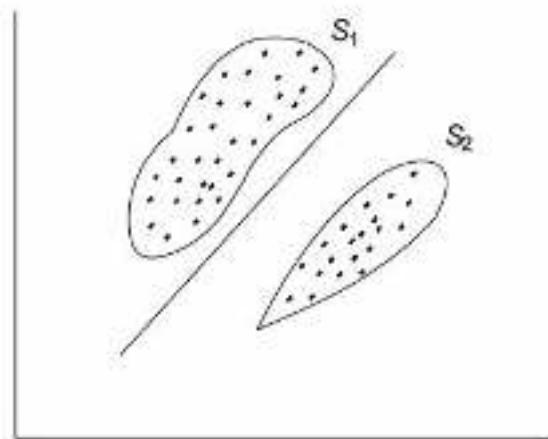
$$W_{ij}^{(k+1)} = W_{ij}^{(k)} + \alpha \cdot x_i \quad (2.13)$$

Here, x_i is the input, w_{ij} is the weight on the connection leading to the output units (R unit), and y_j is the output.

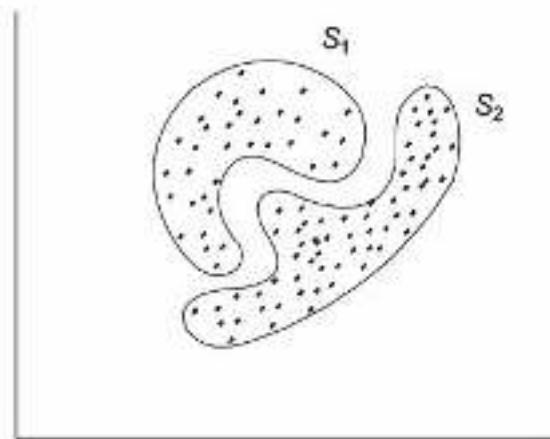
The training algorithm of the perceptron is a supervised learning algorithm where weights are adjusted to minimize error whenever the computed output does not match the target output. Figure 2.13 illustrates a simple perceptron network. A more general multilayer feedforward perceptron is shown in Fig.

2.14.

A basic learning algorithm for training the perceptron is as follows:



(a) Linearly separable patterns



(b) Non-linearly separable patterns

Here, $W(k$

$k)$

$i j + 1)$ is the new adjusted weight, W_{ij}

is the old weight, x_i the

input and α is

the learning rate parameter. Also, small α leads to slow learning and large α

to fast learning. However, large α also runs the risk of allowing weights to oscillate about values which

would result in the correct outputs. For a constant α , the learning algorithm is termed *fixed increment algorithm*. Algorithm 2.1 illustrates the same for a 2-classification problem.

Many variations have been proposed to the perceptron model. The Perceptron Convergence Theorem has been one of the important achievements due to Rosenblatt. However, the observations on the limitations of perceptron by Minsky and Papert (1969), stalled further research on perceptrons until much later, Minsky and Papert pointed out that perceptron would be successful only on problems with a *linearly separable* solution space and cited the XOR

problem as an illustration.

Perceptron and linearly separable tasks

Perceptron cannot handle, in particular, tasks which are not linearly separable.

Sets of points in two dimensional spaces are linearly separable if the sets can be separated by a straight line.

Generalizing, a set of points in n -dimensional space are linearly separable if there is a hyperplane of ($n - 1$) dimensions that separates the sets. Figure 2.15 illustrates linearly separable patterns and non-linearly separable patterns.

Table 2.3 XOR truth table

Inputs	Inputs	Output	
0	0	0	
1	1	0	
0	1	1	
1	0	1	

The diagram shows four arrows originating from the output column of the truth table. The first two arrows point to a box labeled "Even parity", which corresponds to the first two rows of the table where the output is 0. The last two arrows point to a box labeled "Odd parity", which corresponds to the last two rows of the table where the output is 1.

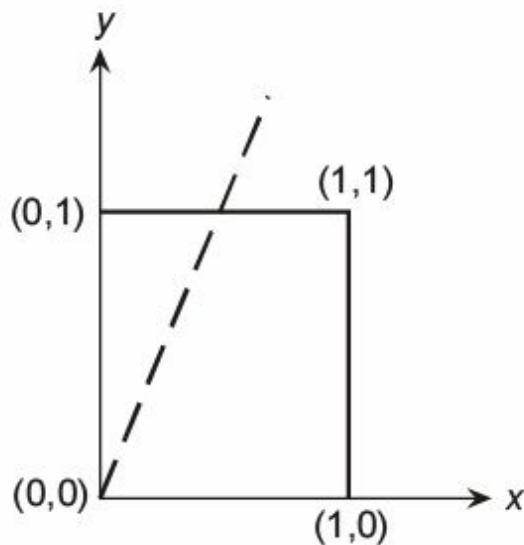


Fig. 2.15 Linearly separable patterns and non-linearly separable patterns.

The perceptron cannot find weights for classification type of problems that are not linearly separable. An example is the XOR (eXclusive OR) problem.

XOR Problem

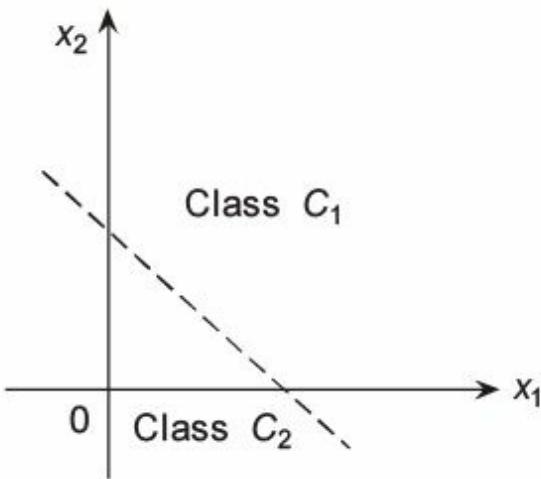
XOR is a logical operation as described by its truth table presented in Table 2.3.

The problem for the ANN is to classify the inputs as *odd parity* or *even parity*. Here, odd parity means odd number of 1 bits in the inputs and even parity refers to even number of 1 bits in the inputs.

This is impossible, since as is evident from Fig. 2.16, the perceptron is unable to find a line separating even parity input patterns from the odd parity input patterns.

Fig. 2.16 The non-linear separable patterns of the XOR problem.

Why is the perceptron unable to find weights for non-linearly separable classification problems? This can be explained by means of a simple



instance.

Consider a perceptron network with two inputs x_1 and x_2 and bias $x_0 = 1$ (refer Fig. 2.17). The weighted sum of the inputs.

$\text{net} = w_0 + w_1 x_1 + w_2 x_2 \quad (2.14)$ represents the equation of a straight line.

The straight line acts as a decision boundary separating the points into classes C_1 and C_2 , above and below the line respectively (refer Fig. 2.18).

Fig. 2.18 A straight line as a decision boundary for a 2-classification problem.

This is what the perceptron aims to do for a problem when it is able to obtain their weights.

Algorithm 2.1

Fixed increment perceptron learning algorithm for a classification problem with n input features (x_1, x_2, \dots, x_n) and two output classes (0/1)

Algorithm Fixed-Incr-Percept-Lrng ($\bar{X}_j, \bar{Y}_j, \bar{W}$)

Step 1: Create a perceptron with $(n + 1)$ input neurons x_0, x_1, \dots, x_n , where $x_0 = 1$ is the bias input. Let O be the output neuron.

Step 2: Initialize $\bar{w} = (w_0, w_1, \dots, w_n)$ to random weights.

Step 3: Iterate through the input patterns \bar{X}_j of the training set using the weight set, (i.e.) compute the weighted sum of inputs $\text{net}_j = \sum_{i=0}^n x_i w_i$ for each input pattern j .

Step 4: Compute the output y_j using the step function

$$y_j = f(\text{net}_j) = 1, \text{net}_j > 0 \\ = 0, \text{otherwise.}$$

Step 5: Compare the computed output y_j with the target output Y_j for each input pattern j . If all the input patterns have been classified correctly, output the weights and exit.

$$W_i^{\text{new}} = W_i^{\text{old}} + \alpha (t - y_j)x_i$$

2.9.2 ADALINE Network

The Adaptive Linear Neural Element Network framed by Bernard Widrow of Stanford University, makes use of supervised learning. Figure 2.19

illustrates a simple ADALINE network. Here, there is only one output neuron and the output values are bipolar (-1 or $+1$). However, the inputs x_i could be binary, bipolar or real valued. The bias weight is w_0 with an input link of $x_0 = 1$.

$= +1$. If the weighted sum of the inputs is greater than or equal to 0 then the output is 1 otherwise it is -1 .

The supervised learning algorithm adopted by the network is similar to the perceptron learning algorithm. Devised by Widrow-Hoff (1960), the learning algorithm is also known as the Least Mean Square (LMS) or Delta rule. The rule is given by

(2.15)

where, α is the learning coefficient, t is the target output, y is the computed output, and x_i is the input.

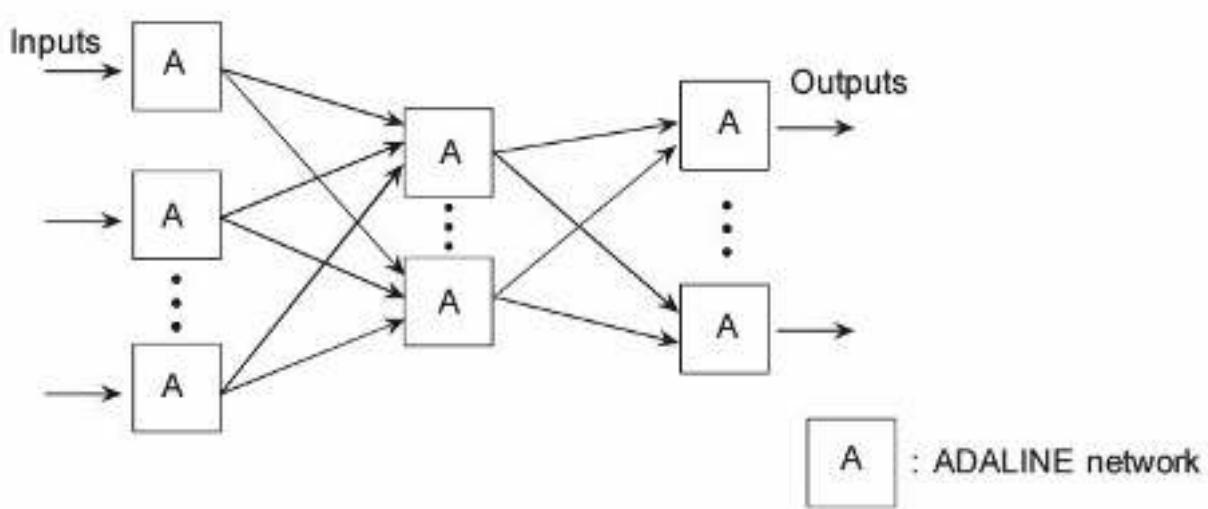
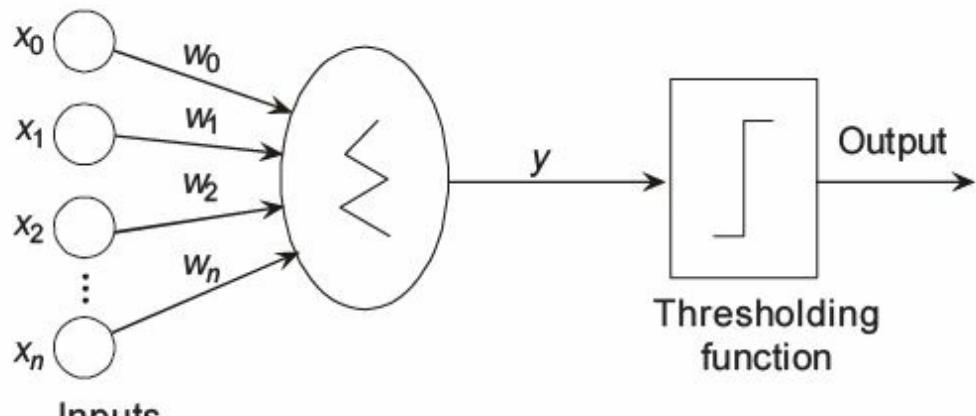


Fig. 2.19 A simple ADALINE network.

ADALINE network has had the most successful applications because it is used virtually in all high speed modems and telephone switching systems to cancel the echo in long distance communication circuits.

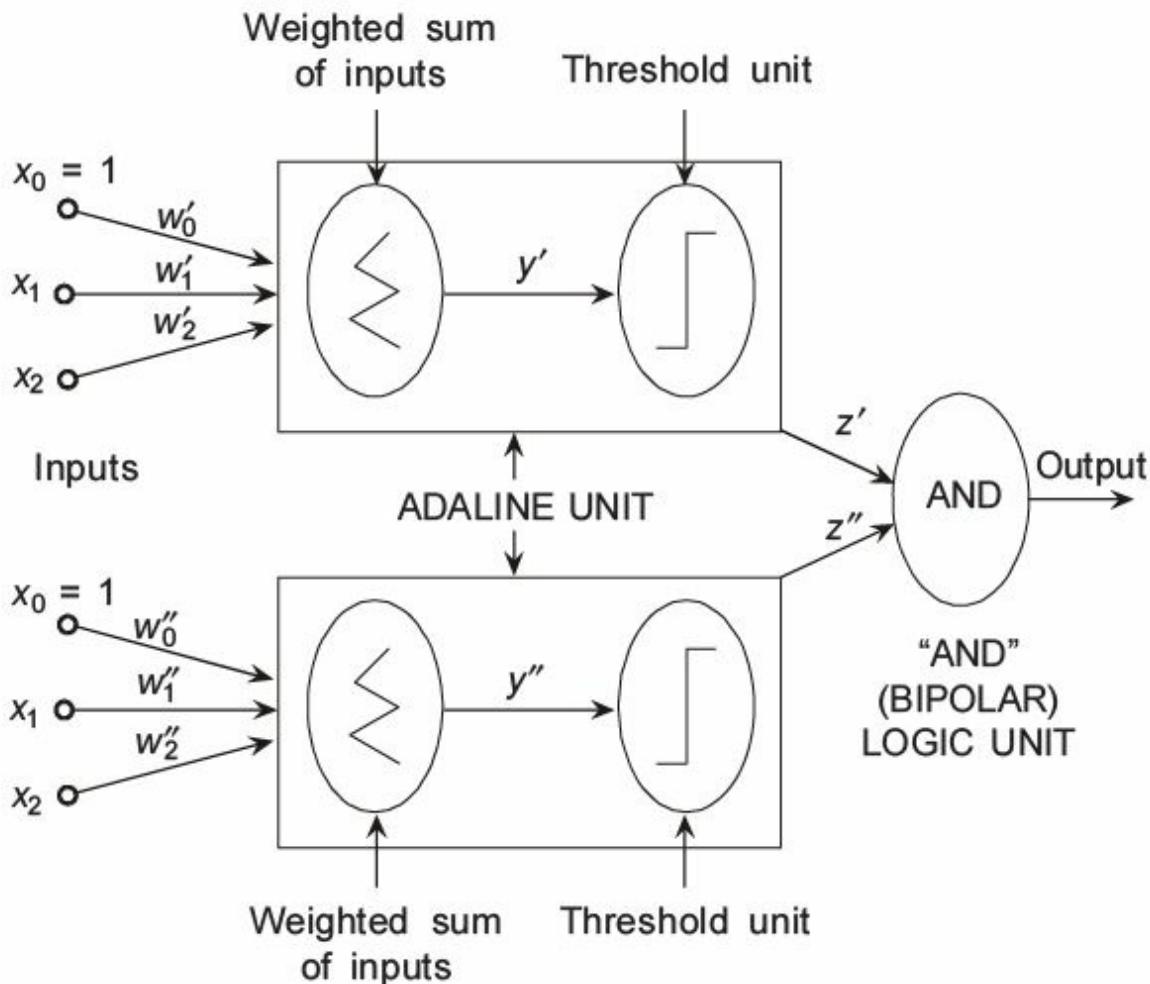
2.9.3 MADALINE Network

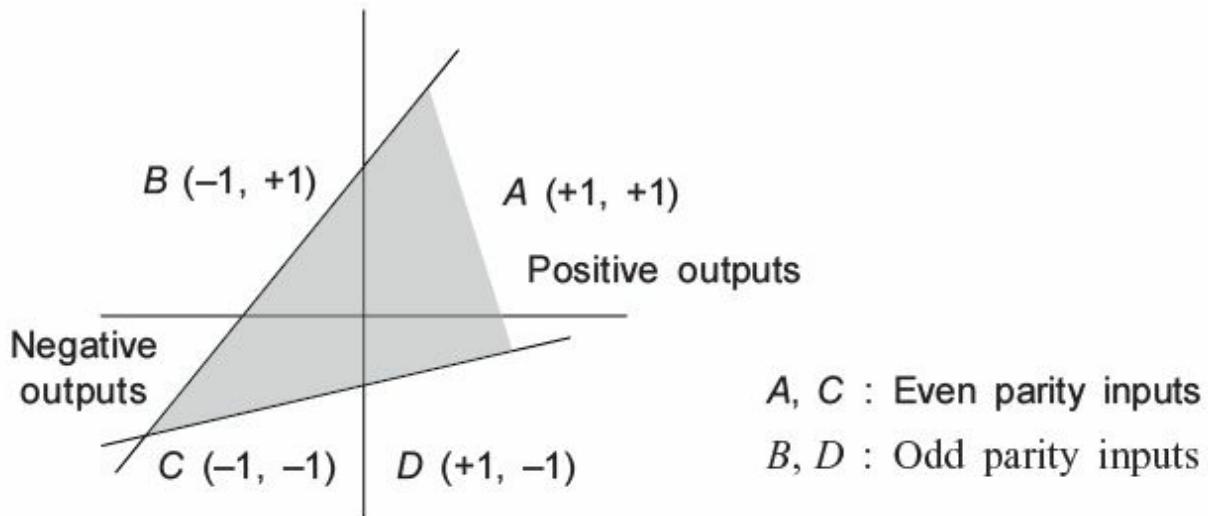
A MADALINE (Many ADALINE) network is created by combining a number of ADALINES. The network of ADALINES can span many layers.

Figure 2.20 illustrates a simple MADALINE network. *The use of multiple ADALINES helps counter the problem of non-linear separability.*

Fig. 2.20 MADALINE network.

For example, the MADALINE network with two units exhibits the capability to solve the XOR problem (refer Fig. 2.21). In this, each ADALINE unit receives the input bits x_1, x_2 and the bias input $x_0 = 1$ as its inputs. The weighted sum of the inputs is calculated and passed on to the bipolar threshold units. The logical ‘and’ing (bipolar) of the two threshold outputs are computed to obtain the final output. Here, if the threshold outputs are





both $+1$ or -1 then the final output is $+1$. If the threshold outputs are different, (i.e.) $(+1, -1)$ then the final output is -1 . Inputs which are of even parity produce positive outputs and inputs of odd parity produce negative outputs.

Figure 2.22 shows the decision boundaries for the XOR problem while trying to classify the even parity inputs (positive outputs) from the odd parity inputs (negative outputs).

Fig. 2.21 A MADALINE network to solve the XOR problem.

Fig. 2.22 Decision boundaries for the XOR problem.

The learning rule adopted by MADALINE network is termed as

‘MADALINE Adaptation Rule’ (MR) and is a form of supervised learning.

In this method, the objective is to adjust the weights such that the error is minimum for the current training pattern, but with as little damage to the learning acquired through previous training patterns.

MADALINE networks have been subject to enhancements over the years.

2.10 SOME APPLICATION DOMAINS

Neural networks have been successfully applied for the solution of a variety of problems. However, some of the common application domains have been listed below:

Pattern recognition (PR)/image processing

Neural networks have shown remarkable progress in the recognition of visual images, handwritten characters, printed characters, speech and other PR based tasks.

Optimization/constraint satisfaction

This comprises problems which need to satisfy constraints and obtain optimal solutions. Examples of such problems include manufacturing scheduling, finding the shortest possible tour given a set of cities, etc. Several problems of this nature arising out of industrial and manufacturing fields have found acceptable solutions using NNs.

Forecasting and risk assessment

Neural networks have exhibited the capability to predict situations from past trends. They have, therefore, found ample applications in areas such as meteorology, stock market, banking, and econometrics with high success rates.

Control systems

Neural networks have gained commercial ground by finding applications in control systems. Dozens of computer products, especially, by the Japanese companies incorporating NN technology, is a standing example. Besides they have also been used for the control of chemical plants, robots and so on.

SUMMARY

NNs are simplified models of the biological nervous systems. An NN

can be defined as a data processing system, consisting of a large number of simple, highly interconnected processing elements (*artificial neurons*), in an architecture inspired by the structure of the cerebral cortex of the brain.

The brain is made up of a massive, highly interconnected network of neurons. Each biological neuron is made up of a cell body—*soma*, with *dendrites* acting as input channels and the *axon* terminating in a specialized contact called *synapse*, as the output channel. The cumulative inputs received by the soma induce the neuron to ‘fire’

resulting in either the excitation or inhibition of other neurons.

An artificial neuron receives n inputs x_1, x_2, \dots, x_n with weights w_1, w_2, \dots, w_n attached to the input links. The weighted sum of the inputs $\sum w_i \cdot x_i$ is computed to be passed on to a nonlinear filter φ called *activation function* to release the output $\varphi(I)$. Here, φ could be a *step function*, *signum function*, *sigmoidal function* or *hyperbolic tangent function*.

The three fundamental classes of NN architectures are, *Single layer feedforward architecture*, *Multilayer feedforward architecture*, and *Recurrent networks architecture*. The learning mechanisms of NNs are broadly classified as *Supervised*, *Unsupervised*, and *Reinforced learning* methods. Supervised and unsupervised learning methods have found expression through rules such as Hebbian learning, Gradient Descent learning Competitive learning, and Stochastic learning. BAM, Boltzmann machine, Cauchy machine, Brain-State-in-a-Box, CPN, Hopfield network, Backpropagation network, ART, Neocognitron, Perceptron, SOFM networks are some of the well known NNs that have turned out to be milestones in NN research.

Amongst the early NN architectures, Rosenblatt’s Perceptron has found a prominent place, though it suffers from the drawback of weight determination only for linearly separable tasks. However, a Multilayer perceptron exhibits the capability to overcome this

problem. Bernard Widrow’s ADALINE and its extension to MADALINE networks have been successful with regard to their applications.

NNs have found wide applications in areas such as pattern recognition, image processing, optimization, forecasting, and control systems to name a few.

PROGRAMMING ASSIGNMENT

P2.1 (a) Implement using C/C++, the Fixed Increment Perceptron Learning algorithm presented in Algorithm 2.1.

(b) The following is a training set for a 2-classification problem.

Iterate the perceptron through the training set and obtain the weights.

Inputs

Classification

X1

X2

0/1

0.25

0.353

0

0.25

0.471

1

0.5

0.353

0

0.5

0.647

1

0.75

0.705

0

0.75

0.882

1

1

0.705

0

1

1

1

P2.2 Attempt solving the XOR problem using the above implementation.

Record the weights. What are your observations?

SUGGESTED FURTHER READING

Plenty of books and journals are available on Neural Networks. *Artificial Neural*

Networks–Theory

and

Applications

(Patterson,

1996),

Fundamentals of Neural Networks, Architectures, Algorithms and Applications (Laurene Fausette, 1994), *Fuzzy and Neural Approaches in Engineering* (Tsoukalas and Uhrig, 1997) are comprehensive titles to begin with.

IEEE Transactions on Neural Networks, *Neural Networks*, *Neuro Computing* and *Neural Computation* are a few of the well known journals.

AI Expert, *IEEE Transactions on Systems, Man and Cybernetics*, and *IEEE Transactions on Pattern Analysis and Machine Intelligence* are some of the well known NN related journals.

REFERENCES

Anderson, J.A. (1983), Cognitive and psychological computation with neural models, *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-13, No. 5, pp. 799–815.

Carpenter Gail, A. and Grossberg Stephen (1987), A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision, Graphics and Image Processing*, 37:54–115.

Carpenter Gail, A. and Grossberg Stephen (1988), The ART of adaptive pattern recognition by a self organizing neural network, *Computer*, 21(3): pp.

77–88, March.

Hebb Donald, O. (1949), *The Organization of Behaviour*, Wiley, NY.

Kosko Bart (1988), Bidirectional Associative Memories, *IEEE Trans. on Systems Man and Cybernetics*, Vol. SMC-18, pp. 49–60.

Laurene Fausett (1994), *Fundamentals of Neural Networks, Architectures, Algorithms and Applications*, Prentice Hall, Englewood Cliffs.

Linsker, R. (1988), Self-organization in a perceptual network, *IEEE*

Computer, Vol. 21, No. 3,

pp. 105–17.

Lippman, R.P. (1987), Introduction to computing with Neural Nets, *IEEE ASSP Magazine*, April, pp. 4–22.

Macgregor, R.J. (1987), *Neural and Brain Modelling*, Academic Press, London.

McCulloch, W.S. and W. Pitts (1943), A logical calculus of the ideas imminent in nervous activity, *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–53.

Minsky, M. and S. Papert (1969), *Perceptrons*, MIT Press, Cambridge, MA.

Patterson Dan, W. (1996), *Artificial Neural Networks: Theory and Applications*, Prentice Hall.

Rosenblatt, Frank (1958), The Perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, 65: 386–408, 1958.

Tsoukalas Lefteri, H. and A. Uhrig Robert (1997), *Fuzzy and Neural*

Approaches in Engineering, John Wiley and Sons, Inc.

Willshaw, D.J. and C. Von der Malsburg (1976), How patterned neural connections can be set up by self-organization, *Proc. of the Royal Society of London Series B*, Vol. 194, pp. 431–445.

Chapter 3

Backpropagation Networks

In Chapter 2, the mathematical details of a neuron at the single cell level and as a network were described. Although single neurons can perform certain simple pattern detection functions, the power of neural computation comes from the neurons connected in a network structure. Larger networks generally offer greater computational capabilities but the converse is not true.

Arranging neurons in layers or stages is supposed to mimic the layered structure of certain portions of the brain.

For many years, there was no theoretically sound algorithm for training multilayer

artificial neural networks. Since single layer networks proved severely limited in what they

could represent (in what they could learn), the entire field went into virtual eclipse. The resurgence of interest in artificial neural network began after the invention of backpropagation algorithm.

Backpropagation is a systematic method of training multilayer artificial neural networks. It is built on high mathematical foundation and has very good application potential. Even though it has its own limitations, it is applied to a wide range of practical problems and has successfully demonstrated its power.

Rumelhart, Hinton and Wilham (1986) presented a clear and concise description

of the backpropagation algorithm. Parker (1982) has also shown to have anticipated

Rumelhart's work. It was also shown elsewhere that Werbos (1974) had described the method still earlier.

In this chapter, we describe the most popular Artificial Neural Network (ANN)

architecture,

the

Multilayer

Feedforward

(MLFF)

network

with

backpropagation (BP) learning. This type of network is sometimes called multilayer perceptron because of its similarity

to perceptron networks with more than one layer. First, we briefly review the

perceptron

model to show how this is altered to form (MLFF) networks. We derive the generalized

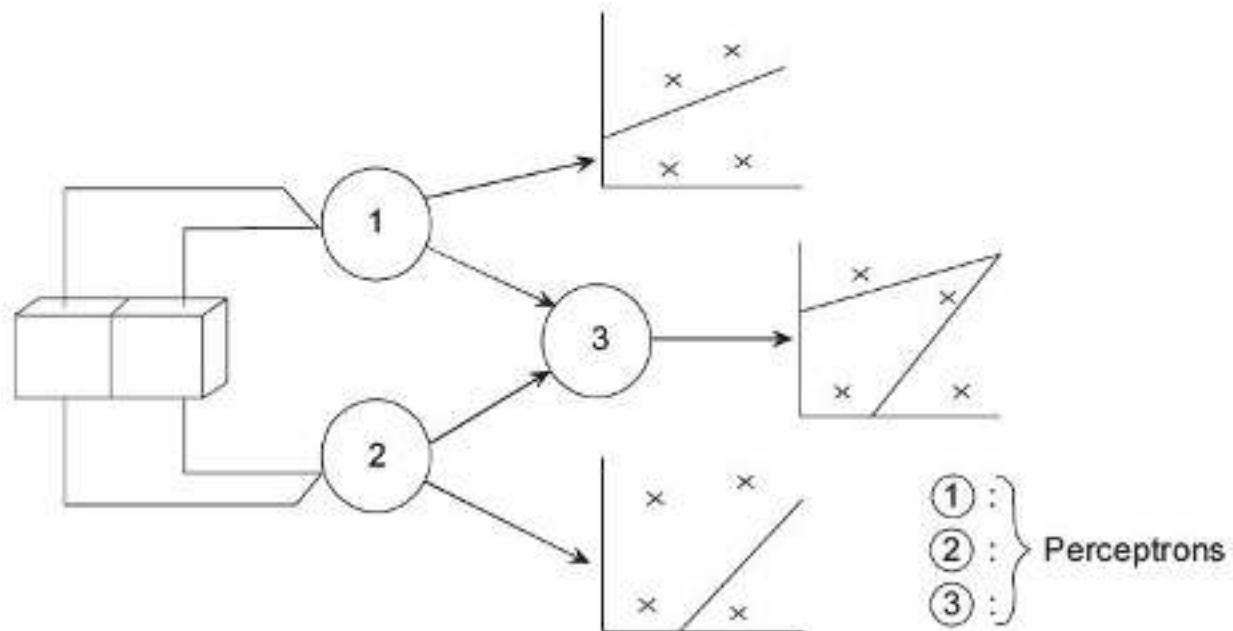
delta (backpropagation) learning rule and see how it is implemented in practice. We

will also examine variations in the learning process to improve the efficiency, and ways to

avoid some potential problems that can arise during training. We will also discuss

optimal parameters' settings and discuss various other training methods. We will also

look at the capabilities and limitations and discuss a number of applications in various engineering fields.



3.1

ARCHITECTURE

OF

A

BACKPROPAGATION

NETWORK

3.1.1 The Perceptron Model

In Chapter 2 (see Section 2.9.1), Rosenblatt's perceptron was introduced and its limitation with regard to the solution of linearly inseparable (or nonlinearly separable) problems was discussed.

The initial approach to solve such linearly inseparable problems was to have more than one perceptron, each set up identifying small linearly separable sections of the inputs. Then, combining their outputs into another perceptron would produce a final indication of the class to which the input belongs. To explain this, let us take the example of XOR problem discussed in Chapter 2. Figure 3.1 illustrates the combination of perceptrons to solve the XOR problem. Even though, it looks that the arrangement shown in Fig.

3.1 can solve the problem. On examination, it is clear that this arrangement of perceptrons in layers will be unable to learn. As explained in Chapter 2, each neuron in the structure takes the weighted sum of inputs, thresholds it and outputs either a one or zero. For the perceptron in the first layer, the input comes from the actual inputs of the problem, while for the perceptron in the second layer the inputs are outputs of the first layer. The perceptrons of the second layer do not know which of the real inputs from the first layer were on or off.

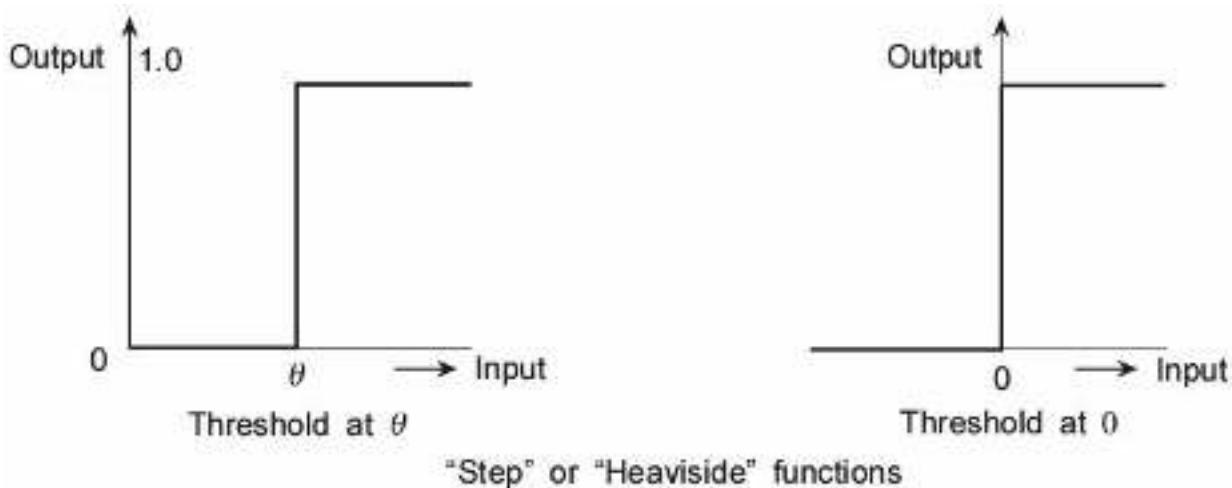


Fig. 3.1 Combining perceptrons to solve XOR problem.

It is impossible to strengthen the connections between active inputs and strengthen the correct parts of the network. The actual inputs are effectively masked off from the output units by the intermediate layer. The two states of

neuron being on or off (shown in Fig. 3.2) do not give us any indication of the scale by which we have to adjust the weights. The hard-hitting threshold functions remove the information that is needed if the network is to successfully learn. Hence, the network is unable to determine which of the input weights should be increased and which one should not and so, it is unable to work to produce a better solution next time. The way to go around the difficulty using the step function as the thresholding process is to adjust it slightly and to use a slightly different nonlinearity.

Fig. 3.2 Hard-hitting threshold function.

3.1.2 The Solution

If we smoothen the threshold function so that it more or less turns on or off as before but has a sloping region in the middle that will give us some information on the inputs, we will be able to determine when we need to strengthen or weaken the relevant weights. Now, the network will be able to learn as required. A couple of possibilities for the new thresholding function are given in Table 3.1. Some of the functions have already been introduced in Chapter 2. Even now, the value of the outputs will practically be one if the input exceeds the value of the threshold a lot and will be practically zero if the input is far less than the threshold. However, in cases when input and threshold are almost same, the output of the neuron will have a value between

$$u(t) = W_1 I_1 + W_2 I_2 + \dots + W_n I_n \quad (3.1a)$$

or

$$u = \langle W \rangle \{I\} \quad (3.1b)$$

zero and one, meaning that the output to the neurons can be related to input in a more informative way.

As seen in Chapter 2, an artificial neuron is developed to mimic the characteristics and functions of a biological neuron. Analogous to a biological neuron, an artificial neuron receives much input representing the output of the other neurons. Each input is multiplied by the corresponding weights analogous to synaptic strengths. All of these weighed inputs are then

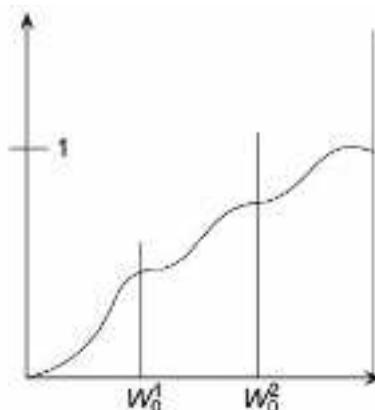
summed up and passed through an activation function to determine the neuron output. The artificial neural model (with new notations to suit the discussion in this chapter) is shown in Fig. 3.3.

Table 3.1 Typical nonlinear activation operators

Type	Equation	Functional form
Linear	$O = gI$ $g = \tan \phi$	
Piecewise Linear	$O = \begin{cases} 1 & \text{if } mI > 1 \\ gI & \text{if } mI < 1 \\ -1 & \text{if } mI > -1 \end{cases}$	
Hard Limiter	$O = \text{sgn}[I]$	
Unipolar Sigmoidal	$O = \frac{1}{(1 + \exp(-\lambda I))}$	
Bipolar Sigmoidal	$O = \tanh[\lambda I]$	

Unipolar Multimodal

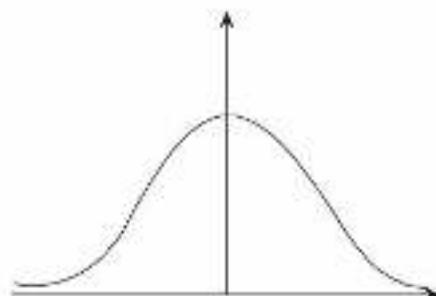
$$O = \frac{1}{2} \left[1 + \frac{1}{M} \sum_{n=1}^M \tanh(\beta^n(I - W_O^n)) \right]$$



Radial Basis Function
(RBF)

$$O = \exp(I)$$

$$I = \left[\frac{-\sum_{i=1}^N (W_i(t) - X_i(t))^2}{2\sigma^2} \right]$$



$$u(t) = W_1 I_1 + W_2 I_2 + \dots + W_n I_n - \theta \quad (3.2a)$$

$$= \sum_{i=0}^n W_i I_i \quad \text{where } W_0 = -\theta; I_0 = 1 \quad (3.2b)$$

The output using the nonlinear transfer function f is given by

$$O = f(u) \quad (3.3)$$

Considering threshold θ , the relative input to the neuron is given by The activation function $f(u)$ is chosen as a nonlinear function to emulate the nonlinear behaviour of conduction current mechanism in a biological neuron.

However, as the artificial neuron is not intended to be a xerox copy of the biological neuron, many forms of nonlinear functions have been suggested and used in various engineering applications. Some of the activation functions along with their mathematical descriptions given in Table 3.1 are most commonly used activation functions in multilayered static neural networks. It is also seen that for sigmoidal functions, the output of a neuron varies continuously but not linearly with the input. Neurons with sigmoidal functions bear a greater resemblance to biological neurons than with other

activation functions. Even if the sigmoidal function is differentiated, it gives continuous values of the output. Hard limiter (see Table 3.1) and radial basis

$$I_I = \begin{Bmatrix} I_{I1} \\ I_{I2} \\ \vdots \\ I_{In} \end{Bmatrix}; \quad O_O = \begin{Bmatrix} O_{O1} \\ O_{O2} \\ \vdots \\ O_{Om} \end{Bmatrix} \quad (3.4)$$

$n \times 1 \qquad m \times 1$

$$\{O_I\} = \{I_I\} \text{ (linear transfer function)} \quad (3.5)$$

$n \times 1 \qquad m \times 1$

$$I_{oj} = W_{1j}I_{I1} + W_{2j}I_{I2} + \dots + W_{nj}I_{In} \quad (3.6)$$

Hence, the input to the output layer can be given as

$$\begin{Bmatrix} I_O \end{Bmatrix} = [W]^T \{O_I\} = [W]^T \{I_I\} \quad (3.7)$$

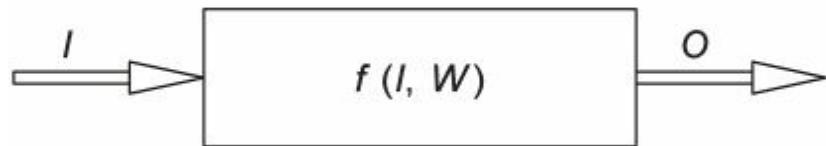
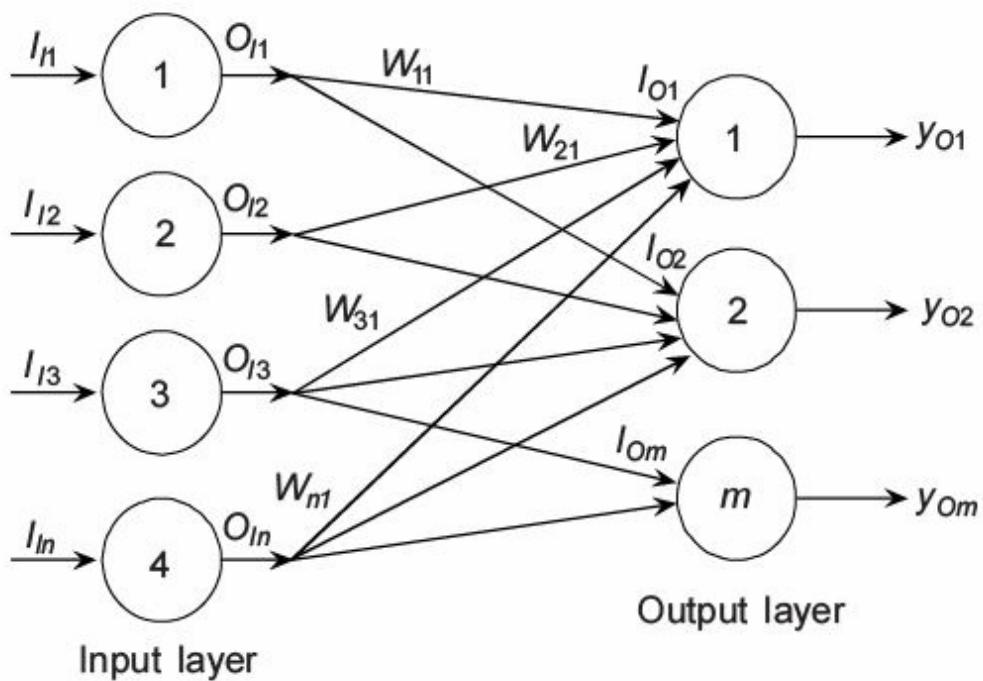
$m \times 1 \qquad m \times n \qquad n \times 1$

functions are also equally popular .

3.1.3 Single Layer Artificial Neural Network

In the preceding section, we have seen the mathematical details of a neuron at a single level. Although a single neuron can perform certain simple pattern detection problems, we need larger networks to offer greater computational capabilities. In order to mimic the layered structure of certain portions of the brain, let us explain the single feedforward neural network as shown in Fig. 3.4(a) and the block diagram is given in Fig. 3.4(b). Consider a single layer feedforward neural network shown in Fig. 3.4(a) consisting of an input layer to receive the inputs and an output layer to output the vectors respectively. The input layer consists of ‘ n ’ neurons and the output layer consists of ‘ m ’ neurons. Indicate the weight of the synapse connecting i th input neuron to the j th output neuron as W_{ij} . The inputs of the input layer and the corresponding outputs of the output layer are given as

Assume, we use linear transfer function for the neurons in the input layer and the unipolar sigmoidal function for the neurons in the output layer (refer Table 3.1).



$$O_{Ok} = \frac{1}{(1 + e^{-\lambda I_{Ok}})} \quad (3.8a)$$

$$\text{or} \quad \{O_O\} = f[WI] \quad (3.8b)$$

Fig. 3.4(a) Single layer feedforward neural network.

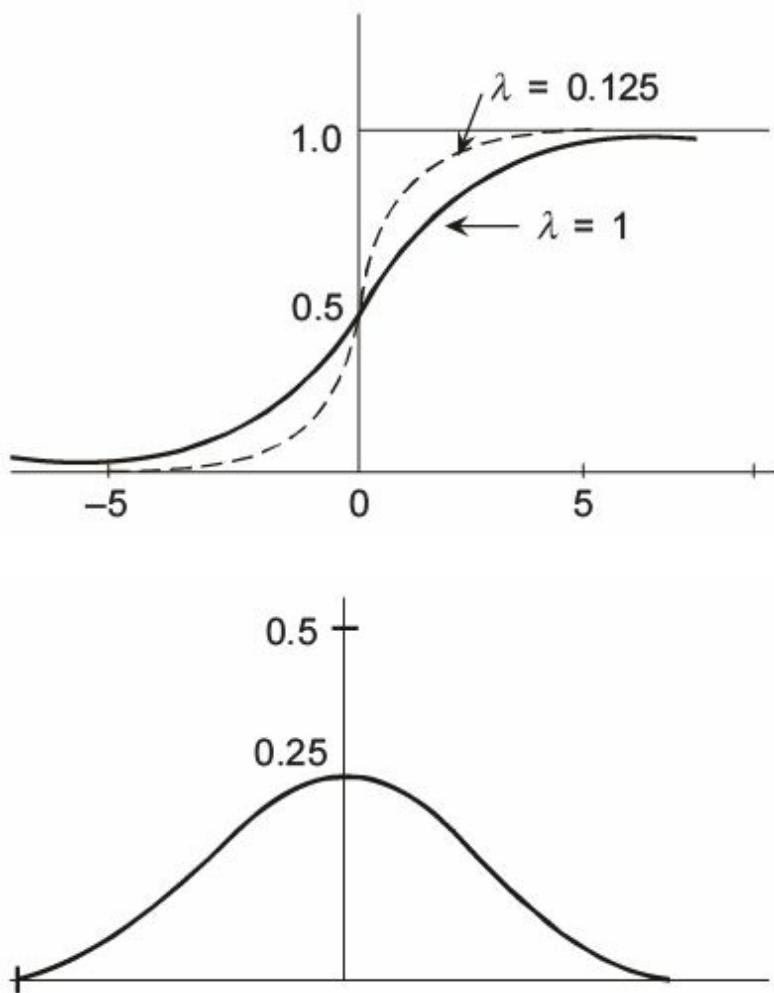
Fig. 3.4(b) Block diagram of a single layer feedforward neural network.

Using the unipolar sigmoidal or squashed-S function as shown in Fig.

3.5(a) and the slope of this function as given in Fig. 3.5(b) for neurons in the output layer, the output is given by

O_{Ok} is evaluated as given in Eq. (3.8a). In Eq. (3.8a), λ is known as sigmoidal gain. The block diagram representation of Eq. (3.8b) is shown in

Fig. 3.4(b). In Eq. (3.7), [W] is called weight matrix and is also known as connection matrix.



$$f(I) = \frac{1}{(1 + e^{-\lambda I})} \quad (3.9a)$$

and

$$f'(I) = \lambda f(I) (1 - f(I)) \quad (3.9b)$$

Fig. 3.5(a) Squashed-S function for various values of λ .

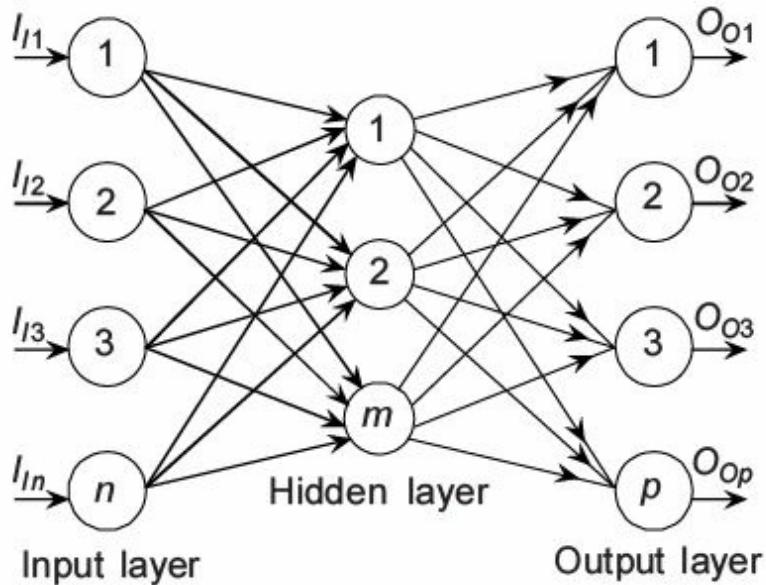
Fig. 3.5(b) Squashed-S slope.

The nonlinear activation function $f[WI]$ in Eq. (3.8a) operates component wise on the activation values ‘ I ’ of each neuron. Each activation value is in

turn a scalar product of the input with respect to weight vectors. The sigmoidal function is given as

3.1.4 Model for Multilayer Perceptron

The adapted perceptrons are arranged in layers and so the model is termed as multilayer perceptron. This model has three layers; an input layer, and output layer, and a layer in between not connected directly to the input or the output and hence, called the hidden layer. For the perceptrons in the input layer, we use linear transfer function, and for the perceptrons in the hidden layer and the output layer, we use sigmoidal or squashed-S functions. The input layer serves to distribute the values they receive to the next layer and so, does not



perform a weighted sum or threshold. Because we have modified the single layer perceptron by changing the nonlinearity from a step function to a sigmoidal function and added a hidden layer, we are forced to alter the learning rules as well. So now, we have a network that should be able to learn to recognize more complex things. The input–output mapping of multilayer perceptron is shown in Fig. 3.6(a) and is represented by

$O = N_3 [N_2 [N_1 [I]]] \dots \dots \dots \quad (3.10)$ In Eq. (3.10), N_1 , N_2 , and N_3 (see Fig. 3.6(b)) represent nonlinear mapping provided by input, hidden and output layers respectively. Multilayer perceptron provides no 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 due to nonlinear activation function of each neuron.

The three-layer network shown in Fig. 3.6(a) and the block diagram shown in Fig. 3.6(b) show that the activity of neurons in the input layers represent the raw information that is fed into the network. The activity of neurons in the hidden layer is determined by the activities of the neurons in the input layer and the connecting weights between input and hidden units. Similarly, the activity of the output units depends on the activity of neurons in the hidden layer and the weight between the hidden and output layers. This structure is interesting because neurons in the hidden layers are free to construct their own representations of the input.

Fig. 3.6(a) Multilayer perceptron.

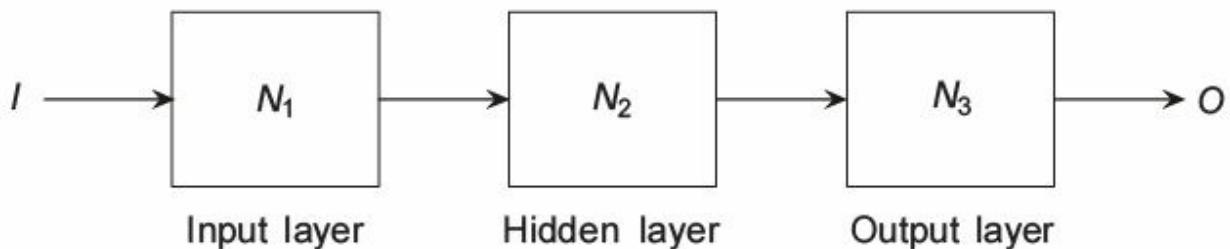
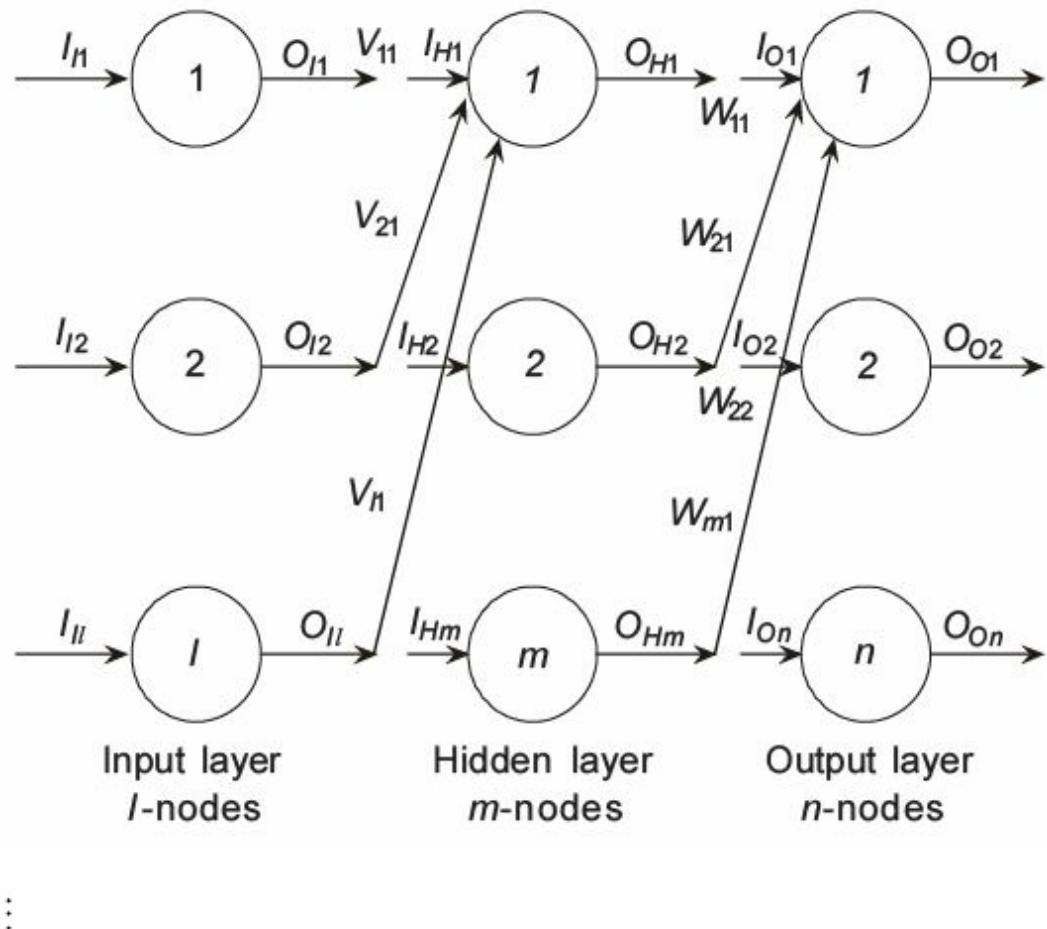


Fig. 3.6(b) Block diagram representing three-layer ANN.



3.2 BACKPROPAGATION LEARNING

Consider the network as shown in Fig. 3.7 where the subscripts I, H, O denote input, hidden and output neurons.

Fig. 3.7 Multilayer feedforward backpropagation network.

Consider a problem in which an “nset” of “l” inputs and the corresponding “nset” of “n” output data is given as shown in Table 3.2.

Table 3.2 “nset” of input and output data

No.

Input

Output

I_1

I_2

...

I_l

O_1

O_2

...

O_n

1

0.3

0.4

...

0.8

0.1

0.56

...

0.82

2

nset

3.2.1 Input Layer Computation

Consider linear activation function the output of the input layer is input of input layer (considering $g = \tan \varphi = 1$). Taking one set of data

$$\{ O \} I = \{ I \} I \quad (3.11)$$

$$\begin{bmatrix} V \\ l \times m \end{bmatrix}$$

$$O_{Ip} = \frac{1}{(1 + e^{-\lambda(I_{Ip} - \theta_p)})} \quad (3.14)$$

$$l \times 1 \dots l \times 1$$

The hidden neurons are connected by synapses to input neurons and (let us denote) Vij is the weight of the arc between i th Input neuron to j th hidden neuron. As shown in Eq. (3.1b), the input to the hidden neuron is the weighted sum of the outputs of the input neurons to get IHp (i.e. Input to the p th hidden neuron) as

$$IHp = V1 pOI1 + V2 pOI2 + \dots + V1 pOIl \quad (3.12) \quad (p = 1, 2, 3, \dots, m)$$

Denoting weight matrix or connectivity matrix between input neurons and hidden neurons as

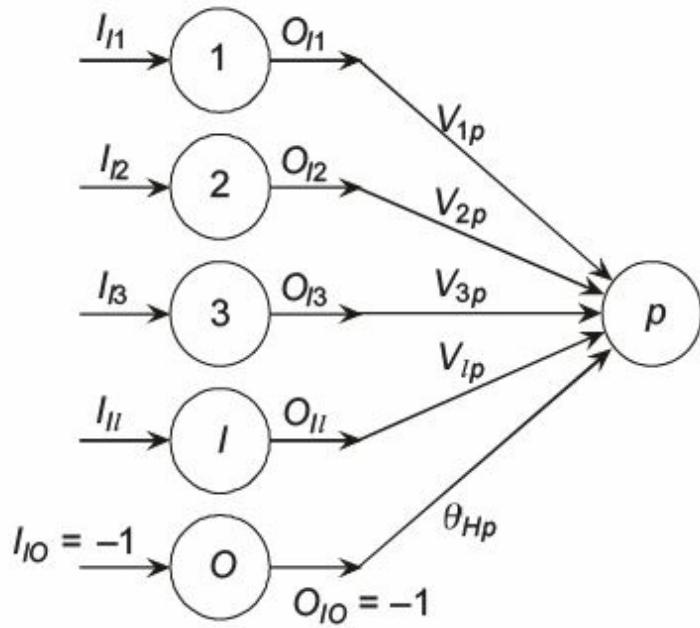
, we can get an input to the hidden neuron as

$$\{ I \} H = [V] T \{ O \} I \quad (3.13) \quad m \times 1 \dots m \times l \quad l \times 1$$

3.2.2 Hidden Layer Computation

Considering sigmoidal function or squashed-S function, the output of the p th hidden neuron is given by

where O_{Hp} is the output of the p th hidden neuron, I_{Hp} is the input of the p th hidden neuron, and θ_{Hp} is the threshold of the p th neuron. A non-zero threshold neuron is computationally equivalent to an input that is always held at -1 and the non-zero threshold becomes the connecting weight values as shown in Fig. 3.8.



$$\{O\}_H = \begin{Bmatrix} - \\ - \\ 1 \\ - \\ - \end{Bmatrix} \frac{1}{(1 + e^{-\lambda(I_{hp} - \theta_{hp})})} \quad (3.15)$$

Fig. 3.8 Treating threshold in hidden layer.

But in our derivations we will not treat threshold as shown in Fig. 3.8 .

Now, output to the hidden neuron is given by

Treating each component of the input of the hidden neuron separately, we get the outputs of the hidden neuron as given by Eq. (3.15).

As shown in Fig. 3.1(b) the input to the output neurons is the weighted sum of the outputs of the hidden neurons. To get IOq (i.e. the input to the q th output neuron)

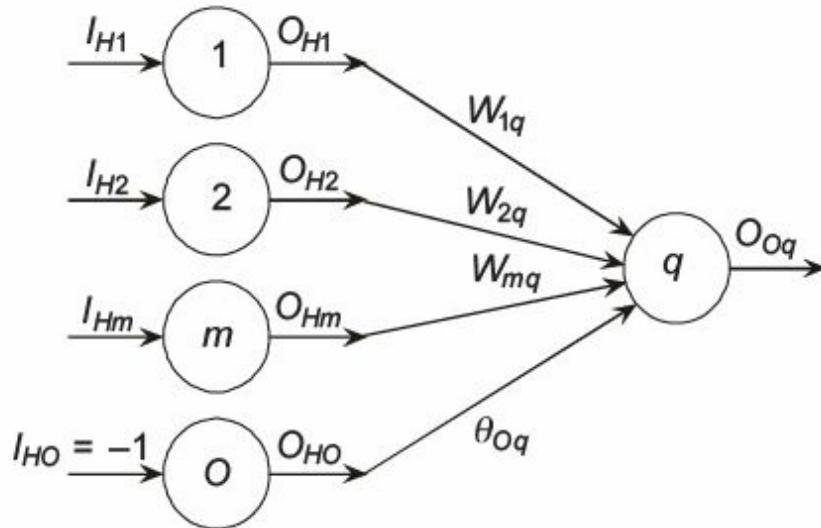
$$IOq = W_1 qOH_1 + W_2 qOH_2 + \dots + W_m qOH_m \quad (3.16) \quad (q = 1, 2, 3, \dots, n)$$

Denoting weight matrix or connectivity matrix between hidden neurons and output neurons as $[W]$, we can get input to the output neuron as

$$\{I\} O = [W] T \{O\} H \quad (3.17) \quad n \times 1 \dots n \times m \dots m \times 1$$

3.2.3 Output Layer Computation

$$O_{Oq} = \frac{1}{(1 + e^{-\lambda(O_{Oq} - \theta_{Oq})})} \quad (3.18)$$



$$\{O\}_O = \left[\begin{array}{c} \vdots \\ \vdots \\ 1 \\ \vdots \end{array} \right] \quad (3.19)$$

$$E_r^1 = \frac{1}{2} e_r^2 = \frac{1}{2} (T - O)^2$$

Considering sigmoidal function, the output of the q th output neuron is given by

where, O_{Oq} is the output of the q th output neuron, IO_q is the input to the q th output neuron, and θ_{Oq} is the threshold of the q th neuron. This threshold may also be tackled again by considering extra O th neuron in the hidden layer with output of -1 and the threshold value θ_{Oq} becomes the connecting weight value as shown in Fig. 3.9.

Fig. 3.9 Treating threshold in output layer.

Hence, the outputs of output neurons are given by

3.2.4 Calculation of Error

Considering any r th output neuron and for the training example we have calculated the output ‘ O ’ for which the target output ‘ T ’ is given in Table 3.2.

Hence, the error norm in output for the r th output neuron is given by (3.20)

$$E_r^1$$

$$E^1 = \frac{1}{2} \sum_{r=1}^n (T_{Or} - O_{Or})^2$$

$$E^1 = \frac{1}{2} \sum_{r=1}^n (T_{Or} - O_{Or})^2$$

where

$= 1/2$ second norm of the error in the r th neuron (' er ') for the given training pattern. The square of the error is considered since irrespective of whether error is positive or negative, we consider only absolute values. The Euclidean norm of error E for the first training pattern is given by (3.21)

Equation 3.21 gives the error function in one training pattern. If we use the same technique for all the training patterns, we get

(3.22)

where E is the error function depending on the $m(1+n)$ weights of [W] and

[V]. This is a classic type of optimization problem. For such problems, an objective function or cost function is usually defined to be maximized or minimized with respect to a set of parameters. In this case, the network parameters that optimize the error function E over the 'nset' of pattern sets

[I nset, t nset] are synaptic weight values [V] and [W] whose sizes are

[V] and [W] (3.23)

$l \times m \dots m \times n$

3.2.5 Training of Neural Network

The synaptic weighting and aggregation operations performed by the synapses and soma respectively, provide a 'similarity measure' between the input vector I and the synaptic weights [V] and [W] (accumulation knowledge base). When a new input pattern that is significantly different from the previously learned pattern is presented to the neural network, the similarity between this input and the existing knowledge base is small. As the neural network learns this new pattern, by changing the strengths of synaptic weights, the distance between the new information and accumulated knowledge decreases as shown in Fig. 3.10. In other words, the purpose of learning is to make " W and V " very similar to given pattern I .

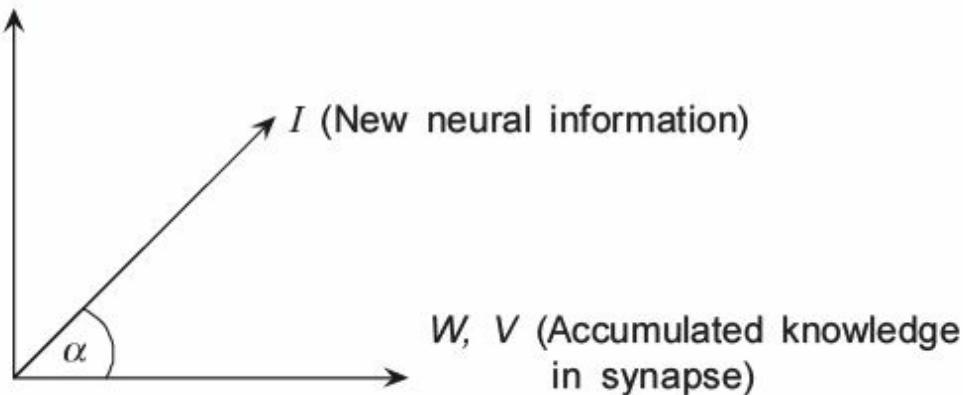


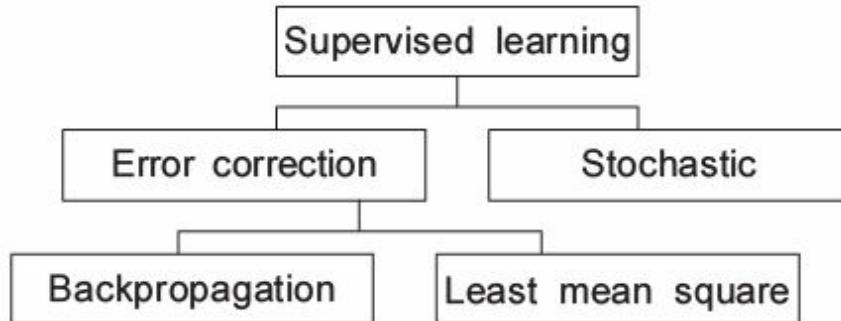
Fig. 3.10 Similarity between new information and past knowledge.

Most of the neural network structures undergo ‘learning procedures’ during which synaptic weights W and V are adjusted. Algorithms for determining the connection strengths to ensure learning are called ‘learning rules’. The objective of learning rules depends upon applications. In classification and functional approximation problems, each cycle of presentation of all cases is usually referred as ‘learning epoch’. However, there has been no generalization as to how a neural network can be trained.

As in Chapter 2, neural network learning algorithms have been classified as *Supervised* and *Unsupervised learning algorithms*. The learning algorithms are shown in Fig. 3.11. Supervised algorithms are also known as ‘error based learning algorithms’ which employ an external reference signal (teacher) and generate an error signal by comparing the reference with the obtained response. Based on error signal, neural network modifies its synaptic connections to improve the system performance. In this scheme, it is always assumed that the desired answer is known

“a priori”. In this backpropagation neural network, we use the procedure of supervised learning of backpropagation. In contrast, as seen in Chapter 2, unsupervised (output based) learning or competitive learning involve adjustment of synaptic weights according to the correlation of the response of two neurons that adjoin it. Error based algorithms need desired responses or training data labelled with target results. If target results are unknown then error based algorithms are useless and learning output based learning algorithm are useful. Figure 3.11 reviews the supervised learning

classification. Let us discuss the supervised learning algorithm for backpropagation neural network.



$$E = \sum_{p=1}^{\text{nset}} E^p(V, W, I)$$

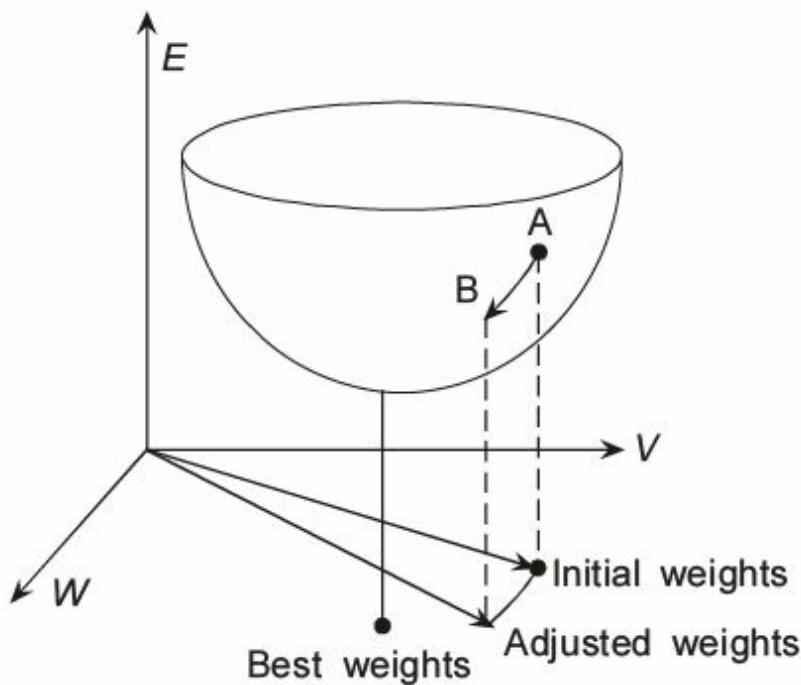


Fig. 3.11 Learning algorithms.

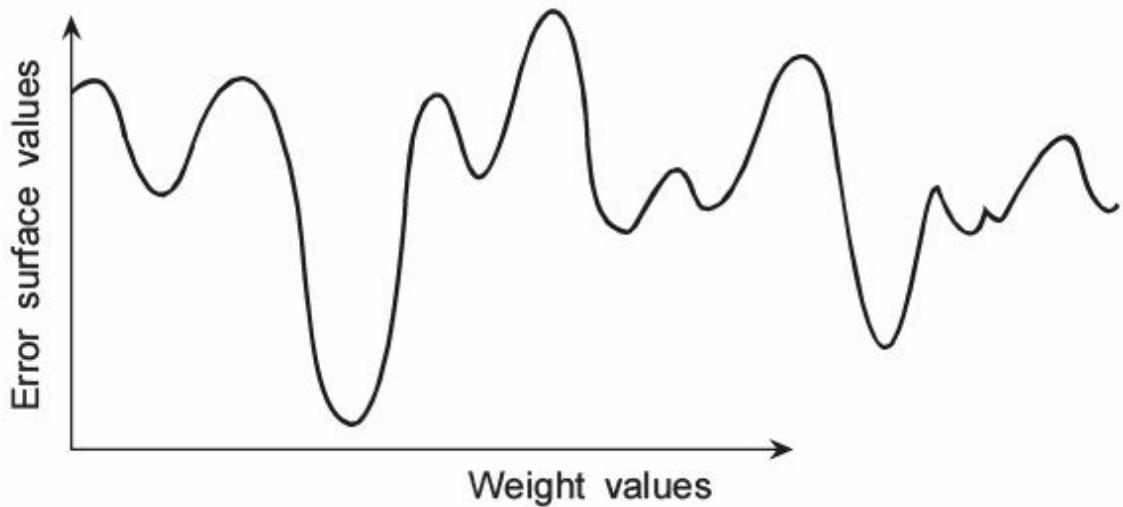
3.2.6 Method of Steepest Descent

The error surface is given by

...(3.24)

and is shown in Fig. 3.12. Multilayer feedforward networks with nonlinear activation functions have mean squared error (MSE) surface above the total Q -dimensional weight space RQ which is not in general, a smooth parabolic surface as in the single layer linear activation case. In general, the error surface is complex and consists of many local and global minima, shown by McInerney and Dhawan (1989) as illustrated in Fig. 3.13.

Fig. 3.12 Euclidian norm of errors.



$$E = \frac{1}{\text{nset}} \sum_{p=1}^{\text{nset}} E^p = \frac{1}{2 \times \text{nset}} \sum_{p=1}^k (T_k^p - O_{Ok}^p)^2$$

Fig. 3.13 Typical error surface for MLFF networks with nonlinear activation function.

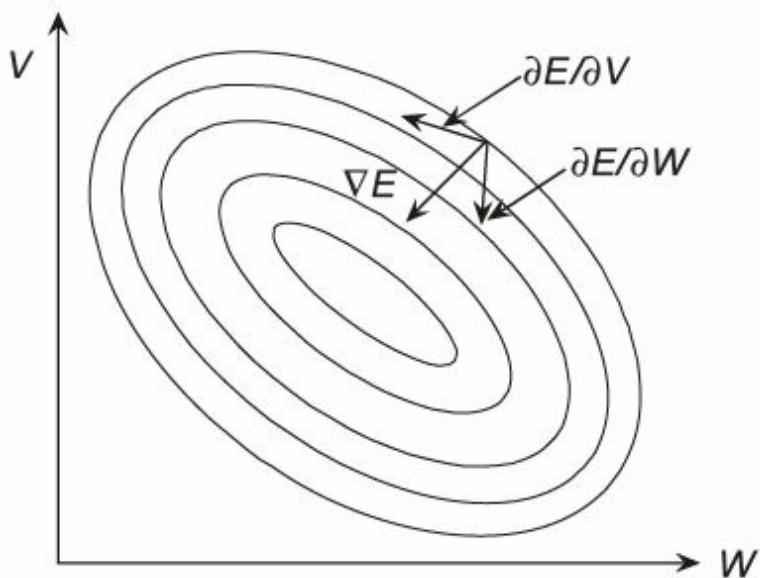
In backpropagation (BP) networks, at any given error value E , including minima regions, there are many permutations of weights which give rise to the same value of E . BP is never assured of finding global minimum as in the simple layer delta rule case. In general, for MLFF network case, the error surface will have many local minima. Sometimes, one can get stuck during the learning process on flat or near flat regions of the error surface.

At the start of the training process, gradient descent search begins at a location with error value E determined by initial weight assignments $W(0)$, $V(0)$ and the training pattern pair (Ip, Op) where, (3.25)

During training, the gradient descent (shown in Fig. 3.12) computations incrementally determine how the weights should be modified at each new location to move most rapidly in the direction opposite to the direction of steepest ascent (a steepest descent). After the incremental adjustments to the weights have been made, the location is shifted to a different E location on the error-weight surface. This process is repeated for each, training pattern (or each epoch $\{(Ip, Op)\}$,

$p = 1, 2, \dots, n_{set}$), progressively shifting the location to lower level until a threshold error value is reached or until a limit on the total number of training cycles is reached.

In moving down the error-weight surface, the path followed is generally not the ideal path. It depends on the shape of the surface and the learning rate coefficient η , which is discussed later. In general, error surface contains many



\overline{AB}

$$\bar{AB} = (V_{i+1} - V_i)\bar{i} + (W_{i+1} - W_i)\bar{j} = \Delta V\bar{i} + \Delta W\bar{j}$$

$$\bar{G} = \frac{\partial E}{\partial V}\bar{i} + \frac{\partial E}{\partial W}\bar{j}$$

$$\bar{e}_{AB} = \frac{1}{|\bar{G}|} \left\{ \frac{\partial E}{\partial V}\bar{i} + \frac{\partial E}{\partial W}\bar{j} \right\}$$

flat areas and troughs where the weights must be changed many times to realize perspective drop in error. From experience, it is found that at the places of steep slopes, larger steps can result in oscillating movements across the slopes. Because of such anomalies, it is difficult to work with steepest descent method to choose the current direction to move, thereby making progress slow and uncertain. It is known that error surface is the summation of quadratic terms which describes elliptical rather than circular contours and gradient will not point directly in the direction of the minimum (see Fig.

3.14).

Fig. 3.14 Direction of descent for two dimensional case.

Since the error surface is steeper along V dimension than W dimension, the derivative of E with respect to V is greater than the derivative with respect to W , resulting in a combined vector shifted more in the direction of V

derivative. Hence, combined vector does not point towards the true minimum. For simplicity, we assume the error surface shown in Fig. 3.12 as truly spherical. From Fig. 3.12, the vector

is written as

(3.26)

The gradient is given by

(3.27)

and hence, the unit vector in the direction of the gradient is given by (3.28)

Hence,

$$\bar{AB} = -\eta \left[\frac{\partial E}{\partial V} \bar{i} + \frac{\partial E}{\partial W} \bar{j} \right]$$

$$\frac{K}{|\bar{G}|}$$

$$\Delta V = -\eta \frac{\partial E}{\partial V}; \quad \Delta W = -\eta \frac{\partial E}{\partial W} \quad (3.31)$$

for the k th output neuron, E_k is given by

$$E_k = \frac{1}{2} (T_k - O_{ok})^2 \quad (3.32)$$

To compute $\frac{\partial E_k}{\partial W_{ik}}$, we apply chain rule of differentiation as

$$\frac{\partial E_k}{\partial W_{ik}} = \frac{\partial E_k}{\partial O_{ok}} \frac{\partial O_{ok}}{\partial I_{ok}} \frac{\partial I_{ok}}{\partial W_{ik}} \quad (3.33)$$

where,

$$\frac{\partial E_k}{\partial O_{ok}} = -(T_k - O_{ok}) \quad (3.34a)$$

and the output of the k th output neuron is given by

$$O_{ok} = \frac{1}{(1 + e^{-\lambda(I_{ok} - \theta_{ok})})} \quad (3.34b)$$

Hence,

$$\frac{\partial O_{ok}}{\partial I_{ok}} = \lambda O_{ok} (1 - O_{ok}) \quad (3.34c)$$

$$\frac{\partial I_{ok}}{\partial W_{ik}}$$

.....(3.29)

where,

$$\eta = ; K \text{ is a constant} \quad (3.30)$$

Comparing Eq. (3.26) with Eq. (3.29) we get

where, T_k is the target output of the k th output neuron and O_{ok} is the computed output of the k th output neuron.

Hence, the derivative of the sigmoidal function is a simple function of outputs. Let us evaluate

as

$$I_{ok} = W_{1k}O_{1H} + W_{2k}O_{2H} + \dots + W_{mk}O_{mH} \quad (3.35)$$

as given by Eq. (3.17)

$$\frac{\partial I_{ok}}{\partial W_{ik}} = O_{Hi} \quad (3.36)$$

Hence,

$$\frac{\partial E_k}{\partial W_{ik}} = -\lambda(T_k - O_{ok}) O_{ok} (1 - O_{ok}) O_{Hi} \quad (3.37)$$

and

$$\Delta W_{ik} = -\eta \frac{\partial E_k}{\partial W_{ik}} \quad (3.38a)$$

Writing in matrix form

$$[\Delta W] = \eta \{O\}_H \langle d \rangle \quad (3.38b)$$

$m \times n \quad m \times 1 \quad 1 \times n$

where

$$\langle d \rangle = \lambda \langle (T_k - O_{ok}) O_{ok} (1 - O_{ok}) \rangle \quad (3.39)$$

$$\frac{\partial E_k}{\partial V_{ij}}$$

$$\frac{\partial E_k}{\partial V_{ij}} = \frac{\partial E_k}{\partial O_{ok}} \frac{\partial O_{ok}}{\partial I_{ok}} \frac{\partial I_{ok}}{\partial O_{hi}} \frac{\partial O_{hi}}{\partial I_{hi}} \frac{\partial I_{hi}}{\partial V_{ij}} \quad (3.40)$$

It is already proved that

$$\frac{\partial E_k}{\partial I_{ok}} = \frac{\partial E_k}{\partial O_{ok}} \frac{\partial O_{ok}}{\partial I_{ok}} = -\lambda(T_k - O_{ok})O_{ok}(1 - O_{ok}) \quad (3.41a)$$

$$\frac{\partial I_{ok}}{\partial O_{hi}} = W_{ik} \quad (3.41b)$$

$$\frac{\partial O_{hi}}{\partial I_{hi}} = \lambda(O_{hi})(1 - O_{hi}) \quad (3.41c)$$

$$\frac{\partial I_{hi}}{\partial V_{ij}} = O_{hi} = I_{ij} \quad (3.41d)$$

Hence,

$$\frac{\partial E_k}{\partial O_{ok}} \frac{\partial O_{ok}}{\partial I_{ok}} \frac{\partial I_{ok}}{\partial O_{hi}} = -W_{ik} d_k = -e_i \quad (3.42)$$

where d_k is given by the Eq. (3.39).

Now we compute

by applying the chain rule of differentiation as

Define d^*

k as

$$d_k^* = -e_l \lambda(O_{Hk}) (1 - O_{Hk}) \quad (3.43a)$$

$$\frac{\partial E_k}{\partial V_g} = -d_i^* I_{ij} \quad (3.43b)$$

or

$$[\Delta V] = \eta \{I\}_I \langle d^* \rangle \quad (3.44)$$

where

$$d_i^* = \lambda e_l (O_{Hk}) (1 - O_{Hk}) \quad (3.45)$$

Combining Eq. (3.38) and Eq. (3.44) we get

$$[\Delta W] = \eta \{O\}_H \langle d \rangle \quad (3.46a)$$

$$[\Delta V] = \eta \{I\}_I \langle d^* \rangle \quad (3.46b)$$

η in Eq. (3.46) is known as learning rate coefficient.

3.2.7 Effect of Learning Rate ‘ η ’

Learning rate coefficient determines the size of the weight adjustments made at each iteration and hence influences the rate of convergence. Poor choice of the coefficient can result in a

failure in convergence. We should keep the coefficient constant through all the iterations for

best results. If the learning rate coefficient is too large, the search path will oscillate and converges more slowly than a direct descent as shown in Fig.

3.15(a). If the coefficient is too small, the descent will progress in small steps significantly increasing the time to converge

(see Fig. 3.15(b)). For the example illustrated in this chapter, the learning coefficient is taken as 0.9 and this seems to be optimistic (see Fig. 3.15(c)).

Jacobs (1988) has suggested the use of adaptive coefficient where the value of the learning coefficient is the function of error derivative on successive updates.

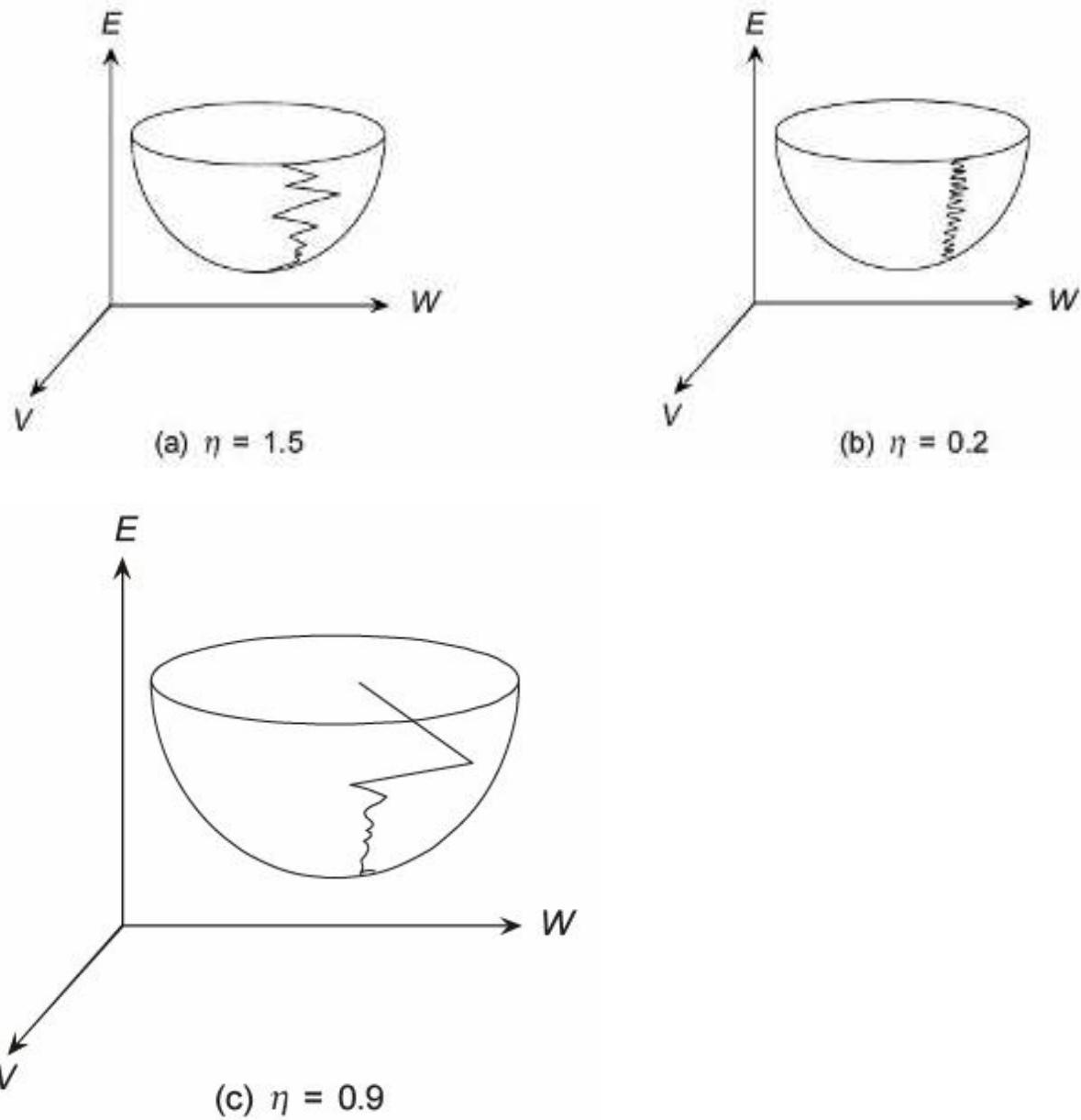


Fig. 3.15 Convergence paths for different learning coefficients.

3.2.8 Adding a Momentum Term

There is another way possible to improve the rate of convergence by adding some inertial or momentum to the gradient expression. This can be accomplished by adding a fraction of the previous weight change to the current weight change. The addition of such a term helps to smooth out the

descent path by preventing extreme changes in the gradients due to local anomalies. A commonly used update rule introduced by Rumelhart et al.

(1986) includes such a momentum term. The updatation equations used by Rumelhart are defined as

$$[\Delta W]^{t+1} = -\eta \frac{\partial E}{\partial W} + \alpha [\Delta W]^t \quad (3.47a)$$

$$[\Delta V]^{t+1} = -\eta \frac{\partial E}{\partial V} + \alpha [\Delta V]^t \quad (3.47b)$$

$$[\Delta W]^{t+1} = \eta \{O\}_H \langle d \rangle + \alpha [\Delta W]^t \quad (3.48a)$$

$$[\Delta V]^{t+1} = \eta \{I\}_I \langle d^* \rangle + \alpha [\Delta V]^t \quad (3.48b)$$

Similarly, the thresholds may be updated as

$$\{\Delta \theta\}_O^{t+1} = \eta \{d\} - \alpha \{\Delta \theta\}_O^t \quad (3.49a)$$

$$\{\Delta \theta\}_H^{t+1} = \eta \{d^*\} + \alpha \{\Delta \theta\}_H^t \quad (3.49b)$$

$$\begin{aligned} [W]^{t+1} &= [W]^t + [\Delta W]^{t+1} \\ [V]^{t+1} &= [V]^t + [\Delta V]^{t+1} \\ [\theta]_O^{t+1} &= [\theta]_O^t + [\Delta \theta]_O^{t+1} \\ [\theta]_H^{t+1} &= [\theta]_H^t + [\Delta \theta]_H^{t+1} \end{aligned} \quad (3.50)$$

where α is defined as the momentum coefficient. The value of α should be positive but less than 1. Typical values lie in the range of 0.5–0.9. But for some problems, Fahlman (1988) used a value for $\alpha = 0$ and showed it to be the best.

Hence, Eq. (3.46) are modified and written as

The weights and thresholds may be updated as

3.2.9 Backpropagation Algorithm

We have already seen the benefit of the middle-hidden layer in an artificial neural network. We understand that the hidden layer allows ANN to develop its own internal representation of this mapping. Such a rich and complex internal representation capability allows the hierarchical network to learn any mapping and not just linearly separable ones. Let us consider the three-layer network with input layer having ‘ l ’ nodes, hidden layer having ‘ m ’ nodes, and an output layer with ‘ n ’ nodes. We consider sigmoidal functions for activation functions for the hidden and output layers and linear activation function for input layer. The number of neurons in the hidden layer may be chosen to lie between 1 and 21. The basic algorithm loop structure is given as

```
Initialize the weights
Repeat
    For each training pattern
        Train on that pattern
    End
Until the error is acceptably low

Step 1: Normalize the inputs and outputs with respect to their
maximum values. It is proved that the neural networks
work better if input and outputs lie between 0-1. For
each training pair, assume there are ' $l$ ' inputs given
by  $\{I\}_l$  and ' $n$ ' outputs  $\{O\}_n$  in a normalised form.
Step 2: Assume the number of neurons in the hidden layer to lie
between  $1 < m < 21$ 
```

Algorithm 3.1 illustrates the step by step procedure of the backpropagation algorithm.

Algorithm 3.1 (Backpropagation Learning Algorithm)

Algorithm BPN()

Step 3: $[V]$ Represents the weights of synapses connecting input neurons and hidden neurons and $[W]$ represents weights of synapses connecting hidden neurons and output neurons. Initialize the weights to small random values usually from -1 to 1. For general problems, λ can be assumed as 1 and the threshold values can be taken as zero,

$$\begin{aligned}[V]^0 &= [\text{random weights}] \\ [W]^0 &= [\text{random weights}] \\ [\Delta V]^0 = [\Delta W]^0 &= [0] \end{aligned} \quad (3.51)$$

Step 4: For the training data, present one set of inputs and outputs. Present the pattern to the input layer (I_I), as inputs to the input layer. By using linear activation function, the output of the input layer may be evaluated as

$$\begin{aligned}\{O\}_I &= \{I\}_I \\ l \times 1 &\quad l \times 1\end{aligned} \quad (3.52)$$

Step 5: Compute the inputs to the hidden layer by multiplying corresponding weights of synapses as

$$\begin{aligned}\{I\}_H &= [V]^T \{O\}_I \\ m \times 1 &\quad m \times l \quad l \times 1\end{aligned} \quad (3.53)$$

Step 6: Let the hidden layer units evaluate the output using the sigmoidal function as

$$\begin{aligned}\{O\}_H &= \begin{bmatrix} * \\ * \\ 1 \\ \frac{1}{(1 + e^{-x_H})} \\ * \\ * \end{bmatrix} \\ m \times 1 &\quad m \times 1\end{aligned} \quad (3.54)$$

Step 7: Compute the inputs to the output layer by multiplying corresponding weights of synapses as

$$\begin{aligned}\{I\}_O &= [W]^T \{O\}_H \\ n \times 1 &\quad n \times m \quad m \times 1\end{aligned} \quad (3.55)$$

Step 8: Let the output layer units evaluate the output using sigmoidal function as

$$\{C\}_0 = \begin{Bmatrix} \cdot \\ \cdot \\ \vdots \\ 1 \\ \hline (1 + e^{-x_0}) \end{Bmatrix} \quad (3.56)$$

The above is the network output.

Step 9: Calculate the error and the difference between the network output and the desired output as for the i th training set as

$$E^P = \frac{\sqrt{\sum (T_j - O_{Oj})^2}}{n} \quad (3.57)$$

Step 10: Find $\{g\}$ as

$$\{d\} = \begin{Bmatrix} \cdot \\ \cdot \\ (T_k - O_{0k})O_{0k}(1 - O_{0k}) \\ \cdot \\ \cdot \end{Bmatrix}_{n \times 1} \quad (3.58)$$

Step 11: Find $[Y]$ matrix as

$$[Y] = \{O\}_H(d) \quad (3.59)$$

$$\text{Step 12: Find } [\Delta g]^{t+1} = \alpha [\Delta g]^t + n[\gamma] \quad (3.60)$$

卷之三

most likely finding for = [W] (at) (3.61%)

$\text{H}_2\text{X} + \text{H}_2\text{Y} \rightarrow \text{H}_3\text{X} + \text{H}_3\text{Y}$

$$(d') = \begin{Bmatrix} \cdot \\ \cdot \\ e_i(O_{hi}) (1 - O_{hi}) \\ \cdot \\ \cdot \end{Bmatrix} \quad (3.61b)$$

Find $[X]$ matrix as

$$[X] = \{O\}_1 \langle d' \rangle = \{I\}_1 \langle d' \rangle \\ 1 \times m \quad 1 \times 1 \quad 1 \times m \quad 1 \times 1 \quad 1 \times m$$

Step 14: Find $[\Delta V]^{t+1} = \alpha [\Delta V]^t + \eta [X]$ (3.63)
 $I \times m \quad I \times m \quad I \times m$

Step 15: Find

$$[V]^{t+1} = [V]^t + [\Delta V]^{t+1} \\ [W]^{t+1} = [W]^t + [\Delta W]^{t+1}$$
 (3.64)

Step 16: Find error rate as

$$\text{error rate} = \frac{\sum E_p}{n_{set}}$$
 (3.65)

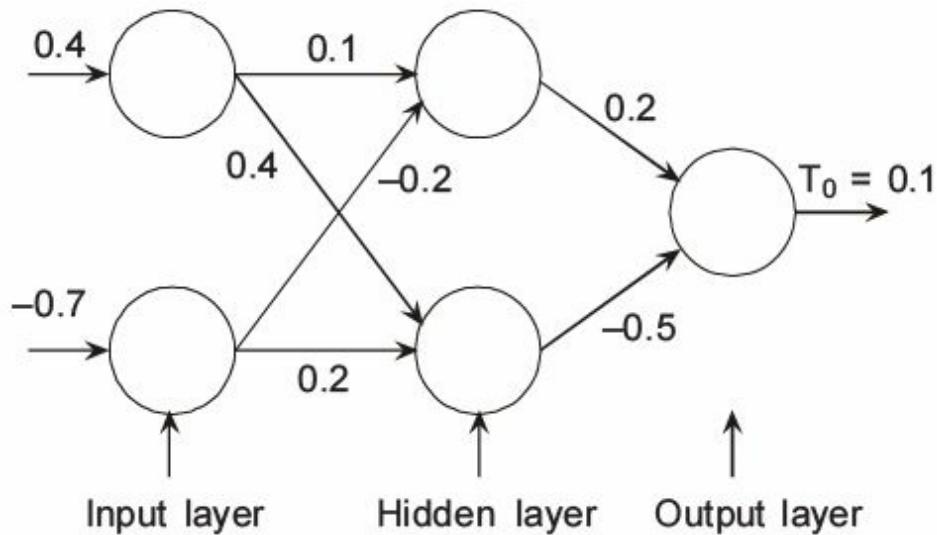
Step 17: Repeat steps 4-16 until the convergence in the error rate is less than the tolerance value.

End Algorithm BPN

Once the process converges, the final weights should be stored in a file.

Now, we are ready to test the neural net. Given any other input, we will be able to get the outputs and this is known as *inference session*. Hence, it is seen that training of an artificial neural network involves two passes. In the forward pass, the input signals propagate from the network input to the output. In the reverse pass, the calculated error signals propagate backwards through the network where they are used to adjust the weights. The calculation of output is carried out layer by layer in the forward direction.

The output of one layer in weighted manner will be the input to the next layer. In the reverse pass, the weights of the output neuron layer are adjusted first since the target value of each output neuron is available to guide the adjustment of associated weights.



3.3 ILLUSTRATION

Consider that for a particular problem there are five training sets as shown in Table 3.3.

Table 3.3 Training sets

S. no.

Inputs

Output

I 1

I 2

O

1

0.4

-0.7

0.1

2

0.3

-0.5

0.05

3

0.6

0.1

0.3

4

0.2

0.4

0.25

5

0.1

-0.2

0.12

In this problem, there are two inputs and one output and already, the values lie between

-1 to 1 and hence, there is no need to normalize the values. Assume two neurons in the hidden layer. The neural network architecture is shown in Fig.

3.16.

With the data of the first training set.

Fig. 3.16 MFNN architecture for the illustration.

$$\text{Step 1: } \{O\}_I = \{I\}_I = \begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix}$$

Step 2: Initialize the weights as

$$[V]^0 = \begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix}_{2 \times 2}; \quad [W]^0 = \begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix}_{2 \times 1}$$

Step 3: Find $\{I\}_H = [V]^T \{O\}_I$ as

$$= \begin{bmatrix} 0.1 & -0.2 \\ 0.4 & 0.2 \end{bmatrix} \begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix} = \begin{Bmatrix} 0.18 \\ 0.02 \end{Bmatrix}$$

$$\text{Step 4: } \{O\}_H = \begin{Bmatrix} \frac{1}{1 + e^{-0.18}} \\ \frac{1}{1 + e^{0.02}} \end{Bmatrix} = \begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix}$$

$$\text{Step 5: } \{I\}_O = [W]^T \{O\}_H = \langle 0.2 - 0.5 \rangle \begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix} = -0.14354$$

$$\text{Step 6: } \{O\}_O = \left(\frac{1}{1 + e^{0.14354}} \right) = 0.4642$$

Step 7: Error = $(T_O - O_O)^2 = (0.1 - 0.4642)^2 = 0.13264$

Step 8: Let us adjust the weights

First find

$$d = (T_O - O_{O1})(O_{O1})(1 - O_{O1}) \\ = (0.1 - 0.4642)(0.4642)(0.5358) = -0.09058$$

$$[Y] = [O]_H \langle d \rangle = \begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix} \langle -0.09058 \rangle = \begin{Bmatrix} -0.0493 \\ -0.0457 \end{Bmatrix}$$

Step 9:

$$[\Delta W]^1 = \alpha [\Delta W]^0 + \eta [Y] \quad (\text{assume } \eta = 0.6) \\ = \begin{Bmatrix} -0.02958 \\ -0.02742 \end{Bmatrix}$$

$$\text{Step 10: } \{e\} = [W] \{d\} = \begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix} (-0.09058) = \begin{Bmatrix} -0.018116 \\ 0.04529 \end{Bmatrix}$$

$$\text{Step 11: } \{d^*\} = \begin{Bmatrix} (-0.018116)(0.5448)(1 - 0.5448) \\ (0.04529)(0.505)(1 - 0.505) \end{Bmatrix} = \begin{Bmatrix} -0.00449 \\ 0.01132 \end{Bmatrix}$$

$$\text{Step 12: } [X] = [O]_I \langle d^* \rangle = \begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix} \langle -0.00449 \quad 0.01132 \rangle \\ = \begin{bmatrix} -0.001796 & 0.004528 \\ 0.003143 & -0.007924 \end{bmatrix}$$

$$\text{Step 13: } [\Delta V]^1 = \alpha [\Delta V]^0 + \eta [X] = \begin{bmatrix} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{bmatrix}$$

$$\text{Step 14: } [V]^1 = \begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix} + \begin{bmatrix} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{bmatrix} = \begin{bmatrix} 0.0989 & 0.04027 \\ -0.1981 & 0.19524 \end{bmatrix}$$

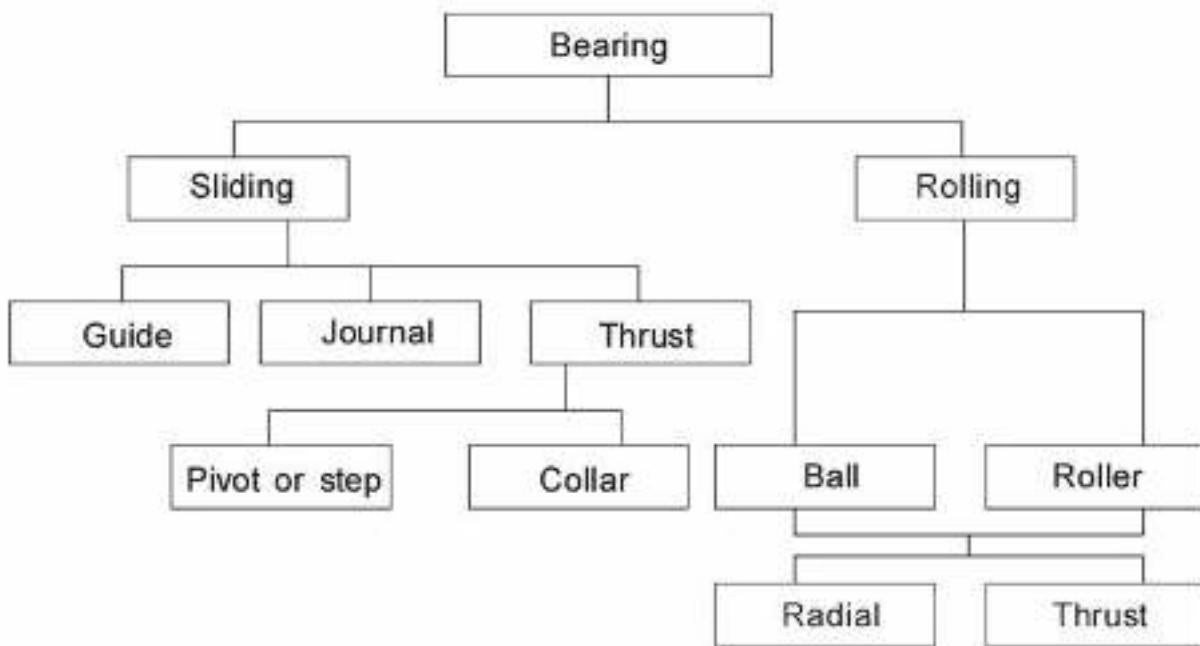
$$[W]^1 = \begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix} + \begin{Bmatrix} -0.02958 \\ -0.02742 \end{Bmatrix} = \begin{Bmatrix} 0.17042 \\ -0.52742 \end{Bmatrix}$$

Step 15: With the updated weights [V] and [W], error is calculated again and next training set is taken and the error will be adjusted.

Step 16: Iterations are carried out till we get the error less than the tolerance.

Step 17: Once weights are adjusted the network is ready for inference.

A computer program “NEURONET” is developed for training the data and inferring the results using backpropagation neural network.



3.4 APPLICATIONS

3.4.1 Design of Journal Bearing

Whenever the machine elements move, there are bearing surfaces, some of which are lubricated easily and completely, some which are lubricated incompletely and with difficulty, and some of which are not lubricated at all.

When the load on the bearing is low and the motion is slow, the bearing is lubricated with oil poured in an oil hole or applying lubricant with some other device from time to time. When either the load, or speed or both are high as in modern high-speed machinery, the lubrication by oil or by other fluids must be designed according to the conditions of operation. When there is a relative motion between two machine parts, one of which supporting the

other, then the supporting member is called *bearing*. The bearings are classified as shown in

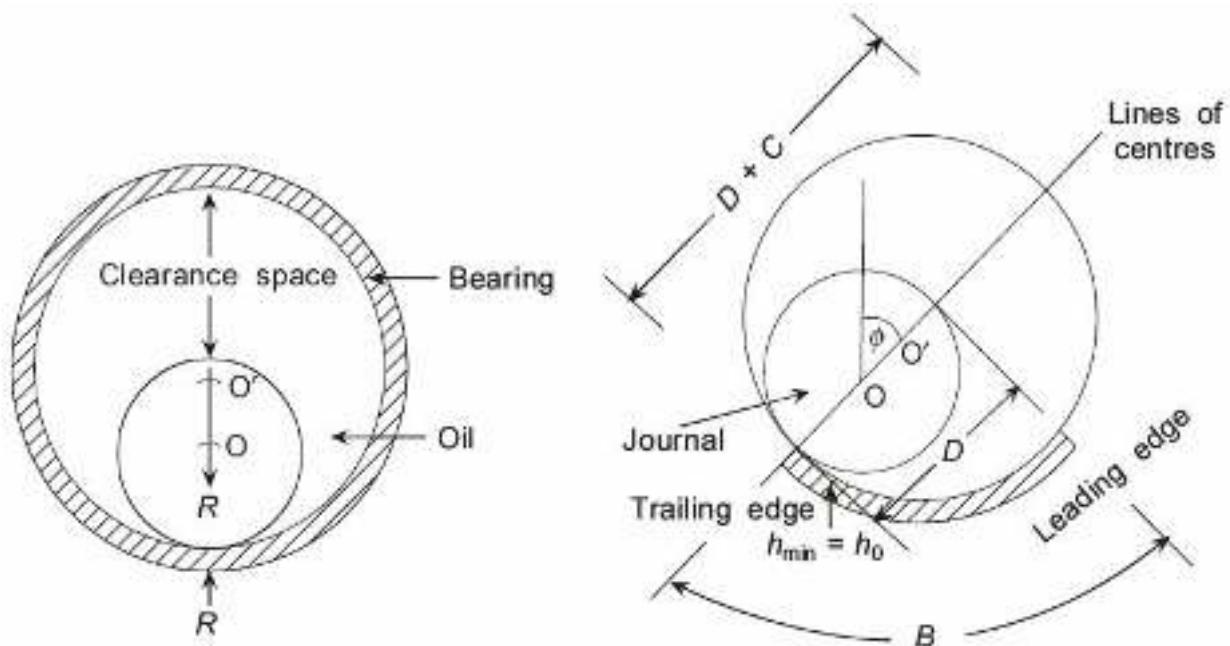
Fig. 3.17(a).

Fig. 3.17(a) Different types of bearings.

Out of the bearings given in Fig. 3.17(a), let us consider *journal bearing*.

Design of journal bearing depends on the load, speed of the journal, clearance in the bearing, length and diameter of the bearing, and the kinds of surface.

The journal bearing is shown in Fig. 3.17(b).



$$\frac{L}{D}; \quad \varepsilon; \quad \frac{h_0}{C_r}; \quad S; \quad \phi; \quad \frac{rf}{C_r}; \quad \frac{q}{rC_r n_s L}; \quad \frac{q_s}{q}; \quad \frac{\rho C \Delta t_0}{P}; \quad \frac{P}{P_{\max}}$$

Fig. 3.17b Journal bearing.

For the design of journal bearing, one has to take into account the end leakage and for various L/D ratios performance variables are plotted with

Summerfield number by Raimondi and Boyd (Kulkarni, 1997) and the results are given in the form of a table (Table 29 of Kulkarni, 1997). The variables included in the table are

where

L —length of the bearing in mm

h_0 —minimum film thickness in mm

D —diameter of the Journal bearing in mm

C —diametrical clearance

Cr —radial clearance

e —eccentricity in mm = $C/2 - h_0$

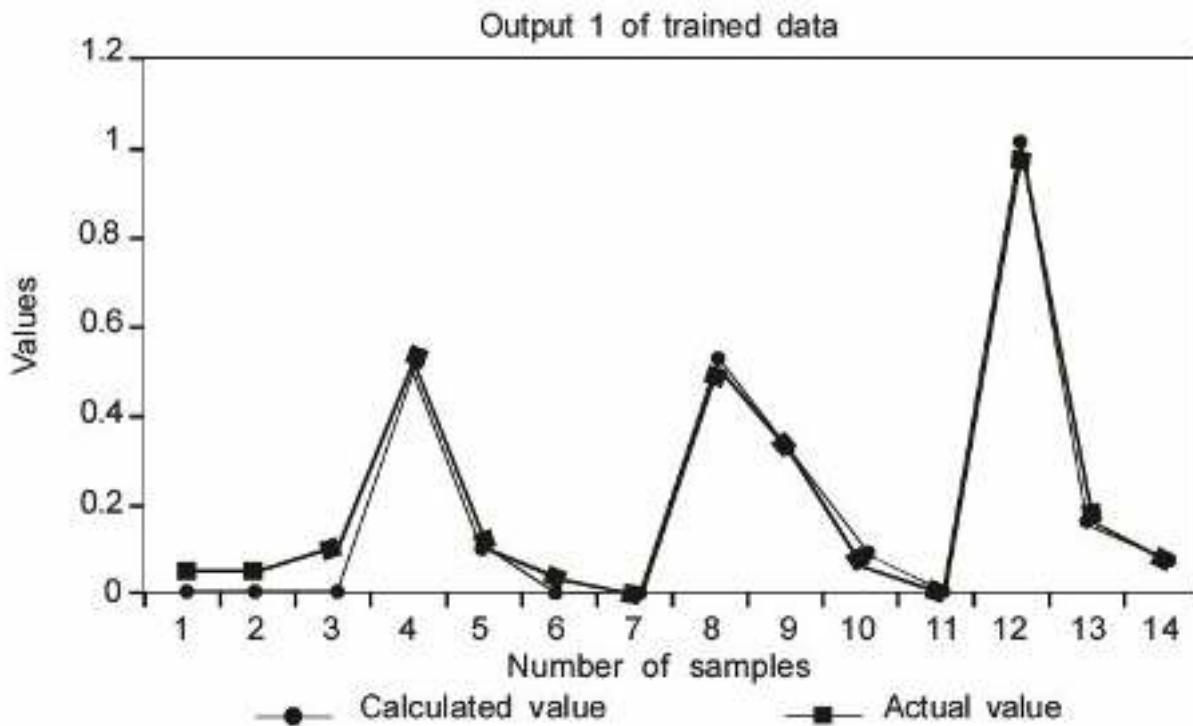
P —bearing pressure on projected area in MPa

P_{max} —maximum pressure in MPa

ns —significant speed in revolutions/s

r —radius of the bearing

$$\varepsilon = e/c$$



$S = (r/c)2\mu N/\rho$ —Sommerfeld number ρ —density of oil in kgf/cu.cm

q —oil flow through the bearing cu.m/s

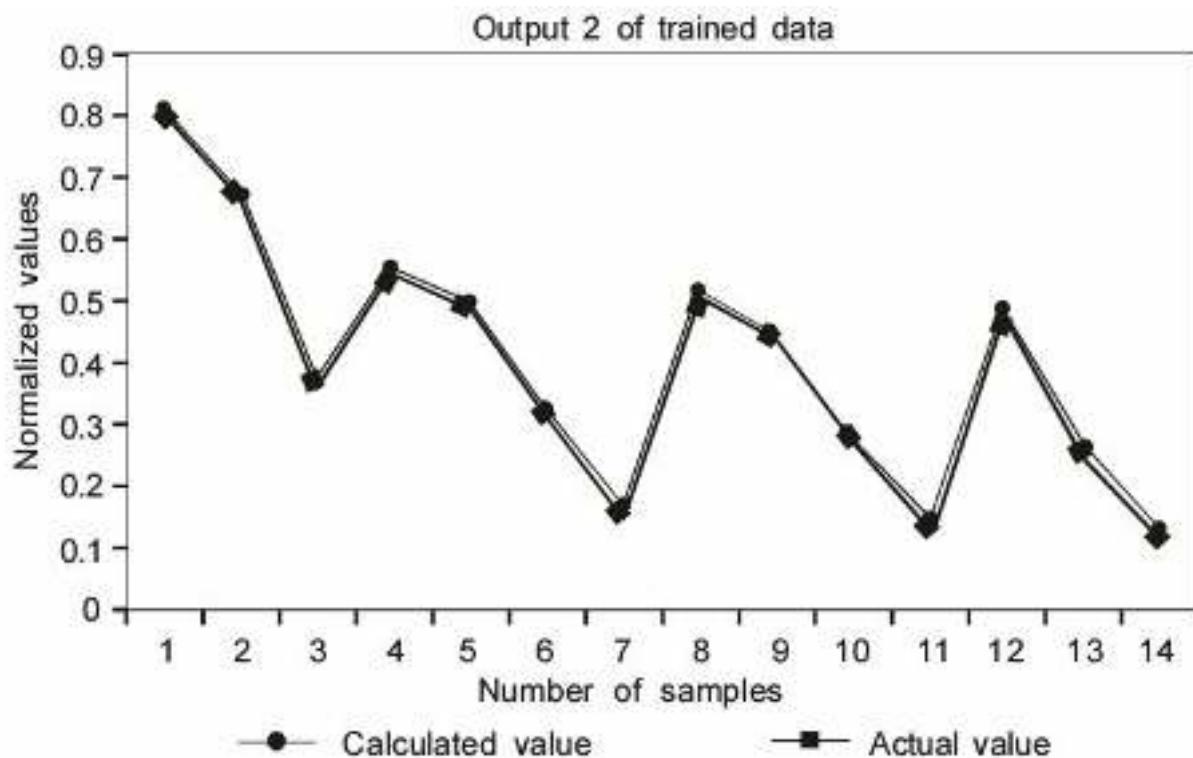
qs —axial flow of oil in cu.m/s

“NEURONET” is used to train the data and infer the results for test data. A backpropagation neural network with 8 input neurons, 8 hidden neurons, and 2 output neurons has been used. The learning of 0.6 and momentum factor of 0.9 have been used. Altogether, 5000 iterations have been performed till the error rate converges to the tolerance. The training data and testing data have been normalized so that inputs and outputs are within the range of 0–1. The training data is given in Table 3.4 and the testing data is given in Table 3.5.

Once the network is trained with the given training data the values are inferred both for training and testing the data . Figures 3.18(a) and 3.18(b) show the performance of the neural network for both the outputs for training data and it is seen that the inferred values, by using neural network, are very close to the actual values. Figures 3.19(a) and 3.19(b) show the performance

of the neural network for both the outputs for test data and it is seen that there is discrepancy at the boundaries.

Fig. 3.18(a) Comparison of values for output $1 = \rho c \Delta t o/(P \times 200)$.



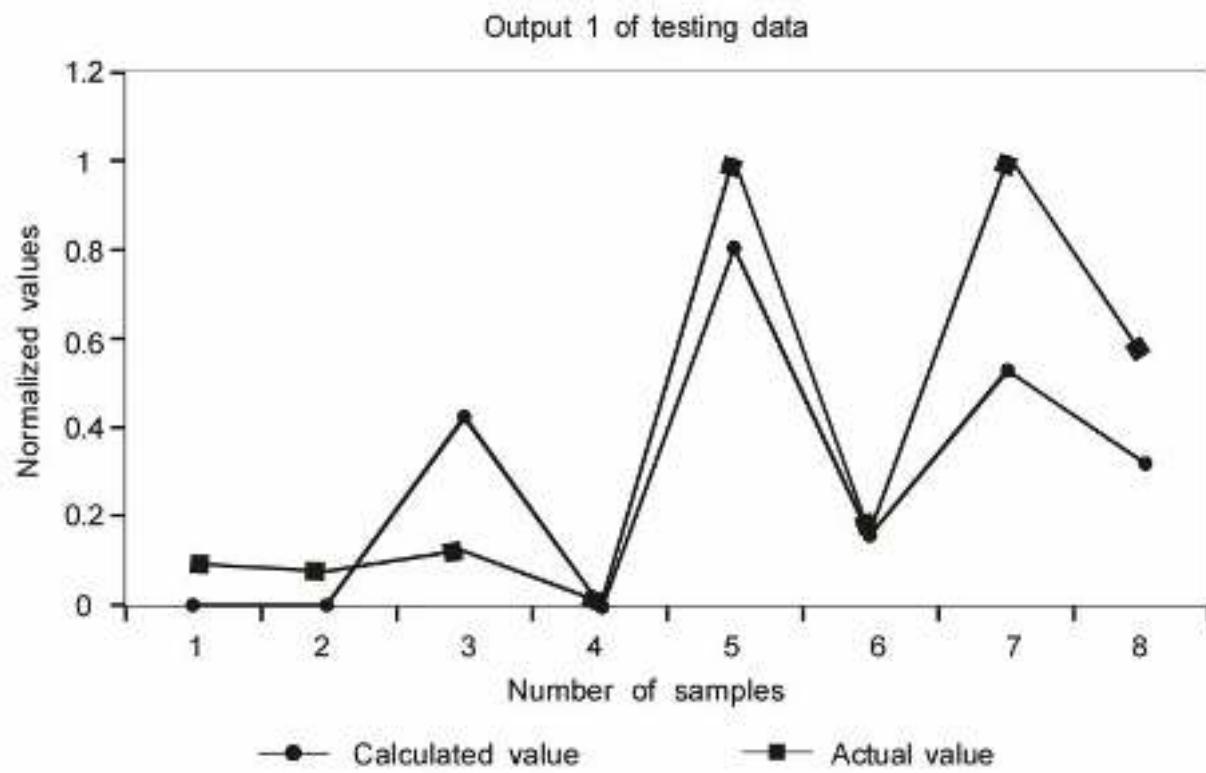


Fig. 3.18(b) Comparison of values for output $2 = P / P_{\max}$.

Fig. 3.19(a) Comparison of values for output $1 = \rho c \Delta t o/(P \times 200)$.

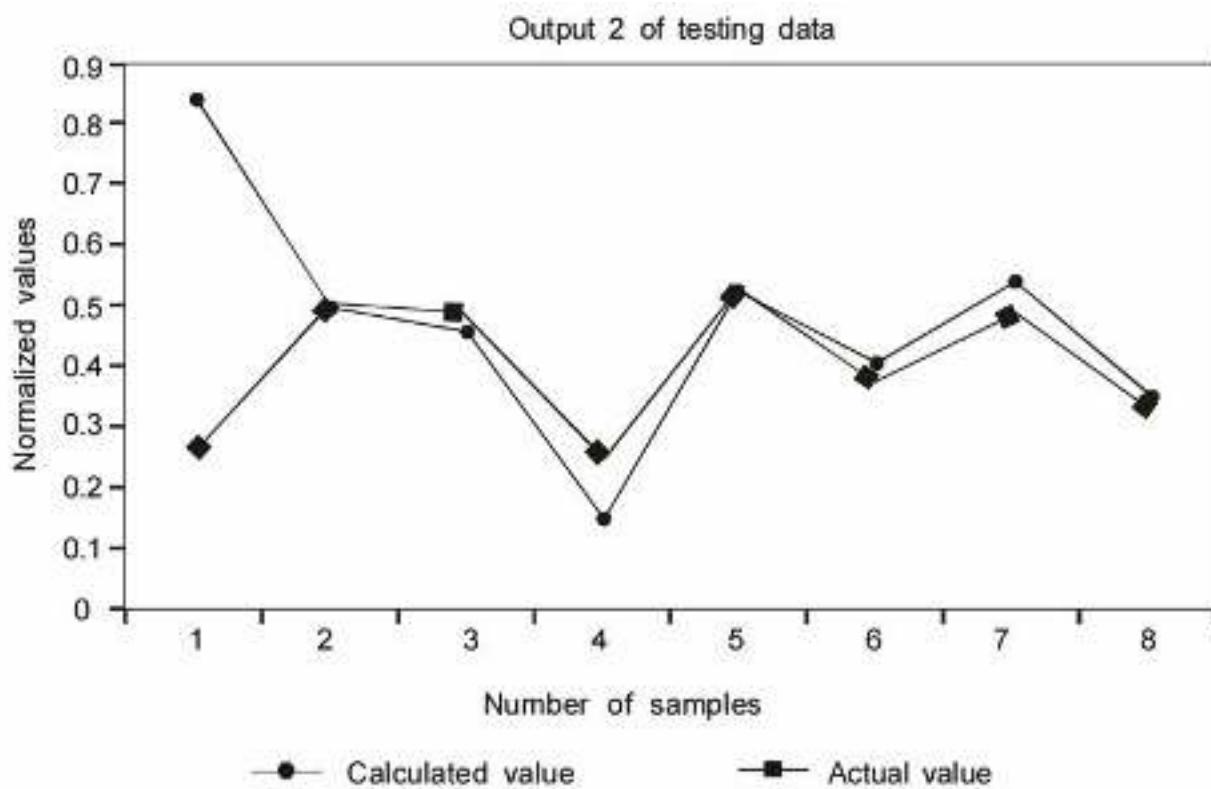


Fig. 3.19(b) Comparison of values for output $2 = P/P_{\max}$.

Table 3.4 Training data (journal bearing problem)

1

2

3

4

5

6

7

8

9

10

Actual Calculated

Actual Calculated

1

0.2

0.8

0.006

0.747

0.0064

0.283

0

0.057

0.0

0.814

0.81

1

0.6

0.4

0.0019

0.603

0.003

0.156

0

0.04865

0.0

0.667

0.67

1

0.9

0.1

0.0005

0.351

0.0018

0.041

0

0.116

0.0

0.358

0.36

0.1

0.1

0.9

0.065

0.883

0.066

0.337

0.15

0.53

0.53

0.54

0.55

0.1

0.4

0.6

0.013

0.701

0.014

0.399

0.497

0.12

0.12

0.489

0.48

0.1

0.8

0.2

0.002

0.402

0.004

0.462

0.892

0.04

0.01

0.313

0.31

0.1

0.97

0.03

0.0002

0.773

0.0012

0.48

0.973

0.013

0

0.152

0.15

0.05

0.2

0.8

0.1

0.832

0.102

0.372

0.318

0.52

0.52

0.506

0.50

0.05

0.4

0.6

0.0385

0.682

0.0425

0.429

0.552

0.343

0.34

0.44

0.44

0.05

0.8

0.2

0.0046

0.37

0.0081

0.541

0.874

0.067

0.0800

0.267

0.26

0.05

0.97

0.03

0.0

0.152

0.0015

0.588

0.98

0.0125

0.03

0.0126

0.12

0.025

0.4

0.6

0.14

0.676

0.152

0.437

0.517

1

1

0.415

0.47

0.025

0.8

0.2

0.014

0.344

0.022

0.56

0.884

0.17

0.17

0.24

0.24

0.025

0.97

0.3

0.01

0.135

0.0023

0.612

0.984

0.087

0.08

0.108

0.11

...

1. $L/10D$; 2. ε ; 3. $\frac{h_0}{C_f}$; 4. $S/20$; 5. $\phi/90$; 6. $r\mu/(400C_r)$

7. $q/(10rC_f n_s L)$; 8. q_d/q ; 9. $\rho C \Delta t_0 / (200P)$; 10. P/P_{\max}

Table 3.5 Test data (journal bearing problem)

1

2

3

4

5

6

7

8

9

10

Actual Calculated

Actual Calculated

1

0.1

0.9

0.012

0.767

0.012

0.303

0

0.099

0

0.26

0.84

1

0.8

0.2

0.001

0.468

0.0024

0.0706

0

0.0795

0

0.495

0.50

0.1

0.2

0.6

0.013

0.701

0.014

0.391

0.497

0.125

0.43

0.485

0.46

0.1

0.9

0.03

0.009

0.293

0.0026

0.479

0.919

0.0045

0

0.247

0.15

0.05

0.1

0.9

0.215

0.906

0.214

0.343

0.173

1.0

0.8

0.523

0.53

0.05

0.6

0.4

0.319

0.534

0.020

0.485

0.73

0.165

0.16

0.365

0.41

0.025

0.2

0.8

0.371

0.834

0.39

0.376

0.33

1.0

0.52

0.489

0.54

0.025

0.6

0.4

0.05

0.519

0.066

0.499

0.746

0.55

0.32

0.334

0.36

3.4.2 Classification of Soil

The objectives of soil exploration and classification are to find the suitability of the soil for the construction of different structures, embankments, subgrades, and wearing surfaces. Soil seldom exists in nature separately as sand, gravel or any other single component but is usually found as mixture with varying proportions of particles of different sizes. Sandy clay has most of the properties of clay but contains significant amount of sand. Many different classification systems of soil exist, out of which many engineers use Bureau of Indian Standards system. The soil has to be classified so as to find its suitability for the construction of dams, roads, highways, and for buildings.

Bureau of Indian Standards system (Reference 8) is based on those characteristics of the soil which indicates how it will behave as a construction material. The classification is given

in Fig. 3.20.

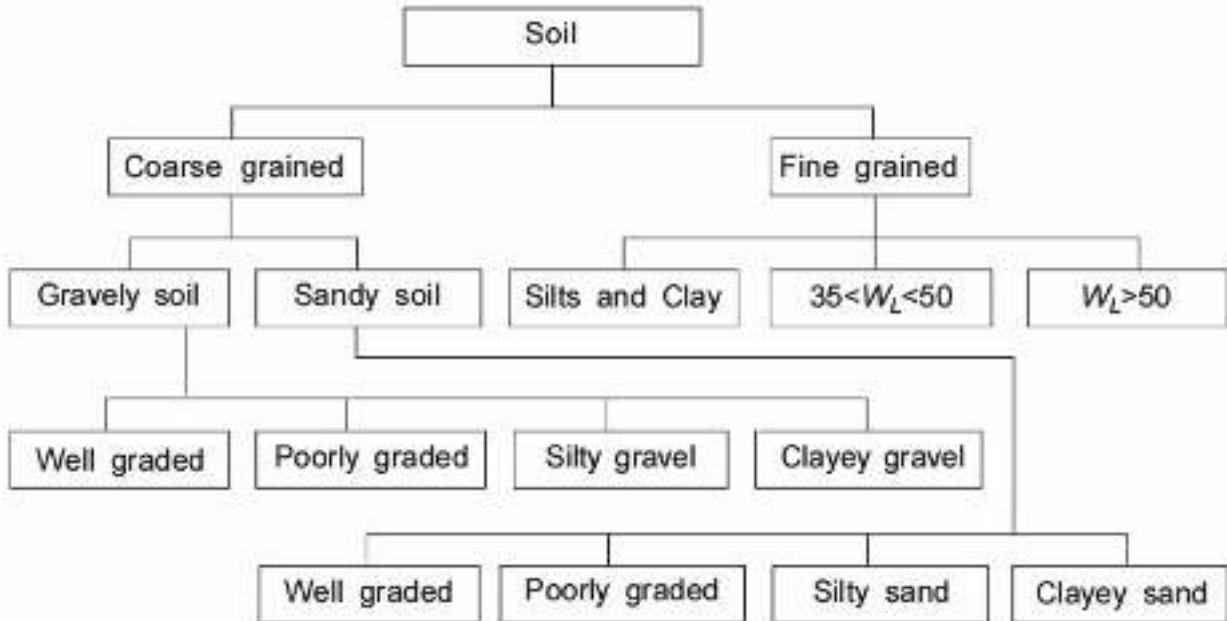


Fig. 3.20 Classification of soil.

The architecture used for grouping the soil is 6-6-1 with 6 input neurons, 6 hidden neurons and 1 output neuron. The six inputs represent colour of the soil, percentage of gravel, percentage of sand, percentage of fine grained particles, liquid limit WL , and plastic limit WP . The output represents the IS classification of the soil. The codes taken for IS classification are 0.1 for clayey sand (SC), 0.2 for clay with medium compressibility (CI), 0.3 for clay with low compressibility (CL), and 0.6 for silt with medium compressibility (MI). The codes taken for colour of the soil are 0.1 for brown, 0.2 for brownish grey , 0.3 for greyish brown, 0.5 for reddish yellow, and 0.7 for yellowish red. All the inputs and output values are normalized. Thirty training sets are taken and the network is trained for 250 iterations with the learning rate and momentum values as 0.6 and 0.9 respectively. The error rate at the end of 250th iteration was found to be 0.0121. Tables 3.6 and 3.7 give sample training data and the test data. The rejection rate was found to be nil for trained set and 8% for untrained set.

Table 3.6 Sample training data for soil classification

(Values in brackets show actual values)

(Fine

(Liquid

(Plastic

Colour of

(Gravel %)

(Sand %)

I.S.

grained

limit %)

limit %)

the soil

18

82

classification

particles %)/84

59

34

0.2

0.111

0.682

0.5

0.508

0.529

0.1(0.1)

0.2

0

0.536

0.666

0.576

0.647

0.292(0.3)

0.1

0

0.329

0.869

0.711

0.735

0.203(0.2)

0.3

0

0.756

0.452

0.491

0.529

0.129(0.1)

0.5

0

0.585

0.619

0.627

0.852

0.608(0.6)

0.2

0

0.524

0.678

0.576

0.676

0.328(0.3)

0.5

0

0.573

0.63

0.61

0.823

0.595(0.6)

0.2

0

0.512

0.69

0.576

0.647

0.296(0.3)

0.1

0

0.341

0.857

0.694

0.705

0.193(0.2)

0.2

0

0.548

0.654

0.576

0.647

0.289(0.3)

0.7

0

0.353

0.845

0.677

1

0.614(0.6)

0.5

0

0.585

0.619

0.61

0.823

0.594(0.6)

0.2

0.222

0.682

0.476

0.508

0.529

0.0842(0.1)

0.1

0

0.317

0.88

0.728

0.764

0.211(0.2)

0.1

0

0.341

0.857

0.711
0.735
0.21(0.2)
0.2
0.166
0.67
0.5
0.525
0.558
0.112(0.1)

...

Table 3.7 Inference results for soil classification (untrained data) 0.1

0
0.304
0.892
0.728
0.754
0.204(0.2)
0.2
0

0.536

0.666

0.576

0.647

0.292(0.3)

0.5

0

0.597

0.607

0.61

.823

0.592(0.6)

0.1

0

0.951

0.261

0.627

0.676

0.0912(0.1)

0.2

0

0.512

0.69

0.593

0.676

0.326(0.3)

0.1

0

0.926

0.285

0.627

0.676

0.0961(0.1)

0.2

0.222

0.658

0.5

0.525

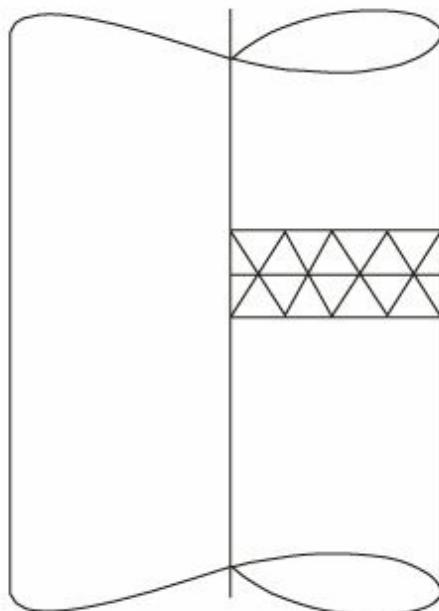
0.529

0.0887(0.1)

0.1
0
0.341
0.857
0.728
0.735
0.206(0.2)

3.4.3 Hot Extrusion of Steel

Hot metal forming has become an attractive process in industry due to its ability to achieve energy and material savings, quality improvement, and development of homogeneous properties throughout the component. In spite of these advantages, the process is rather complicated as it requires careful control and inspection to verify that the final component has the requisite mechanical properties. It in turn demands a peruse simulation and analysis of



the process. The effect of various process parameters such as die angle and velocity of die on factors such as forging head, equivalent stress, and equivalent strain are simulated with a finite element code for hot extrusion of CR45 steel billet. The initial temperature of the billet is 1050 degree centigrade and it is extruded in dies which are maintained at 100 degree centigrade. Hansraj and others (1992) have simulated forward hot extrusion of transmission shaft with various die angles and punch velocities using the six noded triangular elements (Zienkiewicz, 1992). A solid cylinder at 1050

degree centigrade with dies is kept at 100 degree Centigrade. The geometry is axi-symmetric in nature. So, only one half of the part is simulated as shown in Fig. 3.21.

Fig. 3.21 Finite element idealization.

The variation of forging forces with different die angles and punch velocities along with the equivalent strain, equivalent stress, and equivalent strain rate are obtained using finite element analysis and tabulated as shown in Table 3.8. Neural networks are nowadays applied to the simulation of hot extrusion due to the fact that finite element solutions of hot forging are very complex and require a lot of computer time. In a factory where decisions need to be taken quickly, it is advantageous to have a system that can advise an engineer without resorting to large calculations. Hence, one can explore the potential of both the finite element simulations and neural network modelling of hot extrusions so that process decisions can be taken in real time. The neural network architecture consists of three layers—two input neurons, eight neurons in a hidden layer, and five output neurons. Both inputs

and outputs are normalized such that values lie between 0 and 1. The learning rate and momentum factor are taken as 0.6 and 0.8 respectively. The error rate reaches the tolerance value at 3500 iterations. The inputs are die angle and punch velocity and the outputs consist of forging load, maximum equivalent strain, maximum equivalent stress, maximum normal velocity, and equivalent strain rate. Twenty-four data sets are taken for training and eighteen data sets are taken for testing. After the neural network is trained, it is used for inferring. The values obtained using backpropagation neural

network using the program (**NEURONET**) for the training data are given in Table 3.8 and for the test data are given in

Table 3.9. The network predictions for one output, that is, for forging load are shown in Fig. 3.22.

Table 3.8 Trained data (hot extrusion of steel)

Input

Output

Equi max

Max normal

Die

Die vel

Forging

Equivalent

stress

velocity in

ang/90

mm/s/200

load/400

max strain/6

(kg/sq.mm)/0.3

mm/s/5000

actual

calculated

actual

calculated

actual

calculated

actual

calculated

actual

1

0.333

0.875

0.83

0.726

0.74

0.792

0.82

0.683

0.72

0.355

1

0.5

0.6175

0.67

0.843

0.78

0.807

0.76

0.804

0.69

0.486

1

0.66

0.45

0.48

0.76

0.63

0.804

0.89

0.609

0.62

0.66

1

0.83

0.325

0.33

0.252

0.35

0.9

0.95

0.661

0.5

0.925

1

1.0

0.3

0.29

0.5

0.33

0.795

0.72

0.736

0.72

0.359

0.83

0.333

0.837

0.80

0.861

0.72

0.838

0.78

0.7012

0.69

0.442

0.83

0.5

0.55

0.61

0.465

0.63

0.748

0.83

0.514

0.63

0.149

0.83

0.666

0.43

0.42

0.366

0.35

0.881

0.93

0.5536

0.58

0.231

0.83

0.83

0.28

0.32

0.27

0.27

0.78

0.74

0.7528

0.69

0.302

0.83

1.0

0.25

0.26

0.25

0.32

0.07

0.24

0.712

0.79

0.335

0.667

0.667

0.375

0.36

0.363

0.22

0.853

0.77

0.5728

0.65

0.229

0.5

0.333

0.6825

0.7

0.318

0.38

0.789

0.8

0.689

0.6

0.32

0.5

0.5

0.455

0.45

0.24

0.19

0.758

0.79

0.617

0.62

0.2044

0.5

0.66

0.325

0.28

0.148

0.19

0.683

0.59

0.785

0.63

0.175

0.5

0.83

0.195

0.17

0.15

0.18

0.64

0.59

0.5252

0.5

0.1214

0.5

1.0

0.125

0.12

0.128

0.14

0.625

0.59

0.369

0.35

0.1365

0.333

0.333

0.65

0.63

0.271

0.22

0.758

0.76

0.684

0.64

0.247

0.333

0.5

0.275

0.37

0.125

0.18

0.608

0.72

0.4372

0.6

0.0827

0.333

0.66

0.3175

0.22

0.268

0.16

0.751

0.68

0.709

0.48

0.233

0.333

0.83

0.084

0.14

0.113

0.13

0.546

0.62

0.2328

0.36

0.041

0.333

1.0

0.066

0.1

0.108

0.1

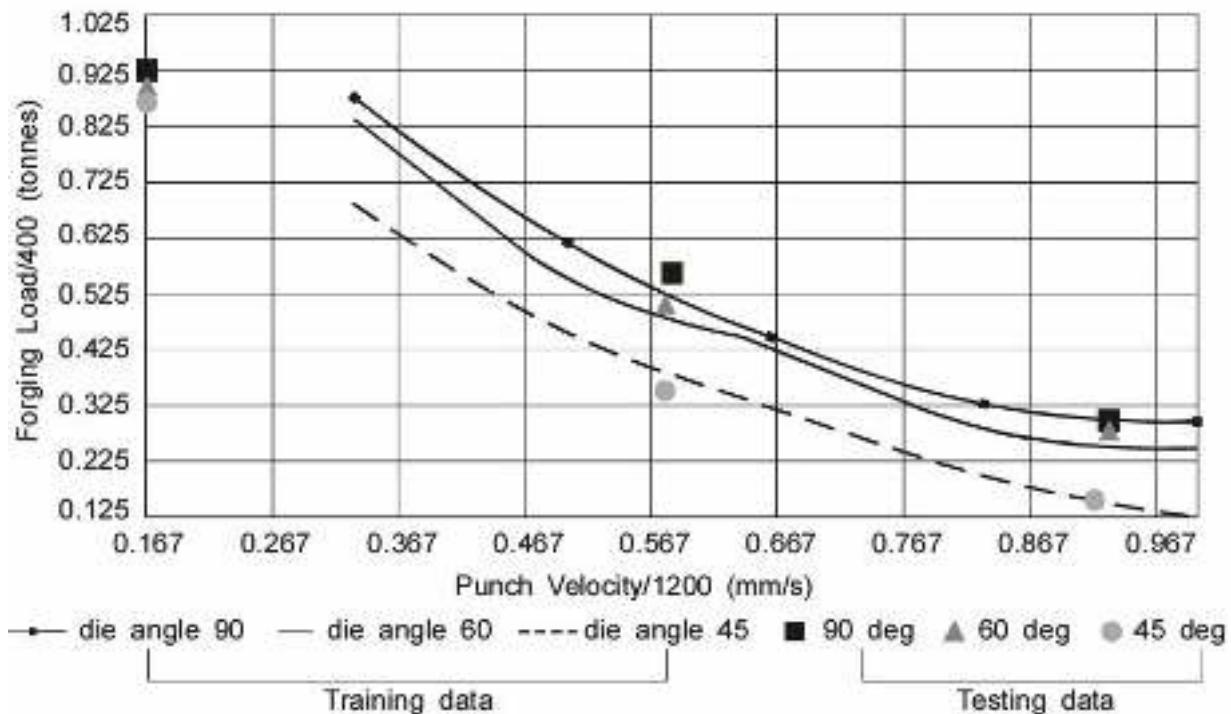
0.53

0.56

0.228

0.27

0.0429



0.167

0.333

0.62

0.63

0.275

0.29

0.686

0.71

0.688

0.72

0.134

0.167

0.5

0.43

0.4

0.276

0.26

0.65

0.66

0.605

0..63

0.081

0.167

0.666

0.2925

0.27

0.21

0.23

0.673

0.57

0.613

0.53

0.108

.

Fig. 3.22 Punch velocity versus forging load.

Table 3.9 Testing data (hot extrusion of steel)

Input

Output

Max

Max

Strain

Die

Velocity

Load

Eq.strain

stress

velocity

rate

angle/90

1200

400

6

300

5000

1200

1.0

0.167

0.93

0.63

0.91

0.74

0.36

1.0

0.583

0.57

0.74

0.80

0.66

0.43

1.0

0.916

0.3

0.31

0.91

0.63

0.50

0.83

0.167

0.91

0.64

0.87

0.73

0.33

0.83

0.583

0.5

0.49

0.9

0.59

0.43

0.83

0.916

0.2

0.3

0.43

0.70

0.23

0.667

0.167

0.9

0.64

0.82

0.70

0.28

0.667

0.583

0.44

0.25

0.88

0.59

0.31

0.667

0.916

0.21

0.25

0.38

0.67

0.19

0.5

0.167

0.88

0.59

0.73

0.67

0.20

0.5

0.583

0.36

0.18

0.67

0.64

0.17

0.5

0.986

0.14

0.16

0.59

0.42

0.15

0.333

0.167

0.86

0.47

0.64

0.67

0.14

0.333

0.583

0.28

0.17

0.70

0.55

0.15

0.333

0.916

0.12

0.11

0.58

0.31

0.06

0.167

0.167

0.87

0.45

0.6

0.78

0.14

0.167

0.583

0.33

0.24

0.61

0.57

0.08

0.167

0.916

0.17

0.18

0.48

0.41

0.03

$$\frac{1.5}{(N_1^2 + N_2^2 + \dots + N_m^2)}$$

3.5 EFFECT OF TUNING PARAMETERS OF THE BACKPROPAGATION NEURAL NETWORK

A proper selection of tuning parameters such as momentum factor, learning coefficient, sigmoidal gain, and threshold value are required for efficient learning and designing of a stable

network. Weight adjustment is made based on the momentum method. Using this, the network tends to follow the bottom of narrow gutters in the error surface (if it exists) rather than

crossing rapidly from side to side. The momentum factor has a significant role in deciding

the values of learning rate that will produce rapid learning. It determines the step size of

change in weights or biases. If momentum factor is zero, the smoothening is minimum

and the entire weight adjustment comes from the newly calculated change. If momentum factor is one, new adjustment is ignored and the previous one is repeated. Between 0 and 1 is a region where the weight adjustment is smoothed by an amount proportional to the momentum factor. Momentum factor of 0.9 has been found to be suitable for most of the problems. The role of momentum factor is to increase the speed of learning without leading to oscillations. The momentum term effectively filters out high frequency variations of the error surface in the weight space, since it adds the effect of past weight changes on the current direction of movement in the weight space.

The choice of learning coefficient is a tricky task in backpropagation algorithm. The range of learning coefficient that will produce rapid training depends on the number and types of input patterns. An empirical formula to select learning coefficient has been suggested by Eaton and Oliver (1992) is given as

$$\eta =$$

$$\dots\dots\dots(3.66)$$

where N_1 is the number of patterns of type 1 and m is the number of different pattern types.

$$\frac{1}{(1 + e^{-\lambda(I+\theta)})}$$

It may be difficult to spot “similar” patterns under such circumstances and the target output is used to determine a pattern’s type. The use of output defined types results in small value of learning coefficient which produces slower but stable training. The largest value of learning coefficient is

obtained if each pattern considered is a separate type. The optimum value lies between these extremes.

A better selection of learning rate is possible if more information is available about

the input patterns. This coefficient must be smaller where there are many input patterns

as compared to when they are few because the step length is controlled by the learning coefficient. If the learning coefficient is large, that is, greater than 0.5, the weights are changed drastically but this may cause optimum combination of weights to be “overshot” resulting in oscillations about the optimum. If the learning is small, that is, less than 0.2, the weights are changed in small increments, thus, causing the system to converge more slowly but with little oscillation. The learning rate has to be chosen as high as possible to allow fast learning without leading to oscillations. The learning rate and error rate for soil mechanics problem are shown in Fig. 3.23 and it is seen that optimum value of learning rate is 0.6.

Sigmoidal gain

If sigmoidal function is selected, the input-output relationship of the neuron can be set as

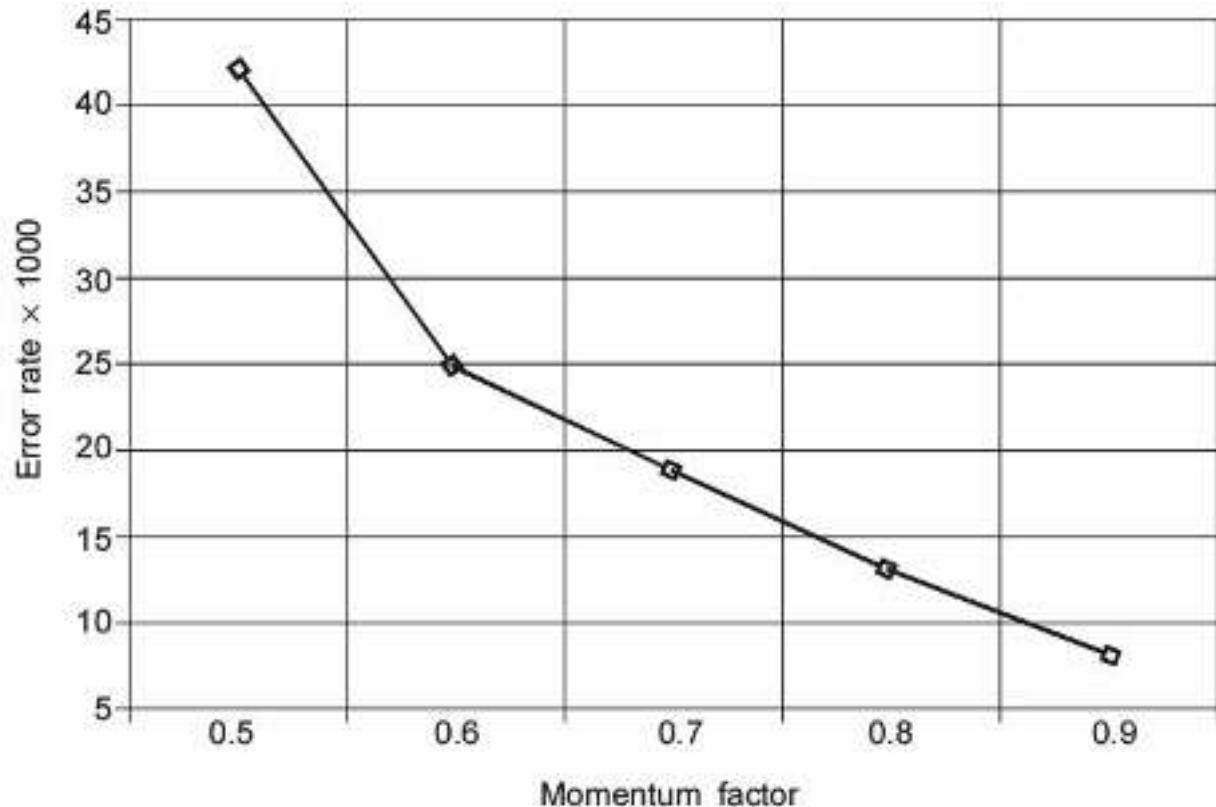
$$O =$$

$$\dots\dots\dots(3.67)$$

where λ is a scaling factor known as *sigmoidal gain*. As shown in Fig. 3.5(a), as the scaling factor increases, it is seen that the input-output characteristic of the analog neuron approaches that of the two-state neuron or the activation function approaches the Sat function.

To get a graded output or a binary output, scaling factor can be varied. The value of sigmoidal gain also affects backpropagation. Improper combinations of the scaling factor, learning rate, and momentum factor might lead to over

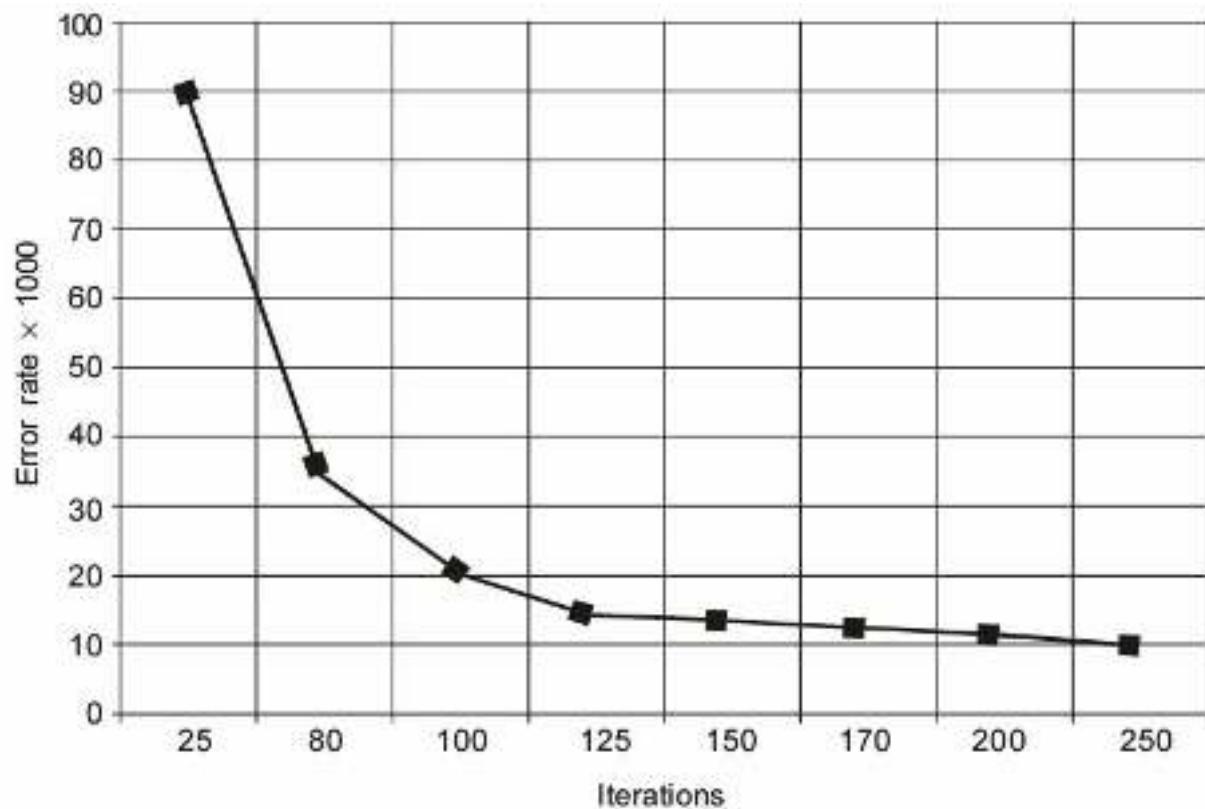
correction and poor convergence. To get graded output, as the scaling factor is increased, learning rate and momentum factor have to be decreased in order to prevent oscillations.



Threshold value

θ in Eq. (3.67) is commonly called as *threshold value* of a neuron, or the *bias* or the *noise factor*. A neuron fires or generates an output if the weighted sum of the input exceeds the threshold value. One method is to simply assign a small value to it and not to change it during training. The other method is to initially choose some random values and change them during training. It is hard to say which method is more efficient. For some problems, assigning a value to the threshold value and holding it constant is preferable. For soil classification problem, the threshold value is taken as zero. Figures 3.23 to 3.27 show the variation of error rate with respect to learning rate, momentum factor, number of iterations, number of hidden neurons, and the number of hidden layers. It is seen that for the soil classification problem, optimum values for the learning rate and momentum factor are taken as 0.6 and 0.9.

Fig. 3.24 Momentum factor versus error rate.



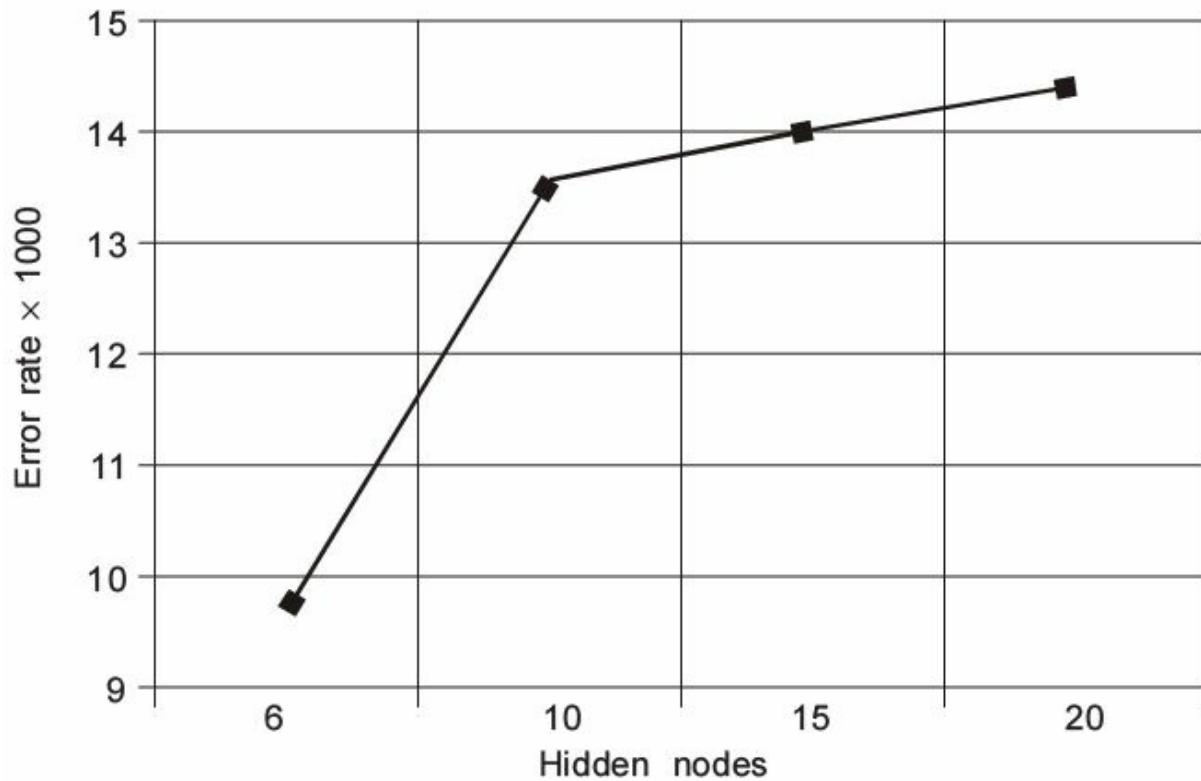


Fig. 3.25 Iterations versus error rate.

Fig. 3.26 Hidden nodes versus error rate.

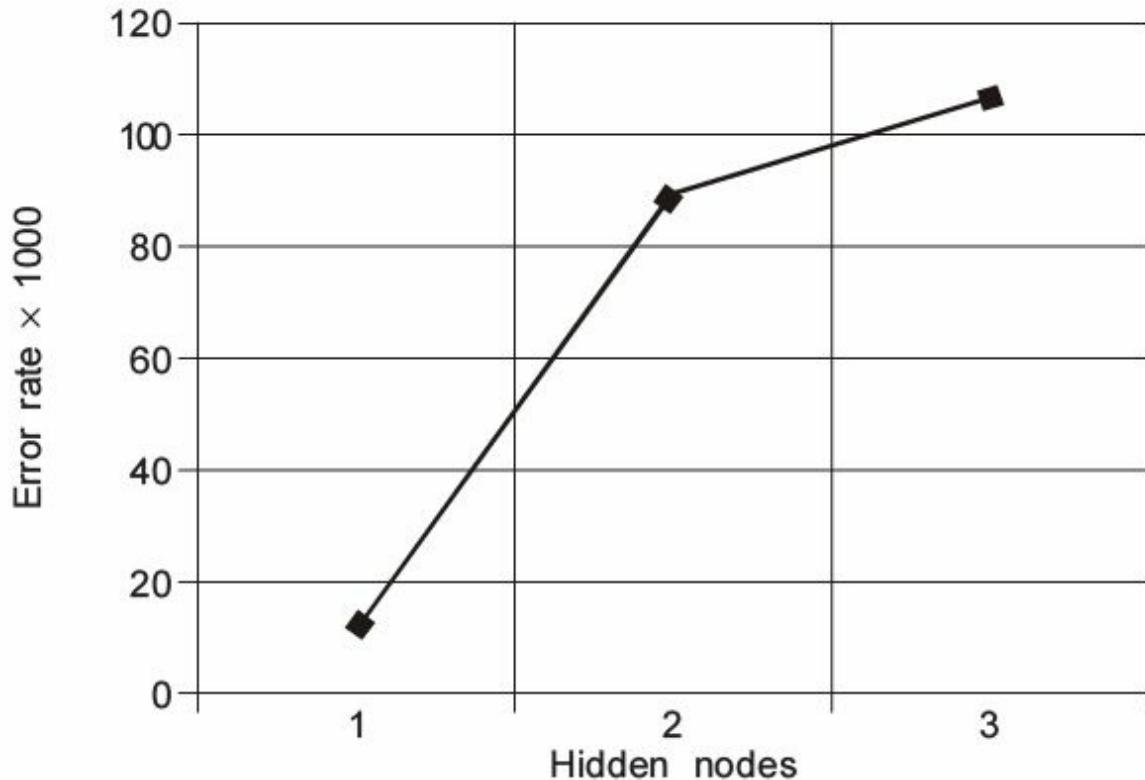


Fig. 3.27 Hidden nodes versus error rate.

$$\frac{l_2}{(l_1 + l_3)}$$

$$\frac{10T}{(l_1 + l_3)}$$

3.6 SELECTION OF VARIOUS PARAMETERS IN BPN

3.6.1 Number of Hidden Nodes

The problem at hand decides the number of nodes in the first and third layers.

There is no general criterion about deciding the number of hidden nodes. The guiding criterion is to select the minimum nodes which would not impair the network performance so that the memory demand for storing the

weights can be kept minimum. Mirchandani and Cao (1989) have proved that the number of separable regions in the input space, M , is a function of the number of hidden nodes H in BPN and $H = M - 1$. Huang and Huang (1991) argue that in terms of learning efficiency, the optimal number of hidden neurons to realize a binary valued function is experimentally found out to be $H = K - 1$, where K is the number of elements in the learning set. They have also proved that this is the least upper bound on the number of hidden neurons needed to realize an arbitrary real valued function defined by set with K elements.

When the number of hidden nodes is equal to the number of training patterns, the learning could be fastest (weight vectors associated with each input and output pair can be algebraically combined). In such cases, BPN simply remembers training patterns losing all generalization capabilities. Hence, as far as generalization is concerned, the number of hidden nodes should be small compared to the number of training patterns (say 10:1). There are results available in literature, which relate the dimensionality of the problems and the network size based on the Vapnik–Chervonenkis dimension (VCdim) of probability theory. These results also indirectly help in the selection of number of hidden nodes for a given number of training patterns. A rough estimate of VCdim for BPN is given by its number of weights which is equal to $l_1 \cdot l_2 + l_2 \cdot l_3$, where l_1 and l_3 denote input and output nodes and l_2

denotes hidden nodes. Assume the training samples T to be greater than VCdim. Now if we accept the ratio of 10:1

$$10 \cdot T =$$

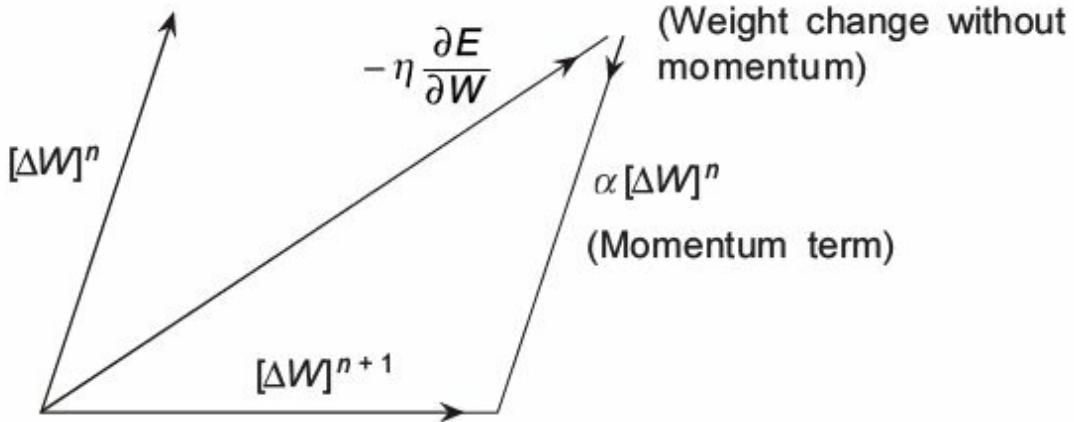
$$\dots \quad (3.68a)$$

$$l_2 =$$

$$\dots \quad (3.68b)$$

which yields the value for l_2 .

$$[\Delta W]^{n+1} = -\eta \frac{\partial E}{\partial W} + \alpha [\Delta W]^n \quad (3.69)$$



3.6.2 Momentum Coefficient α

We have already seen that another method of reducing the training time is the use of momentum factor because it enhances the training process. The influence of the momentum on weight change is shown in Fig. 3.28 where

Fig. 3.28 Influence of momentum term on weight change.

This process works well for many problems but not so well in others. The momentum also overcomes the effect of local minima. The use of momentum term will often carry a weight change process through one or local minima and get it into global minima. This is perhaps its most important function.

There is a substantial number of advanced algorithms or other procedures that have been proposed as a means to speed up the training of backpropagation networks. Sejnowski and Rosenberg (1987) proposed a similar momentum method that used exponential smoothening. Parker (1987) proposed a method called “second order” backpropagation that uses second derivative to produce more accurate estimation of weight change. The computational requirements were great and were generally viewed as not being cost effective compared to other methods. The main drawback of the BPN is the long and sometimes uncertain training time. This may be due to poor choice of training coefficients and the initial random distribution of

weights. However, in most cases failure to train BPN is usually due to local minima or network paralysis where training virtually ceases due to operation in the flat region of sigmoidal function. Backpropagation should never be used in situations where inputs are continuously changing because then the process may never converge.

3.6.3 Sigmoidal Gain λ

In some problems, when the weights become large and force the neuron to operate in a region where sigmoidal function is very flat, a better method of coping with network paralysis is to adjust the sigmoidal gain. By decreasing this scaling factor, we effectively spread out sigmoidal function on wide range so that training proceeds faster.

3.6.4 Local Minima

Tsoukalas and Uhrig (1997) have recommended a procedure for getting over the problem of local minima. One of the most practical solutions involves the introduction of a “shock” which changes all weights by specific or random amounts. If this fails, then the most practical solution is to rerandomize the weights and start the training all over.

Second method of backpropagation is used until the process seems to stall.

Simulated annealing is then used to continue training until local minima has been left behind. After

this, simulated annealing is stopped and BPN continues until global minimum is reached.

In most of the cases, only a few simulated annealing cycles of this two-stage process are needed. Usually in such a procedure, the final training step is BPN to minimize the overall error of the process.

3.6.5 Learning Coefficient η

The learning coefficient cannot be negative because this would cause the change of weight vector to move away from ideal weight vector position. If

the learning coefficient is zero, no learning takes place and hence, the learning coefficient must be positive. If learning coefficient is equal to 2 then the network is unstable and if the learning coefficient is greater than 1, the weight vector will overshoot from its ideal position and oscillate. Hence, the learning coefficient must be between zero and one. Larger values for the learning coefficient are used when input data patterns are close to the ideal otherwise small values are used. If the nature of the input data patterns is not known, it is better to use moderate values.

$$\Delta X = \frac{-k \frac{\partial f}{\partial X}}{\sqrt{\left(\frac{\partial f}{\partial X}\right)^2 + \left(\frac{\partial f}{\partial Y}\right)^2}} \quad (3.70)$$

3.7 VARIATIONS OF STANDARD BACKPROPAGATION

ALGORITHM

3.7.1 Decremental Iteration Procedure

In conventional steepest descent method, the minimum is achieved by varying the variable as

when a step reveals no improvement, the value of ‘ k ’ is reduced and the process is continued. Similarly in BPN also, when no improvement is perceived in decrease of error, the training

can be continued with different set of learning coefficients and momentum factors, further decreasing the error. Usually the training of the network reaches a plateau and the error

might increase, leading to overfitting. Training can be dispensed with at this stage and then continued with previous weights using reduced momentum factor and learning coefficient. Usually learning coefficient is halved and the momentum factor is reduced by a small value.

This method is applied to one example and the error rate is given for different values of learning coefficient in Table 3.10.

Table 3.10 Error rate for different η , α

Iteration

Error

Remark

η

α

1

0.256

0.6

0.9

5

0.085

6

0.290

STOP

0.3

0.9

1

0.085

4

0.009

5

0.056

STOP

0.15

0.9

1

0.009

4

0.001

$$\Delta W = W(\text{new}) - W(\text{old}) - \eta \frac{\partial E}{\partial W} \quad (3.71)$$

The weight update equation used in ABP algorithm is

$$W(\text{new}) = W(\text{old}) - \frac{\rho(E) \left(\frac{\partial E}{\partial W} \right)}{\left\| \frac{\partial E}{\partial W} \right\|^2} \quad (3.72)$$

$$\begin{pmatrix} \eta \\ \eta E \\ \eta \tanh(E) \\ \eta \tanh^{-1}\left(\frac{E}{E_0}\right) \end{pmatrix}$$

5

0.006

STOP

0.075

0.9

1

0.001

4

0.0005

STOP

3.7.2 Adaptive Backpropagation (Accelerated Learning)

The use of Adaptive Backpropagation (ABP) is developed by Alexander et al. (1994) and

Atiya et al. (1992) for the development of a fault detection and identification system for a process composed of direct current motor, a centrifugal pump, and the associated piping system. In the ABP learning algorithm, the network weight update rule is chosen such that the error function is forced to behave

in a certain manner that accelerates convergence. In a standard BPN, the weight update is given by

The learning function $\rho(E)$ depends on the total error E . Though there are several choices for the form of $\rho(E)$, the most commonly considered functions are

$$\begin{aligned} \rho(E) = \\ \dots\dots\dots(3.73) \end{aligned}$$

where η and E_0 are constant, non negative numbers representing the learning rate and the error normalization factors respectively. Since $\rho(E) = \eta E$

seems to provide most acceleration, it is normally used. Hence, the weight update rule is given by

$$\Delta W = \frac{-\eta E \left(\frac{\partial E}{\partial W} \right)}{\left\| \frac{\partial E}{\partial W} \right\|^2} \quad (3.74)$$

$$E_{n+1} = E_n + \Delta W \frac{\partial E}{\partial W} + \Delta V \frac{\partial E}{\partial V} + \frac{\Delta W^2}{2} \frac{\partial^2 E}{\partial W_i \partial W_j} + \Delta W \Delta V \frac{\partial^2 E}{\partial W_i \partial V_j} + \frac{\Delta V^2}{2} \frac{\partial^2 E}{\partial V_i \partial V_j}$$

One can combine decremental method with adaptive backpropagation procedure since in this procedure, the dependence of the learning functions is on the instantaneous value of the total error thereby leading to in faster convergence. Furthermore, the algorithm introduces a number of additional tuning parameters found in other variants of BPN algorithm and it uses only the current value of the error gradient term to determine the error gradient norm. The disadvantage of this algorithm is that it has a jumpy behaviour when approaching local minima but this indirectly helps to avoid entrapment near local minima and enhances further learning.

3.7.3 Genetic Algorithm Based Backpropagation

Conventional BPN makes use of a weight updating rule based kind of gradient descent technique to determine their weights. Genetic algorithm on the other hand turns out to be robust search and optimization technique outperforming gradient based techniques in obtaining solutions to problems; acceptable fairly accurately and quickly. Rajasekaran and Vijayalakshmi Pai (1996) code the weights with the help of GA following a real coded system.

They made use of the principle of *survival of the fittest* to create an offspring by making use of genetic operators such as crossover and mutation. The reproduction is based on the fitness function and in this case it is the error function. This technique has been applied to many engineering problems.

3.7.4 Quick Prop Training

Fahlman's (1988) quick prop training algorithms are one of the most effective algorithms in overcoming the step size problem in backpropagation.

A second order method related to Newton's method is used to update the weights in place of simple gradient descent as

.....

(3.75)

Fahlman reports that quick prop consistently outperforms backpropagation.

Quick prop's weight update procedure depends on two approximations—1.

small changes in one weight produce relatively little effect on the error gradient observed at other weights and 2. the error function with respect to each weight is locally quadratic.

3.7.5 Augmented BP Networks

The architecture is that of a standard backpropagation network with augmented networks, i.e. logarithmic neurons and exponential neurons added to neural network's input and output layers. The principle of augmenting the network is—

the augmented neurons are highly sensitive in the boundary domain, thereby facilitating the construction of accurate mapping in the model's boundary domain.

the network denotes each input variable with multiple input neurons, thus allowing a highly interactive functions on hidden neurons to be easily formed.

Therefore, the hidden neurons can more easily construct an accurate network output for a high interaction mapping model. The architecture of the augmented neural network is shown in Fig. 3.29.

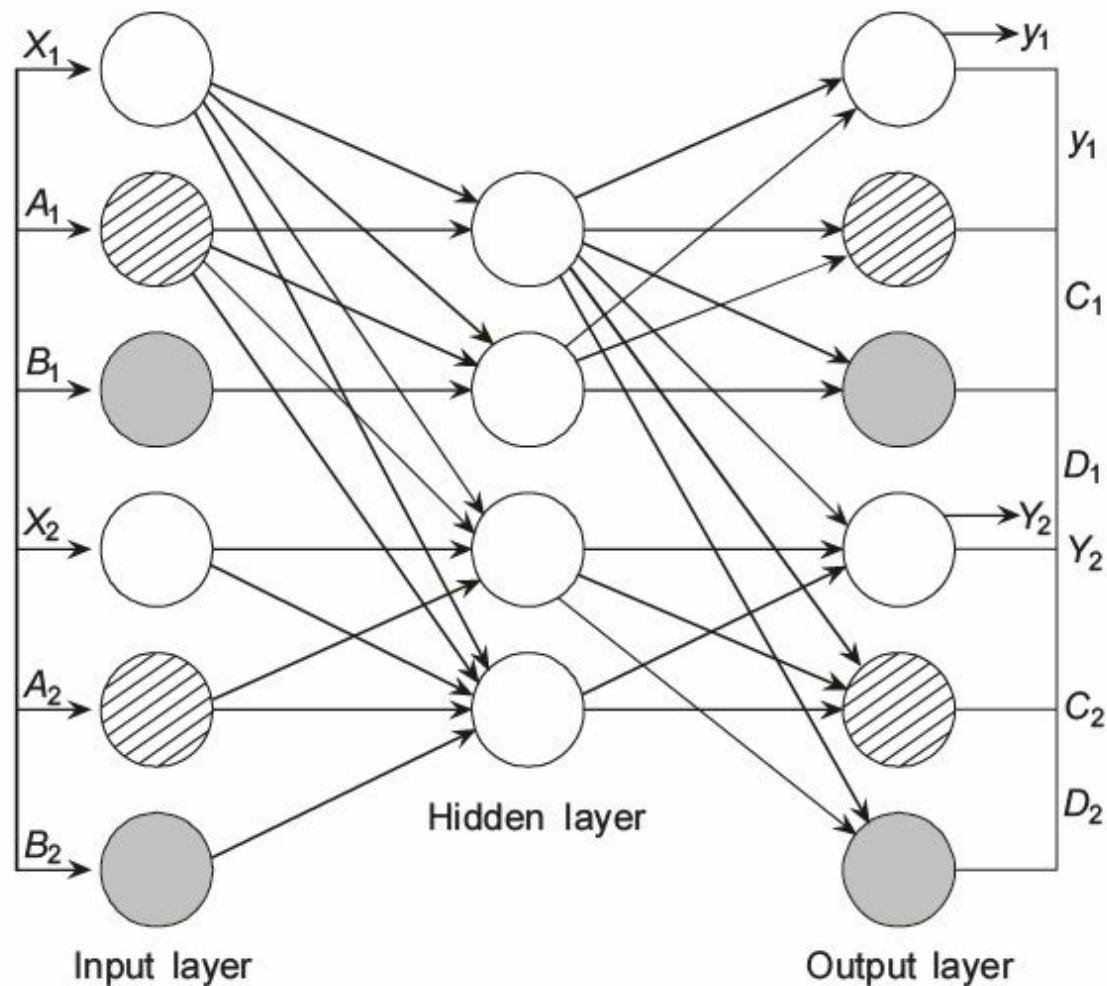


Fig. 3.29 Architecture of augmented neural network.

The architecture of the augmented neural network is that of a standard backpropagation network. However, Fig. 3.29 shows that logarithmic neurons and exponential neurons are added to the network's input and output layers. The logarithmic neuron in the input layer receives a natural logarithm transformation of the corresponding input value of the training data under the consideration

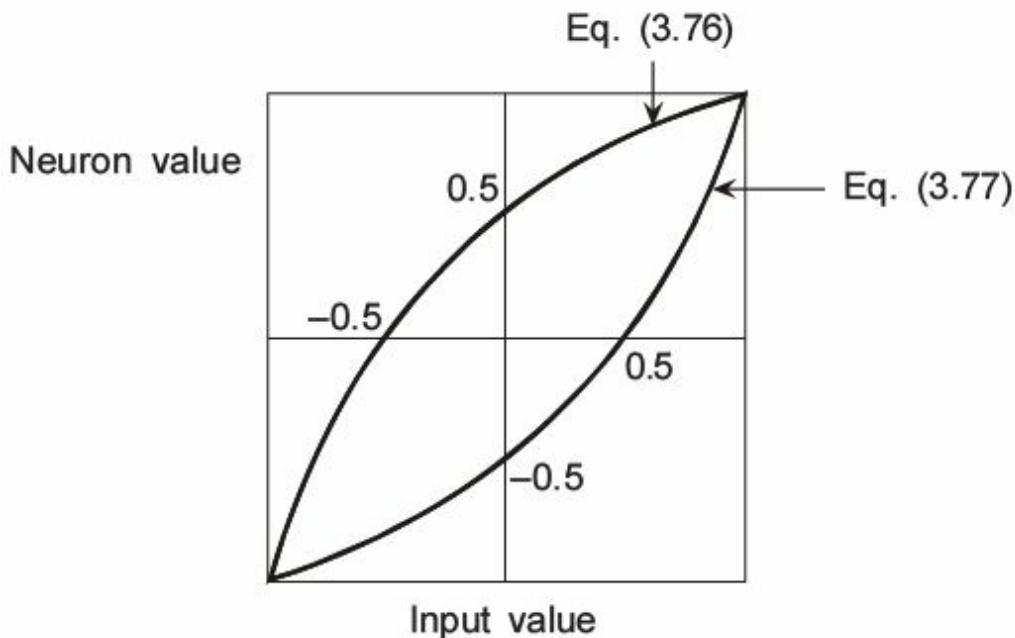
$$A_i = \ln(1.175 X_i + 1.543) \dots\dots\dots(3.76)$$

where X_i is the i th input value of training data and A_i the output of i th logarithm neuron in the input layer.

The input layer's exponent neurons receive natural exponent transformation of the corresponding input value by the training data under the following formula

$$B_i = 0.851 \exp(X_i) - 1.313 \dots\dots\dots(3.77)$$

where B_i is the output of the i th exponent neuron in the input layer and this transformation is shown in Fig. 3.30.



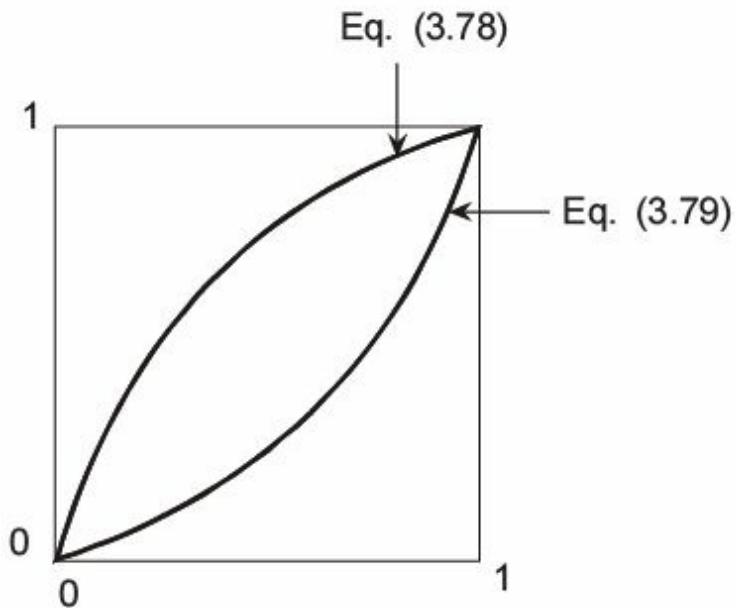


Fig. 3.30 Transfer function of augmented neuron or input layer.

The output layers' logarithm neuron and exponent neuron transform output as

$$C_j = \ln(1.718 Y_j + 1) \dots \dots \dots (3.78)$$

$$D_j = \exp(0.6931 Y_j - 1) \dots \dots \dots (3.79)$$

The transformation is shown in Fig. 3.31.

Fig. 3.31 Transfer function of augmented neuron or output layer.

The principle of working of augmented neural network is as follows: The logarithm neuron is highly sensitive in a small value domain. On the other hand, the exponent neuron is highly sensitive in a large value domain. This feature can facilitate the construction of an accurate mapping in the network's boundary domain.

$$\begin{aligned} \eta_{t+1} &= r_\eta \eta_t \leq \eta_{\min} \\ \alpha_{t+1} &= r_\alpha \alpha_t \leq \alpha_{\min} \end{aligned} \quad (3.82)$$

In BPN, the output Y can be represented as

$$Y = Y [H(X)] \dots \dots \dots (3.80)$$

Here, $r \eta$, and $r \alpha$ are the reduction factors for learning rate and momentum factor and η_{\min} , α_{\min} are the minimum bounds for learning rate and momentum factors respectively.

Cheng Yeh (1998) applied the augmented neural network to a structural engineering problem and compared network's performance with other backpropagation networks in

Table 3.11.

Table 3.11 Performance of various BP networks

Neural networks

RMS

Training set

Testing set

Standard BPN

0.01290

0.01982

Augmented neural network

0.00850

0.01631

BPN with delta bar delta

0.01357

0.02041

Projection NN

0.01343

0.02067

General regression NN

0.02574

0.03534

Radial basis function NN

0.01485

0.02043

Modular NN

0.01322

0.02399

Hence, logarithmic neuron and exponent neuron in the network provide enhanced network architecture capable of markedly improving the network's performance.

3.7.6 Sequential Learning Approach for Single Hidden Layer Neural Networks

Zhang and Morris (1998) proposed the method of sequential learning approach for single hidden layer neural networks. In this method, hidden neurons are added one at a time. The procedure starts with one hidden neuron and sequentially increases the number of hidden neurons until the model error is sufficiently small. When adding a neuron, the new information introduced by this neuron results from that part of its output vector which is orthogonal to the space spanned by the output vectors of previously added hidden neurons. The classical *Gram-Schmidt orthogonalization* method is used at each step to form a set of orthogonal bases for the space spanned by output vectors and hidden neurons. Hidden layer weights are found through optimization while output layer weights are obtained from least square regression. In this architecture, it is possible to determine the necessary number of hidden neurons required. An additional advantage of this method is that it can be used to build and train neural networks with mixed types of hidden neurons and thus, to develop hybrid models. By using mixed types of neurons, it is found that more accurate neural networks with a smaller number of hidden neurons can be developed than those in conventional networks.

3.8 RESEARCH DIRECTIONS

Research work in the area of backpropagation neural networks may be grouped under the following sections.

3.8.1 New Topologies

At present, neural networks are classified into static and dynamic nets. Many investigations have suggested modifications for both the varieties. Among dynamic sets, recurrent nets have been quite popular. They may be considered as a sequence of error propagation networks where the input and

output vectors are divided into internal and external portions and this network operates by concatenating the input and output vectors. There are also other types of time delay networks, continuous time Hopfield nets, discrete time Hopfield nets, and so on. Variations of static nets such as radial basis function networks are also in existence. Many investigators are experimenting to develop novel topologies with desirable features.

3.8.2 Better Learning Algorithms

It is generally agreed that the backpropagation algorithm is expensive in computer time and is suboptimal in performance. This has led to many investigations of developing faster algorithms involving less computation. A group of methods have been suggested in which instead of weight update on output error, algorithm based on some random or heuristic variations or perturbations is obtained. Among these MRIII algorithms based on derivative estimation by weight perturbation, random optimisation, and technique based on genetic algorithms are typical. There have also been other popular approaches such as second order weight adjusting layer by layer optimization and so on. The size of the network is naturally a prime concern and some algorithms have been developed to prune the network to minimum size.

3.8.3 Better Training Strategies

It is necessary to cut down the training time. It is also found that training time depends on the model in the hidden layer, learning rate, momentum factors, the distribution of training patterns in the input space, and the training

algorithm. Many investigators have suggested modular architecture for BPN which could be used to make training faster. Sometimes fine-tuning is also necessary.

3.8.4 Hardware Implementation

The concern of this area is designing efficient hardware which exploits the special features of neural computing such as parallel distributed processing.

Interested readers may consult two special Issues of *IEEE Transactions of Neural Networks* of May 1992 and also May 1993 on neural network hardware.

3.8.5 Conscious Networks

The computing world has lot to gain from neural networks. Their ability to learn by examples makes them very flexible and powerful. Furthermore, there is no need to devise an algorithm in order to perform a specific task; there is no need to understand the internal mechanism of the task. They are also very well suited for real time systems because of their fast response and computational time, which are due to their parallel architecture. Neural networks also contribute to other areas of research such neurology and psychology. They are regularly used to model parts of living organisms and to investigate the internal mechanisms of the brain. Perhaps the most exciting aspect of neural networks is that the possibility that some day “Conscious” networks might be developed which are a realistic possibility.

SUMMARY

Architecture of backpropagation networks is discussed.

Various types of nonlinear activation operators are illustrated.

Model for multilayer perceptron is described.

Backpropagation algorithm is given in matrix form.

Different types of training of artificial neural networks are discussed and backpropagation algorithm is given in detail.

The effect of various parameters on tuning of the network is illustrated.

A toy problem is taken and BPN algorithm is illustrated step by step.

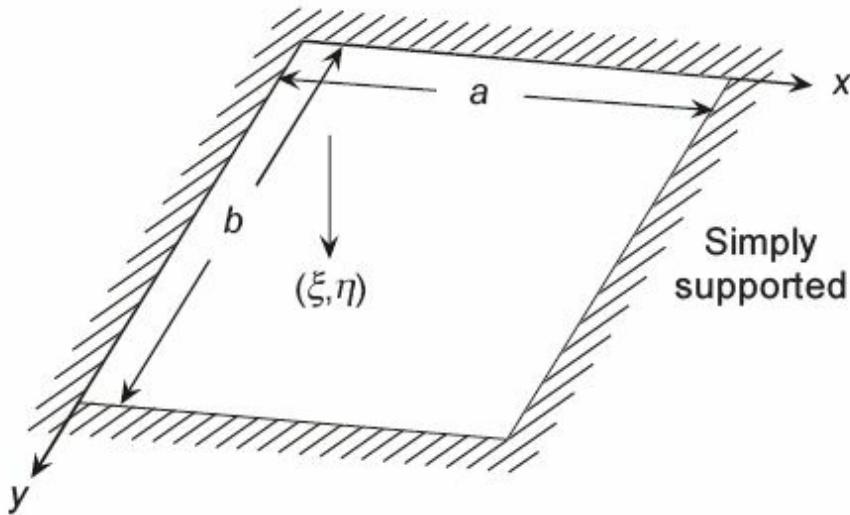
Three real life examples (1) from Mechanical engineering, (2) Civil engineering, and

(3) Metallurgy are solved using BPN and the performance is compared.

Variations in backpropagation networks and the effect of various parameters on training are illustrated.

Many variations of BPN algorithm such as decremental iterative procedure, adaptive backpropagation, GA based BPN, quick prop training, augmented BPN, and sequential learning single hidden layer neural network are discussed.

Research directions in the area of BPN are given for future work.



PROGRAMMING ASSIGNMENT

P3.1 Use “MATLAB” tool box “NEURONET” to solve the following problem using backpropagation training.

Consider a simply supported plate of sides ‘ a and b ’. A concentrated unit load is applied at (x, η) as shown in Fig. P3.1. The program ANSYS (Analysis System) is used to determine maximum moments in X and Y direction and their locations.

Fig. P3.1 Simply supported plate.

Run the program given in “**NEURONET**” with 4 neurons as inputs (a , b , x , η) and 6 neurons as outputs i.e. (MX , max, X_{mx} , Y_{mx} , MY , max, X_{my} , Y_{my})

(a) Train the neural net with one hidden layer, two hidden layers etc.

and vary the number of neurons in the hidden layer.

(b) Observe the performance with different learning rates, sigmoidal gain, and momentum factors and draw the curves depicting the error rate vs these factors and also error rate with iterations for particular problem.

(c) Find the values of maximum moments and their positions for a simply supported plate of $6\text{m} \times 8\text{m}$ with a concentrated load acting at the centre. The data for training is given in the Table P3.1.

Table P3.1 Data for plate problem

Input

Output

Plate DIM

Load POS

M

X

X

x

mx

Y_{mx}

My

my

*Y**my*

a

b

x

η

\max

8

8

\max

8

8

8

8

4

4

0.32

0.5

0.5

0.32

0.5

0.5

8

8

6.4

2

0.27

0.77

0.28

0.28

0.77

0.28

8

8

1.6

1.6

0.25

0.22

0.22

0.25

0.22

0.22

7.2

8

1.12

6.4

0.26

0.16

0.57

0.23

0.16

0.57

7.2

8

4.96

1.6

0.26

0.65

0.22

0.26

0.65

0.22

8

6.4

4

3.2

0.31

0.5

0.4

0.33

0.5

0.4

5.6

8

1.12

1.6

0.2

0.11

0.22

0.23

0.11

0.22

5.6

8

2.8

6.0

0.3

0.35

0.33

0.29

0.35

0.73

8

4.8

1.6

0.96

0.22

0.22

0.14

0.25

0.22

0.14

8

4

4

2.0

0.26

0.5

0.25

0.32

0.5

0.25

4

8

1.04

6.4

0.26

0.14

0.77

0.23

0.14

0.77

P3.2 Modify the program “**NEURONET**” given in CD-ROM by including logarithmic and exponential input and output neurons and study the performance for the above problem. After training, infer to get the result for the following problem whose data is given

in Table P3.2

Table P3.2

Input

Desired output

Plate DIM

Load POS

M

X

X

x

mx

Ymx

My

my

Y_{my}

\max

\max

a

b

x

η

8

8

8

8

8

8

6

4

0.3

0.73

0.5

0.29

0.73

0.5

7.2

8

1.76

6.4

0.26

0.25

0.73

0.26

0.25

0.3

4.8

8

4.0

6.0

0.3

0.3

0.73

0.28

0.3

0.73

P3.3 The training data for a particular problem is given in Table P9.3. Use **NEURONET** to train the data. Show step by step output to input, hidden, and output neurons as well as error. How the weights W and V

are modified? Use learning coefficient and momentum factors as 0.6 and 0.9 respectively.

Table P3.3

Input

Output

0.16

0.12

0.08

0.04

0.2

0.12

0.24

0.16

0.08

0.4

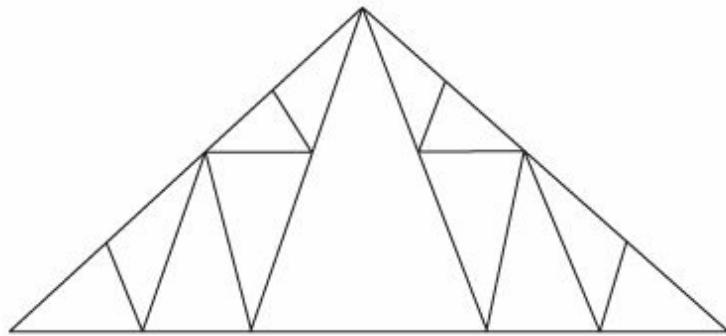
0.08

0.16

0.24

0.12

0.6



0.04

0.08

0.12

0.16

0.8

P3.4 Derive the backpropagation training algorithm for the case where the activation function is an arctan function.

P3.5 Derive the backpropagation training algorithm for the neurons in the hidden layer using logistic function and the neurons in the output layer using linear function.

P3.6 In the design of roof trusses, preliminary dimensions are to be assumed to calculate dead weight of the truss for analysis. A neural network approach is to be incorporated for the preliminary design of trusses. The input variables are fixed as span, slope, access provided or not and spacing. For the Fink Truss shown in Fig. P3.2 the scaled input and output are given in the form of table shown in Table P3.4.

Fig. P3.2 Fink truss.

Table P3.4

Output of different type of areas

Input

7000

Span

Slope

Spacing

Access

Type 1

Type 2

Type 3

Type 4

40

40

10

0.225

0.6

1

0.4

0.3245

0.3245

0.136

0.136

0.225

0.65

0

0.4

0.3245

0.3245

0.136

0.136

0.4

0.6625

1

0.7

0.4674

0.4674

0.247

0.1934

0.25

0.75

0

0.3

0.3245

0.3245

0.1368

0.1368

0.5

0.45

1

0.5

0.685

0.685

0.193

0.299

0.5

0.6

1

0.6

0.628

0.628

0.299

0.247

0.45

0.75

0

0.5

0.3245

0.3245

0.193

0.1368

0.45

0.6

1

0.4

0.3577

0.3577

0.193

0.164

0.5
0.65
1
0.3
0.3245
0.3245
0.193
0.136
0.4
0.6
0
0.6
0.3245
0.3245
0.164
0.136

Access = 1 means access provided and 0 means no access provided.

Train the network with

the given data and infer to find the areas of the members of a truss of span 32 m with a

slope of 20 degrees and access not provided with a spacing of 3 m (use “NEURONET”).

REFERENCES

- Alexander, G.P., V. Muthusami and A.F. Atiya (1994), Incipient Fault Detection and Identification in Process Systems Using Accelerated Neural Network Learning, *Nuclear Technology*, Vol. 105, pp. 145–161.
- Anonymous (1987), *Compendium of Indian Standards on Soil Engineering*, SP-36, (Part I), Bureau of Indian Standards.
- Atiya, A.F., A.G. Parlos, J. Muthusami, B. Fernandez and W.K. Tsai (1992), Accelerated Learning in Multilayer Networks, *Proc. of Int. Jol. Conf. Neural Networks*, Baltimore, Maryland, Vol. 3, p. 925.
- Eaton, H.A.C. and T.L. Oliver (1992), Learning Coefficient Dependence on Training Set Size, *Neural Networks*, Vol. 5, pp. 283–288.
- Fahlman, S.E. (1988a), An Empirical Study of Learning Speed in Backpropagation Networks, *Carnegie Mellon Report*, No CMU-Cs, pp. 88–162.
- Fahlman, S.E. (1988b), Learning Variations of Backpropagation—An Empirical Study, *Proc. of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, San Mater, CA.
- Hansraj, K., I. Fourment, T. Curpez and J.L. Chenot (1992), Simulation of Industrial Forging of Axi-symmetric Parts, *Int. Jol. of Engg. Computation*.

Huang, S. and Y. Huang (1991), Bounds on the Number of Hidden Neurons in Multilayer Perceptrons, *IEEE Trans. on Neural Networks*, **2(1)**, pp.

47–55.

Jacobs, R.A. (1988), Increased Rates of Convergence through Learning Rate Adaptation, *Neural Networks*, Vol. I, No. 4, pp. 295–307.

Kulkarni, S.G. (1997), *Machine Design*, Tata McGraw-Hill, New Delhi.

McInerney, M. and A.P. Dhawan (1993), Use of Genetic Algorithm with Backpropagation in Training of Feedforward Neural Networks, *IEEE*

Proceedings of the Intl. Conf. on Artificial Neural Networks and Genetic Algorithms, Innsbruck, Springer-Verlag, Wien, pp. 203–208.

Mirchandani, G. and W. Cao (1989), On Hidden Nodes for Neural Networks,

IEEE Transactions on Circuits and Systems, **36(5)**, pp. 661–664.

Parker, D.B. (1982), Learning Logic , *Invention Report S 81–84*, File 1, Office of Technology, Licensing, Stanford University, Stanford, CA.

Parker, D. (1987), Optimal Algorithms for Adaptive Networks, Second Order Backpropagation, Second Order Direct Propagation and Second Order Hebbian Learning, *Proc. of IEEE First Intl. Conf. on Neural Networks*, Vol II, Sandiego, Ca, pp. 593–600.

Rajasekaran, S. and G.A. Vijayalakshmi Pai (1996), Genetic Algorithm based Weight Determination for Backpropagation Networks, *Proc. Trends in Computing*, Tata McGraw-Hill,

pp. 73–80.

Rumelhart, D.E., G.E. Hinton and R.J. Williams (1986), Learning Internal Representations by Error Propagation, *Parallel Distributed Processing*, Vol. I, pp. 318–362, Cambridge, MA; MIT Press.

Sejnowski, T. and C. Rosenberg (1987), Parallel Networks that Learn to Pronounce English Text, *Complex Systems*, Vol. 1, pp. 143–168.

Tsoukalas, L.H. and R.E. Uhrig (1997), Fuzzy and Neural Approaches in Engineering, John Wiley & Sons, USA.

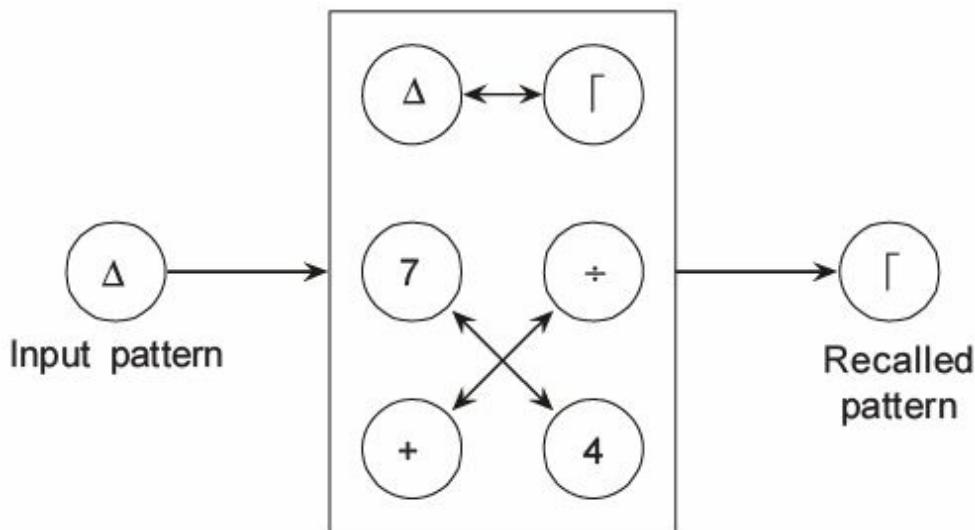
Werbos, P.J. (1974), Beyond Regression , New Tools for Prediction and Analysis in the Behavioural Sciences, *Masters Thesis*, Harvard University.

Yeh, C.I. (1998), Structural Engineering Applications with Augmented Neural Networks, *Computer Aided Civil and Infrastructure Engineering*, Vol. 13, pp. 83–90.

Zhang, J. and A.J. Morris (1998), A Sequential Learning Approach for Single Hidden Layer Neural Networks, *Neural Networks*, Vol. 11, No. 1, pp. 65–80.

Zienkiewicz, O.C. (1992), Flow Formulation for Numerical Solution of Forming Process, *Numerical Analysis of Forming Process*, John Wiley & Sons, pp. 1–44.

Chapter 4



Associative Memory

Associative Memories, one of the major classes of neural networks, are faint imitations of the human brain's ability to associate patterns. An *Associative Memory* (AM) which belongs to the class of single layer feedforward or recurrent network architecture depending on its association capability, exhibits Hebbian learning.

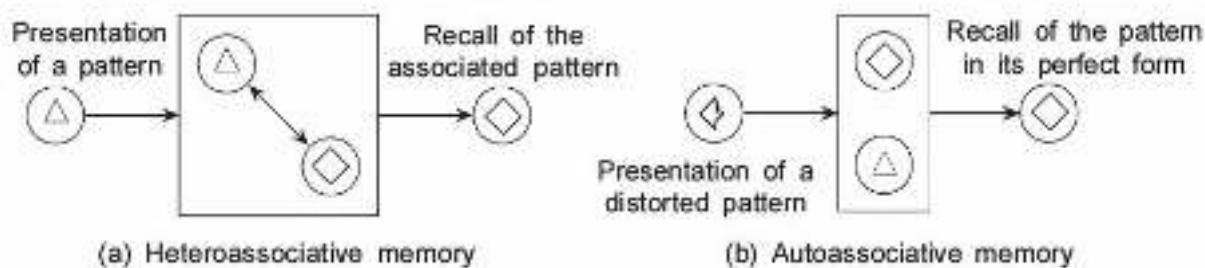
In this chapter, the association memory network is discussed.

Autocorrelators and heterocorrelators are first introduced. Wang et al's *multiple training encoding strategy* and *exponential bidirectional associative memory* are presented next. The AM's capability to associate real-coded patterns is elaborated. Finally, the application of AM in character recognition and fabric defect identification is discussed.

An *assssociate memory* is a storehouse of associated patterns which are encoded in some form. When the storehouse is *triggered* or *incited* with a pattern, the associated pattern pair is *recalled* or *output*. The input pattern could be an exact replica of the stored pattern or a distorted or partial representation of a stored pattern. Figure 4.1 illustrates the working of an associative memory.

Fig. 4.1 The working of an associative memory.

In the figure, (Δ, \lceil) , $(7, 4)$, and $(+, \Pi)$ are associated pattern pairs. The associations represented using ‘ \leftrightarrow ’ symbol are stored in the memory. When the memory is triggered for instance, with a Δ , the associated pattern \lceil is retrieved automatically.



\Re^m with \Re^n . Thus, for $\bar{u} \in \Re^m$ and $\bar{v} \in \Re^n$,

$$\bar{v} = g(\bar{u}) \quad (4.1)$$

Quite often g tends to be a general nonlinear matrix type operator resulting in

$$\bar{v} = M(\bar{u}) \quad (4.2)$$

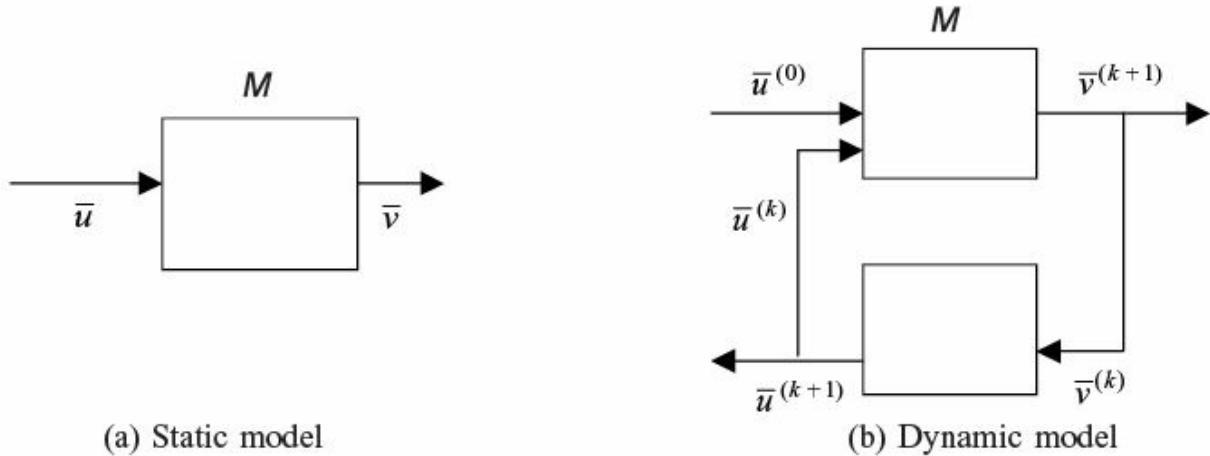
If the associated pattern pairs (x, y) are different and if the model recalls a y given an x or vice versa, then it is termed as *heteroassociative memory*. On the other hand, if x and y refer to the same pattern, then the model is termed as *autoassociative memory*. While heteroassociative memories are useful for the association of patterns, autoassociative memories are useful for image refinement, that is, given a distorted or a partial pattern, the whole pattern stored in its perfect form can be recalled. Autoassociative correlation memories are known as *autocorrelators* and heteroassociative correlation memories are known as *heterocorrelators*. Figure 4.2 illustrates heteroassociative and autoassociative memories.

Fig. 4.2 ‘Hetero’ and ‘auto’ correlators.

An associative memory can therefore be thought of as a mapping g between a pattern space

The operator M has different forms for different memory models. The algorithm which computes M is known as the *recordin g* or *storage algorithm*.

In most cases, M is computed using the input pattern vectors. Based on the principle of recall, associative memory models may be classified into *static* and *dynamic* networks. While static networks recall an output given an input in one feedforward pass, dynamic networks recall through an input/output feedback mechanism which takes time. Static networks are *non-recurrent* and dynamic networks are termed *recurrent*. Figure 4.3 illustrates static and dynamic associative memories.



2

11

$$\bar{v}^{(k)} = M(\bar{u}^{(k)})$$

$$\bar{v}^{(k+1)} = M'(\bar{u}^{(k)}, \bar{v}^{(k)})$$

Fig. 4.3 Static and dynamic associative memories.

Observe that in Fig. 4.3(a) for a static model, the associated for the input pattern is recognized in one feedforward pass whereas for a dynamic network, as shown in Fig. 4.3(b), the following recursive formulae are put to work until an equilibrium state is reached.

(4.3)

and

(4.4)

$$T = \sum_{i=1}^m [A_i^T] [A_i] \quad (4.5)$$

Here, $T = [t_{ij}]$ is a $(p \times p)$ connection matrix and $A_i \in \{-1, 1\}^p$.

The autocorrelator's recall equation is a vector-matrix multiplication followed by a pointwise nonlinear threshold operation. The recall equation is given by

$$a_j^{\text{new}} = f(a_i t_{ij}, a_j^{\text{old}}) \quad \forall j = 1, 2, \dots, p \quad (4.6)$$

where $A_i = (a_1, a_2, \dots, a_p)$ and the two parameter bipolar threshold function is

$$f(\alpha, \beta) = \begin{cases} 1, & \text{if } \alpha > 0 \\ \beta, & \text{if } \alpha = 0 \\ -1, & \text{if } \alpha < 0 \end{cases} \quad (4.7)$$

$$T = \sum_{i=1}^3 [A_i^T]_{4 \times 1} [A_i]_{1 \times 4} = \begin{bmatrix} 3 & 1 & 3 & -3 \\ 1 & 3 & 1 & -1 \\ 3 & 1 & 3 & -3 \\ -3 & -1 & -3 & 3 \end{bmatrix}$$

4.1 AUTOCORRELATORS

Autocorrelators, now most easily recognized by the title of *Hopfield Associative Memory* (HAM), were introduced as a theoretical notation by Donald Hebb (1949) and rigorously analyzed by Amari (1972, 1977). Other researchers who studied their dynamics include Little (1974), Little and Shaw (1978), and Hopfield (1982).

First order autocorrelators obtain their connection matrix (indicative of the association of the pattern with itself) by multiplying a pattern's element with every other pattern's elements. A first order autocorrelator stores M bipolar patterns A_1, A_2, \dots, A_m by summing together m outer products as **Example 4.1** (Working of an autocorrelator)

Consider the following patterns

$$A_1 = (-1, 1, -1, 1)$$

$$A_2 = (1, 1, 1, -1)$$

$$A_3 = (-1, -1, -1, 1)$$

which are to be stored as an autocorrelator.

The connection matrix,

$$HD(x, y) = \sum_{i=1}^n |x_i - y_i|$$

Recognition of stored patterns

The autocorrelator is presented a stored pattern $A_2 = (1, 1, 1, -1)$. The computation of Eq. (4.6) yields,

a_{new}

1

$$= f(3 + 1 + 3 + 3, 1) = 1$$

a_{new}

2

$$= f(6, 1) = 1$$

a_{new}

3

$$= f(10, 1) = 1$$

a_{new}

4

$$= f(-10, 1) = -1$$

This is indeed the vector itself. Also, in the retrieval of $A\ 3 = (-1, -1, -1, 1)$ (a new

new

new

new

1

, a 2

, a 3

, a 4

$$) = (-1, -1, -1, 1)$$

yielding the same vector.

Recognition of noisy patterns

Consider a vector $A' = (1, 1, 1, 1)$ which is a distorted presentation of one among the stored patterns.

We proceed to find the proximity of the noisy vector to the stored patterns using the Hamming distance measure. The Hamming distance (HD) of a vector X from Y , given $X = (x_1, x_2, \dots, x_n)$ and $Y = (y_1, y_2, \dots, y_n)$ is given by, (4.8)

Thus, the HD of A' from each of the patterns in the stored set is as follows:
 $HD(A', A\ 1) = 4$

$$HD(A', A\ 2) = 2$$

$$HD(A', A\ 3) = 6$$

It is evident that the vector A' is closer to A_2 and therefore resembles it, or in other words, is a noisy version of A_2 .

Now, the computations using Eq. (4.6) yield

$$\begin{aligned}(a_1^{\text{new}}, a_2^{\text{new}}, a_3^{\text{new}}, a_4^{\text{new}}) &= (f(4,1), f(4,1), f(4,1), f(-4,1)) \\ &= (1, 1, 1, -1) \\ &= A_2\end{aligned}$$

Hence, in the case of partial vectors, an autocorrelator results in the refinement of the pattern or removal of noise to retrieve the closest matching stored pattern.

$$M = \sum_{i=1}^N X_i^T Y_i \quad (4.9)$$

4.2 HETEROCORRELATORS: KOSKO'S DISCRETE BAM

As Kosko and others (1987a, 1987b), Cruz. Jr. and Stubberud (1987) have noted, the bidirectional associative memory (BAM) is a two-level nonlinear neural network based on earlier studies and models of associative memory (Kohonen 1972; Palm, 1980; Nakano, 1972; Kohonen, 1977; Anderson, 1983; Kohonen and Oja, 1976; Hirai, 1983).

Kosko extended the unidirectional autoassociators to bidirectional processes. One important performance attribute of discrete BAM is its ability to recall stored pairs particularly in the presence of noise.

Bidirectional Associative Memory (BAM) introduced by Kosko (1987b) has the following operations:

1. There are N training pairs $\{(A_1, B_1), (A_2, B_2), \dots, (A_i, B_i), \dots, (A_n, B_n)\}$

where,

$$A_i = (a_{i1}, a_{i2}, \dots, a_{in})$$

$$Bi = (bi\ 1, bi\ 2, \dots, bip)$$

Here, a_{ij} or b_{ij} is either in the ON or OFF state.

2. In the binary mode, ON = 1 and OFF = 0 and in the bipolar mode, ON = 1 and OFF = -1

We frame the correlation matrix

To retrieve the nearest (Ai, Bi) pair given any pair (α, β) , the recall equations are as follows:

Starting with (α, β) as the initial condition, we determine a finite sequence $(\alpha', \beta'), (\alpha'', \beta''), \dots$ until an equilibrium point $(\alpha F, \beta F)$, is reached.

Here,

$$\beta' = \phi(\alpha M) \quad (4.10)$$

$$\alpha' = \phi(\beta' M^T) \quad (4.11)$$

$$\phi(F) = G = g_1, g_2, \dots, g_n \quad (4.12)$$

$$F = (f_1, f_2, \dots, f_n) \quad (4.13)$$

$$g_i = \begin{cases} 1 & \text{if } f_i > 0 \\ 0 & \text{(binary)} \\ -1 & \text{(bipolar)} \\ \text{previous } g_i & f_i = 0 \end{cases} \quad (4.14)$$

$$M = X_1^T Y_1 + X_2^T Y_2 + \dots + X_n^T Y_n + X'^T Y' \quad (4.15)$$

In the case of deletion, we subtract the matrix corresponding to $(X_j^T Y_j)$ from the matrix M .

$$\text{i.e., } (\text{New}) M = M - (X_j^T Y_j) \quad (4.16)$$

4.2.1 Addition and Deletion of Pattern Pairs

Given a set of pattern pairs (X_i, Y_i), for $i = 1, 2, \dots, n$ and their correlation matrix M , a pair

(X' , Y') can be added or an existing pair (X_j, Y_j) can be erased or deleted from the memory model.

In the case of addition, the new correlation matrix M is The addition and deletion of information contributes to the functioning of the system as a typical human memory exhibiting learning and forgetfulness.

4.2.2 Energy Function for BAM

A pair (A, B) defines the state of a BAM. To store a pattern, the value of the energy function for that particular pattern has to occupy a minimum point in the energy landscape. Also, adding new patterns ought not to destroy the previously stored patterns.

The stability of a BAM can be proved by identifying a Lyapunov or energy function E with each state (A, B). In the autoassociative case, Hopfield identified an appropriate E (actually, Hopfield defined half this quantity) as

$$E(A) = -AMAT \quad (4.17)$$

However, Kosko proposed an energy function,

$$\begin{aligned} A_1 &= (100001) & B_1 &= (11000) \\ A_2 &= (011000) & B_2 &= (10100) \\ A_3 &= (001011) & B_3 &= (01110) \end{aligned}$$

$$E(A, B) = -AMBT \quad (4.18)$$

for the bidirectional case and this for a particular case $A = B$ corresponds to *Hopfield's autoassociative energy function*.

Also, when a paired pattern (A, B) is presented to BAM, the neurons change states until a

bidirectionally stable state (Af, Bf) is reached. Kosko proved that such a stable state is reached for any matrix M which corresponds to the local minimum of the energy function.

Kosko proved that each cycle of decoding lowers the energy E if the energy function for any point (α, β) is given by

$$E = -\alpha M \beta T, \quad (4.19)$$

However, if the energy E evaluated using the coordinates of the pair (Ai, Bi) , i.e. $E = -A$

$$T \\ iMBi \quad (4.20)$$

does not constitute a local minimum, then the point cannot be recalled, even though one starts with $\alpha = Ai$. In this aspect, Kosko's encoding method does not ensure that the stored pairs are at a local minima.

Example 4.2 (Working of Kosko's BAM)

Suppose one has $N = 3$ with the pattern pairs given by

Converting these to bipolar forms

$$\begin{array}{l} X_1 = (-1 -1 -1 -1 -1 \ 1) \\ X_2 = (-1 \ 1 \ 1 -1 -1 -1) \\ X_3 = (-1 -1 \ 1 -1 \ 1 \ 1) \end{array} \quad \begin{array}{l} Y_1 = (1 \ 1 -1 -1 -1) \\ Y_2 = (1 -1 \ 1 -1 -1) \\ Y_3 = (-1 \ 1 \ 1 \ 1 -1) \end{array}$$

The matrix M is calculated as

$$M = X_1^T Y_1 + X_2^T Y_2 + X_3^T Y_3 = \begin{bmatrix} 1 & 1 & -3 & -1 & 1 \\ 1 & -3 & 1 & -1 & 1 \\ -1 & -1 & 3 & 1 & -1 \\ -1 & -1 & -1 & 1 & 3 \\ -3 & 1 & 1 & 3 & 1 \\ -1 & 3 & -1 & 1 & -1 \end{bmatrix}$$

Let us suppose we start with $\alpha = X_3$ hoping to retrieve the associated pair Y_3 .

$$\begin{aligned} \alpha M &= (-1 -1 \ 1 -1 \ 1 \ 1) \ (M) = (-6 \ 6 \ 6 \ 6 \ -6) \\ \beta' &= \phi(\alpha M) = (-1 \ 1 \ 1 \ 1 \ -1) \\ \beta' M^T &= (-5 \ -5 \ 5 \ -3 \ 7 \ 5) \\ \phi(\beta' M^T) &= (-1 \ -1 \ 1 -1 \ 1 \ 1) = \alpha' \\ \alpha' M &= (-1, -1, 1 -1, 1 \ 1) \ M = (-6 \ 6 \ 6 \ 6 \ -6) \\ \phi(\alpha' M) &= \beta'' = (-1 \ 1 \ 1 \ 1 \ -1) \\ &\quad - \beta' \end{aligned}$$

$$\begin{array}{ll} A_1 = (000111001) & B_1 = (010000111) \\ A_2 = (111001110) & B_2 = (100000001) \\ A_3 = (110110101) & B_3 = (101001010) \end{array}$$

Conversion of these to bipolar forms yield

$$\begin{array}{ll} X_1 = (-1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ -1 \ 1) & Y_1 = (-1 \ 1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1) \\ X_2 = (1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1) & Y_2 = (1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1) \\ X_3 = (1 \ 1 \ -1 \ 1 \ 1 \ -1 \ 1 \ -1 \ 1) & Y_3 = (1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1 \ 1 \ -1) \end{array}$$

The correlation matrix M is calculated as

$$M = X_1^T Y_1 + X_2^T Y_2 + X_3^T Y_3$$

Here, β' is same as Y_3 . Hence, $(\alpha F, \beta F) = (X_3, Y_3)$ is the desired result.

Example 4.3 (Incorrect recall by Kosko's BAM)

Consider the pattern pairs,

$$\begin{bmatrix} 3 & -3 & 1 & -1 & -1 & 1 & -3 & -1 & -1 \\ 3 & -3 & 1 & -1 & -1 & 1 & -3 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & -3 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & 3 & -1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & 3 & -1 \\ -1 & 1 & -3 & -1 & -1 & -3 & 1 & -1 & 3 \\ 3 & -3 & 1 & -1 & -1 & 1 & -3 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & -3 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & 3 & -1 \end{bmatrix}$$

$$M =$$

Suppose one starts with $\alpha = X 2$, then the calculations for the retrieval of $Y 2$
yield

$$\alpha M = (13 -13 -5 1 1 -5 -13 -19 5)$$

$$\phi(\alpha M) = \beta' = (1 -1 -1 1 1 -1 -1 -1 1)$$

$$\beta' MT = (5 5 11 -11 -11 5 5 11 -11)$$

$$\phi(\beta' MT) = \alpha' = (1 1 1 -1 -1 1 1 1 -1)$$

$$\alpha' M = (13 -13 -5 1 1 -5 -13 -19 5)$$

$$\phi(\alpha' M) = \beta'' = (1 -1 -1 1 1 -1 -1 -1 1)$$

$$\text{Here, } \beta'' = \beta'$$

Hence, the cycle terminates with

$$\alpha F = \alpha = X 2 (-1 1 1 1 -1 -1 1 1 1)$$

and

$$\beta F = \beta' = (1 -1 -1 1 1 -1 -1 -1 1)$$

This however, is an incorrect pattern pair to be recalled.

Now, a computation of the energy functions for ($X 2, Y 2$) and ($\alpha F, \beta F$) yield $E 2 = -71$, $EF = -75$

It could be shown that ($X 2, Y 2$) is not at its local minimum by evaluating E at a point which is one Hamming distance away from $Y 2$.

Thus, consider $Y 2' = (1 -1 -1 -1 1 -1 -1 -1 1)$ where the fifth component -1 of $Y 2$ has been changed to 1. Now,

$$\begin{aligned} E &= -X 2 M Y 2' T \\ &= -73 \end{aligned}$$

which is lower than $E 2$, confirming the hypothesis that ($X 2, Y 2$) is not at its local minimum of E .

Summarizing, BAM cannot guarantee the recall of a particular training pair or several training pairs since the correlation matrix M used by Kosko does not guarantee that the energy of a training pair is at its local minimum. A pair P_i can be recalled if and only if this pair is at a local minimum of the energy surface (Kosko, 1988).

4.3 WANG ET AL.'S MULTIPLE TRAINING ENCODING STRATEGY

Wang et al. (1990a, 1990b) proposed the *Multiple Training Encoding Strategy* which is an enhancement of the encoding strategy proposed by

Kosko.

To recover a pair (A_i, B_i) using multiple training of order q , one augments the matrix M with a matrix P defined as

$$P = (q - 1) X T$$

$$i \quad Y_i \quad (4.21)$$

Here, X_i, Y_i are the bipolar forms of (A_i, B_i).

The new value of the energy function E evaluated at A_i, B_i then becomes $E'(A$

$$T$$

$$T$$

$$T$$

$i, B_i) = -A_i M B_i - (q - 1) A_i X_i Y_i B_i \quad (4.22)$ The augmentation therefore implies adding ($q - 1$) more pairs located at (A_i, B_i) to the existing correlation matrix. As a result, the energy E' can be reduced to an arbitrarily low value by a suitable choice of q . Also, this ensures that the energy at (A_i, B_i) does not exceed that at points which are one Hamming distance away from this location. Algorithm 4.1 illustrates the working of the model.

Algorithm 4.1 (Wang et al.'s Multiple Training Encoding Strategy)

```

Algorithm Mul_Tr_Encod (N,  $\bar{X}$ ,  $\bar{Y}$ ,  $\bar{q}$ )

    /* N: No. of stored pattern sets */
    /*  $\bar{X}, \bar{Y}$ : the bipolar pattern pairs*/
    /*  $\bar{X} = (\bar{X}_1, \bar{X}_2, \dots, \bar{X}_N)$  where  $\bar{X}_i = (x_{i_1}, x_{i_2}, \dots, x_{i_n})$  */
    /*  $\bar{Y} = (\bar{Y}_1, \bar{Y}_2, \dots, \bar{Y}_N)$  where  $\bar{Y}_j = (y_{j_1}, y_{j_2}, \dots, y_{j_m})$  */
    /*  $\bar{q}$  is the weight vector  $(q_1, q_2, \dots, q_N)$  */

    Step 1: Initialize correlation matrix M to null matrix
             $M \leftarrow [0]$ 

    Step 2: Compute the correlation matrix M as,
            for i  $\leftarrow 1$  to N
                 $M \leftarrow M \oplus [q_i * \text{TRANSPOSE } (\bar{X}_i) \otimes \bar{Y}_i]$ ;
            end
                /*  $\oplus$  : Matrix addition.
                    $\otimes$  : Matrix multiplication.
                    $*$  : Scalar Multiplication */

    Step 3: Read input bipolar pattern  $\bar{A}$ 

    Step 4: Compute  $A\_M$  where  $A\_M \leftarrow \bar{A} \otimes M$ ;

    Step 5: Apply threshold function  $\phi$  to  $A\_M$  to get  $\bar{B}'$ , i.e.,
             $\bar{B}' \leftarrow \phi(A\_M)$ ;
            /*  $\phi$  is as defined in 4.11-4.14 */

    Step 6: Output  $\bar{B}'$  which is the associated pattern pair.

END Mul_Tr_Encod

```

Example 4.4 (Working of Multiple Training Encoding Strategy) For the pattern pairs considered in Example 4.3, let the pair to be recalled be (X

T

$2, Y 2)$. Choosing $q = 2$, so that $P = X 2 Y 2$, the augmented correlation matrix becomes

$$M = X T$$

$$T$$

$$1 \ Y 1 + 2 \ X 2 \ Y 2 + X 3 \ Y 3$$

$$\begin{bmatrix} 4 & -4 & 0 & -2 & -2 & 0 & -4 & -2 & 0 \\ 4 & -4 & 0 & -2 & -2 & 0 & -4 & -2 & 0 \\ 2 & -2 & -2 & 0 & 0 & -2 & -2 & -4 & 2 \\ -2 & 2 & 2 & 0 & 0 & 2 & 2 & 4 & -2 \\ -2 & 2 & 2 & 0 & 0 & 2 & 2 & 4 & -2 \\ 0 & 0 & -4 & -2 & -2 & -4 & 0 & -2 & 4 \\ 4 & -4 & 0 & -2 & -2 & 0 & -4 & -2 & 0 \\ 2 & -2 & -2 & 0 & 0 & -2 & -2 & -4 & 2 \\ -2 & 2 & 2 & 0 & 0 & 2 & 2 & 4 & -2 \end{bmatrix}$$

$$\begin{bmatrix} 4 & -4 & 0 & -2 & -2 & 0 & -4 & -2 & 0 \\ 4 & -4 & 0 & -2 & -2 & 0 & -4 & -2 & 0 \\ 2 & -2 & -2 & 0 & 0 & -2 & -2 & -4 & 2 \\ -2 & 2 & 2 & 0 & 0 & 2 & 2 & 4 & -2 \\ -2 & 2 & 2 & 0 & 0 & 2 & 2 & 4 & -2 \\ 0 & 0 & -4 & -2 & -2 & -4 & 0 & -2 & 4 \\ 4 & -4 & 0 & -2 & -2 & 0 & -4 & -2 & 0 \\ 2 & -2 & -2 & 0 & 0 & -2 & -2 & -4 & 2 \\ -2 & 2 & 2 & 0 & 0 & 2 & 2 & 4 & -2 \end{bmatrix}$$

ie., $M =$

Now given $\alpha = X 2$ the corresponding $\beta = Y 2$ is correctly recalled. The computations are as follows:

$$\alpha M = (22 -22 -14 -8 -8 -14 -22 -28 14)$$

$$\beta' = \phi(\alpha M) = (1 -1 -1 -1 -1 -1 -1 -1 1)$$

$$\beta' MT = (18 18 16 -16 -16 18 18 16 -16)$$

$$\alpha' = \phi(\beta' MT) = (1 1 1 -1 -1 1 1 1 -1)$$

$$= \alpha$$

However, it is not possible to recall $(X 1, Y 1)$ for the same M . This is so, since choosing

$$\alpha = X 1,$$

$$\alpha M = (-22 22 6 4 4 6 22 24 -6)$$

$$\beta' = (-1 1 1 1 1 1 1 1 -1)$$

To tackle this, M needs to be augmented further.

$$\text{Defining } M = 2 X T$$

$$T$$

$$T$$

$$1 Y 1 + 2 X 2 Y 2 + 2 X 3 Y 3, \text{ we get } M =$$

We now observe that all three pairs can be correctly recalled.

$$M = \sum_{i=1}^n q_i X_i^T Y_i \quad (4.23)$$

$$\begin{array}{ccccccccc}
 X_1 & = & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\
 X_2 & = & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 \\
 X_3 & = & 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 \\
 X_4 & = & 1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1
 \end{array}
 \quad
 \begin{array}{ccccccccc}
 Y_1 & = & 1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 \\
 Y_2 & = & 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\
 Y_3 & = & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
 Y_4 & = & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1
 \end{array}$$

For $\alpha = X 1 = (-1 -1 -1 1 1 1 -1 -1 1)$

$$\alpha M = (-31 31 -3 -5 -5 -3 31 33 3)$$

$$\varphi(\alpha M) = \beta' = (-1 1 -1 -1 -1 -1 1 1 1)$$

$$= Y 1$$

For $\alpha = X 2 = (1 1 1 -1 -1 1 1 1 -1)$

$$\alpha M = (29 -29 -7 -1 -1 -7 -29 -35 7)$$

$$\varphi(\alpha M) = \beta' = (1 -1 -1 -1 -1 -1 -1 -1 1)$$

$$= Y 2$$

For $\alpha = X 3 = (1 1 -1 1 1 -1 1 -1 1)$

$$\alpha M = (1 -1 13 -5 -5 13 -1 1 17 -13)$$

$$\varphi(\alpha M) = \beta' = (1 -1 1 -1 -1 1 -1 1 -1)$$

$$= Y 3$$

Thus, the multiple training encoding strategy ensures the correct recall of a pair for a suitable augmentation of M . Generalizing the correlation matrix, for a correct recall of all training pairs, we write

where q_i 's are positive real numbers. This modified correlation matrix is called the *generalized correlation matrix*.

The necessary and sufficient conditions for the weights qi such that all training pairs will be correctly recalled, has been discussed by Wang et al. (1991).

Example 4.5 (Generalized correlation matrix for multiple training encoding strategy)

Consider the pattern pairs

$$\begin{bmatrix} 8 & 2 & 4 & -5 & 2 & -4 & 1 & -5 & 8 & -1 \\ 5 & -1 & 1 & -8 & 5 & -1 & 4 & -2 & 5 & -4 \\ 5 & -1 & 1 & -2 & 5 & -1 & -2 & -8 & 5 & 2 \\ -2 & -8 & 2 & -1 & 4 & -2 & 5 & -1 & -2 & -5 \\ -2 & 4 & 2 & 5 & -8 & -2 & -1 & 5 & -2 & 1 \\ -5 & 1 & -1 & 2 & -5 & 1 & 2 & 8 & -5 & -2 \\ -1 & 5 & -5 & 4 & -1 & 5 & -8 & -2 & -1 & 8 \\ -4 & 2 & -8 & 1 & 2 & 8 & -5 & 1 & -4 & 5 \\ 8 & 2 & 4 & -5 & 2 & -4 & 1 & -5 & 8 & -1 \\ 1 & -5 & 5 & -4 & 1 & -5 & 8 & 2 & 1 & -8 \\ 1 & -5 & 5 & 2 & 1 & -5 & 2 & -4 & 1 & -2 \\ -2 & -8 & 2 & -1 & 4 & -2 & 5 & -1 & -2 & -5 \\ 2 & 8 & -2 & 1 & -4 & 2 & -5 & 1 & 2 & 5 \\ -1 & 5 & -5 & -2 & -1 & 5 & -2 & 4 & -1 & 2 \\ -1 & 5 & -5 & 4 & -1 & 5 & -8 & -2 & -1 & 8 \end{bmatrix}$$

The generalized correlation matrix M for $q_1 = 3/2$, $q_2 = 3/2$, $q_3 = 2$, and $q_4 =$

3 is given by

$M =$

The computations for the retrieval of all training pairs yield

$$\alpha = X_2 = (1 \ 1 \ -1 \ -1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1 \ -1)$$

$$\alpha M = (20 \ 26 \ 16 \ -23 \ -22 \ -16 \ 19 \ 25 \ 20 \ -19)$$

$$\beta = (1 \ 1 \ 1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1) = Y_2$$

$$\alpha = X_4 = (1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1)$$

$$\alpha M = (36 \ -54 \ 48 \ -39 \ 42 \ -48 \ 51 \ -39 \ 36 \ -51)$$

$$\beta = (1 \ -1 \ 1 \ -1 \ 1 \ -1 \ 1 \ -1) = Y_4$$

$$\alpha = X_3 = (1 \ 1 \ 1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1 \ 1)$$

$$\alpha M = (30 \ 48 \ -30 \ -15 \ 12 \ 30 \ -45 \ -27 \ 30 \ 45)$$

$$\beta = (1 \ 1 \ -1 \ -1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1) = Y_3$$

$$\alpha = X_1 = (1 \ -1 \ 1 \ -1 \ 1 \ -1 \ 1 \ -1 \ 1 \ -1 \ 1 \ -1 \ 1 \ -1 \ 1)$$

$$\alpha M = (28 \ 34 \ 8 \ 17 \ -14 \ -8 \ -37 \ -31 \ 28 \ 37)$$

$$\beta = (1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ 1 \ 1) = Y_1$$

\hat{N}

$$(a_{i_1}, a_{i_2}, \dots, a_{i_n})$$

$$(b_{i_1}, b_{i_2}, \dots, b_{i_p})$$

$$y_k = \begin{cases} 1 & \text{if } \sum_{i=1}^N y_{i_k} b^{X_i X} \geq 0 \\ -1 & \text{if } \sum_{i=1}^N y_{i_k} b^{X_i X} < 0 \end{cases} \quad (4.24)$$

$$x_k = \begin{cases} 1 & \text{if } \sum_{i=1}^N x_{i_k} b^{Y_i Y} \geq 0 \\ -1 & \text{if } \sum_{i=1}^N x_{i_k} b^{Y_i Y} < 0 \end{cases} \quad (4.25)$$

Here, b is a positive number, $b > 1$ and “.” represents the inner product operator of X and X_b , Y and Y_b

i.e., for $X = (x_1, x_2, \dots, x_n)$ and $X_b = (x_{i_1}, x_{i_2}, \dots, x_{i_b})$

4.4 EXPONENTIAL BAM

The capacity (k) of a specific BAM structure is defined to be the maximum number of training pairs selected from a uniform distribution which can be recalled with a minimum specified probability $P = 1 - k$.

Wang and Don (1995) proposed a BAM structure with an exponential form and it is therefore termed eBAM. eBAM has higher capacity for pattern pair storage than conventional BAMs. The model takes advantage of the exponential nonlinearity in the evolution equations causing a significant increase in the signal-to-noise ratio. The energy, as a result decreases as the recall process is in progress, ensuring the stability of the system. The increase in the signal-to-noise ratio also enhances the capacity of BAM.

4.4.1 Evolution Equations

Suppose we are given N training pairs $\{(A_1, B_1), (A_2, B_2), \dots, (A_n, B_n)\}$

where $A_i =$

and $B_i =$

and if X_i, Y_i are the bipolar

modes of the training pattern pairs A_i and B_i respectively, given by $X_i \in \{-1, 1\}^n$ and $Y_i \in \{-1, 1\}^p$.

Then, we use the following equations in the recall process of eBAM.

$$\langle X \cdot X_i \rangle = \sum_{j=1}^n x_j x_{i_j}$$

$$X_1 = (-1 -1 -1 -1 -1 1 -1 -1 1 1 1 -1 1 1 1 1), \quad Y_1 = (1 -1 -1)$$

$$X_2 = (-1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 1 1 1 1), \quad Y_2 = (-1 1 -1)$$

$$X_3 = (-1 -1 -1 -1 -1 -1 1 -1 -1 1 1 1 1 1 1 1), \quad Y_3 = (-1 -1 1)$$

To retrieve Y_2 corresponding to X_2

$$\langle X_2 \cdot X_1 \rangle = 12$$

$$\langle X_2 \cdot X_2 \rangle = 16$$

$$\langle X_2 \cdot X_3 \rangle = 12$$

Choosing $h = 2$,

$$b^{\langle X_2 \cdot X_t \rangle}, \text{ for } t = 1, 2, 3 \\ = (4096, 65536, 4096)$$

Computing Y using (4.24) yields

$$Y = \left((1 -1 -1) \begin{pmatrix} 4096 \\ 65536 \\ 4096 \end{pmatrix}, (-1 1 -1) \begin{pmatrix} 4096 \\ 65536 \\ 4096 \end{pmatrix}, (-1 -1 1) \begin{pmatrix} 4096 \\ 65536 \\ 4096 \end{pmatrix} \right) \\ = (-65536, 57344, -65536) \\ = (-1 1 -1) \\ = Y_2.$$

(4.26)

The reasons for using an exponential scheme are to enlarge the attraction radius of every stored pattern pair and to augment the desired pattern in the

recall reverberation process.

Example 4.6 (Working of eBAM)

Consider the set of pattern pairs

In the case of noisy patterns, eBAM retrieves the closest pair among the stored patterns which is associated with the noisy pattern.

Example 4.7 (Recall of noisy vectors by eBAM)

For the set of pattern pairs considered in Example 4.6. Consider the retrieval of

$$X = (1 -1 -1 -1 -1 1 1 -1 1 1 -1 1 1 1 -1)$$

Now the Hamming distance of X from each of X_1, X_2, X_3 is $HD(X, X_1) = 8$

$$HD(X, X_2) = 4$$

$$HD(X, X_3) = 10$$

Observe that X is closer to X_2 by way of its distance.

Now, the application of eBAM's evolution equations results in

$$\langle X \cdot X_1 \rangle = 8$$

$$\langle X \cdot X_2 \rangle = 12$$

$$\langle X \cdot X_3 \rangle = 8$$

Choosing $b = 2$, the vector retrieved is

$$Y = (-1 1 -1)$$

which is Y_2 and is the desired result.

Given a set of N training pairs (A_i, B_i) , $i = 1, 2, \dots, N$ where $A_i = (a_{i_1}, a_{i_2}, \dots, a_{i_n})$ and $B_i = (b_{i_1}, b_{i_2}, \dots, b_{i_n})$, $a_{i_j}, b_{i_j} \in \mathfrak{R}$, the normalized input pattern (\hat{A}_i) is given by

$$(\hat{A}_i) = \frac{A_i}{\| A_i \|}$$

where,

$$\| A_i \| = \sqrt{\sum_{j=1}^n (a_{ij})^2} \quad (4.27)$$

Needless to say, (\hat{A}_i) $i=1, 2, \dots, n$ are vectors of unit length.

4.5

ASSOCIATIVE

MEMORY

FOR

REAL-CODED

PATTERN PAIRS

Most associative memory architectures insist on binary or bipolar pattern pairs. Rajasekaran and Pai (1998) proposed an associative memory model termed *simplified Bidirectional Associative Memory* (sBAM) which not only associates patterns represented in bipolar forms but also those that are real-coded. However, the evolution equations demand the normalization of real-coded input pattern vectors to vectors of unit length, before their application.

Thus, sBAM proceeds in two steps, namely input normalization and the application of evolution equations.

4.5.1 Input Normalization

4.5.2 Evolution Equations

Let α be the vector to be submitted to retrieve the associated pair β . Now α

could represent a stored pattern or a noisy or an unknown pattern.

Irrespective of the case, the general system of equations is given as: Frame the correlation vector

$M_{n \times 1} = [m_j]$ given by

$$m_j = \phi(\langle \alpha \cdot \hat{A}_j \rangle), \quad j = 1, 2, \dots, N \quad (4.28)$$

where,

$$\phi(x) = \begin{cases} 1, & \text{if } x = \max(\langle \alpha \cdot \hat{A}_j \rangle) \\ 0, & \text{otherwise} \end{cases} \quad (4.29)$$

Now β is retrieved as

$$\beta_k = \sum_{l=1}^N (\hat{a}_{lk} m_k), \quad k = 1, 2, \dots, p \quad (4.30)$$

where β_k is the k th component of the vector β and \hat{a}_{lk} is the k th component of \hat{A}_l .

$$m_d = \phi(\langle \alpha \cdot \hat{A}_d \rangle) = 1,$$

$$\hat{B}_d$$

$$\hat{B}_l$$

$$\hat{B}_s$$

$$\begin{bmatrix} m_1 = 0 \\ m_2 = 0 \\ \vdots \\ \vdots \\ m_s = 0 \\ \vdots \\ m_l = 0 \\ \ddots \\ m_N = 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} m_1 = 0 \\ m_2 = 0 \\ \vdots \\ \vdots \\ m_s = 1 \\ \vdots \\ m_l = 0 \\ \ddots \\ m_N = 0 \end{bmatrix}$$

It is to be observed that the computation of Eq. (4.30) is unnecessary since for some d , when

the corresponding

is the vector to be

retrieved.

Also, in the event of noisy vectors, it is to be noted that there could exist an l , $s \in \{1, 2, \dots, N\}$, $l \neq s$ such that $ml = ms = 1$. In such a case, it implies that the noisy vector α is close to $\hat{\alpha}_l$ and $\hat{\alpha}_s$ by the same “distance”. Thus, if $d_{\alpha l}$ is the Euclidean distance between α and $\hat{\alpha}_l$ and $d_{\alpha s}$ that between α and $\hat{\alpha}_s$, then $d_{\alpha l} = d_{\alpha s}$. The system, therefore, could retrieve or depending on whether matrix M is chosen to be

(4.31)

assuming $s < l$. Algorithm 4.2 illustrates the working of sBAM.

Algorithm 4.2 (Simplified Bi-directional Associative Memory)

```

Algorithm sBAM (N, X, Y)

/* N is the number of pattern sets */
/* (X, Y) : are the pattern pair sets where
   X = {X1, X2, ..., XN} and Y = {Y1, Y2, ..., YN} */
/* Xi = (Xi1, Xi2, ..., Xin) and Yj = (Yj1, Yj2, ..., Yjm) */

Step 1: Normalize (X, Y)

  for i ← 1 to N
    Xi(norm) =  $\frac{\bar{X}_i}{\|\bar{X}_i\|}$ ;
    Yi(norm) =  $\frac{\bar{Y}_i}{\|\bar{Y}_i\|}$ ;
  end.

Step 2: Input  $\bar{A}$  the pattern vector whose associated pair is
to be recalled. Obtain its normalized equivalent

$$\bar{A}^{(norm)} = \frac{\bar{A}}{\|\bar{A}\|}$$


Step 3: Compute the inner product of  $\bar{A}^{(norm)}$  with

$$\bar{X}_i^{(norm)} \quad i = 1, 2, \dots, N$$

  for j ← 1 to N
    Zi =  $\bar{A}^{(norm)} \cdot \bar{X}_i^{(norm)}$  /* Here, ".·." is the inner product
                                operator           */
  end
  Let Z = {Zi} i = 1, 2, ..., N

Step 4: Apply threshold function  $\phi$  on Z to obtain the
correlation vector M,
  M =  $\phi(Z) = (m_1, m_2, \dots, m_N)$ 
  /*  $\phi$  is as defined in Equation 4.28 */

Step 5: Output  $\bar{Y}_k$  where k is such that  $m_k = \max_{i=1,2,\dots,N} (m_i)$ 

END sBAM.

```

Example 4.8 (Retrieval of stored patterns by sBAM)

A_1 :	(18.82, -26.32, 110.40, 62.80)	B_1 :	(30, 50, 40)
A_2 :	(56.55, 24.82, 98.67, -16.84)	B_2 :	(20, 40, 30)
A_3 :	(11.26, -54.78, -84.86, 100.96)	B_3 :	(10, 30, 20)

The normalized training pairs \hat{A}_i are given by

$$\begin{aligned}\hat{A}_1 &: (0.1436, -0.2008, 0.8423, 0.4791) \\ \hat{A}_2 &: (0.4808, 0.2110, 0.8389, -0.1432) \\ \hat{A}_3 &: (0.0786, -0.3824, -0.5924, 0.7049)\end{aligned}$$

Retrieval of vector $\alpha = (0.4808, 0.2110, 0.8389, -0.1432)$ (which is the stored pattern \hat{A}_2) yields the following calculations:

$$\langle \alpha \cdot \hat{A}_1 \rangle = 0.6647$$

$$\langle \alpha \cdot \hat{A}_2 \rangle = 1$$

$$\langle \alpha \cdot \hat{A}_3 \rangle = -0.6408$$

$$\langle \alpha \cdot \hat{A}_1 \rangle = -0.0657$$

$$\langle \alpha \cdot \hat{A}_2 \rangle = -0.6390$$

$$\langle \alpha \cdot \hat{A}_3 \rangle = -0.9999$$

Consider the following pattern pairs:

On applying the threshold function φ , the correlation vector M is given by $M = (0, 1, 0)$ which retrieves B_2 , which is the correct pair.

Example 4.9 (Retrieval of an unknown pattern by sBAM)

For the set of pairs considered, let us suppose A' (an unknown pattern) is to be presented for retrieval. The system now retrieves the B corresponding to A among the stored pattern pairs which is close to A' .

Let $\alpha = (10.26, -53.80, -81.75, 98.96)$ be an unknown pattern. It may be observed that on an inspection α is ‘closer’ to \hat{A}_3 among $\hat{A}_1, \hat{A}_2, \hat{A}_3$.

The normalized $\alpha = (0.0735, -0.385, -0.5858, 0.7091)$

The recall calculations are

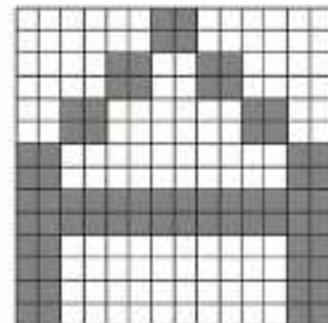
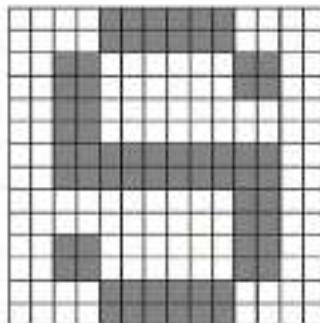
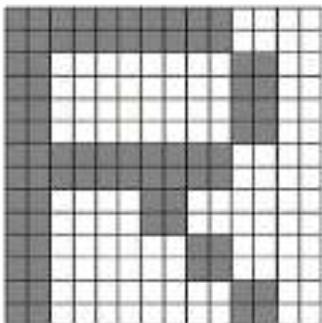
Now, $M = (0, 0, 1)$. Therefore B_3 is retrieved which is again the appropriate pair. It may also be verified that the system has retrieved that pattern which is closest to that of the stored pattern pairs, in the case of noisy patterns.

$$d = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

The Euclidean distance measure d between two real vectors $X = (x_1, x_2, \dots, x_n)$ and

$Y = (y_1, y_2, \dots, y_n)$ is given by The Euclidean distance measure di of the normalized α from A_i , $i = 1, 2, 3$

respectively, is $(d_1, d_2, d_3) = (1.4599, 1.8105, 0.0098)$. Choosing the minimum, we obtain α closer to \hat{A}_3 and therefore the system has appropriately recalled B_3 .



4.6 APPLICATIONS

In this section, the applications of associative memories to real world problems are illustrated. We discuss two applications, namely

Recognition of characters (using bipolar coding)

Fabric defect identification (using real coding)

These applications serve to demonstrate ways in which associative memories can be used for the solution of problems.

4.6.1 Recognition of Characters

Consider a set of English alphabetical characters such as A, B, C, \dots which are to be recognized. The objective is to allow the associative memory model identify the characters presented.

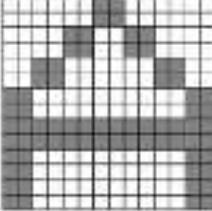
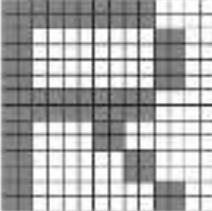
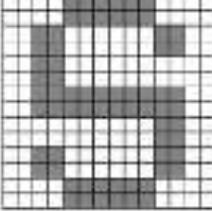
The characters are engraved in a 14×14 grid as shown in Fig. 4.4. These characters are to be associated with their ASCII equivalents. Thus, the (X, Y) pattern pairs which are to be associated using the associative memory model are the grid patterns and their ASCII equivalents.

The grid patterns are represented as a bipolar vector of 196 components. If the pixel in the grid is shaded, the vector component is 1 otherwise it is -1 .

Also, the ASCII numbers of the characters have been represented using their bipolar equivalents. Figure 4.5 shows the bipolar coding of the sample characters, and their ASCII equivalents.

Fig. 4.4 Characters engraved in a grid.

Making use of Wang et al.'s associative memory model and choosing $q = 1$ =

Character	X Bipolar equivalent	Y		
		ASCII value	Binary equivalent	Bipolar equivalent
	$(-1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1$ $1 1 1 1 1 1 1 1 1 1 1 1$ $-1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1$ \dots $1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1$ $1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1)$	65	(1000001)	$Y_1 = (1 -1 -1 -1 -1 -1 1)$
	$(1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1$ $1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1$ $1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1$ \dots $1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 1$ $1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1)$	82	(1010010)	$Y_2 = (1 -1 1 -1 1 1 -1)$
	$(1 -1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 -1$ $-1 -1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 -1$ $-1 -1 1 1 -1 -1 -1 -1 -1 1 1 1 -1 -1$ \dots $-1 -1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 -1$ $-1 -1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 -1)$	83	(1010011)	$Y_3 = (1 -1 1 -1 -1 1 1)$

$2, q 2 = 3$, and

$q 3 = 2$, we frame the correlation matrix $[M]196 \times 7$ given by $M = 2 X T$

T

T

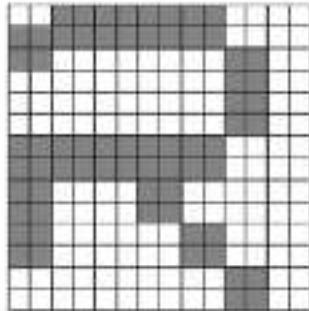
$$1 Y_1 + 3 X_2 Y_2 + 2 X_3 Y_3$$

Fig. 4.5 Bipolar equivalent of the pattern pairs.

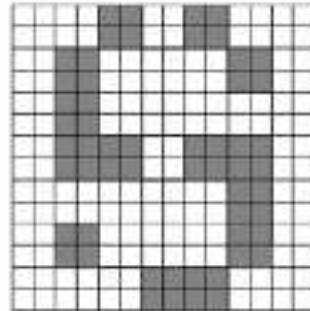
The recall of X_1, X_2, X_3 yields the following results: $X_1 \cdot M = (332, -332, -452, -332, -332, -452, 308)$ $\varphi(X_1 \cdot M) = (1 -1 -1 -1 -1 -1 1)$ (on application of the threshold function φ defined in Eqs. (4.12–4.14)

= Y 1

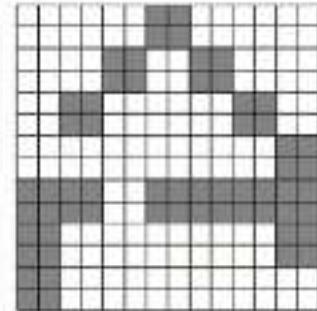
$$X 2 \cdot M = (652, -652, 636, -652, -652, 636, -524) \quad \varphi(X 2 \cdot M) = (1 -1 1 -1 -1 1 -1)$$



R' = Distorted R



S' = Distorted S



A' = Distorted A

= Y 2

$$X 3 \cdot M = (404, -404, 548, -404, -404, 548, 238) \quad \varphi(X 3 \cdot M) = (1 -1 1 -1 -1 1 1)$$

= Y 3

Recall of noisy characters

Consider the set of noisy characters as shown in Fig. 4.6.

Fig. 4.6 Noisy characters.

The recall of the bipolar vectors corresponding to A' , R' and S' which are partially distorted versions of the respective characters A , R and S yields $A' \cdot M = (356, -356, -332, -356, -356, -332, 284) \quad \varphi(A' \cdot M) = (1 -1 -1 -1 -1 -1 1 1)$

= Y 1 (character A)

$$R' \cdot M = (644, -644, 596, -644, 596, -388)$$

$$\varphi(R' \cdot M) = (1 -1 1 -1 -1 1 -1)$$

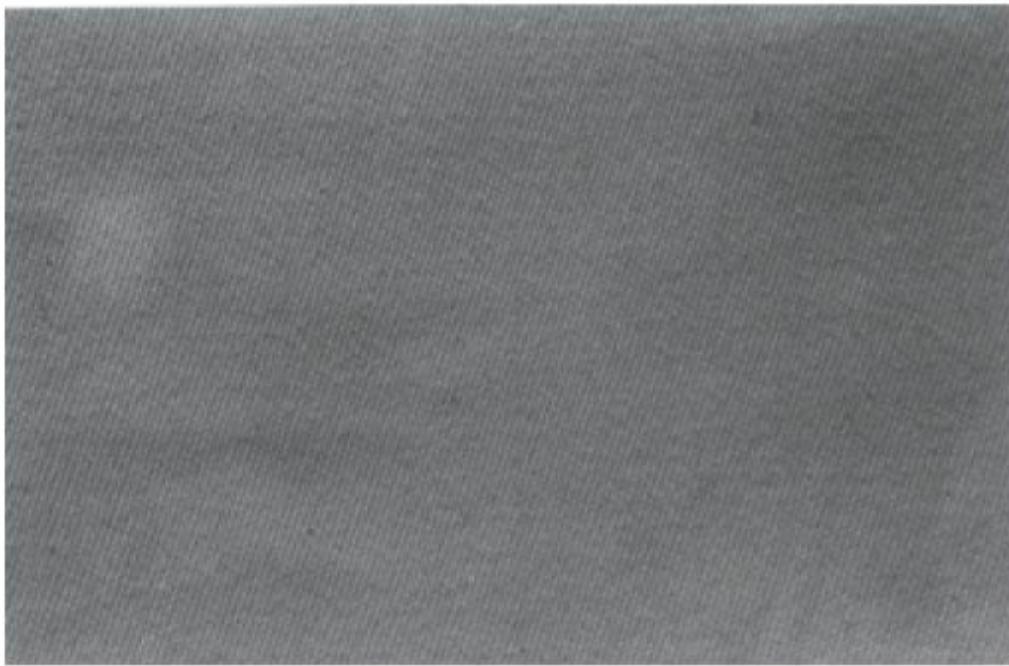
= Y 2 (character R)

$$S' \cdot M = (324, -324, 404, -324, -324, 404, 252) \varphi(S' \cdot M) = (1 -1 1 -1 -1 \\ 1 1)$$

= Y 3 (character S)

4.6.2 Fabric Defect Identification

Inspection of fabrics for defects depends on human sight and the results are greatly dependent on the mental and physical condition of the inspector.



Textile engineers have therefore begun to seek assistance from computers and in recent years have found *neural computing* to hold a lot of potential in handling a wide range of problems in textile engineering.

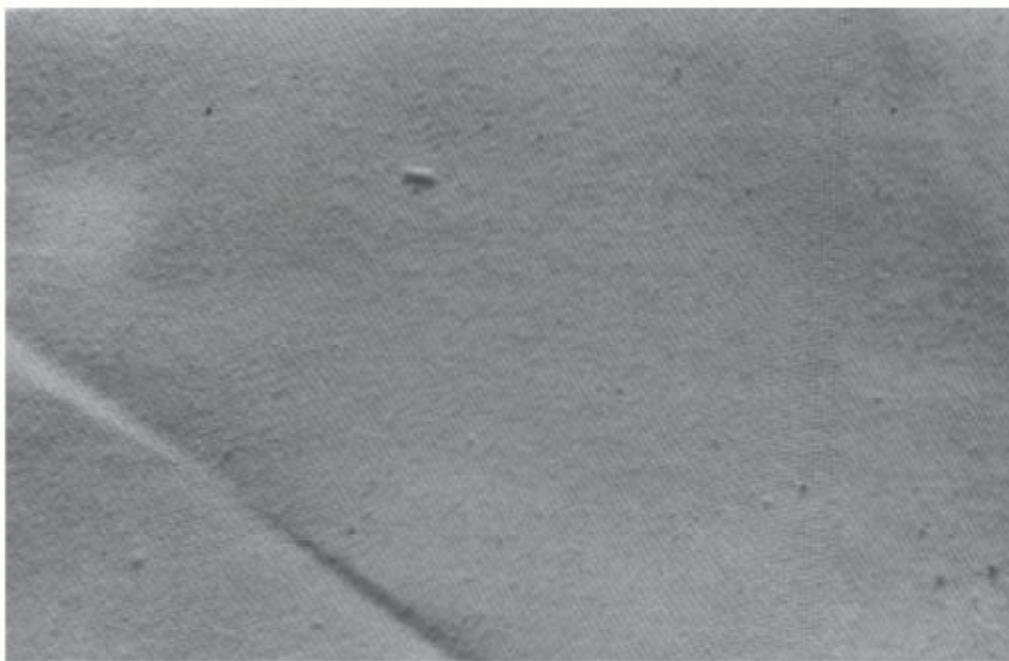
Thus, for the problem of fabric defect identification, Tsai et al. (1995) applied conventional multilayer perceptron using the backpropagation algorithm as a learning strategy. However, an elegant solution has been suggested by Rajasekaran (1997) by making use of a self-organizing network namely *training free counterpropagation network*, which is an improvement

(Rajasekaran and Pai, 1997) over Hecht Nielsen's counterpropagation network.

The sBAM scheme offers an elegant solution to the fabric defect identification problem. The defects to be identified are *nep*, *broken end*, *broken pitch*, and *oil stain*. For purposes of classification, the categories are identified by numbers, namely 1—normal, 2—*nep*, 3—*broken end*, 4—*broken pitch*, and 5—*oil stain*. Figure 4.7(a) illustrates a normal fabric and Figs.

4.7(b)–4.7(d) show some kinds of fabric defects.

Fig. 4.7(a) Normal fabric.



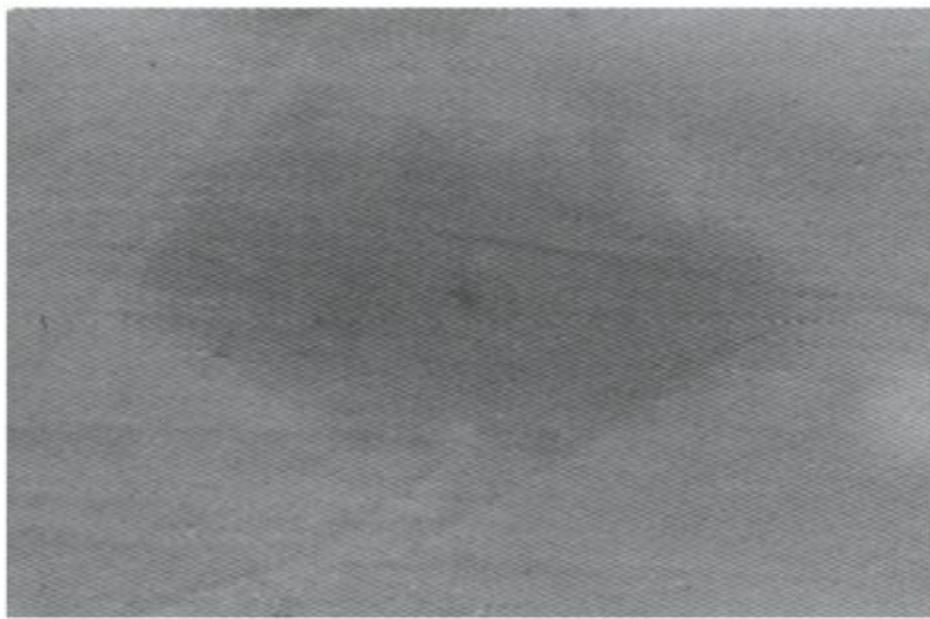
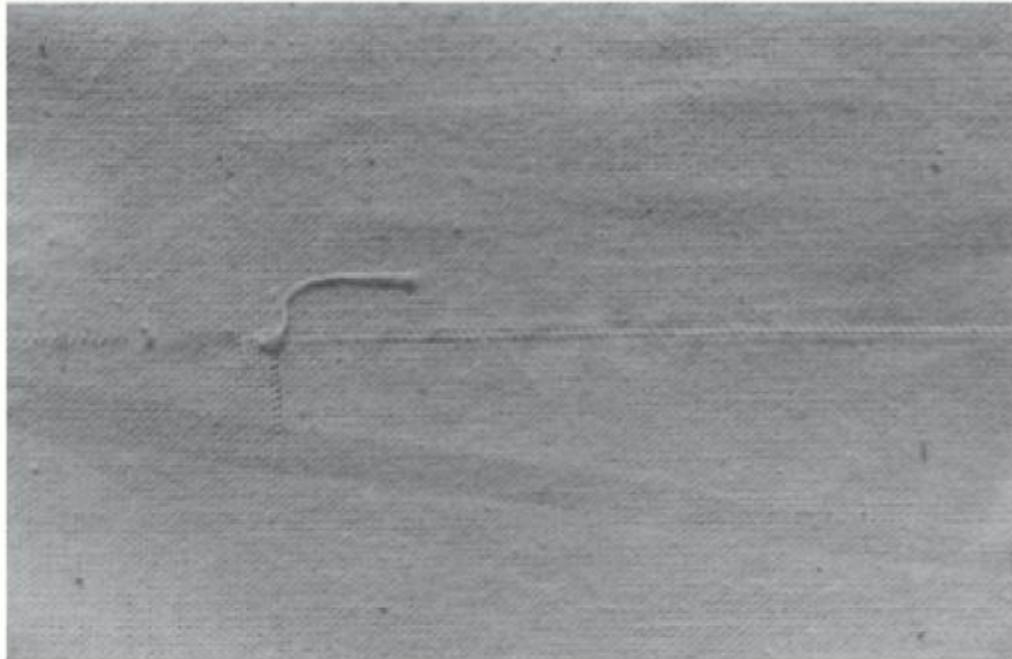


Fig. 4.7(b) Nep.

Fig. 4.7(c) Broken end.

Fig. 4.7(d) Oil stain.

Tsai et al. employed a gray level co-occurrence matrix (Sobus et al., 1997) to obtain the feature parameters $f_1, f_2, f_3, f_4, f_5, f_6$ for each fabric defect.

Among the feature parameters, f_1, f_2, f_3 , and f_4 are the contrast (CON) measurements of the texture images along $0^\circ, 45^\circ, 90^\circ$ and 135° , when the spatial displacement d is equal to 1. Here, f_5, f_6 are the contrast values at $d =$

$12, \theta = 0^\circ$ and $d = 16, \theta = 90^\circ$ respectively. Here, θ is the direction angle.

Tsai et al.'s experimental data has been used for the associative memory model. Table 4.1 refers to a sample set of input representing some defects and stored in the associative memory model. Table 4.2 shows the results of the defects identified by sBAM, when a testing set comprising unknown instances of the various defects was presented for retrieval. The sBAM model is able to identify fabric defects with 100% accuracy.

Table 4.1 Sample set of stored patterns presented to sBAM for fabric defect identification f_1

f_2

f_3

f_4

f_5

f_6

Defects

0.3978

0.6433

0.3704

0.4430

0.3484

0.3811

1

0.3920

0.6464

0.3532

0.4221

0.3352

0.3859

1

0.3887

0.6363

0.3601

0.4202

0.3220

0.3257

1

0.3851

0.6228

0.3567

0.4361

0.3496

0.3371

1

0.3529

0.5768

0.3219

0.3865

0.4417

0.4725

2

0.3465

0.584

0.3225

0.3819

0.4740

0.5255

2

0.3467

0.5767

0.3130

0.3782

0.3845

0.4925

2

0.3537

0.5642

0.3182

0.3918

0.4358

0.5035

2

0.3159

0.5158

0.3214

0.3981

0.5433

0.3301

3

0.3354

0.5356

0.3373

0.4095

0.5594

0.3677

3

0.3534

0.5655

0.3275

0.4129

0.5210

0.3301

3

0.3761

0.5795

0.3399

0.4324

0.5290

0.3305

3

0.3765

0.6080

0.3098

0.3824

0.3198

0.3578

4

0.3840

0.5953

0.3123

0.3920

0.3165

0.4022

4

0.3854

0.6023

0.3101

0.3890

0.3154

0.3635

4

0.3873

0.5970

0.3074

0.3944

0.3554

0.3735

4

0.3592

0.4453

0.3000

0.3543

0.4973

0.4100

5

0.4049

0.4874

0.3207

0.3977

0.5187

0.4240

5

....

Table 4.2 Results of the sample testing set (unstored patterns) presented for fabric defect identification Defect

Actual

f_1

f_2

f_3

f_4

f_5

f_6

identified by sBAM

defect

0.3900

0.6402

0.3584

0.4205

0.3726

0.3434

1

1

0.4026

0.6362

0.3601

0.4320

0.3438

0.3442

1

1

0.3879

0.6161

0.3419

0.4153

0.3228

0.3547

1

1

0.3689

0.6188

0.3483

0.4026

0.4393

0.4813

2

2

0.3789

0.6173

0.3447

0.4042

0.3954

0.4213

2

2

0.3663

0.6173

0.3444

0.4045

0.4439

0.4788

2

2

0.3881

0.6345

0.3569

0.4305

0.4214

0.5121

2

2

0.3509

0.5957

0.3507

0.4079

0.5432

0.3107

3

3

0.3661

0.5915

0.3361

0.4137

0.4808

0.2884

3

3

0.3717

0.5968

0.3237

0.4003

0.4708

0.3376

3

3

0.3723

0.5821

0.2097

0.3695

0.3453

0.3765

4

4

0.3836

0.6022

0.3054

0.3861

0.3383

0.3429

4

4

0.4000

0.4976

0.3254

0.3969

0.5242

0.4233

5

5

0.2626

0.3115

0.2417

0.2633

0.4584

0.3841

5

5

0.4051

0.5158

0.3361

0.4082

0.6228

0.6095

5

5

(\bar{a}_i, \bar{b}_i)

4.7 RECENT TRENDS

The bidirectional associative memory is a two-layer nonlinear recurrent network associating pattern pairs

where $i = 1, 2, \dots, N$. The model can be

generalized to display multiple association of pattern vectors (ai, bi, ci, \dots) where $i = 1, 2, \dots, N$. Such an associative memory model termed *multiple association memory* has been proposed by Hagiwara (1990). Most learning methods for BAM do not consider the basin of attraction seriously which may restrict the application of a BAM as a CAM (*content addressable memory*). Wang et al. (1994) proposed a learning algorithm for BAM with optimal stability which guarantees the storage of training patterns with basins of attraction as large as possible.

An asymmetric BAM with asymmetric feedforward, feedback connections, pattern nonorthogonality and relatively large capacity has been proposed by Xu et al. (1994). A BAM model which uses an optimal associative memory matrix in place of the standard Hebbian or quasicorrelation matrix has been suggested by Wang (1996). Iku and Makoto (1996) have proposed a complex associative memory and Lee and Wang (1998), a multivalued bidirectional associative memory.

$$T = \sum_{i=1}^m A_i^T \cdot A_i$$

$$M = \Sigma X_i^T \cdot Y_i$$

$$M = \Sigma q_i X_i^T \cdot Y_i$$

SUMMARY

Associative memories are a class of neural network architectures which store associated patterns in some form and recall them when they are incited with their associated pairs. The associative mapping is quite often a general nonlinear matrix type operator. The patterns presented to the network could be an exact replica of the stored patterns or a noisy version of the same.

The associative memory architectures can be classified into *heteroassociative* and *autoassociative* models, depending on the pattern set processed, and into

static and *dynamic* models, based on the recall mechanism employed to retrieve a pattern.

First order autocorrelators obtain their connection matrix by multiplying a pattern's element with every other pattern's element, i.e.

The recall equation is a vector-matrix multiplication followed by a pointwise nonlinear threshold function.

Kosko extended the unidirectional autoassociators to bidirectional associative processes. Making use of a correlation matrix computed from the pattern pairs, the system proceeds to retrieve the nearest pattern pair given any pair (α, β) , with the help of recall equations. However, Kosko's encoding method does not ensure that the stored pairs are at a local minimum and hence, results in incorrect recall.

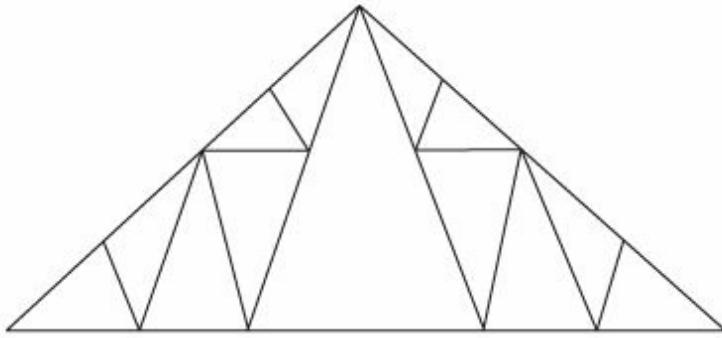
Wang et al.'s *multiple training encoding strategy* which is an enhancement of Kosko's network ensures correct recall of pattern pairs. The generalized correlation matrix suggested is

Wang and Don's eBAM reports higher capacity for pattern pair storage than conventional BAMs. The exponential nonlinearity in the evolution equations results in the decrease of energy, thereby ensuring correct recall.

Most BAM architectures employ bipolar/binary encoding of pattern pairs which are to be associated. Rajasekaran and Pai suggested sBAM

which associates pattern pairs that are not only bipolar coded but also real-coded. The patterns, however, are to be normalized before application of the evolution equations.

Finally, two applications, namely recognition of characters and fabric defect identification have been discussed. The former employs bipolar encoding and the latter, real encoding of pattern pairs.



PROGRAMMING ASSIGNMENT

P4.1 Fink Truss Design. In the design of roof trusses, the initial areas of members are essential for the analysis. If the truss configuration is determinate then the initially assumed areas affect the deflection of the truss system and if it is indeterminate, it then affects the forces in the members in addition to deflection. In case, the assumed area deviates much from the actual area necessary, the system has to be reanalyzed and redesigned and the process is to be repeated until the area taken for the analysis and design are the same.

The problem here is to obtain the initial area of truss configuration. For the Fink Truss shown in Fig. P4.1, span, angle, type of access ('provided'/'not provided'), and the spacing of the trusses are taken as inputs. However, it is assumed that the access is provided, thereby reducing the number of inputs to three. The outputs are the areas of the four regions, namely top chord members, bottom chord members, and the two web members (area 1–area 4). Table P4.1 illustrates a sample set of data to be recorded by the associative memory model.

Fig. P4.1 Fink truss.

Table P4.1

Span

Angle

Spacing

Area 1

Area 2

Area 3

Area 4

10

26

4

2275

2275

959

959

10

20

3

2275

2275

1225

959

18

28

4

2275

2275

1456

959

12

22

3

2275

2275

959

959

20

26

3

2275

2275

1351

959

15

26

3

2275

2275

1148

959

16

23

3

2275

2275

1148

959

(i) Normalize the input-output data sets of Table P4.1.

(ii) Implement Algorithm 4.2 for simplified bidirectional associative memory.

(iii) Test for the outputs retrieved corresponding to the data set given in Table P4.2 (after normalization).

Table P4.2

Span

Angle

Spacing

Area 1

Area 2

Area 3

Area 4

10

24

4

2275

2275

959

959

16

26.52

7

3269

3269

1729

1351

12

20

3

2275

2275

959

959

18

25

4

2506

2506

1351

959

P4.2 Satellite Image Identification. Wang et al. (1990) applied their associative memory model to a satellite image identification system.

Here, a set of satellite images are to be identified. A sample set of images and their associated pairs have been illustrated in Fig. P4.2.

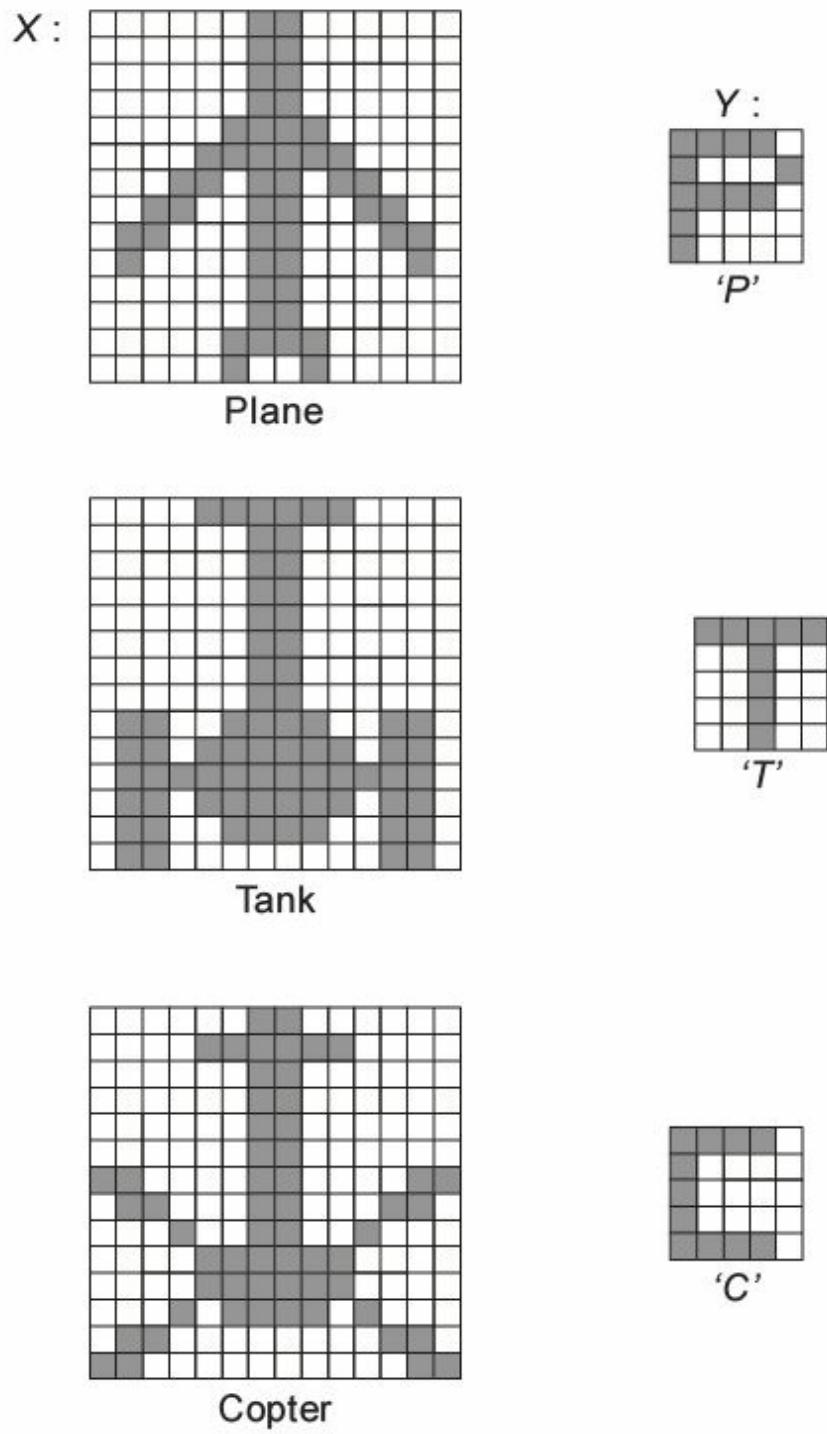


Fig. P4.2 Satellite images and their associated pairs.

- (i) Design a graphics user interface to accept these images.
- (ii) Convert the images into their bipolar equivalents.

(iii) Implement Wang et al.'s multiple training encoding strategy (Algorithm 4.1).

(iv) Test for the recognition of stored patterns (X).

(v) Test for the recognition of noisy images (X'), a sample of which is shown in

Fig. P4.3.

$X :$

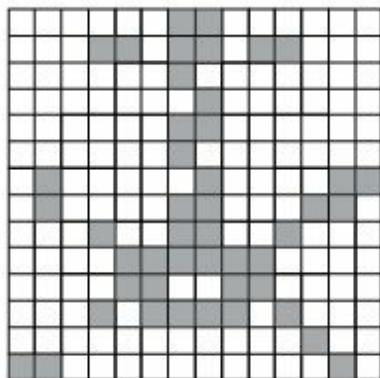
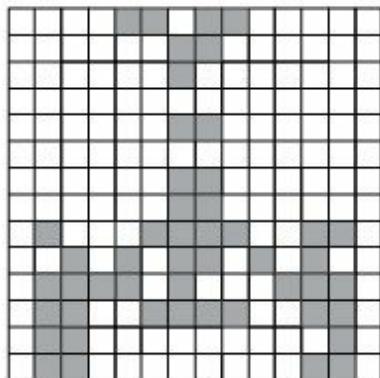
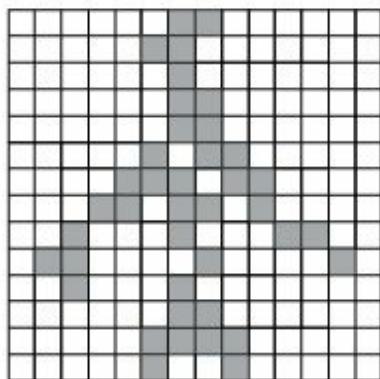


Fig. P4.3 Noisy satellite images.

REFERENCES

Amari, S. (1972), Learning Patterns and Other Pattern Sequences by Self-Organizing Nets of Threshold Elements, *IEEE Trans. Comput.*, Vol. C-21, pp. 1197–1206.

Amari, S. (1977), Neural Theory of Association and Concept Formation, *Biol. Cybern.*, Vol. 26, pp. 175–185.

Anderson, J.A. (1983), Cognitive and Psychological Computation with Neural Models, *IEEE Trans. Syst., Man, Cybern.*, Vol. SMC-13, No. 5, pp. 799–815.

Cruz, Jr. J.B. and A.R. Stubberud (1987), Intelligent Control of Variably Configured Systems using Neural Networks, *in Proc. IEEE TENCON 87*, pp. 893–989.

Hagiwara, M. (1990), Multidimensional Associative Memory, *Proc. 1990 IEEE Joint Conf. On Neural Networks*, IEEE, Vol. 1, pp. 3–6.

Hebb, D.O. (1949), *The Organization of Behaviour*, John Wiley, NY.

Hirai, Y. (1983), A Model of Human Associative Processor (HASP), *IEEE Trans. Syst. Man, Cybern.*, Vol. SMC-13, No. 5, pp. 851–863.

Iku Nemoto, and Kubono Makoto (1996), Complex Associative Memory, *Neural Networks*, (9)2, pp. 253–261.

Hopfield, J.J. (1982), Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proc. of the National Academy of Sciences*, Vol. 79, pp. 2554–2558.

Kohonen, T. (1972), Correlation Matrix Memories, *IEEE Trans. Comput.*, Vol. 21, pp. 353–359.

Kohonen, T. (1977), *Associative Memory—A System Theoretical Approach*, Springer Verlag.

Kohonen, T. and E. Oja (1976), Fast Adaptive Formation of Orthogonalizing Filters and Associative Memory in Recurrent Networks of Neuron like Elements, *Biolog. Cybern.* , Vol. 21, pp. 85–95.

Kosko, B. (1987a), Constructing an Associative Memory, *Byte*, Vol. 12, No. 10, pp. 137–144.

Kosko, B. (1987b), Adaptive Bidirectional Associative Memories, *Appl. Opt.*, Vol. 26, No. 23, pp. 4947–4960.

Kosko, B. (1988), Bidirectional Associative Memories, *IEEE Trans. Syst. Man, Cybren.* , Vol. 18, No. 1, pp. 49–60.

Donq-Liang Lee and Wen-June Wang (1998), A Multivalued Bidirectional Associative Memory Operations on a Complex Domain, *Neural Networks*, (11)9, pp. 1623–1635.

Little, W. (1974), The Existence of Persistent States in the Brain, *Math. Biosci.*, Vol. 19, pp. 101–120.

Little, W. and G. Shaw (1978), Analytical Study of the Memory Storage Capacity of a Neural Network, *Math. Biosci.*, Vol. 39, pp. 281–289.

Nakano, K. (1972), Associatron—a Model of Associative Memory, *IEEE Trans. Syst., Man, Cybren.*, Vol. SMC-2, No. 3, pp. 380–388.

Palm, G. (1980), On Associative Memory, *Biolog. Cybern.*, Vol. 36, pp. 19–31.

Rajasekaran, S. (1997), Training Free Counterpropagation Neural Network to Pattern Recognition in Fabric Defects, *Textile Research J.*, 67(6), pp. 401–405.

Rajasekaran, S. and G.A. Vijayalakshmi Pai (1997), Training Free Counterpropagation Network as Static Heteroassociative Memories, *Indian Journal of Engineering and Material Sciences*, Vol. 4, Dec., pp. 245–253.

Rajasekaran, S. and G.A. Vijayalakshmi Pai (1998), Simplified Bidirectional Associative Memory for the Retrieval of Real Coded Patterns, *Eng. Int. Syst.*, 4, pp. 237–243.

Sobus, J.B., Pourdeyhimi, J. Gerde and Y. Ulcay (1991), Assessing Changes in Texture Periodicity due to Appearance Loss in Carpets: Gray Level Cooccurrence Analysis, *Textile Research J.*, 61(10), pp. 557–567.

Tsai, I.S., C.H. Lin, and J.J. Lin (1995), Applying an Artificial Neural Network to Pattern Recognition in Fabric Defects, *Textile Research J.*, 65(3), pp. 123–130.

Zheng-ou-Wang (1996), A Bidirectional Associative Memory based on Optimal Linear Associative Memory, *IEEE Trans. on Computers*, Vol. 45, No. 10.

Wang, Y.F., B. Cruz Jose Jr. and H. Mulligan James Jr. (1990), Two Coding Strategies for Bidirectional Associative Memory, *IEEE Trans. On Neural Networks*, Vol. 1, No. 1, March, pp. 81–92.

Wang, Y.F., B. Cruz Jose Jr., and H. Mulligan James Jr. (1990), On Multiple Training for Bidirectional Associative Memory, Letters, *IEEE Trans. on Neural Networks*, Vol. 1, No. 3, September, pp. 275–276.

Wang, Y.F., B. Cruz Jose Jr., and H. Mulligan James Jr., (1991), Guaranteed Recall of all Training Pairs for Bidirectional Associative Memory, *IEEE Trans. on Neural Networks*, Vol. 2, No. 6, November.

Wang Tao, Zhuang Xinhua, and Xing Xiaoliang (1994), Designing Bidirectional Associative Memories with Optimal Stability, *IEEE Trans. Syst. Man, Cybern.*, Vol. 24, No. 5, May.

Wang Chua-Chin and Don Hon Son (1995), An Analysis of High Capacity Discrete Exponential BAM, *IEEE Trans. on Neural Networks*, Vol. 6, No. 2, March, pp. 492–496.

Zong-Ben Xu, Yee Leung, and Xiang-Wei He (1994), Asymmetric Bidirectional Associative Memories, *IEEE Trans. Syst. Man, Cybern.*, Vol. 24, No. 10, October.

Chapter 5

Adaptive Resonance Theory

5.1 INTRODUCTION

One of the main goals of computer science is to develop an intelligent machine that can

perform satisfactorily in unaided fashion in a complex environment.

Paradigms such as

Adaptive Resonance Theory (ART) that learn in an unsupervised fashion represent an attempt to fulfil the goal. ART, the Adaptive Resonance Theory

was introduced by Stephen Grossberg (1988) in 1976. Currently, ART headquarters are located at The Centre for Adaptive Systems and Deptt. of Cognitive and Neural Systems, Boston University (<http://cns-web.bu.edu>).

The term *resonance* refers to the so called resonant state of the network in which a category prototype vector matches the current input vector so close enough that the orienting system will not generate a reset signal in the other *attentional* layer. The networks learn only in their resonant states. The architecture of ART is based on the idea of *adaptive resonant feedback* between two layers of nodes as developed by Grossberg (1988). In case of ART paradigm, autonomous learning and pattern recognition proceed in a stable fashion in response to an arbitrary sequence of input patterns. In this paradigm, self-regulatory control structure is embedded into competitive learning mode. ART is capable of developing stable clustering of arbitrary sequences of input patterns by self-organization.

5.1.1 Cluster Structure

Patterns can be viewed as points of N-dimensional feature space and we expect a pattern similar in some respects. On the basis of class membership or other attribute value, it would be close to each other in the pattern space.

Thus, pattern belonging to class C_1 would cluster more closely to one another than any pattern belonging to class C_i . Of course, in many practical applications these clusters overlap.

$$d = \|X^p - C_j\| = \left[\sum_{i=1}^N (X_i^p - C_{ji})^2 \right]^{1/2}$$

$$\|X^p - C_k\| < \|X^p - C_j\| = \begin{cases} j = 1, \dots, M \\ j \neq k \end{cases}$$

$$\|X^p - C_k\|$$

$$\|X^p - C_k\| < \rho$$

$$\|X^p - C_k\| > \rho$$

$$C_x = \frac{1}{N_x} \sum_{x \in S_n} X$$

Many unsupervised learning algorithms try to identify several prototypes of exemplars that can serve as cluster centres. K-means algorithm, ISODATA algorithm, and Vector Quantization (VQ) technique (see Pao, 1989 and Tou and Gonzals, 1974), are examples of decision theoretical approaches for cluster formation. ART structure is a neural network for cluster formation in an unsupervised learning domain. In these architectures, the number of output nodes cannot be accurately determined in advance.

5.1.2 Vector Quantization

It is customary to cluster input vectors based on distance functions within an Euclidean space. VQ presented in this section and ART presented in the following sections of this chapter present two distinct approaches to the dynamic allocation of cluster centres. VQ is non-neural approach whereas ART is a neural network approach.

To begin with, in VQ, since no cluster has been allocated, the first pattern will force the creation of cluster to hold it. Whenever a new input pattern is encoded, the Euclidean distance between it and any allocated cluster is calculated. If we designate the p th input vector as X_p and j th cluster as C_j then Euclidean distance d is calculated as

(5.1)

The cluster closest to the input is determined such that

(5.2)

where M is the number of allocated clusters.

Once the closest cluster k has been determined, the distance must be tested against the threshold distance ρ as

pattern assigned n th cluster

a new cluster is allocated to p (5.3)

and every time the cluster centre must be updated as

...(5.4)

A program **VECQUANT** is written in Fortran for cluster formation.

There are two drawbacks of the method, namely

1. Sensitivity to sequence of presentation of input and
2. Arbitrary selection of threshold distance at which new clusters are created.

Example 5.1

Figure 5.1 shows 12 points in a two dimensional Euclidean space. The aim is to cluster these utilizing VQ. The input patterns, i.e. X and Y coordinates of 12 points are given as (Table 5.1).

Table 5.1 Coordinates of 12 points

Points

X

Y

Points

X

Y

1

2

3

7

6

4

2

3

3

8

7

4

3

2

6

9

2

4

4

3

6

10

3

4

5

6

3

11

2

7

6

7

3

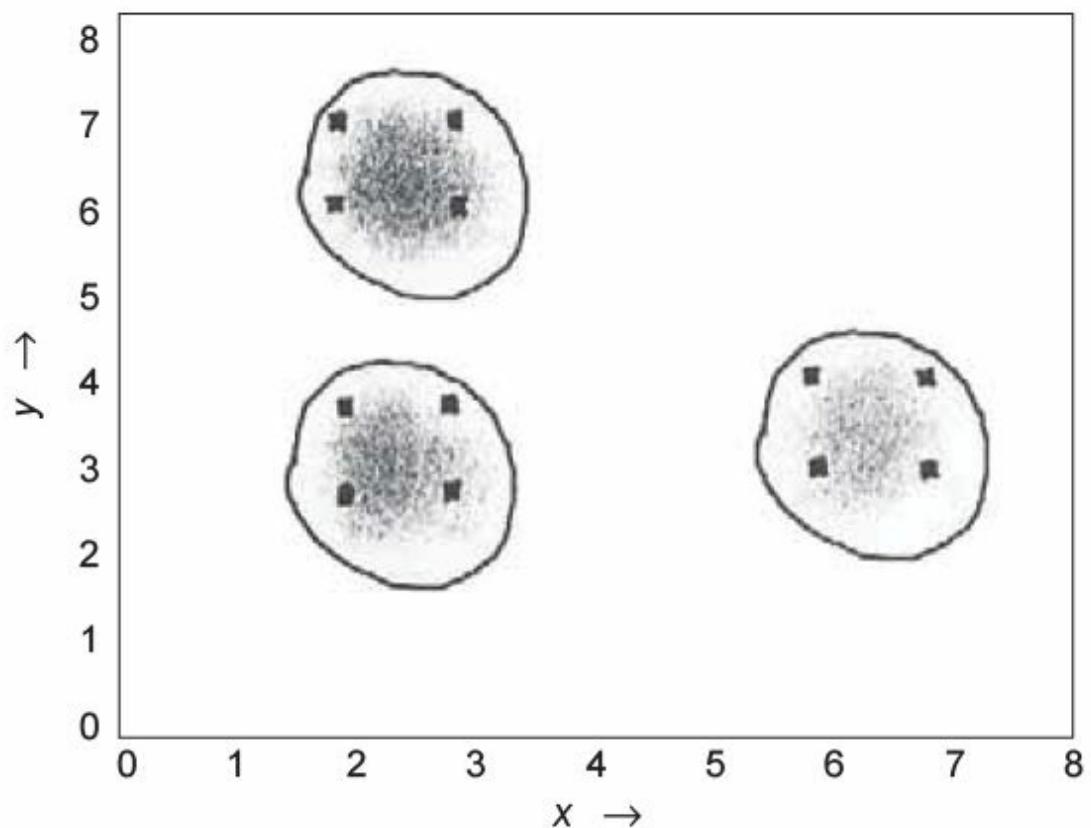
12

3

7

Let us take a threshold distance of 2.0. It is to be noted that there are three clusters with centres $C_1 = (2.5, 3.5)$; $C_2 = (2.5, 6.5)$; and $C_3 = (6.5, 3.5)$. The cluster membership list can also be seen as $S(1) = \{1, 2, 9, 10\}$, $S(2) = \{3, 4, 11, 12\}$, and $S(3) = \{5, 6, 7, 8\}$. The corresponding results are shown

graphically in Fig. 5.1(a). Also note that the clusters identified are very much what humans might expect from looking at the patterns.



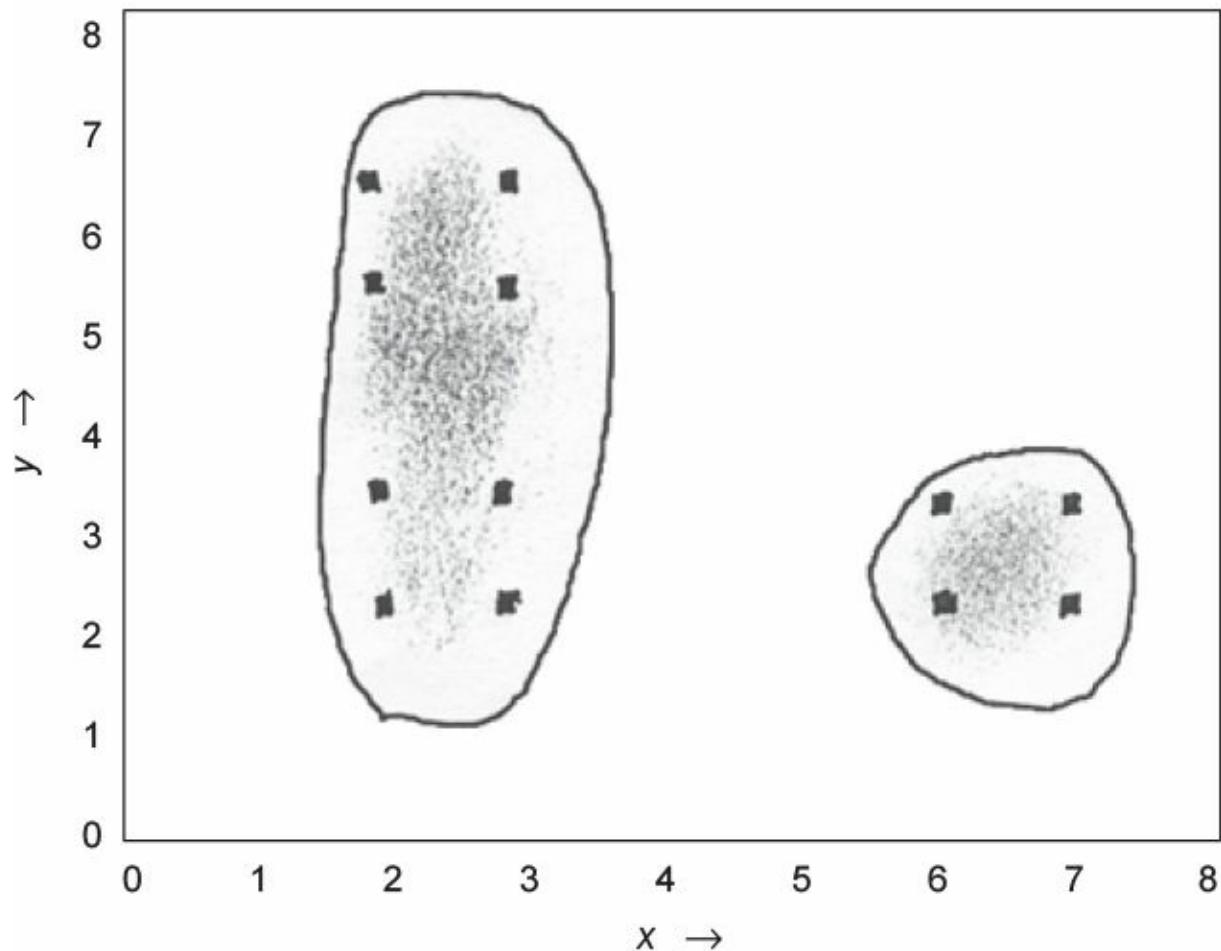
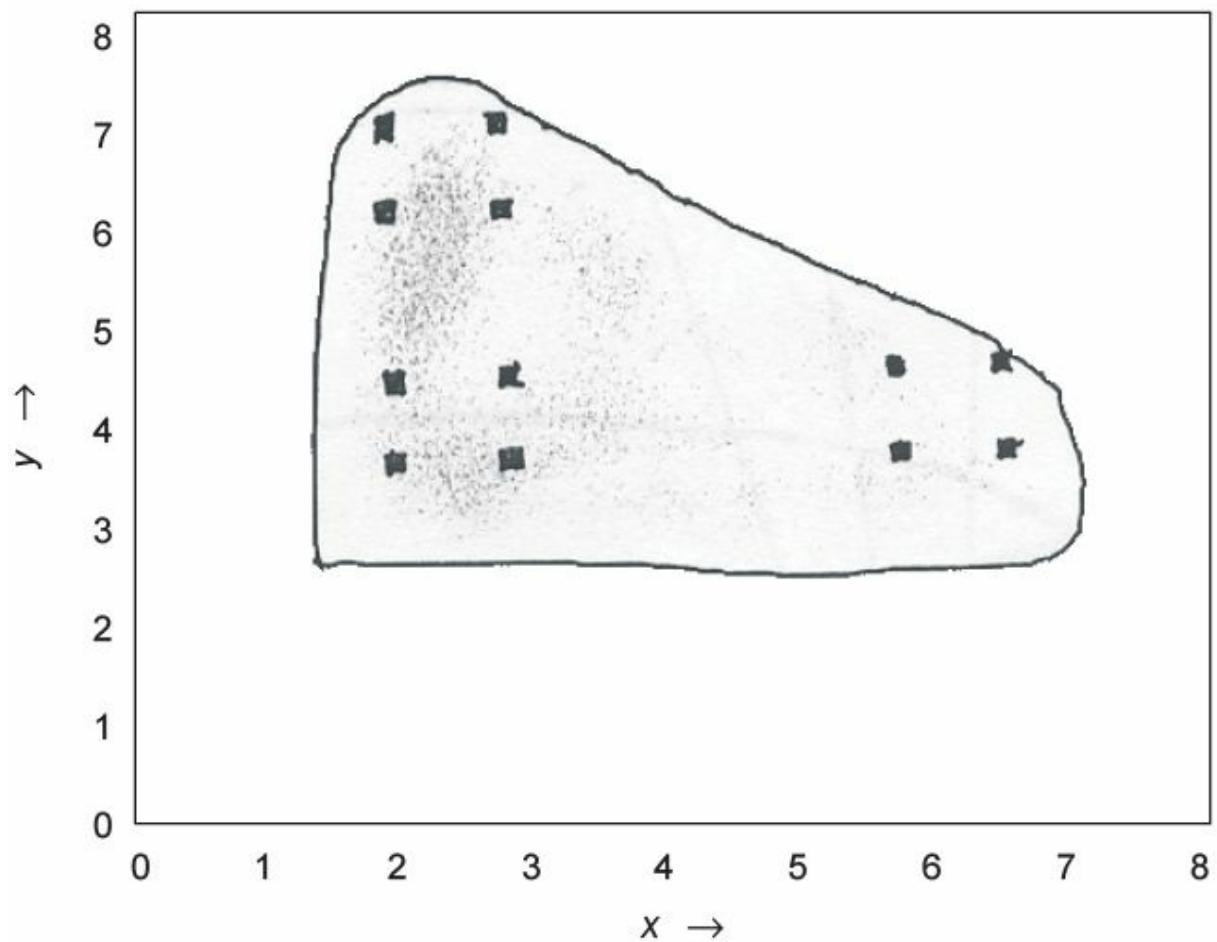


Fig. 5.1(a) Input pattern for VQ. Example 5.1 (Threshold distance 2).

Fig. 5.1(b) Input pattern for VQ. Example 5.1 (Threshold distance 3.5).

The same example is repeated with a threshold distance of 3.5. It is to be noted that there are two clusters with cluster centres $C_1 = \{2.5, 5\}$ and $C_2 = \{6.5, 3.5\}$. The cluster membership list can now be seen as $S(1) = \{1, 2, 3, 4\}$,



$\{9, 10, 11, 12\}$ and $S(2) = \{5, 6, 7, 8\}$. Again, these results are shown graphically in Fig. 5.1(b). From this it is clear that choosing very large threshold distance may easily obscure meaningful categories. Conversely, choosing too low threshold may lead to the proliferation of many meaningful categories. If the threshold distance is 4.5, all patterns are grouped into one category as $S(1) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. The results are shown in Fig. 5.1(c) followed by outputs for all the three cases.

Fig. 5.1(c) Input pattern for VQ. Example 5.1 (Threshold distance 4.5).

```
*****
pattern of      1
assigned to cluster 1
the new cluster centres are
cluster      1      2.000000 3.000000
cluster      1      1
*****  

pattern      2
belongs to the cluster      1
distance      1.000000
cluster centre 2.500000 3.000000
*****  

pattern      3
belongs to the cluster      2
distance      3.041381
cluster centre 2.000000 6.000000
*****  

pattern      4
belongs to the cluster      2
distance      1.000000
cluster centre 2.500000 6.000000
*****  

pattern      5
belongs to the cluster      3
distance      3.500000
cluster centre 6.000000 3.000000
*****  

pattern      6
belongs to the cluster      3
distance      1.000000
cluster centre 6.500000 3.000000
*****  

pattern      7
belongs to the cluster      3
distance      1.118034
cluster centre 6.333333 3.333333
*****  

pattern      8
belongs to the cluster      3
distance      9.428090E-01
cluster centre 6.500000 3.500000
*****  

pattern      9
belongs to the cluster      1
distance      1.118034
cluster centre 2.333333 3.333333
*****
```

FOR THRESHOLD DISTANCE OF 2

```

pattern      10
belongs to the cluster    1
distance      9.428092E-01
cluster centre  2.500000  3.500000
*****
pattern      11
belongs to the cluster    2
distance      1.118034
cluster centre  2.333333  6.333333
*****
pattern      12
belongs to the cluster    2
distance      9.428090E-01
cluster centre  2.500000  6.500000
*****
```

```

pattern of      1
assigned to cluster 1
the new cluster centres are
cluster      1      2.000000  3.000000
cluster      1      1
```

```

pattern      2
belongs to the cluster    1
distance      1.000000
cluster centre  2.500000  3.000000
*****
pattern      3
belongs to the cluster    1
distance      3.041381
cluster centre  2.333333  4.000000
*****
pattern      4
belongs to the cluster    1
distance      2.108185
cluster centre  2.500000  4.500000
*****
pattern      5
belongs to the cluster    2
distance      3.807887
cluster centre  6.000000  3.000000
*****
```

FOR THRESHOLD DISTANCE OF 3.5

```

pattern      6
belongs to the cluster      2
distance      1.000000
cluster centre  6.500000  3.000000
*****
pattern      7
belongs to the cluster      2
distance      1.118034
cluster centre  6.333333  3.333333
*****
pattern      8
belongs to the cluster      2
distance      9.428090E-01
cluster centre  6.500000  3.500000
*****
pattern      9
belongs to the cluster      1
distance      7.071068E-01
cluster centre  2.400000  4.400000
*****
pattern      10
belongs to the cluster      1
distance      7.211102E-01
cluster centre  2.500000  4.333333
*****
pattern      11
belongs to the cluster      1
distance      2.713137
cluster centre  2.428571  4.714286
*****
pattern      12
belongs to the cluster      1
distance      2.356060
cluster centre  2.500000  5.000000
*****
*****  

pattern of      1
assigned to cluster 1
the new cluster centres are
cluster      1      2.000000  3.000000
cluster      1      1
*****
```

FOR THRESHOLD DISTANCE OF 4.5

pattern	2	
belongs to the cluster		1
distance	1.000000	
cluster centre	2.500000	3.000000

pattern	3	
belongs to the cluster		1
distance	3.041381	
cluster centre	2.333333	4.000000

pattern	4	
belongs to the cluster		1
distance	2.108185	
cluster centre	2.500000	4.500000

pattern	5	
belongs to the cluster		1
distance	3.807887	
cluster centre	3.200000	4.200000

pattern	6	
belongs to the cluster		1
distance	3.984972	
cluster centre	3.833333	4.000000

pattern	7	
belongs to the cluster		1
distance	2.166667	
cluster centre	4.142857	4.000000

pattern	8	
belongs to the cluster		1
distance	2.857143	
cluster centre	4.500000	4.000000

pattern	9	
belongs to the cluster		1
distance	2.500000	
cluster centre	4.222222	4.000000

pattern	10	
belongs to the cluster		1
distance	1.222222	
cluster centre	4.100000	4.000000

pattern	11	
belongs to the cluster		1
distance	3.661967	
cluster centre	3.909091	4.272727

pattern	12	
belongs to the cluster		1
distance	2.874798	
cluster centre	3.833333	4.500000

5.1.3 Classical ART Networks

ART networks were developed by Carpenter and Grossberg (1987, 1991).

One form ART1, is designed for clustering binary vectors and the other, ART2 accepts analog or continuous valued vectors. These nets cluster inputs by unsupervised learning. One can present input patterns in any order. Each time when a pattern is presented, an appropriate cluster unit is chosen and the cluster's weights are adjusted to let the cluster unit to learn the pattern. The weights on the cluster unit may be considered to be an exemplar (or code vector) for the patterns placed on the cluster.

When the net is trained, one can present training pattern several times. A pattern may be placed on one cluster unit for the first time and then on a different cluster when it is presented later due to changes in the weights for the first cluster if it has learned other patterns in the mean time. We find in ART architecture, a pattern oscillating among different cluster units at different stages of training, indicating an unstable net.

Stability of the network means that a pattern should not oscillate among different cluster units at different stages of training. Some nets achieve stability by gradually reducing the learning rate as the same set of training set presented many times.

Plasticity While training patterns are presented many times, this does not allow the net to learn readily a new pattern that is presented for the first time after a number of training epochs have already taken place.

Plasticity is the ability of the net to respond to learn new pattern equally well at any stage of learning.

Usually, adaptive resonance theory nets are designed to be both stable and plastic. This has been described by Stephen Grossberg as *Stability-Plasticity Dilemma*. The dilemma poses a series of questions, some of which are as follows:

1. How can a learning system remain adaptive (plastic) in response to significant input yet stable in response to irrelevant input?
2. When does the system know to switch between its plastic and stable modes?
3. What is the method by which the system can retain previously learned information while learning new things?

ART seeks to provide answers for these questions. It is an extension of competitive learning scheme. Nodes compete with one another based on certain criteria and the winner is said to classify the input pattern in the competitive system.

In order to solve *Stability-plasticity dilemma*, it is necessary to add a feedback mechanism between the competitive layer and the input layer of the network. This feedback mechanism facilitated the learning of new information without destroying old information and hence, automatic switching between stable and plastic modes.

This approach results in two neural networks suitable particularly for pattern classification problems in realistic environment. Also, attention has been paid to structuring ART nets so that neural process can control the rather intricate operation of these sets. This requires a number of neurons, in addition to the input units, cluster units, and units for the comparison of the input signal with the cluster unit's weights.

ART1—This is a binary version of ART. It can cluster binary input vectors.

ART2—This is an analogous version of ART. It can cluster real value input vectors.

ART2A—This network is an ART extension that incorporates a chemical transmitter to control search process in a hierarchical ART structure.

ARTMAP—This is a supervised version of ART that can learn arbitrary mapping of binary patterns.

Fuzzy ART—It is a synthesis of ART and fuzzy logic.

Fuzzy ARTMAP—This is a supervised fuzzy art.

Distributed ART and ARTMAP (dart and dartmap)—These models learn distributed code representation in the F_2 layer. In the case of winner take all F_2 layers, they are equivalent to fuzzy ART and ARTMAP. Besides the above, there are many ART adaptations such as ARTMAP1C, GAUSSIAN

ARTMAP and Hierarchical ART models, ARBO ART, CASCADE Fuzzy ART,

HART-J, HART-S, SMART, LAPART, MART, PROBART, RRMAP, TD-

ART are some of the architectures of hierarchical models.

5.1.4 Simplified ART Architecture

The ART network is an unsupervised vector classifier that accepts input vectors which are classified according to the stored patterns they mostly resemble. It also provides for a mechanism allowing adaptive expansion of the output layer of neurons until an adequate size is reached based on the number of classes, inherent in the observation. The ART network can adaptively create a new neuron corresponding to an input pattern if it is determined to be sufficiently different from existing clusters. This determination, called a *vigilance test*, is incorporated into the adaptive backward network. Thus, the ART architecture allows the user to control the degree of similarity of patterns placed in the same cluster . Figure 5.2 shows the simplified configuration of the ART architecture.

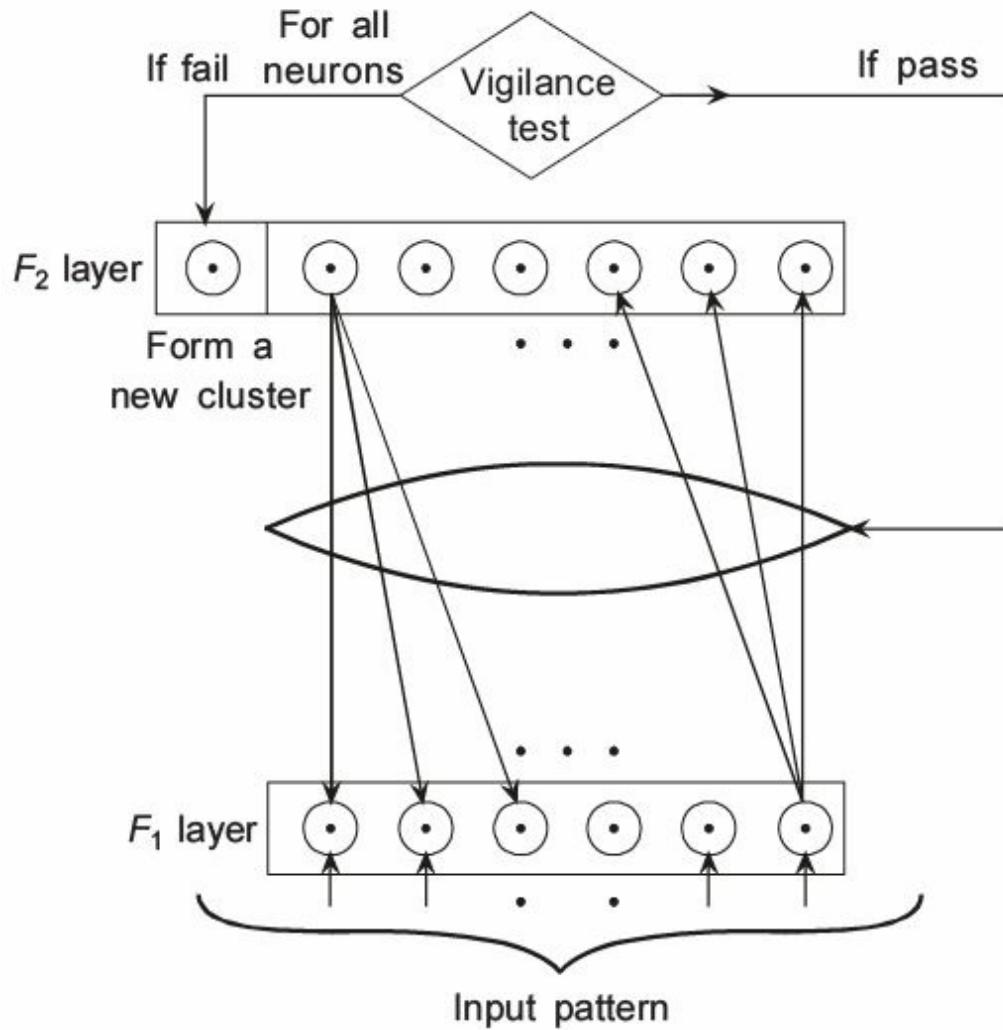


Fig. 5.2 Simplified ART architecture.

The basic architecture of ART involves three groups of neurons, an input processing field (*F₁* layer), the cluster units (*F₂* layer), and a mechanism to control the degree of similarity of patterns placed on the same cluster (a reset mechanism). To control the similarity of patterns placed on the same cluster, there are two sets of connections (each with its own weights) between each unit of input of *F₁* layer and the cluster unit of *F₂* layer, and this is known as *bottom-up weights*. Similarly, *F₂* layer is connected to *F₁* layer by *top-down weights*.

The *F₂* layer is a competitive layer. The cluster unit with the large net input becomes the candidate to learn the input pattern setting all other *F₂* units to

zero. The reset unit makes the decision whether or not the cluster unit is allowed to learn the input pattern depending on how similar its top-down weight vector is to the input vector and to this decision. If the cluster unit is not allowed to learn that it is inhibited, a new cluster unit is selected as the candidate.

Basically, there are two learning methods *fast learning* and *slow learning*.

In fast learning, weight update during resonance occurs rapidly whereas in slow learning, weight changes occur slowly relative to the duration of a learning trial. Fast learning is used in ART1 whereas slow learning is appropriate in ART2.

5.2 ART1

As mentioned in the introduction to this chapter, ART1 network requires binary input vector, that is, they must have components of the set $\{0, 1\}$. This restriction may appear to limit the utility of the network but there are many problems having data that can be cast into binary format. Later on, we will see in one example how real input can be handled in ART1.

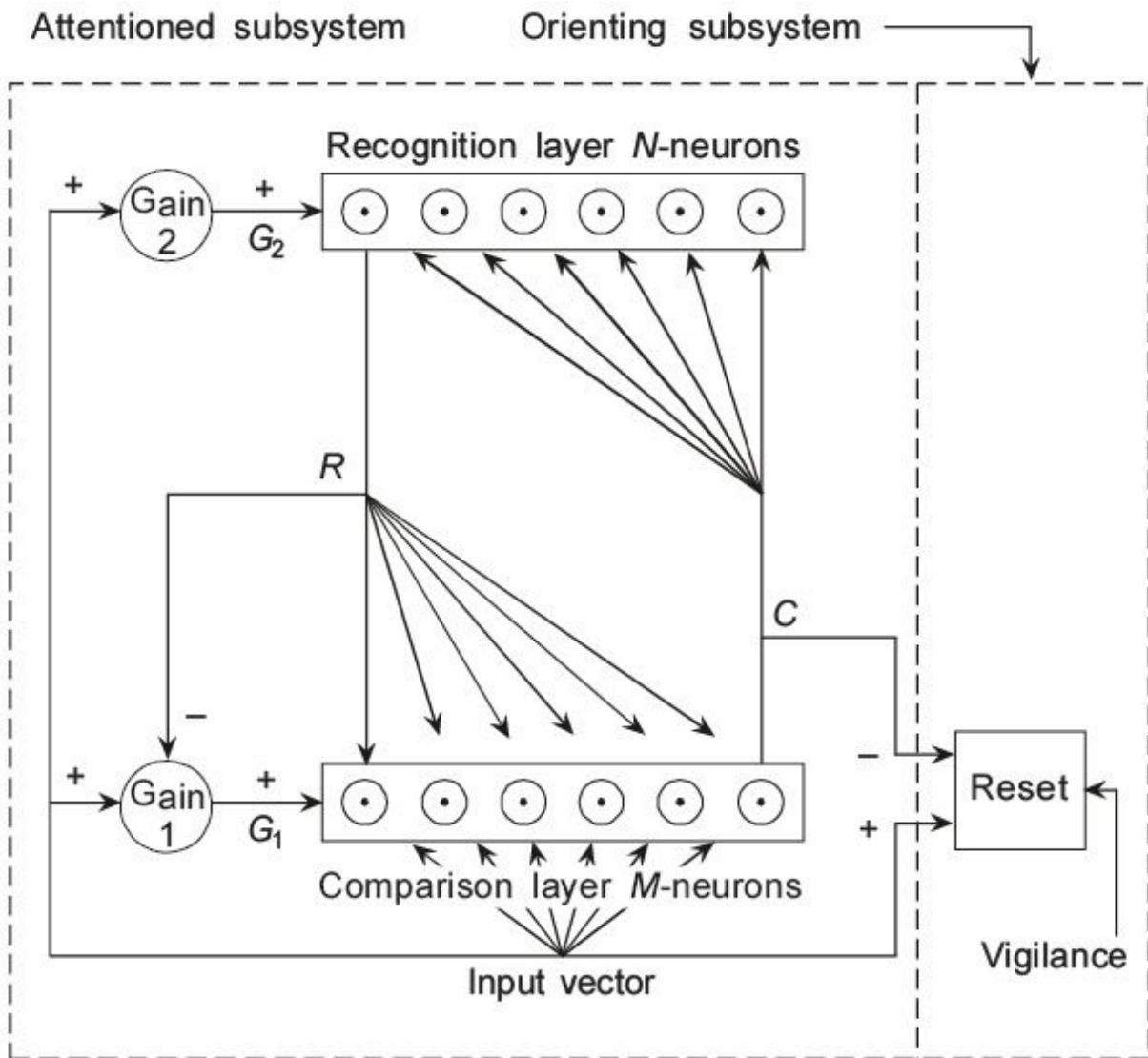
5.2.1 Architecture of ART1

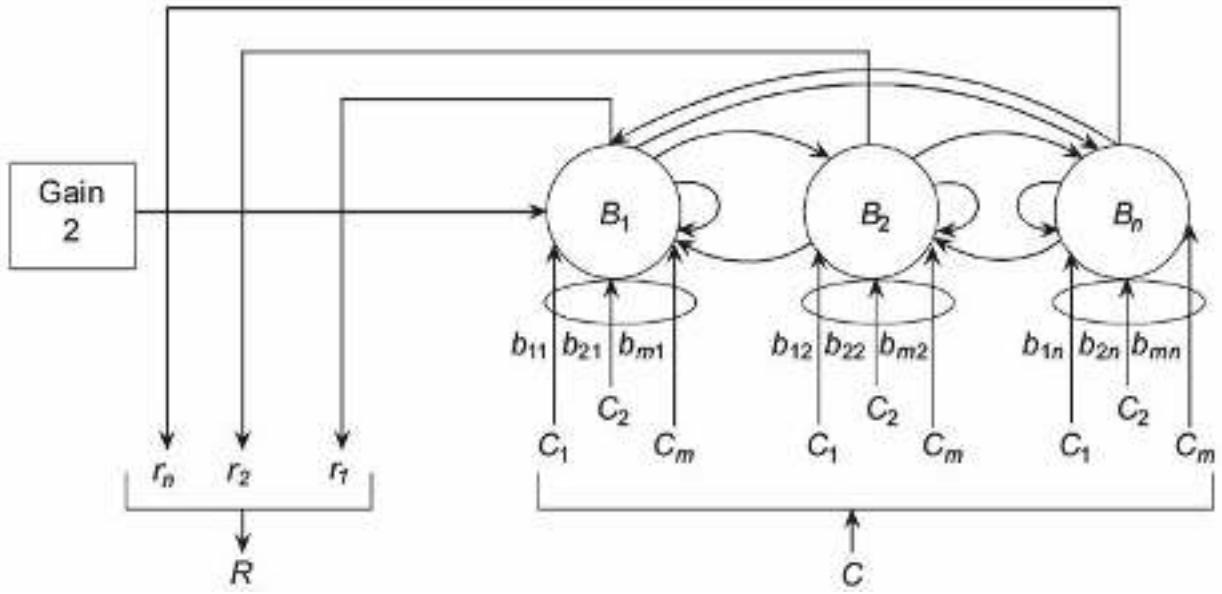
The neural network for ART1 model consists of the following:

- (a) A layer of neuron called F_1 layer (input layer or comparison layer), (b) A node for each layer as a gain control unit,
- (c) A layer of neurons called F_2 layer (output layer or recognition layer), (d) Bottom-up connection from F_1 to F_2 layer, (e) Top-down connection from F_2 to F_1 layer,
- (f) Inhibitory connection (negative weights) from F_2 layer to gain control,
- (g) Excitatory connection (positive weights) from gain control to a layer, (h) Inhibitory connection from F_1 layer to reset node, and (i) Excitatory connection from reset node to F_2 layer.

The ART1 architecture shown in Fig. 5.3 consists of two layers of neurons called *comparison layer* and the *recognition layer*. Usually, the classification decision is indicated by a single neuron in the recognition layer that fires. The neurons in the comparison layer respond to input features in the pattern. The synaptic connections (weights) between these two layers are modifiable in both the directions. According to learning rules, the recognition layer neurons have inhibitory connections that allow for competition. These two layers constitute *attentioned system*.

The network architecture also consists of three additional modules labelled *Gain1*, *Gain2*, and *Reset* as shown in Fig. 5.3. In the attentioned subsystem, if the match of input pattern with any of the prototype stored occurs, resonance is established. The orienting subsystem is responsible for sending mismatch between bottom-up and top-down patterns on the recognition layer. The recognition layer response to an input vector is compared to the original input





vector through a mechanism called *vigilance*. When vigilance falls below a threshold, a new category must be created and the input vector must be stored into that category. The recognition layer follows the *winner take all* paradigm. The recognition layer is shown in Fig. 5.4.

Fig. 5.3 ART1 network.

Fig. 5.4 Recognition layer.

$$\sum_{i=1}^N b_{ij} C_i$$

$$f(\text{net}_j) = \begin{cases} 1 & \text{for } \text{net}_j > \text{net}_i, \text{ for all } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

$$P_i = \sum_{j=1}^N t_{ji} r_j \quad \text{for } i = 1, \dots, M$$

5.2.2 Special Features of ART1 Models

One special feature of an ART1 model is that a two-third rule is necessary to determine the activity of the neuron in the F_1 layer. There are three input sources to each neuron in the F_1 layer. They are—the external input, the output of gain control, and the output of F_2 layer neurons. The gain control unit and 2/3 rule together ensure proper response from the input layer neurons. A second feature is that the vigilance parameter is used to determine the activity of the reset unit, which is activated whenever there is no match found among existing patterns during classification.

Considering Fig. 5.4

$$\text{net } j =$$

$$(5.5)$$

$$rj =$$

where C_i is the output of the i th comparison layer neuron, f is a step function and thus, rj results in a binary value. M is the number of neurons in the comparison layer. Figure 5.5 shows the comparison layer.

As shown in Fig. 5.5, each neuron i in the comparison layer receives the following three inputs:

- (a) A component or the input pattern X , X_i
- (b) The gain signal G_1 is a scalar (binary value), thus, the same value is input to each neuron
- (c) A feedback signal from the recognition layer is a weighted sum of the recognition layer outputs. Thus,

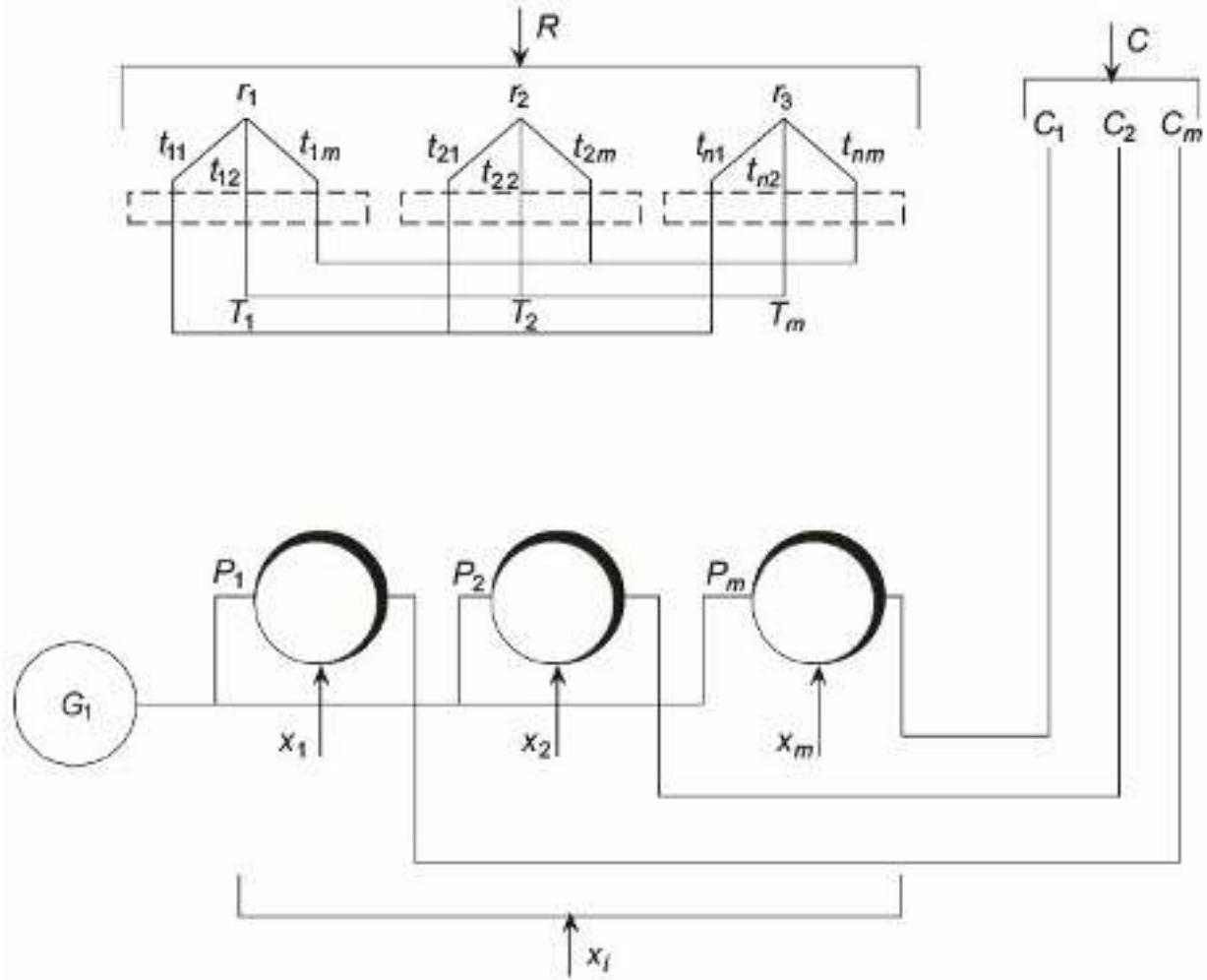
$$(5.6)$$

where rj is the output of the j th recognition layer neuron and N is the number of neurons in the recognition layer, T_j is the weight vector associated with the recognition layer neuron j , vector \mathbf{C} represents the output of

comparison layer, gain G_1 is one when the \mathbf{R} vector is zero and the logical OR of the components of the input vector X is one as seen from Eq. (5.3) as

$$G_1 = (\bar{r}_1 | \bar{r}_2 | \dots | \bar{r}_n) (X_1 | X_2 | X_3 | \dots | X_M)$$

$$G_2 = (X_1 | X_2 | \dots | X_m)$$



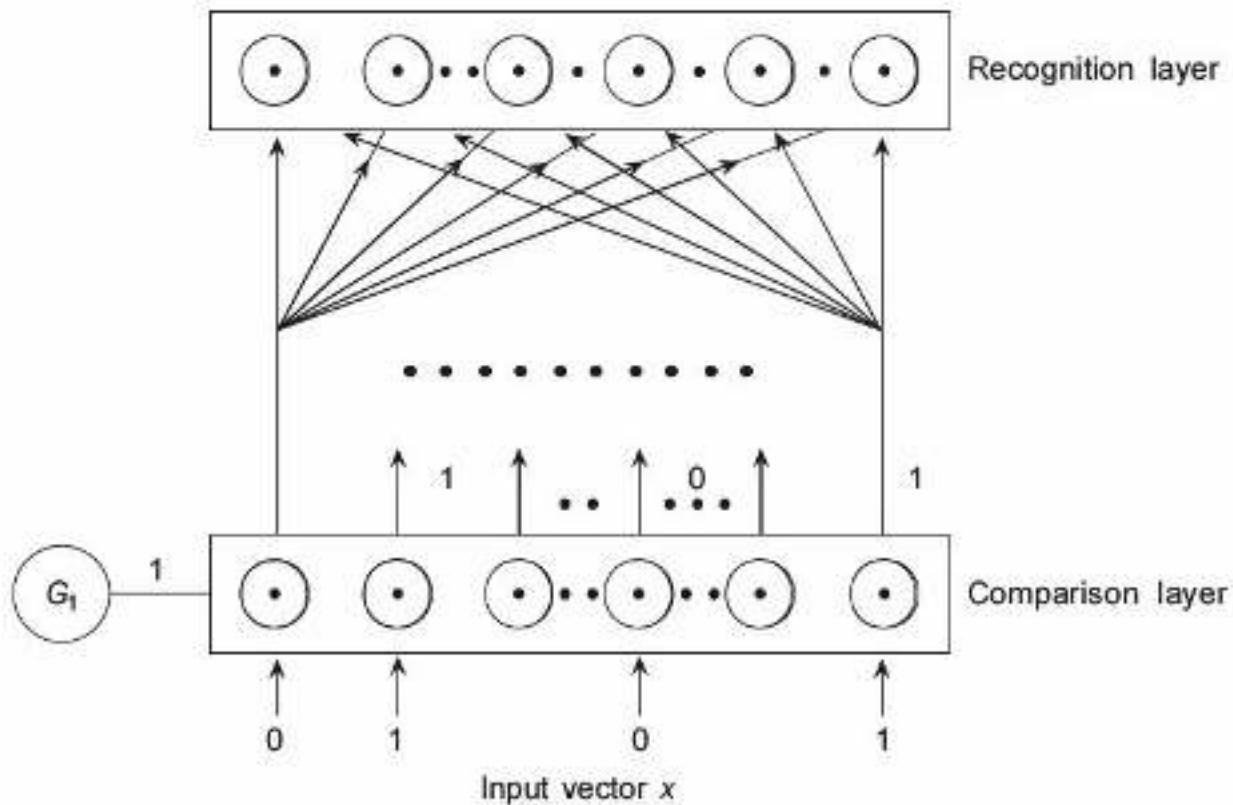
(5.7a)

Gain G_2 is one when the logical OR of the components of the input vector X is one as seen from Eq. (5.4) as.

(5.7b)

The steps of ART1 operations are given in Figs. 5.6–5.9.

Fig. 5.5 ART1 comparison layer.



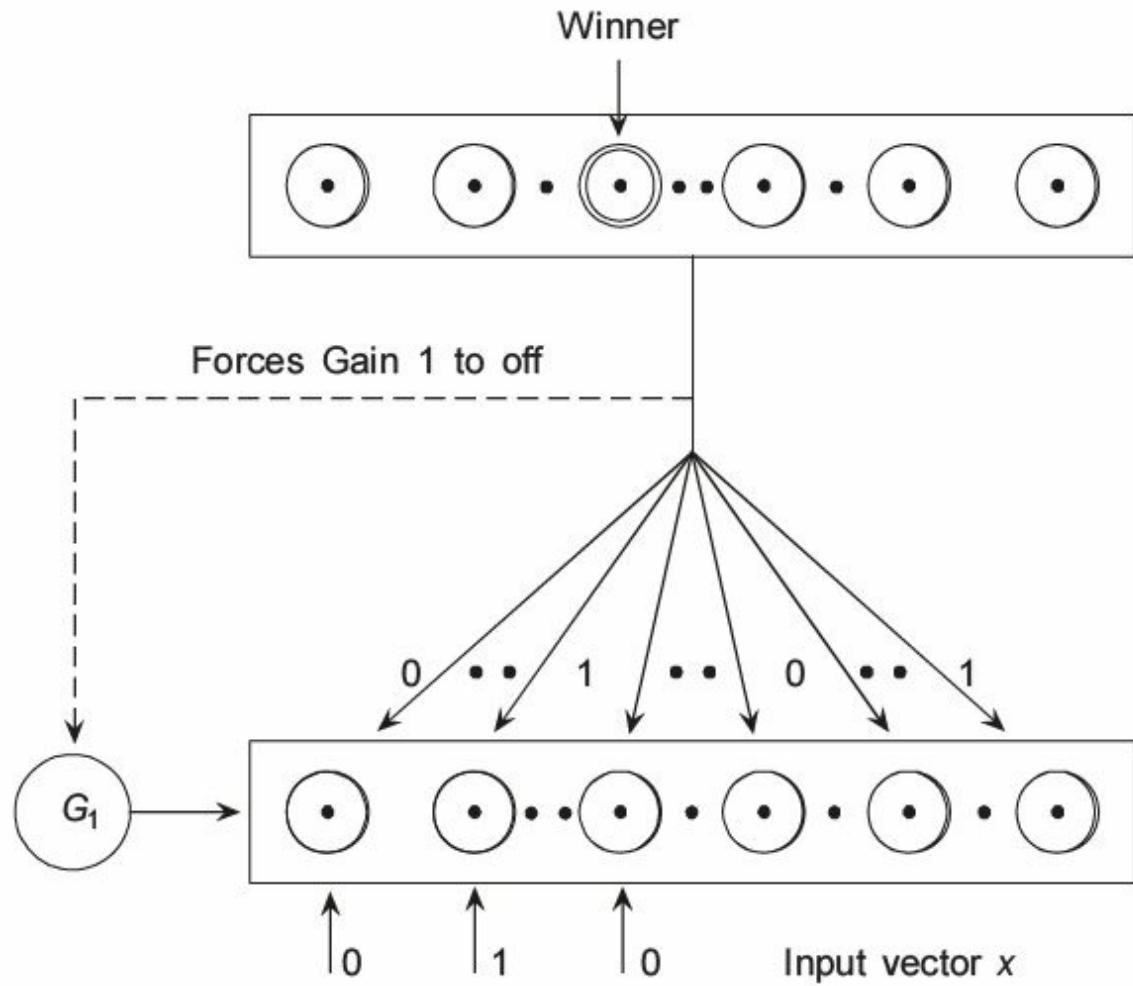
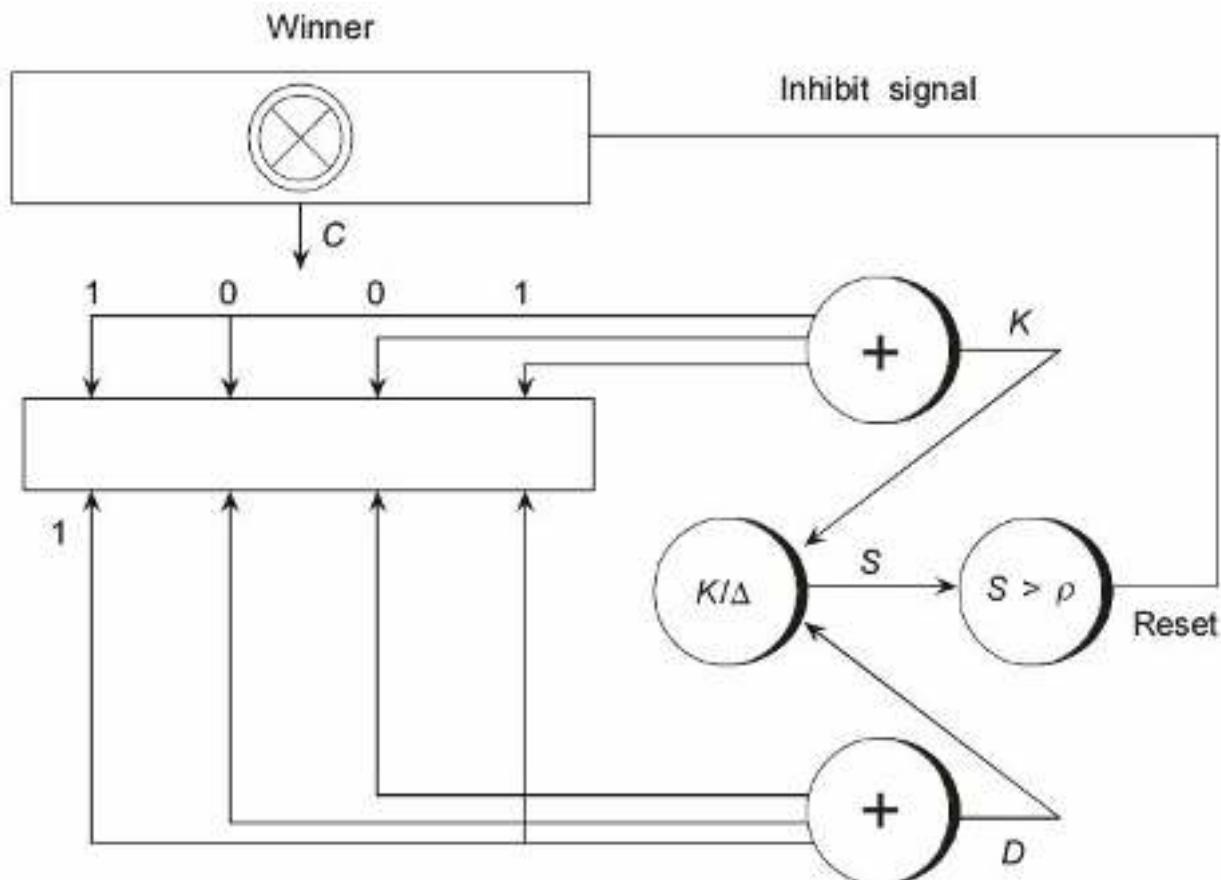


Fig. 5.6 Step 1. $G_1 = 1$ The input vector is passed through the comparison layer to the recognition layer.

Fig. 5.7 Step 2. The best neuron of the recognition layer has been selected as winner, the winner sends its signal through its top-down weights.



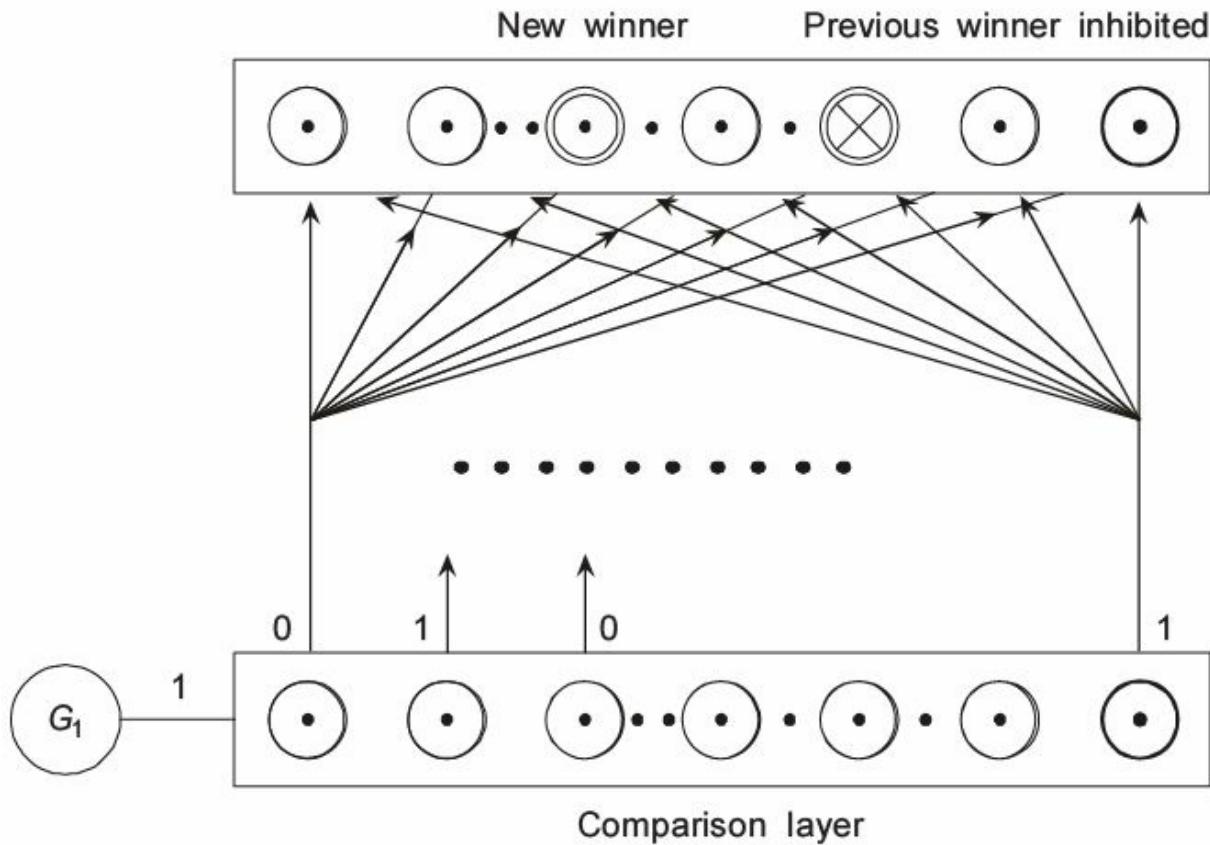


Fig. 5.8 Step 3. The input vector X and P vector in recognition layer compared. Vigilance failed.

Winning neuron is inhibited.

Fig. 5.9 Step 4. Previous winning neuron is disabled. New winner is selected.

The reader may refer for the mathematics of the dynamics of the system to the textbook by Freeman and Skapura (Freeman and Skapura, 1991).

$$A \geq 0$$

$$C \geq 0$$

$$D \geq 0$$

$$\max(D, 1) < B < D + 1$$

$$L > 1, \text{ typically } L = 2$$

$$0 < \rho \leq 1$$

$$td_{ij}^0 > \frac{B-1}{D}$$

$$bu_{ij}^0 < \frac{L}{L-1+M}$$

$$x_i(0) = \frac{-B}{1+C}$$

$$|I| = \sum_{i=1}^M I_i$$

5.2.3 ART1 Algorithm

To begin with, we must determine the size of the F_1 and F_2 layers as No. of units in $F_1 = M$

$$\text{No. of units in } F_2 = N \quad (5.8)$$

Other parameters must be chosen according to the following constraints.

$$(5.9)$$

The parameter B must be chosen to satisfy the above constraint to implement 2/3 rule successfully to distinguish between top-down and bottom-up patterns.

Initialization. Top-down weights must be initialized as

$$[td] M \times N = \text{top-down weight matrix} \quad (5.10) \quad (5.11)$$

Bottom-up weights must be initialized as

$$[bu] N \times M = \text{bottom-up weights} \quad (5.12) \quad 0 <$$

(5.13)

The activities on F_2 are initialized to zero but according to our chosen model, F_1 activities are initialized to

(5.14)

All input patterns must be binary $I_i \in \{0, 1\}$. The norm of the vector is equal to the sum of the components.

(5.15)

Step 1: Apply an input vector I to F_1 and F_1 activities are calculated as

$$X_i = \frac{I_i}{1 + A(I_i + B) + C} \quad (5.16)$$

Step 2: Calculate the output vector for F_1

$$S_i = h(X_i) = \begin{cases} 1 & \text{if } X_i > 0 \\ 0 & \text{if } X_i \leq 0 \end{cases} \quad (5.17)$$

Step 3: Propagate S forwards to F_2 and calculate the activities according to

$$\{T\}_{N \times 1} = [bul]_{N \times N} \{S\}_{N \times 1} \quad (5.18)$$

Step 4 : Only the winning F_2 node has a nonzero output.

$$u_j = \begin{cases} 1 & T_j = \max \{T_k\} \forall k \\ 0 & \text{otherwise} \end{cases}$$
$$\{u\}_{N \times 1} \quad (5.19)$$

We shall assume that winning node is J .

Step 5: Propagate the output from F_2 back to F_1 . Calculate the net inputs from F_2 to F_1 as

$$\{V\}_{N \times 1} = [td]_{N \times N} \{u\}_{N \times 1} \quad (5.20)$$

Step 6: Calculate new activities according to

$$X_i = \frac{I_i + DV_i - B}{1 + A(I_i + DV_i) + C} \quad (5.21)$$

Step 7: Determine new output values $\{S\}$ as in step 2.

Step 8: Determine the degree of match between input pattern and the top-down template as

$|I|$ will be equal to number of non zero components of the vector.

Algorithm 5.1 illustrates the steps to be followed.

Algorithm 5.1 (Art1 Algorithm)

$$\frac{|S|}{|I|} = \frac{\sum_{i=1}^M S_i}{\sum_{i=1}^N I_i} \quad (5.22)$$

Step 9: If the above value is $< \rho$ mark J as inactive, zero the outputs of F_2 and return to step 1 using the original pattern and if the above value is greater than or equal to ρ then continue.

Step 10: Update bottom-up weights on J only as

$$[bu]_{J,i} = \begin{cases} \frac{L}{L - 1 + |S|} & \text{if } i \text{ is active} \\ 0 & \text{if } i \text{ is inactive} \end{cases} \quad (5.23)$$

Step 11: Update the top-down weights coming from J only to all F_1 units.

$$[td]_{i,J} = \begin{cases} 1 & \text{if } i \text{ is active} \\ 0 & \text{if } i \text{ is inactive} \end{cases} \quad (5.24)$$

Step 12: Remove the input pattern. Restore all inactive F_2 units. Return to step 1 with a new input pattern.

END ART1.

$$[I] = \text{Input} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

5.2.4 Illustration

Example 5.2

To see the algorithm, let us perform a step by step calculation for a small example problem.

We shall choose the dimension of F_1 and F_2 as $M = 5, N = 6$ respectively.

Choose the values for the following parameters as

$$A = 1; B = 1.5; C = 5; D = 0.9; \rho = 0.9$$

Let us take the first input vector as

$$\{I_1\} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\frac{0.2 + (B - 1)}{D}$$

$$\frac{0.2 + 0.5}{0.9}$$

$$\begin{bmatrix} 0.756 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0.756 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0.756 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0.756 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0.756 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \end{bmatrix}_{5 \times 6}$$

$$\left(\frac{L}{(L-1+M)} \right) - 0.1$$

$$\left(\frac{5}{9} \right) - 0.1$$

$$\begin{bmatrix} 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \end{bmatrix}_{6 \times 5}$$

Let us initialize top-down weights by adding positive value of 0.2 to ($B -$

1)/ D giving

=

= 0.756

[td] =

Since $M = 5$ and $L = 5$, weights on F_2 units are all initialized to slightly less than the given value (say 0.1) which is obtained as

=

= 0.456

[bu] =

We can now begin actual processing. We shall start with simple input vector as

$$\langle I 1 \rangle T = \langle 0 0 0 1 0 \rangle T$$

Step 1: After the input vector is applied, the F_1 activities become (see step 1 of the algorithm)

$$\langle X 1 \rangle T = \langle 0 0 0 0.118 0 \rangle T$$

Step 2: The output vector S is written as

$$\begin{Bmatrix} 0.455 \\ 0.455 \\ 0.455 \\ 0.455 \\ 0.455 \\ 0.455 \end{Bmatrix}$$

$$\{V\}_{5 \times 1} = [td]_{5 \times 6} \{u\}_{6 \times 1} = \begin{Bmatrix} 0.756 \\ 0.756 \\ 0.756 \\ 0.756 \\ 0.756 \\ 0.756 \end{Bmatrix}_{6 \times 1}$$

$$\frac{|S|}{|I|} = 1 > \rho = 0.9$$

$$\langle S \rangle T = \langle 0 0 0 1 0 \rangle T$$

Step 3: Propagating this output vector to F_2 the net inputs to all F_2 units will be identical.

$$\{T\}_{6 \times 1} = [bu]_{6 \times 5} \{S\}_{5 \times 1}$$

=

Step 4: Calculate $\{u\}$ as

$$\langle u \rangle T = \langle 1 0 0 0 0 0 \rangle T$$

Since all unit activities are equal, simply take the first unit as winner Step 5.
Calculate $\{V\}$ as

Step 6: Calculate new activity values on F_1 as

$$\langle X \rangle T = \{-0.123 -0.123 -0.123 0.023 -0.123\} T$$

Step 7: Only unit 4 has a positive activity and hence new outputs are

$$\langle S \rangle T = \{0 0 0 1 0\} T$$

Step 8: Calculate

Step 9: There is no reset and resonance reached.

Step 10: Update bottom-up weight matrix as

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \end{bmatrix}_{6 \times 5}$$

$$\begin{bmatrix} 0 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\ 1 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \end{bmatrix}_{5 \times 6}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.833 & 0 & 0.833 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \end{bmatrix}_{6 \times 5}$$

$$[bu] =$$

Step 11: Update top-down weights as

$$[td] =$$

That completes the cycle of the first input pattern. Now, let us apply second pattern that is orthogonal to I_1 as

$$\langle I_2 \rangle T = \langle 0 0 1 0 1 \rangle T$$

$$\langle T \rangle T = \langle 0 0.911 0.911 0.911 0.911 0.911 \rangle T$$

Unit 1 definitely loses. We select unit 2 as winner.

$$\langle u \rangle T = \langle 0 1 0 0 0 0 \rangle T$$

$$\langle V \rangle T = [td]\{ u \} = \langle 0.756 0.756 0.756 0.756 0.756 \rangle T$$

$$\langle X \rangle T = \langle -0.123 -0.123 0.0234 -0.123 -0.0234 \rangle T$$

The resulting output matches the input vector $\langle 0 0 1 0 1 \rangle T$ and hence there is no reset. Now, the bottom-up matrix is given by

$$[bu] =$$

Now, the top-down matrix is given by

$$\begin{bmatrix} 0 & 0 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0 & 0 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0 & 1 & 0.756 & 0.756 & 0.756 & 0.756 \\ 1 & 0 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0 & 1 & 0.756 & 0.756 & 0.756 & 0.756 \end{bmatrix}_{5 \times 6}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \\ 0.456 & 0.456 & 0.456 & 0.456 & 0.456 \end{bmatrix}_{6 \times 5}$$

$$\begin{bmatrix} 0 & 0 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0 & 0 & 0.756 & 0.756 & 0.756 & 0.756 \\ 0 & 0 & 0.756 & 0.756 & 0.756 & 0.756 \\ 1 & 0 & 0.756 & 0.756 & 0.756 & 0.756 \\ 1 & 1 & 0.756 & 0.756 & 0.756 & 0.756 \end{bmatrix}_{5 \times 6}$$

$$[td] =$$

Now, let us apply the third input vector as

$$\langle I 3 \rangle T = \langle 0 0 0 0 1 \rangle T$$

$$\langle u \rangle T = \langle 0 1 0 0 0 0 \rangle T$$

$$\langle V \rangle T = \langle 0 0 1 0 1 \rangle T$$

In this case, the equilibrium activities are

$$\langle X \rangle T = \langle -0.25 \ -0.25 \ -0.087 \ -0.25 \ .0.0506 \rangle T$$

with only one positive activity. The new output pattern is $\langle 0 \ 0 \ 0 \ 0 \ 1 \rangle$ which exactly matches with input pattern. So, no reset occurs.

Even though unit 2 on F_2 had previously encoded an input pattern, it gets recoded now to match the new input pattern that is a subset of the original pattern. The new weight matrices are

$$[bu] =$$

$$[td] =$$

If we return to the superset vector $\langle 0 \ 0 \ 1 \ 0 \ 1 \rangle T$, the initial forward propagation to F_2 yields activities as

$$X = \langle -0.25 \ -0.25 \ -0.25 \ 0.0506 \ -0.25 \rangle$$

The outputs are $\langle 0 \ 0 \ 0 \ 1 \ 0 \rangle$. This time, resonance has reached pattern 1 on unit 1.

A program ART1 developed in Fortran is given} in CD-ROM (attached with this book) to classify patterns for the binary input. If we use the program we get the following output.

resonance has been reached on unit 1 pattern 1

resonance has been reached on unit 2 pattern 2

resonance has been reached on unit 3 pattern 3

network not stable

resonance has been reached on unit 1 pattern 1

reset with pattern 2 on unit 2

resonance has been reached on unit 3 pattern 2

resonance has been reached on unit 3 pattern 3

network not stable

resonance has been reached on unit 1 pattern 1

reset with pattern 2 on unit 3

reset with pattern 2 on unit 2

resonance has been reached on unit 4 pattern 2

resonance has been reached on unit 2 pattern 3

network not stable

resonance has been reached on unit 1 pattern 1

resonance has been reached on unit 4 pattern 2

resonance has been reached on unit 2 pattern 3

network stable

Even if any pattern is repeated, we will be able to recognize the pattern.

ART1 is an elegant theory that addresses stability-plasticity dilemma. The network relies on resonance. It is a

self-organizing network and does the categorization by associating individual neuron of the F 2 layer with individual patterns. By employing so called 2/3

rule, it ensures stability in learning process.

5.3 ART2

5.3.1 Architecture of ART2

ART2 networks self organize stable recognition categories in response to arbitrary sequences of analog (Grey-scale, continuous-valued) input patterns, as well as binary input patterns. On the surface, it looks that the main difference between ART1 and ART2 is that ART2 accepts input vectors whose components can have any real numbers as their value. But in execution, ART2 network is considerably different from ART1 network. The capability of recognizing analog patterns represents a significant enhancement to the system. ART2 also recognizes the underlying similarity of identical patterns superimposed on constant backgrounds having different levels. Figure 5.10 shows the ART2 architecture where the comparison layer of F_1 layer is split into several sublayers. Additionally, the orienting subsystem has also accommodated real-valued data. ART2 includes the following:

- (a) Allowance for noise suppression,
- (b) Normalization, i.e. contrast to enhance the significant parts of the pattern,
- (c) Comparison of top-down and bottom-up signals needed to reset the mechanism, and
- (d) Dealing with real-valued data that may be arbitrarily close to one another.

The learning laws of ART2 are much simpler even though network is complicated.

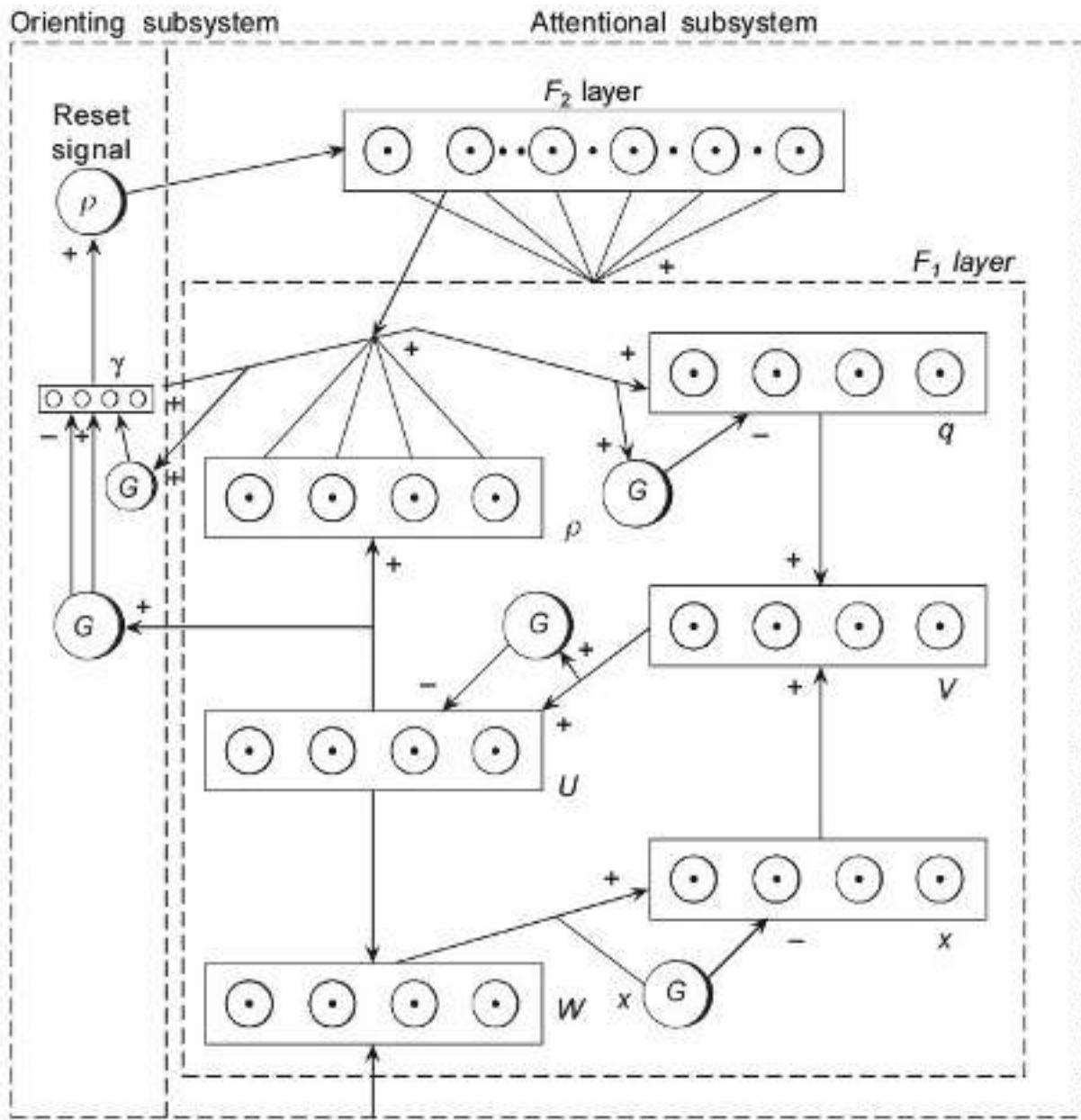


Fig. 5.10 ART2 architecture.

Carpenter and Grossberg (1987), the developers of ART2 architecture have been developing various architectures of ART2 and in this chapter, we will describe one such architecture. The reader may refer to the mathematics of ART2 architecture in the book by Freeman and Skapura (1991).

5.3.2 ART2 Algorithm

The size of the F_1 and F_2 layers is determined as Number of units in F_1
layer = M

Number of units in F_2 layer = N

$$0 \leq d \leq 1; cd/(1-d) \leq 1; 0 \leq \theta \leq 1; 0 \leq \rho \leq 1; e \ll 1$$

$$\begin{bmatrix} \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha \end{bmatrix}$$

$$a \leq \frac{0.5}{\{(1-d)\sqrt{M}\}}$$

The parameters are chosen according to the following constraints $a, b > 0$

(5.25)

Top-down weights are initialized to zero as $[td] = [0]$. The top-down weight matrix consists of

M rows and N columns.

Bottom-up weights are initialized as

$$[bu] =$$

(5.26)

where

The steps to be followed are illustrated in Algorithm 5.2.

Algorithm 5.2 (ART2 Algorithm)

Step 1: Initialize all layer and sublayer outputs to zero vectors and establish a cycle counter initialized to a value 1.

Step 2: Apply an input pattern I to the W layer of F_1 . The output of this layer is

$$w_i = I_i + au_i \quad (5.27)$$

Step 3: Propagate forward to X sublayer as

$$X_i = \frac{w_i}{e + \|w\|} \quad (5.28)$$

Where $\|w\|$ denotes the second norm of (w) given by

$$\|w\| = \sqrt{w_1^2 + w_2^2 + w_3^2 + \dots + w_n^2} \quad (5.29)$$

Step 4: Propagate forward to the v sublayer as

$$v_i = f(X_i) + b f(q_i) \quad (5.30)$$

$$\text{where, } f(X) = \begin{cases} 0 & 0 \leq X \\ X & X > 0 \end{cases} \quad (5.31)$$

It is to be noted that the second term is zero on the first pass through as q is zero at that time.

Step 5: Propagate to u sublayer as

$$u_i = v_i / (e + \|v\|) \quad (5.32)$$

Step 6: Propagate forward to p sublayer as

$$p_i = u_i + d (td_{i,j}) \quad (5.33)$$

where j th node on F_1 is the winner if the competition layer F_2 is inactive, i.e. $p_i = u_i$. Similarly, the network is still in its initial configuration $p_i = u_i$ because $td(i,j) = 0$.

Step 7: Propagate to q th sublayer as

$$q_i = p_i / (e + \|p\|) \quad (5.34)$$

Step 8: Repeat steps 2 to 7 as necessary to stabilize values on F_1 .

Step 9: Calculate the output of r th layer as

$$r_i = (u_i + cp_i) / (e + \|u\| + \|cp\|) \quad (5.35)$$

Step 10: Determine whether a reset signal condition is reached if $\rho/(c + \|r\|) > 1$.

Then send reset signal to F_2 . Mark any active F_2 node as ineligible for competition, reset the cycle counter to one and return to step 2. If there is no reset, and the cycle counter is 1, increment the cycle counter and continue with step 11. If there is no reset and the cycle counter is greater than one then skip to step 14 as resonance has been established.

Step 11: Propagate the output of p sublayer to the F_2 layer and calculate the net inputs to F_2 , as

$$T_j = \sum p_i bu(j, i) \quad (5.36)$$

Step 12: Only the winning F_2 node has normalized output as

$$g(T_j) = \begin{cases} d & T_j = \max_k \{T_k\} \\ 0 & \text{otherwise} \end{cases} \quad (5.37)$$

Any node marked as ineligible by previous reset signals does not participate in the competition.

Step 13: Repeat steps 6 to 10.

Step 14: Modify bottom-up weights on the winning F_2 unit as

$$bu(j, i) = u_i/(1 - d) \quad (5.38)$$

Step 15: Modify top-down weights coming from the winning F_2 unit as

$$td(i, j) = u_i/(1 - d) \quad (5.39)$$

Step 16: Remove the input vector. Restore all inactive F_2 units.
Return to step 1 with a new input pattern.

$$[I] = \begin{bmatrix} 0.2 & 0.7 & 0.1 & 0.5 & 0.4 \\ 0.8 & 2.8 & 0.4 & 2.0 & 1.6 \\ 0.1 & 0.3 & 1.2 & 2.0 & 4.0 \end{bmatrix}$$

$$\rho = 0.9; \theta = 0.7; a = 10; b = 10; c = 0.1; d = 0.9; tol = 0.001$$

$$\begin{bmatrix} 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \end{bmatrix}_{6 \times 5}$$

5.3.3 Illustration

Example 5.3

Let us perform step by step calculation for the following example.

Let us define a set of three input vectors as

Let us initialize the following variables.

We shall assume the dimensions of F_1 and F_2 as $M = 5, N = 6$ respectively.

Let us choose the first input vector as

$$\langle I_1 \rangle T = \langle 0.2 \ 0.7 \ 0.1 \ 0.5 \ 0.4 \rangle T$$

The top-down weights are initialized as zero; $[td] = [0]$.

The bottom-up weights are initialized according to Eq. (5.26) as

$$[bu] =$$

Using $\langle I_1 \rangle T = \langle 0.2 \ 0.7 \ 0.1 \ 0.5 \ 0.4 \rangle T$ as the input vector, before propagation to F_2 we have

$$\langle p \rangle T = \langle 0 \ 1 \ 0 \ 0 \ 0 \rangle T.$$

Propagating the vector forward to F_2 yields a vector of activities across the F_2 units of

$$\langle T \rangle T = \langle 2.236 \ 2.236 \ 2.236 \ 2.236 \ 2.236 \rangle T$$

Because all the activities are the same, the first unit becomes the winner and the activity vector becomes

$$\langle T \rangle T = \langle 2.236 \ 0 \ 0 \ 0 \ 0 \rangle T$$

and the output of the F_2 layer is a vector given by

$$\begin{bmatrix} 0 & 9.998 & 0 & 0 & 0 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\ 2.236 & 2.236 & 2.236 & 2.236 & 2.236 \end{bmatrix}_{6 \times 5}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 9.98 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{5 \times 6}$$

$\langle 0.9 \ 0 \ 0 \ 0 \ 0 \rangle$

We must propagate this output vector back to F_1 and cycle through the layers again. Since the top-down weights are all initialized to zero, there is no change on the sublayers of F_1 . Resonance is established on unit by pattern 1.

The bottom-up weights will be

$[bu] =$

and the top-down matrix will be

$[td] =$

Notice the expected similarity between the first row of bottom-up matrix and the first column of the top-down matrix.

By continuing this example further, we get the output as.

resonance established on Unit 1 with pattern 1

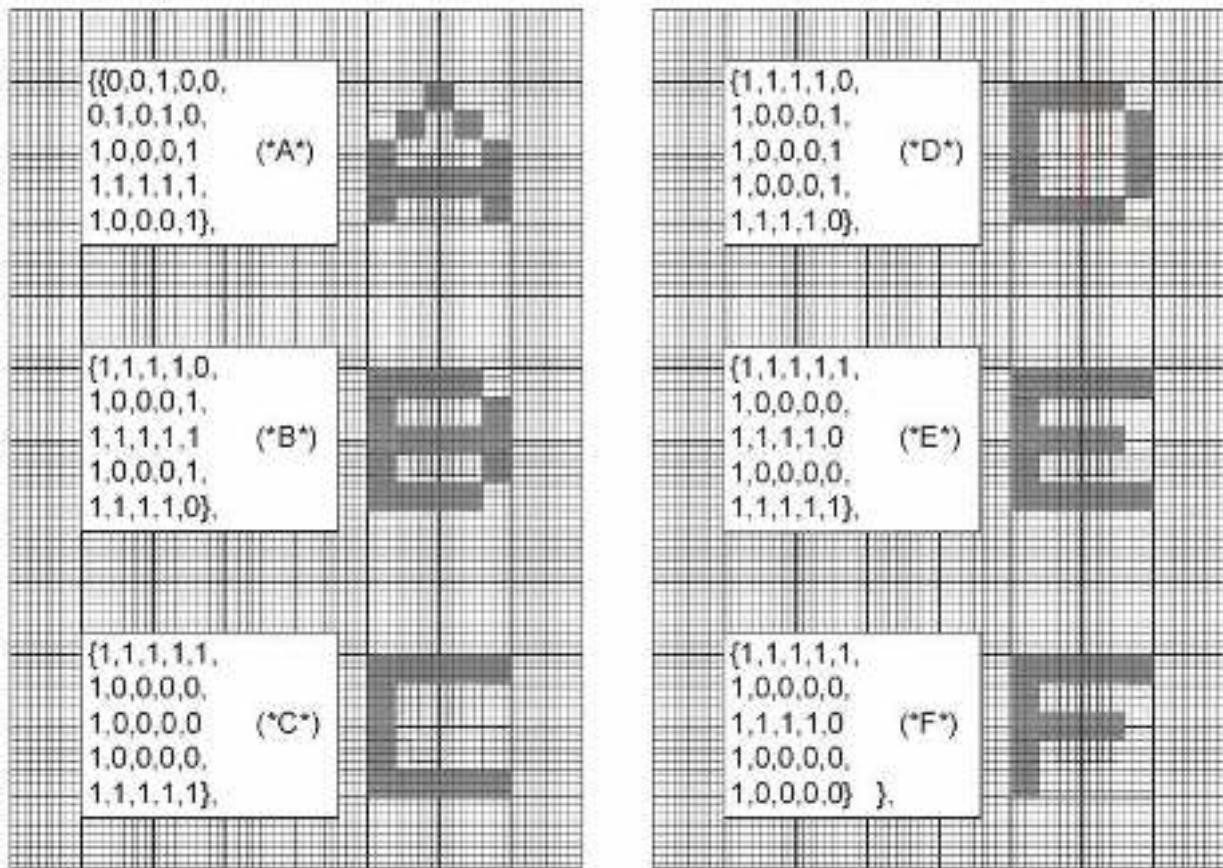
resonance established on Unit 1 with pattern 2

resonance established on Unit 2 with pattern 3

A program ART2 in Fortran given in CD-ROM (attached with this book) is written to recognize analog patterns.

From the above example, we can see that ART2 network varies from ART1 network primarily in the implementation of F_1 layer. Rather than a single

layer structure, unit F 1 layer consists of a number of sublayers that serve to remove noise to enhance contrast, and to normalize the analog input patterns. All signals propagating through ART2 network must be modelled as floating point numbers.



5.4 APPLICATIONS

5.4.1 Character Recognition Using ART1

Let us construct the list of input vectors that correspond to the first six characters of English language. The letters are five by five pixel representation and appear in Fig. 5.11. The ART1 implementation is used to classify the patterns. The inputs used are $A = 1$, $B = 1.5$, $C = 5$, $D = 0.9$, $L = 25$, $\rho = 0.9$. ART1 program given in CD-ROM of this book is able to identify all the patterns without any difficulty. This network requires a total of 16

resets during encoding process and when $L = 3$, 22 resets are required to classify the patterns. It is clear that this parameter changes from 16 to 22.

There are many more such experiments that one can perform to gain experience with ART1 architecture. We can also construct a noisy version of these letters to see how the network responds once trained with noise-free letters.

Fig. 5.11 Binary representation of alphabets.

5.4.2 Classification of Soil (Rajasekaran et al., 2001) The potential of ART1 based pattern recognizer to recognize real data has been studied. The example classification of soils studied in Chapter 3 is taken here and we will select only four data from Table 3.6 and identify their IS

classification. The data is given in Table 5.2.

Table 5.2 Soil data

Colour

Gravel %

Sand %

Finegrain %

Liquidlimit

Plasticlimit

IS

of soil

18

82

84

58

34

type

0.2

0.111

0.682

0.5

0.508

0.529

1

0.1

0

0.329

0.869

0.711

0.735

2

0.2

0

0.529

0.670

0.576

0.676

3

0.7

0

0.353

0.845

0.677

1

4

Colour of soil

0.1—Brown

0.2—Brownish grey

0.7—Yellowish red

IS type

1—Clayey sand

2—Clay with medium compressibility

3—Clay with low compressibility

4—Silt with medium compressibility

First the real data is converted to integer and given as inputs to ART1. ART1 is able to identify the soil as 1, 2, 3, 4. The output of the program is as follows

resonance has been reached on unit 1 pattern 1

reset with pattern 2 on unit 1

resonance has been reached on unit 2 pattern 2

resonance has been reached on unit 3 pattern 3

reset with pattern 4 on unit 3

reset with pattern 4 on unit 1

resonance has been reached on unit 4 pattern 4

network not stable

reset with pattern 1 on unit 4

reset with pattern 1 on unit 3

resonance has been reached on unit 1 pattern 1

reset with pattern 2 on unit 1

resonance has been reached on unit 2 pattern 2

reset with pattern 3 on unit 4

reset with pattern 3 on unit 1

resonance has been reached on unit 3 pattern 3

reset with pattern 4 on unit 1

resonance has been reached on unit 4 pattern 4

network stable

5.4.3 Prediction of Load from Yield Patterns of Elastic-Plastic Clamped Square Plate

Example 5.4

Whang (1969) has developed finite element displacement method for elastic-plastic analysis of bilinear strain hardening orthotropic plates and shells considering elastic unloading also. Figure 5.12 shows the uniformly loaded isotropic plate with clamped edges with properties—thickness,

$t = 1$; elastic modulus, $E = 30000$; Poisson's ratio = 0.3; plastic modulus, Ep

= 300; normal yield stress = 30; shear yield stress = 17.3.

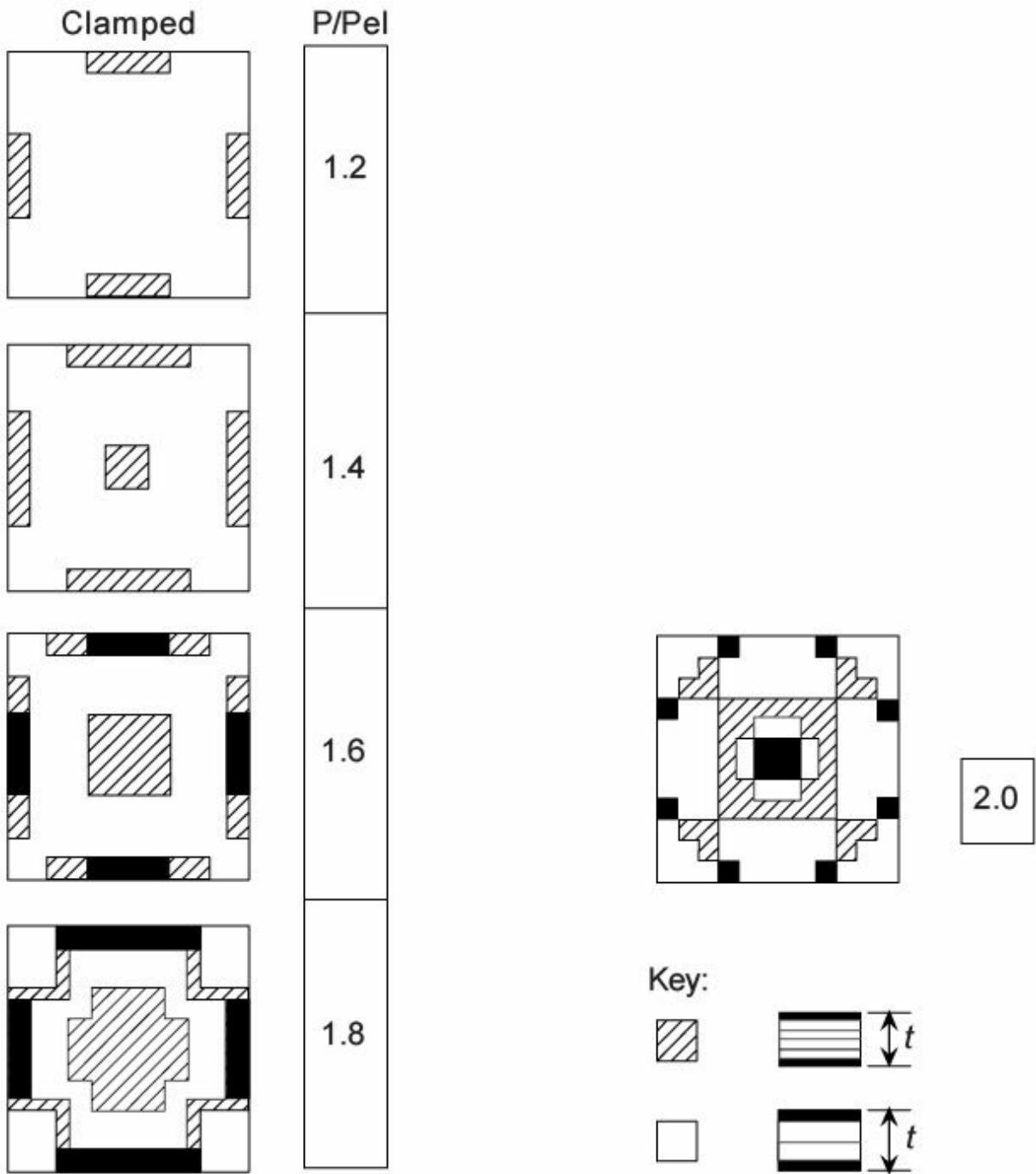
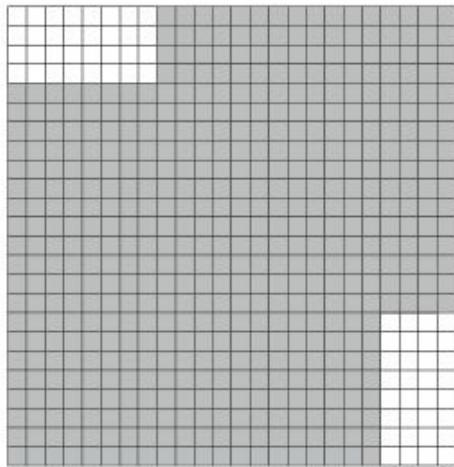


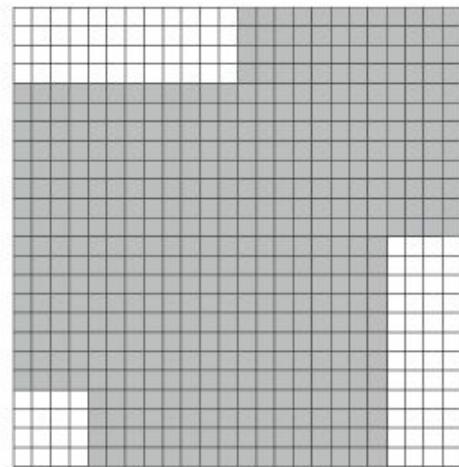
Fig. 5.12 Yield pattern of isotropic plates (clamped).

Considering the doubly symmetric nature of the pattern, only a quarter of the image is presented to ART1 for training. For each pattern, using the *feature extractor* to be discussed in Chapter 13, seven moment invariants are extracted and they are real numbers. These numbers are converted to binary

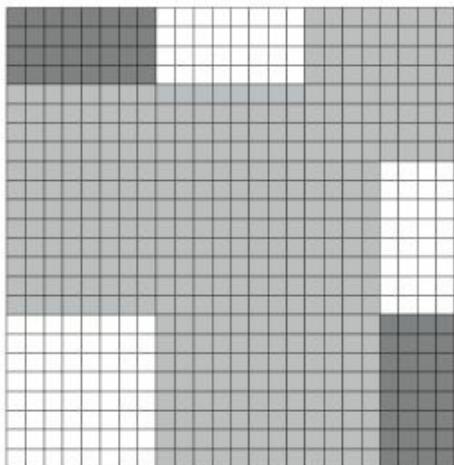
values. In the moment extraction process, various colours in the pattern can be considered by giving different values for $f(x, y)$ (see Chapter 13). Figure 5.13 shows 1/4th images of the patterns which are trained using ART1. The output for such a run as well as the moment invariants and the binary input for the six moment invariants are



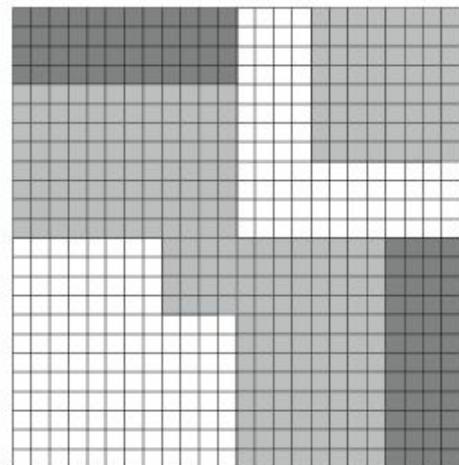
Pattern 1



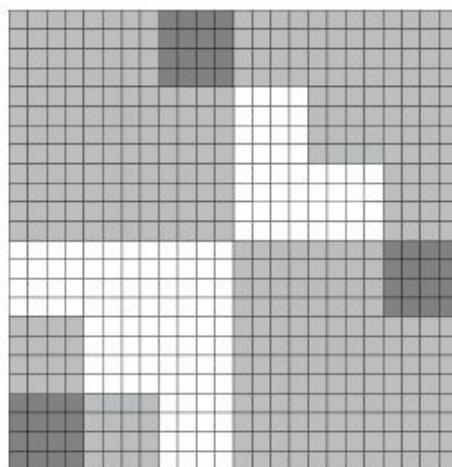
Pattern 2



Pattern 3



Pattern 4



Pattern 5

Fig. 5.13 One-fourth images of the patterns.

Output of the Example 5.4

```
*****
```

resonance has been reached on unit 1 pattern 1

reset with pattern 2 on unit 1

resonance has been reached on unit 2 pattern 2

```
*****  
SIX MOMENT INVARIANTS FOR THE 5 PATTERNS WHERE 6TH PATTERN IS THE COPY  
OF PATTERN 1 (7TH MOMENT INVARIANT IS EQUAL TO ZERO)  
*****
```

```
.2347043E+00, .9712145E+00, .3654507E-01, .4060563E-02, -.9374095E-03, -.742054E-01  
.7064392E+00, .1652542E+00, .6775361E+00, .8672835E-01, .5949809E-01, -.5931095E-01  
.6791413E+00, .9357532E-02, .7339421E+00, .2883887E-02, -.2947404E-03, -.4157953E-03  
.9984985E+00, .1446497E-01, .5226685E-01, .7694976E-02, -.8123569E-04, .6714713E-03  
.9966066E+00, .8169467E-01, .9659074E-02, .2780784E-02, -.5542211E-05, .4928858E-03  
.2347043E+00, .9712145E+00, .3654507E-01, .4060563E-02, -.9374095E-03, -.1742054E-01
```

reset with pattern 3 on unit 2

reset with pattern 3 on unit 1

resonance has been reached on unit 3 pattern 3

reset with pattern 4 on unit 1

reset with pattern 4 on unit 3

reset with pattern 4 on unit 2

resonance has been reached on unit 4 pattern 4

reset with pattern 5 on unit 4

reset with pattern 5 on unit 1

reset with pattern 5 on unit 3

reset with pattern 5 on unit 2

resonance has been reached on unit 5 pattern 5

resonance has been reached on unit 1 pattern 6

network not stable

resonance has been reached on unit 1 pattern 1

reset with pattern 2 on unit 5

reset with pattern 2 on unit 4

resonance has been reached on unit 2 pattern 2

resonance has been reached on unit 3 pattern 3

resonance has been reached on unit 4 pattern 4

reset with pattern 5 on unit 4

resonance has been reached on unit 5 pattern 5

resonance has been reached on unit 1 pattern 6

network stable

```
*****  
INPUT DATA TO ART1 PROGRAM—BINARY EQUIVALENT OF MOMENT INVARIANTS  
(6 X 12 = 72 BITS)  
*****
```

```
00011110000001111000101000001001010000000001000100000000001100000100011  
010110100110000101010010010101101011000010110001000001111001100001111001  
010101101110000000010011010111011110000000001011000000000001000000000000  
011111111000000000011101000001101011000000001111100000000000000000000000001  
01111111100100001010011100000001001100000000010110000000000000000000000001  
00011110000001111000101000001001010000000001000100000000001100000100011
```

```
*****  
PARAMETERS TO BE READ FOR THE PROGRAM  
*****
```

```
1, 1.5, 5, 0.9, 1000, .99, 50  
i.e., a = 1, b = 1.5, c = 5, d = 0.9, L = 1000, rho = 0.99, ite = 50  
*****
```

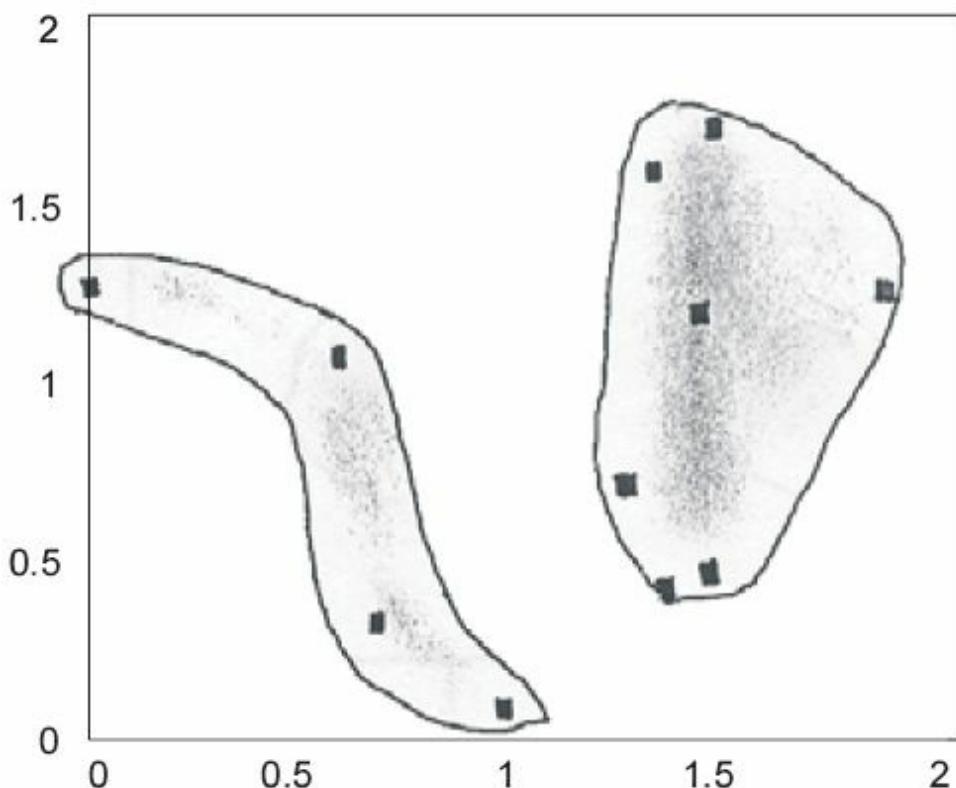
ART1 is able to classify monochrome or color images. Rajasekaran and Amalraj (2002) have applied ART1 architecture augmented with moment based feature extractor to recognize patterns colour-noisy and noise-free for two experimental problems discussed here. The first one is concerned with the recognition of satellite images and the second one is concerned with colour images.

5.4.4 Chinese Character Recognition—Some Remarks

Kim et al. (1996) proposed an online *Chinese character recognition* method using ART2 architecture. Strokes and primitive components of Chinese characters are usually warped into a cursive form and classifying them is very difficult. To deal with such cursive strokes, they considered them as a recognition unit and automatically classified them using ART2 neural network. Character recognition is achieved by traversing the Chinese character database with a sequence of recognized strokes and the positional relations between the strokes. They tested 1800 basic characters used daily in Korea and found a good recognition rate of 93.13%.

Gan and Lua (1992) have also applied ART2 architecture to the problem of character recognition of Chinese characters. ART2 was chosen because they used a real-valued feature set. The feature set contains 12 geometric features including intersection, turning points, and horizontal and vertical strokes. In

this application, ART network was not used as the final classifier, but rather served to divide 3755 Chinese characters into 7 groups in preparation of final recognition stage. A best case classification accuracy of 97.23% for the training set and 90.25% for test set was obtained.



5.5 SENSITIVENESS OF ORDERING OF DATA

The ART architecture is sensitive to the order in which the patterns are presented to the network. Kung (1993) has shown through an experiment that ART2 yields a different clustering on the same input when the patterns are presented in the reverse order. Figure 5.14(a) shows the patterns space containing patterns presented in reverse order and Fig. 5.14(b) when it is present in actual order. In both cases, the threshold value is kept constant at 1.5.

Fig. 5.14(a) Cluster of patterns (reverse order) (Kung, 1993).

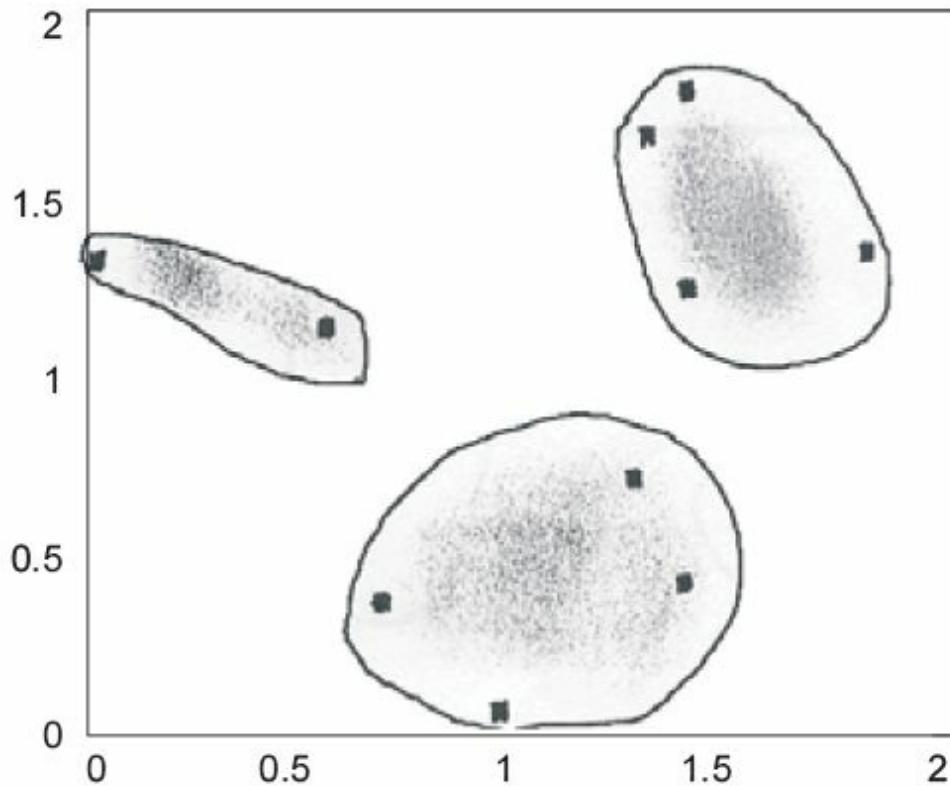


Fig. 5.14(b) Cluster of patterns (in order) (Kung, 1993).

Carpenter and Grossberg (1987b) have applied ART2 architecture to the problem of categorization of analog patterns which were drawn as a graph of functions. Fifty input patterns were classified as 34 clusters.

SUMMARY

The advantages of ART over competing pattern recognition technique are as follows:

ART exhibits stability and is not perturbed by an arbitrary barrage of inputs.

The network adapts to reflect the type of patterns not frequently observed in the environment by updating the category prototypes adequately.

The ART architecture can easily integrate with other hierarchical theories of cognition. It is sensitive to the order in which the patterns are presented.

ART models are based on unsupervised learning for adaptive clustering whereas ARTMAP architecture performs supervised learning, as we will see in later chapters. By mapping categories of one input space on to categories of another input space, both the categories are determined as two separate ART systems.

ART models belong to the class of match-based learning as opposed to error-based learning of backpropagation networks.

In match-based learning the weights are adjusted only when the external input matches one of the stored prototypes. Thus, match-based learning tends to group similar patterns whereas error-based learning tends to discriminate dissimilar patterns.

In this chapter, vector quantization, and ART1 and ART2 models have been discussed with algorithms and examples.

PROGRAMMING ASSIGNMENT

P5.1 Cluster all the 5 bit binary vectors except the all zero vector $\langle 0\ 0\ 0\ 0\ 0 \rangle$ T using ART1 algorithm. Study the effect of vigilance parameter on the resulting clusters.

P5.2 Implement the ART1 simulator. Test it using the example data presented in Example P5.1. Does simulator generate the same data values described in the example?

P5.3 Using ART1 simulator, recognize the pattern of alphabets G to L.

P5.4 Convert the real data in Illustration 5.3.3 to binary form and use ART1 simulator to identify the patterns.

P5.5 Implement the ART2 simulator. Test it using the data presented in Illustration 5.3.3. Describe the activity levels at each sublayer on $F\ 1$ at different periods during the signal propagation process.

P5.6 Run ART2 program using different values of theta and threshold value and study the results.

P5.7 Using ART2 simulator, describe what happens when all the inputs in a training pattern are scaled by a small random noise value and are presented to the network after training. Does ART2 network correctly classify the input data?

SUGGESTED FURTHER READING

Carpenter, G.A. and S. Grossberg (1987a), Stable Self Organization of Pattern Recognition Codes for Analog Input Patterns, *Proc. First Int.*

Conference on Neural Networks, San Diego (IEEE, New York).

Carpenter, G.A. and S. Grossberg (1987b), Invariant Pattern Recognition and Recall by an Attentive Self Organizing ART Architecture in a Non-stationary World. *Proc. of Intl. Conf. on Neural Networks*, San Diego (IEEE, New York).

Pandya, A.S. and R.B. May (1996), *Pattern Recognition with Neural Networks in C++*, CRC press, USA.

REFERENCES

Carpenter, G.A. and S. Grossberg (1987b), ART2—Self Organization of Stable Category Recognition Codes for Analog Input Patterns, *Applied Optics*, 26, pp. 4919–4930.

Carpenter, G.A., S. Grossberg, and D.B. Rosen (1991), ART2A—An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition, *Neural Networks*, 4, pp. 495–504.

Freeman, J.A. and D.M. Skapura (1991), *Neural Networks*, Addison Wesley.

Gan, K.W. and K.T. Lua (1992), Chinese Character Classification using Adaptive Resonance Network, *Pattern Recognition*, Vol. 25, No. 8.

Grossberg, S. (1988), *Neural Networks and Natural Intelligence*, MIT Press Cambridge, MA, USA.

Kim, K.J., J.W. Jung, and S.K. Kim (1996), Online Character Recognition using ART based Stroke Classification, *Pattern Recognition Letters*, 17, pp. 1311–1322.

Kung, S.Y. (1993), *Digital Neural Networks*, Prentice–Hall, Englewood Cliff, NJ.

Pao, Y.H. (1989), *Adaptive Pattern Recognition Neural Networks*, Addison Wesley, MA.

Rajasekaran, S., D. Suresh, and G.A. Vijayalakshmi Pai (2002), Application of Sequential Learning Neural Networks to Civil Engineering Modeling Problems, *Engineering with Computers*, Vol. 8, pp. 138–147.

S. Rajasekaran, and R. Amalraj (2002), Image Recognition using ART1

Architecture Augmented with Moment based Feature Extractor, accepted to *Neuro Computing* for possible publication.

J.T. Tou, and R.C. Gonzalz (1974), *Pattern Recognition Principles*, Addison Wesley, Reading, MA.

Whang, B. (1969), Elasto-Plastic Orthotropic Plates and Shells, *Proc. of the Symposium on Application of Finite Element Methods in Civil Engg.* , ASCE, U.S.A., November,

pp. 481–515.

PART 2

FUZZY LOGIC

- Fuzzy Set Theory
- Fuzzy Systems

Chapter 6

Fuzzy Set Theory

Problems in the real world quite often turn out to be complex owing to an element of uncertainty either in the parameters which define the problem or in the situations in which the problem occurs.

Although probability theory has been an age old and effective tool to handle uncertainty, it can be applied only to situations whose characteristics are based on random processes, that is, processes in which the occurrence of events is strictly determined by chance. However, in reality, there turn out to be problems, a large class of them whose uncertainty is characterized by a nonrandom process. Here, the uncertainty may arise due to partial information about the problem, or due to information which is not fully reliable, or due to inherent imprecision in the language with which the problem is defined, or due to receipt of information from more than one source about the problem which is conflicting.

It is in such situations that *fuzzy set theory* exhibits immense potential for effective solving of the uncertainty in the problem. *Fuzziness* means

‘vagueness’. Fuzzy set theory is an excellent mathematical tool to handle the uncertainty arising due to vagueness. Understanding human speech and recognizing handwritten characters are some common instances where fuzziness manifests.

It was Lotfi A. Zadeh who propounded the fuzzy set theory in his seminal paper (Zadeh, 1965). Since then, a lot of theoretical developments have taken place in this field. It is however, the Japanese who seem to have fully exploited the potential of fuzzy sets by commercializing the technology.

More than 2000 patents have been acquired by the Japanese in the application of the technique and the area spans a wide spectrum, from consumer products and electronic instruments to automobile and traffic monitoring systems.

6.1 FUZZY VERSUS CRISP

Consider the query, “Is water colourless?” The answer to this is a definite *Yes/ True*, or *No/ False*, as warranted by the situation. If “yes”/“true” is accorded a value of 1 and “no”/“false” is accorded a value of 0, this statement results in a 0/1 type of situation. Such a logic which demands a binary (0/1) type of handling is termed *crisp* in the domain of fuzzy set theory. Thus, statements such as “Temperature is 32°C”, “The running time of the program is 4 seconds” are examples of crisp situations.

On the other hand, consider the statement, “Is Ram honest?” The answer to this query need not be a definite “yes” or “no”. Considering the degree to which one knows Ram, a variety of answers spanning a range, such as

“extremely honest”, “extremely dishonest”, “honest at times”, “very honest”

could be generated. If, for instance, “extremely honest” were to be accorded a value of 1, at the high end of the spectrum of values, “extremely dishonest” a value of 0 at the low end of the spectrum, then, “honest at times” and “very honest” could be assigned values of 0.4 and 0.85 respectively. The situation is therefore so fluid that it can accept values between 0 and 1, in contrast to the earlier one which was either a 0 or 1. Such a situation is termed *fuzzy*.

Figure 6.1 shows a simple diagram to illustrate fuzzy and crisp situations.

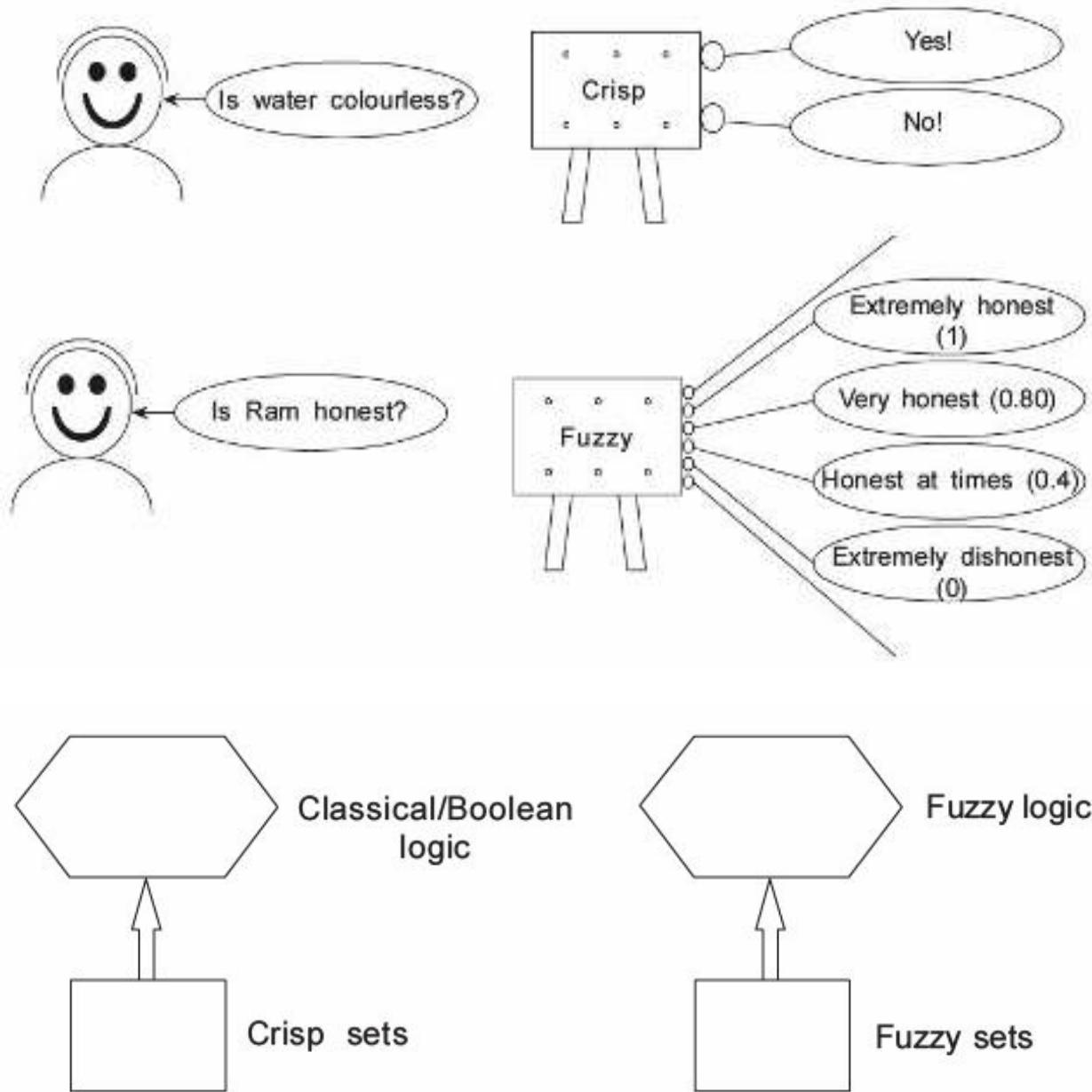


Fig. 6.1 Fuzzy versus crisp.

Classical set theory also termed *crisp set theory* and propounded by George Cantor is fundamental to the study of fuzzy sets. Just as Boolean logic had its roots in the theory of crisp sets, fuzzy logic has its roots in the theory of fuzzy sets (refer Fig. 6.1).

Fig. 6.2 Crisp sets and fuzzy sets.

We now briefly review crisp sets and its operations before a discussion on fuzzy sets is undertaken.

6.2 CRISP SETS

Universe of discourse

The *universe of discourse* or *universal set* is the set which, with reference to a particular context, contains all possible elements having the same characteristics and from which sets can be formed. The universal set is denoted by E .

Example

- (i) The universal set of all numbers in Euclidean space.
- (ii) The universal set of all students in a university.

Set

A set is a *well defined* collection of objects. Here, well defined means the object either belongs to or does not belong to the set (observe the “crispness” in the definition).

A set in certain contexts may be associated with its universal set from which it is derived.

Given a set A whose objects are $a_1, a_2, a_3, \dots, a_n$, we write A as $A = \{ a_1, a_2, \dots, a_n \}$. Here,

a_1, a_2, \dots, a_n are called the *members* of the set. Such a form of representing a set is known as *list form*.

Example

$$A = \{ \text{Gandhi, Bose, Nehru} \}$$

$$B = \{\text{Swan, Peacock, Dove}\}$$

A set may also be defined based on the properties the members have to satisfy. In such a case, a set A is defined as

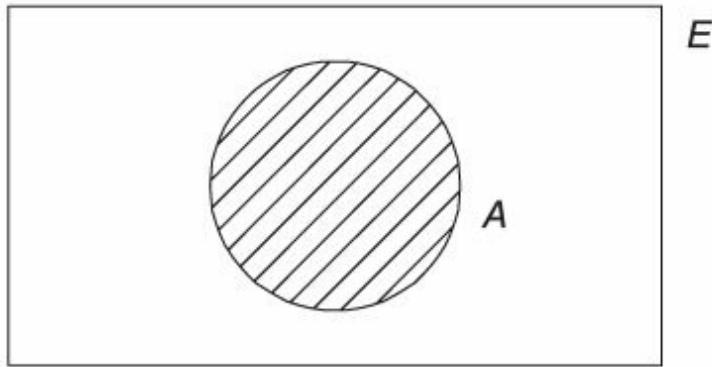
$$A = \{ x \mid P(x) \} \quad (6.1)$$

Here, $P(x)$ stands for the property P to be satisfied by the member x . This is read as ‘ A is the set of all X such that $P(x)$ is satisfied’.

Example

$$A = \{ x \mid x \text{ is an odd number} \}$$

$$B = \{ y \mid y > 0 \text{ and } y \bmod 5 = 0 \}$$



Venn diagram

Venn diagrams are pictorial representations to denote a set. Given a set A defined over a universal set E , the Venn diagram for A and E is as shown in Fig. 6.3.

Fig. 6.3 Venn diagram of a set A .

Example

In Fig. 6.3, if E represents the set of university students then A may represent the set of female students.

Membership

An element x is said to be a member of a set A if x belongs to the set A . The membership is indicated by ‘ \in ’ and is pronounced “belongs to”. Thus, $x \in A$ means x belongs to A and $x \notin A$ means x *does not belong to* A .

Example

Given $A = \{4, 5, 6, 7, 8, 10\}$, for $x = 3$ and $y = 4$, we have $x \in A$ and $y \notin A$

Here, observe that each element either belongs to or does not belong to a set. The concept of membership is definite and therefore crisp (1—belongs to, 0—does not belong to). In contrast, as we shall see later, a fuzzy set accommodates membership values which are not only 0 or 1 but *anything* between 0 and 1.

Cardinality

The number of elements in a set is called its cardinality. Cardinality of a set A is denoted as $n(A)$ or $|A|$ or $\#A$.

Example

If $A = \{4, 5, 6, 7\}$ then $|A| = 4$

Family of sets

A set whose members are sets themselves, is referred to as a family of sets.

Example

$A = \{\{1, 3, 5\}, \{2, 4, 6\}, \{5, 10\}\}$ is a set whose members are the sets $\{1, 3, 5\}$, $\{2, 4, 6\}$, and

$\{5, 10\}$.

Null Set/Empty Set

A set is said to be a *null set* or *empty set* if it has no members. A null set is indicated as \emptyset or $\{\}$ and indicates an impossible event. Also, $|\emptyset| = 0$.

Example

The set of all prime ministers who are below 15 years of age.

Singleton Set

A set with a single element is called a *singleton set*. A singleton set has cardinality of 1 .

Example

If $A = \{ a \}$, then $|A| = 1$

Subset

Given sets A and B defined over E the universal set, A is said to be a *subset* of B if A is fully contained in B , that is, every element of A is in B .

Denoted as $A \subset B$, we say that A is a subset of B , or A is a *proper subset* of B . On the other hand, if A is contained in or equivalent to that of B then we denote the subset relation as $A \subseteq B$. In such a case, A is called the *improper subset* of B .

Superset

Given sets A and B on E the universal set, A is said to be a *superset* of B if every element of B is contained in A .

Denoted as $A \supset B$, we say A is a superset of B or A contains B . If A contains B and is equivalent to B , then we denote it as $A = B$.

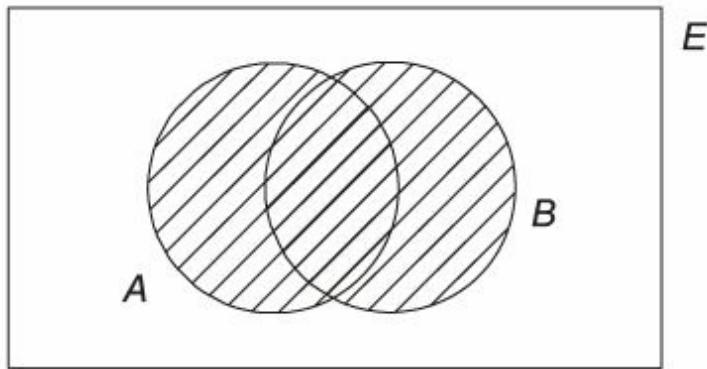
Example

Let $A = \{3, 4\}$ $B = \{3, 4, 5\}$ and $C = \{4, 5, 3\}$

Here, $A \subset B$, and $B \supset A$

$C \subseteq B$, and $B \supseteq C$

$$A \cup B = \{x/x \in A \text{ or } x \in B\}$$



$$A \cap B = \{x/x \in A \text{ and } x \in B\}$$

Power set

A *power set* of a set A is the set of all possible subsets that are derivable from A including null set.

A power set is indicated as $P(A)$ and has cardinality of $|P(A)| = 2^{|A|}$

Example

Let $A = \{3, 4, 6, 7\}$

$$P(A) = \{\{3\}, \{4\}, \{6\}, \{7\}, \{3, 4\}, \{4, 6\}, \{6, 7\}, \{3, 7\}, \{3, 6\}, \{4, 7\}, \{3, 4, 6\}, \{3, 4, 7\}, \{3, 6, 7\}, \{3, 4, 6, 7\}, \emptyset\}$$

Here, $|A| = 4$ and $|P(A)| = 2^4 = 16$.

6.2.1 Operations on Crisp Sets

Union (\cup)

The union of two sets A and B ($A \cup B$) is the set of all elements that belong to A or B or both.

(6.2)

Example

Given $A = \{a, b, c, 1, 2\}$ and $B = \{1, 2, 3, a, c\}$, we get $A \cup B = \{a, b, c, 1, 2, 3\}$

Figure 6.4 illustrates the Venn diagram representation for $A \cup B$

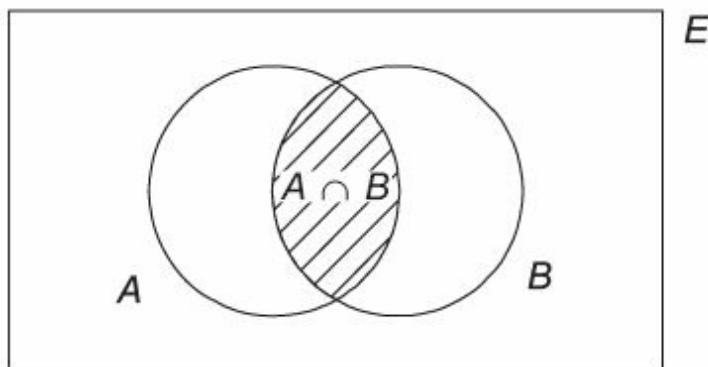
Fig. 6.4 Venn diagram for $A \cup B$.

Intersection (\cap)

The intersection of two sets A and B ($A \cap B$) is the set of all elements that belong to A and B

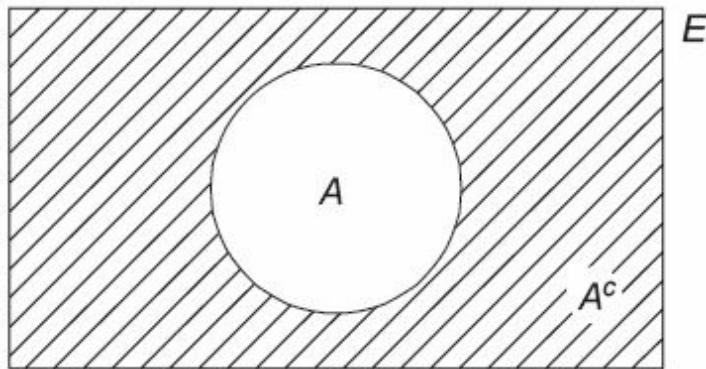
(6.3)

Any two sets which have $A \cap B = \emptyset$ are called *Disjoint Sets*.



$$A(\bar{A} | A^c)$$

$$A^c = \{x/x \notin A, x \in E\}$$



$$A - B = \{x \mid x \in A \text{ and } x \notin B\}$$

Example

Given $A = \{a, b, c, 1, 2\}$ and $B = \{1, 2, 3, a, c\}$, we get $A \cap B = \{a, c, 1, 2\}$

Figure 6.5 illustrates the Venn diagram for $A \cap B$

Fig. 6.5 Venn diagram for $A \cap B$.

Complement (c)

The complement of a set

is the set of all elements which are in E but

not in A .

(6.4)

Example

Given $X = \{1, 2, 3, 4, 5, 6, 7\}$ and $A = \{5, 4, 3\}$, we get $A^c = \{1, 2, 6, 7\}$

Figure 6.6 illustrates the Venn diagram for A^c .

Fig. 6.6 Venn diagram for Ac.

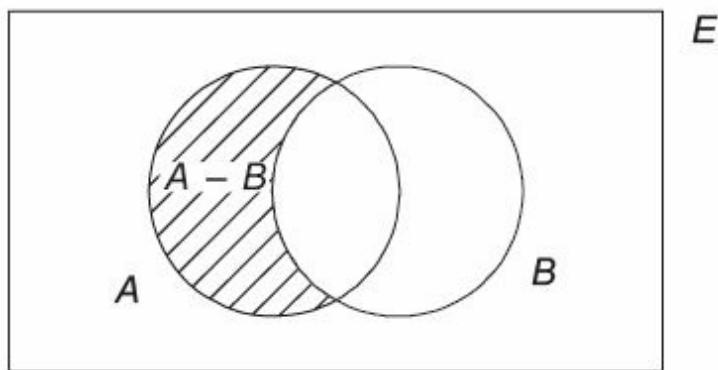
Difference (-)

The difference of the set A and B is $A - B$, the set of all elements which are in A but not in B .

(6.5)

Example

Given $A = \{ a, b, c, d, e \}$ and $B = \{ b, d \}$, we get $A - B = \{ a, c, e \}$



<i>Commutativity:</i>	$A \cup B = B \cup A$	
	$A \cap B = B \cap A$	(6.6)
<i>Associativity:</i>	$(A \cup B) \cup C = A \cup (B \cup C)$	
	$(A \cap B) \cap C = A \cap (B \cap C)$	(6.7)
<i>Distributivity:</i>	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	
	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	(6.8)
<i>Idempotence:</i>	$A \cup A = A$	
	$A \cap A = A$	(6.9)
<i>Identity:</i>	$A \cup \emptyset = A$	
	$A \cap E = A$	
	$A \cap \emptyset = \emptyset$	
	$A \cup E = E$	(6.10)
<i>Law of Absorption:</i>	$A \cup (A \cap B) = A$	
	$A \cap (A \cup B) = A$	(6.11)
<i>Transitivity:</i> If $A \subseteq B, B \subseteq C$ then $A \subseteq C$		(6.12)
<i>Involution:</i>	$(A^c)^c = A$	(6.13)
<i>Law of the Excluded Middle:</i>	$A \cup A^c = E$	(6.14)
<i>Law of Contradiction:</i>	$A \cap A^c = \emptyset$	(6.15)
<i>De Morgan's laws:</i>	$(A \cup B)^c = A^c \cap B^c$	
	$(A \cap B)^c = A^c \cup B^c$	(6.16)

Figure 6.7 illustrates the Venn diagram for $A - B$.

Fig. 6.7 Venn diagram for $A - B$.

6.2.2 Properties of Crisp Sets

The following properties of sets are important for further manipulation of sets.

All the properties could be verified by means of Venn diagrams.

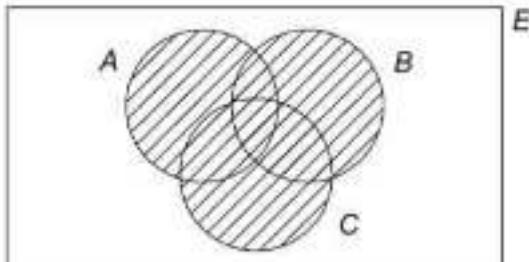
Example 6.1

Given three sets A, B , and C . Prove De Morgan's laws using Venn diagrams.

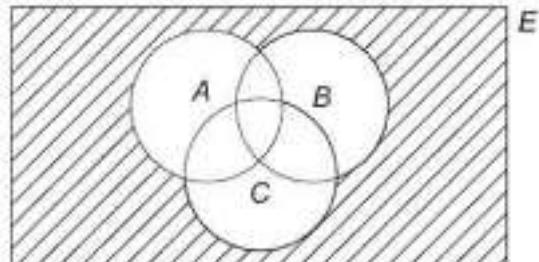
Solution

To prove De Morgan's laws, we need to show that

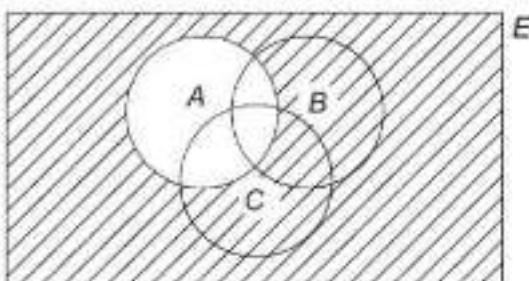
$$(i) (A \cup B \cup C)^c = A^c \cap B^c \cap C^c \quad (ii) (A \cap B \cap C)^c = A^c \cup B^c \cup C^c$$



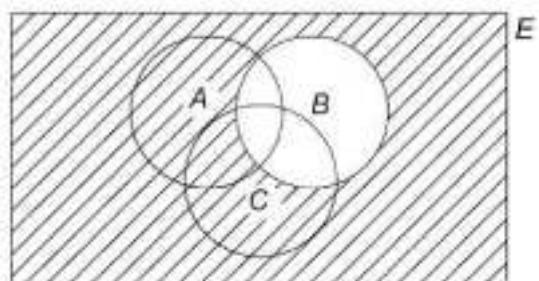
(a) $A \cup B \cup C$



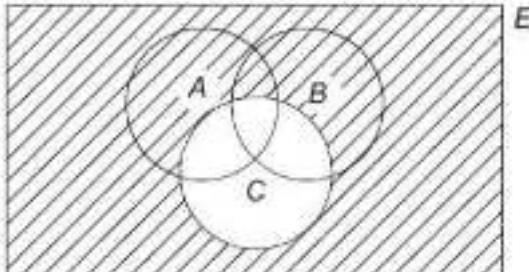
(b) $(A \cup B \cup C)^c$



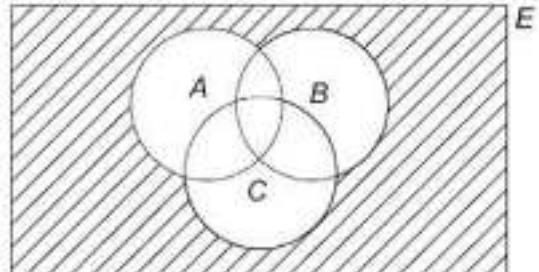
(c) A^c



(d) B^c

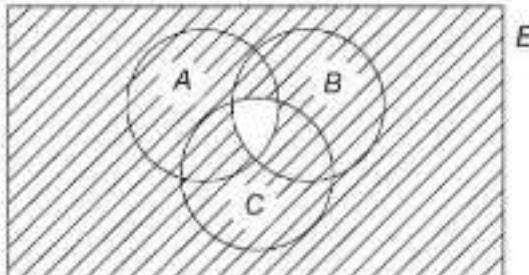


(e) C^c

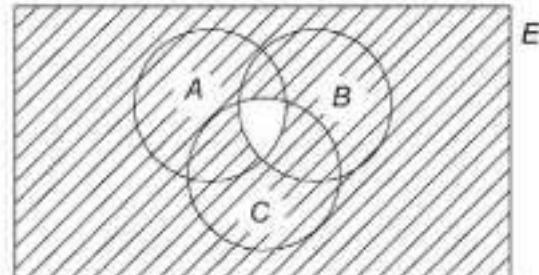


(f) $A^c \cap B^c \cap C^c$

(i) Here, it can be seen that $(A \cup B \cup C)^c = A^c \cap B^c \cap C^c$.



(g) $(A \cap B \cap C)^c$



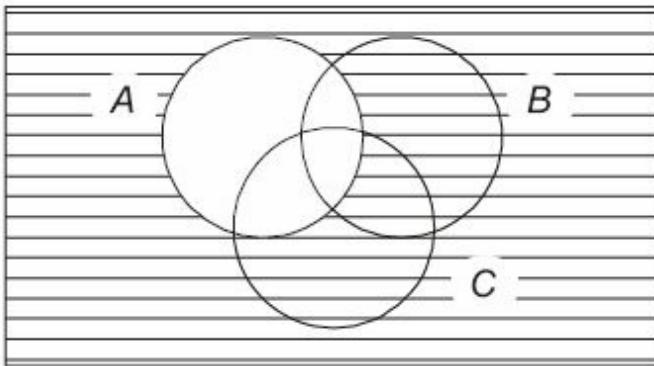
(h) $(A^c \cup B^c \cup C^c)$

(ii) Figures (g) to (h) show that $(A \cap B \cap C)^c = A^c \cup B^c \cup C^c$.

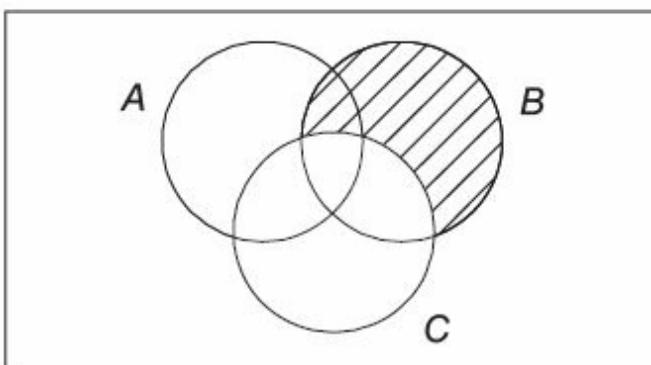
Example 6.2

Let the sets A , B , C , and E be given as follows: E = all students enrolled in the university cricket club.

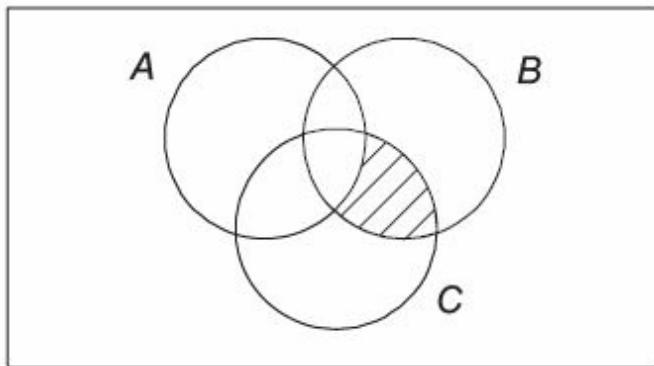
A = male students, B = bowlers, and C = batsmen.



(a) Female students



(b) Bowlers who are not batsmen



(c) Female students who can both bowl and bat

Draw individual Venn diagrams to illustrate (a) female students (b) bowlers who are not batsmen (c) female students who can both bowl and bat.

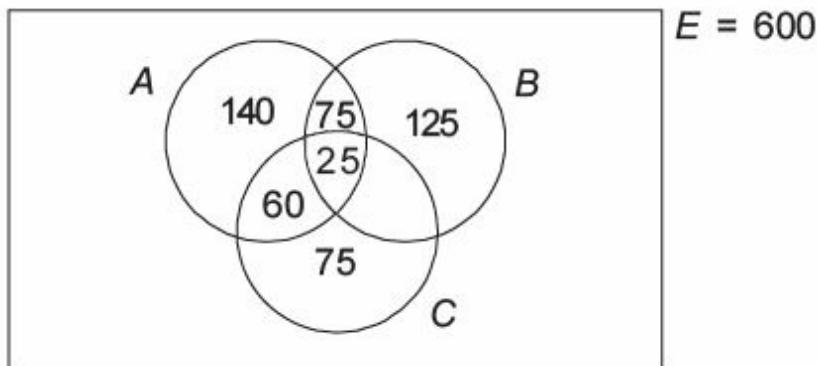
Solution

Example 6.3

In Example 6.2, assume that $|E| = 600$, $|A| = 300$, $|B| = 225$, $|C| = 160$. Also, let the number of male students who are bowlers ($A \cap B$) be 100, 25 of whom are batsmen too ($A \cap B \cap C$), and the total number of male students who are batsmen ($A \cap C$) be 85.

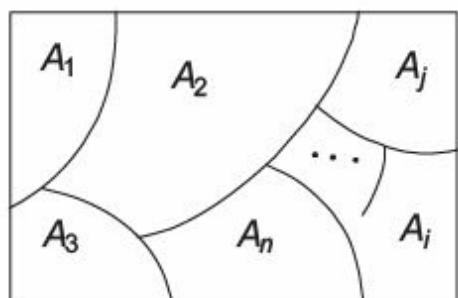
Determine the number of students who are: (i) Females, (ii) Not bowlers, (iii) Not batsmen,
(iv) Females and who can bowl but not bat.

Solution



$$A_i \cap A_j = \emptyset \text{ for each pair } (i, j) \in I, i \neq j$$

$$\bigcup_{i \in I} A_i = A$$



From the given data, the Venn diagram obtained is as follows:

- (i) No. of female students $|Ac| = |E| - |A| = 600 - 300 = 300$
- (ii) No. of students who are not bowlers $|Bc| = |E| - |B| = 600 - 225 = 375$
- (iii) No. of students who are not batsmen $|Cc| = |E| - |C| = 600 - 160 = 440$
- (iv) No. of female students who can bowl $|Ac \cap B| = 125$ (from the Venn diagram)

6.2.3 Partition and Covering

Partition

A *partition* on A is defined to be a set of non-empty subsets A_i , each of which is pairwise disjoint and whose union yields the original set A .

Partition on A indicated as $\Pi(A)$, is therefore

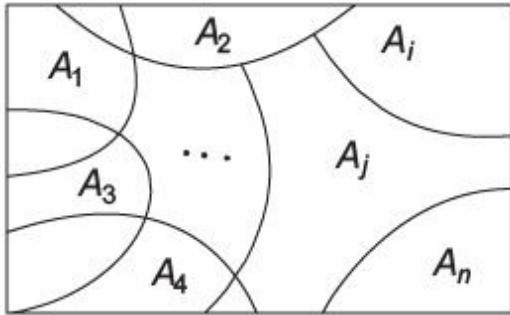
- (i)
(6.17)
- (ii)

The members A_i of the partition are known as blocks (refer Fig. 6.8).

Fig. 6.8 Partition of set A .

Example

Given $A = \{a, b, c, d, e\}$, $A_1 = \{a, b\}$, $A_2 = \{c, d\}$ and $A_3 = \{e\}$, which gives



$$|A| = |\bigcup_{i=1}^n A_i| = \sum_{i=1}^n |A_i|$$

$$\sum_{i=1}^3 |A_i| = 2 + 2 + 1 = 5$$

$$A_1 \cap A_2 = \emptyset, A_1 \cap A_3 = \emptyset, A_2 \cap A_3 = \emptyset$$

Also, $A_1 \cup A_2 \cup A_3 = A = \{a, b, c, d, e\}$

Hence, $\{A_1, A_2, A_3\}$, is a partition on A .

Covering

A *covering* on A is defined to be a set of non-empty subsets A_i whose union yields the original

set A . The non-empty subsets need not be disjoint (Refer Fig. 6.9).

Fig. 6.9 Covering of set A .

Example

Given $A = \{a, b, c, d, e\}$, $A_1 = \{a, b\}$, $A_2 = \{b, c, d\}$, and $A_3 = \{d, e\}$. This gives

$$A_1 \cap A_2 = \{b\}$$

$$A_1 \cap A_3 = \emptyset$$

$$A_2 \cap A_3 = \{d\}$$

Also, $A_1 \cup A_2 \cup A_3 = \{a, b, c, d, e\} = A$ Hence, $\{A_1, A_2, A_3\}$ is a covering on A .

Rule of Addition

Given a partition on A where $A_i, i = 1, 2, \dots, n$ are its non-empty subsets then,

.(6.18)

Example

Given $A = \{a, b, c, d, e\}$, $A_1 = \{a, b\}$, $A_2 = \{c, d\}$, $A_3 = \{e\}$, $|A| = 5$, and

$$|A| = \left| \bigcup_{i=1}^n A_i \right| = \sum_{i=1}^n |A_i| - \sum_{\substack{i=1 \\ i \neq j}}^n \sum_{j=1}^n |A_i \cap A_j|$$

$$|A| = \left| \bigcup_{i=1}^n A_i \right| = \sum_{i=1}^n |A_i| - \sum_{\substack{i=1 \\ i \neq j}}^n \sum_{j=1}^n |A_i \cap A_j|$$

Rule of Inclusion and Exclusion

Rule of addition is not applicable on the covering of set A , especially if the subsets are not pairwise disjoint. In such a case, the rule of inclusion and exclusion is applied.

Example

$$\begin{aligned} \text{Given } A \text{ to be a covering of } n \text{ sets } A_1, A_2, \dots, A_n, \text{ for } n = 2, \\ |A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2| \end{aligned} \quad |A| =$$

(6.19)

$$\text{for } n = 3, \quad |A| = |A_1 \cup A_2 \cup A_3| = |A_1| + |A_2| + |A_3| - |A_1 \cap A_2| - |A_2 \cap A_3| - |A_1 \cap A_3| + |A_1 \cap A_2 \cap A_3| \quad (6.20)$$

Generalizing,

(6.21)

Example 6.4

Given $|E| = 100$, where E indicates a set of students who have chosen subjects from different streams in the computer science discipline, it is found that 32

study subjects chosen from the Computer Networks (CN) stream, 20 from the Multimedia Technology (MMT) stream, and 45 from the Systems Software (SS) stream. Also, 15 study subjects from both CN and SS streams, 7 from both MMT and SS streams, and 30 do not study any subjects chosen from either of the three streams.

Find the number of students who study subjects belonging to all three streams.

Solution

Let A, B, C indicate students who study subjects chosen from CN, MMT, and SS streams respectively. The problem is to find $|A \cap B \cap C|$.

The no. of students who do not study any subject chosen from either of the

$$\begin{aligned}
 &\text{i.e.} & |A^c \cap B^c \cap C^c| &= 30 \\
 \Rightarrow & & |(A \cup B \cup C)^c| &= 30 \quad (\text{using De Morgan's laws}) \\
 \Rightarrow & & |E| - |A \cup B \cup C| &= 30 \\
 \Rightarrow & & |A \cup B \cup C| &= |E| - 30 \\
 & & & = 100 - 30 = 70
 \end{aligned}$$

From the principle of inclusion and exclusion,

$$\begin{aligned}
 &|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C| \\
 \Rightarrow & |A \cap B \cap C| = |A \cup B \cup C| - |A| - |B| - |C| + |A \cap B| + |B \cap C| + |A \cap C| \\
 & & = 70 - 32 - 20 - 45 + 15 + 7 + 10 \\
 & & = 5
 \end{aligned}$$

three

streams = 30.

Hence, the no. of students who study subjects chosen from all the three streams is 5.

$$A = \{(x, \mu_{\tilde{A}}(x)), x \in X\}$$

$$\sum_{x_i \in X} \mu_{\tilde{A}}(x_i)/x_i$$

$$\int_X \mu_{\tilde{A}}(x)/x$$

$$\tilde{A} = \{(g_1, 0.4) (g_2, 0.5) (g_3, 1) (g_4, 0.9) (g_5, 0.8)\}$$

6.3 FUZZY SETS

Fuzzy sets support a flexible sense of membership of elements to a set. While in crisp set theory, an element either belongs to or does not belong to a set, in fuzzy set theory many degrees of membership (between 0 and 1) are allowed.

Thus, a membership function $\mu_A(x)$

A

is associated with a fuzzy set \tilde{A} such that

the function maps every element of the universe of discourse X (or the reference set) to the interval $[0, 1]$.

Formally, the mapping is written as $\mu_{\tilde{A}}(x) : X \rightarrow [0, 1]$

A *fuzzy set* is defined as follows:

If X is a universe of discourse and x is a particular element of X , then a fuzzy set A defined on X may be written as a collection of ordered pairs (6.23)

where each pair $(x, \mu_{\tilde{A}}(x))$ is called a singleton. In crisp sets, $\mu_{\tilde{A}}(x)$ is dropped.

An alternative definition which indicates a fuzzy set as a union of all $\mu_{\tilde{A}}(x)/x$ singletons is given by

$A =$

in the discrete case (6.24)

and

$A =$

in the continuous case (6.25)

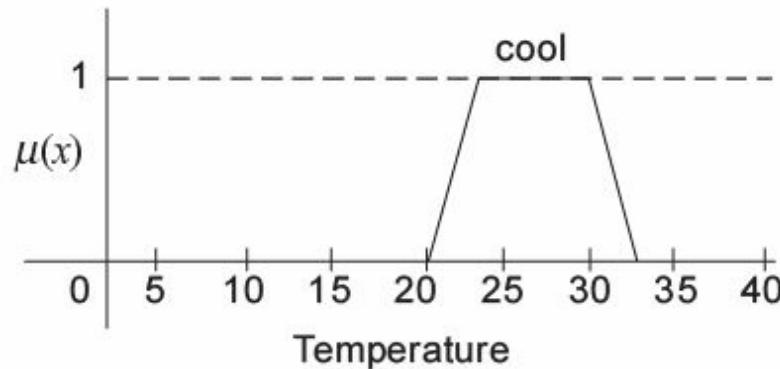
Here, the summation and integration signs indicate the union of all $\mu_{\tilde{A}}(x)/x$ singletons.

Example

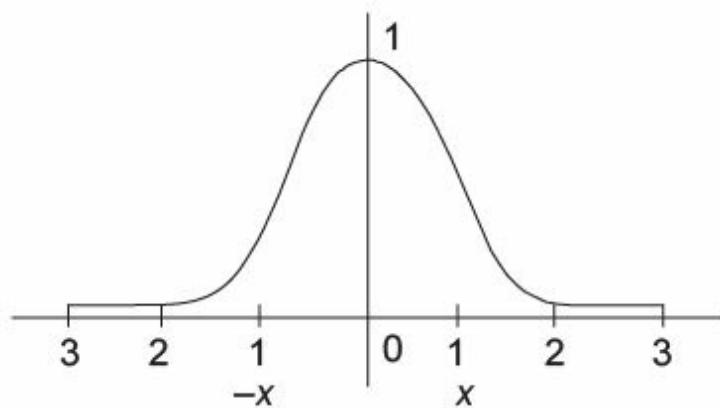
Let $X = \{g_1, g_2, g_3, g_4, g_5\}$ be the reference set of students. Let \tilde{A} be the fuzzy set of “smart” students, where “smart” is a fuzzy linguistic term.

Here \tilde{A} indicates that the smartness of g_1 is 0.4, g_2 is 0.5 and so on when graded over a scale of 0–1.

Though fuzzy sets model vagueness, it needs to be realized that the definition of the sets varies according to the context in which it is used. Thus,



$$\mu_{\tilde{A}}(x) = \frac{1}{(1+x)^2}$$



the fuzzy linguistic term “tall” could have one kind of fuzzy set while referring to the height of a building and another kind of fuzzy set while referring to the height of human beings.

6.3.1 Membership Function

The membership function values need not always be described by discrete values. Quite often, these turn out to be as described by a continuous function.

The fuzzy membership function for the fuzzy linguistic term “cool” relating to temperature may turn out to be as illustrated in Fig. 6.10.

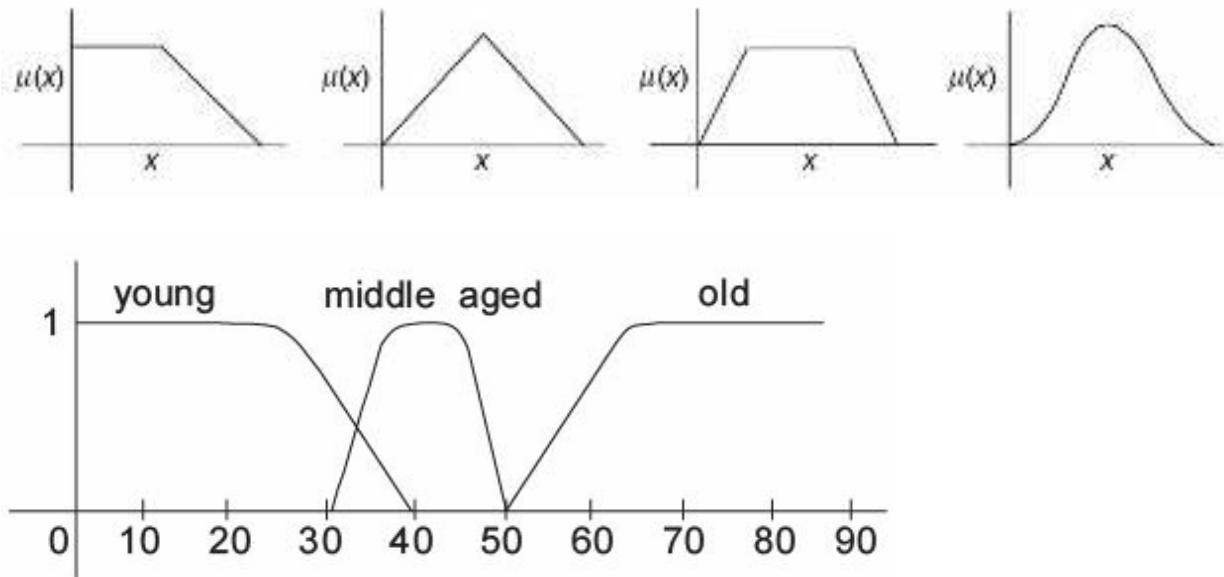
Fig. 6.10 Continuous membership function for “cool”.

A membership function can also be given mathematically as

The graph is as shown in Fig. 6.11.

Fig. 6.11 Continuous membership function dictated by a mathematical function.

Different shapes of membership functions exist. The shapes could be triangular, trapezoidal, curved or their variations as shown in Fig. 6.12.



\tilde{B}

\tilde{B}

\tilde{B}

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$$

Fig. 6.12 Different shapes of membership function graphs.

Example

Consider the set of people in the following age groups

0–10

40–50

10–20

50–60

20–30

60–70

30–40

70 and above

The fuzzy sets “young”, “middle-aged”, and “old” are represented by the membership function graphs as illustrated in Fig. 6.13.

Fig. 6.13 Example of fuzzy sets expressing “young”, “middle-aged”, and “old”.

6.3.2 Basic Fuzzy Set Operations

Given X to be the universe of discourse and \tilde{A} and \tilde{B} to be fuzzy sets with $\mu_A(x)$ and $\mu_B(x)$ as their respective membership functions, the basic fuzzy set operations are as follows:

Union

The union of two fuzzy sets \tilde{A} and \tilde{B} is a new fuzzy set $\tilde{A} \cup \tilde{B}$ also on X with a membership function defined as

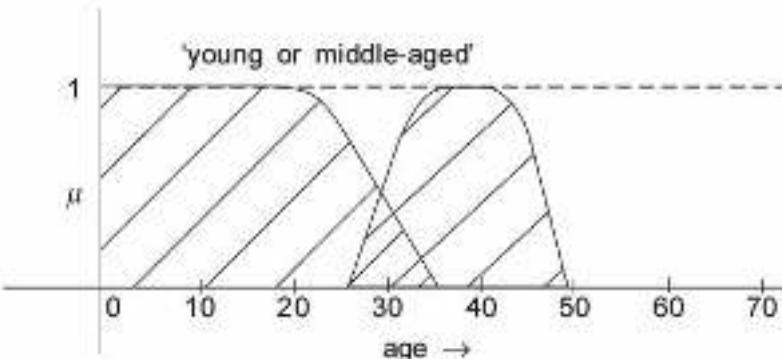
.(6.26)

Example

\tilde{A}

\tilde{B}

$\tilde{A} \cup \tilde{B}$:



if

$$A = \{(x_1, 0.5), (x_2, 0.7), (x_3, 0)\} \quad \text{and} \quad \tilde{B} = \{(x_1, 0.8), (x_2, 0.2), (x_3, 1)\}$$

$$\tilde{A} \cup \tilde{B} = \{(x_1, 0.8), (x_2, 0.7), (x_3, 1)\}$$

since,

$$\begin{aligned}\mu_{\tilde{A} \cup \tilde{B}}(x_1) &= \max(\mu_{\tilde{A}}(x_1), \mu_{\tilde{B}}(x_1)) \\ &= \max(0.5, 0.8) \\ &= 0.8\end{aligned}$$

$$\mu_{\tilde{A} \cup \tilde{B}}(x_2) = \max(\mu_{\tilde{A}}(x_2), \mu_{\tilde{B}}(x_2)) = \max(0.2, 0.7) = 0.7$$

$$\mu_{\tilde{A} \cup \tilde{B}}(x_3) = \max(\mu_{\tilde{A}}(x_3), \mu_{\tilde{B}}(x_3)) = \max(0, 1) = 1$$

\tilde{B}

\tilde{B}

$$\mu_{\tilde{A} \cap \tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$$

\tilde{B}

Let \tilde{A} be the fuzzy set of young people and \tilde{B} be the fuzzy set of middle-aged people as illustrated in Fig. 6.13. Now $\tilde{A} \cup \tilde{B}$, the fuzzy set of “young or middle-aged” will be given by

In its discrete form, for x_1, x_2, x_3

Intersection

The intersection of fuzzy sets \tilde{A} and \tilde{B} is a new fuzzy set $\tilde{A} \cap \tilde{B}$

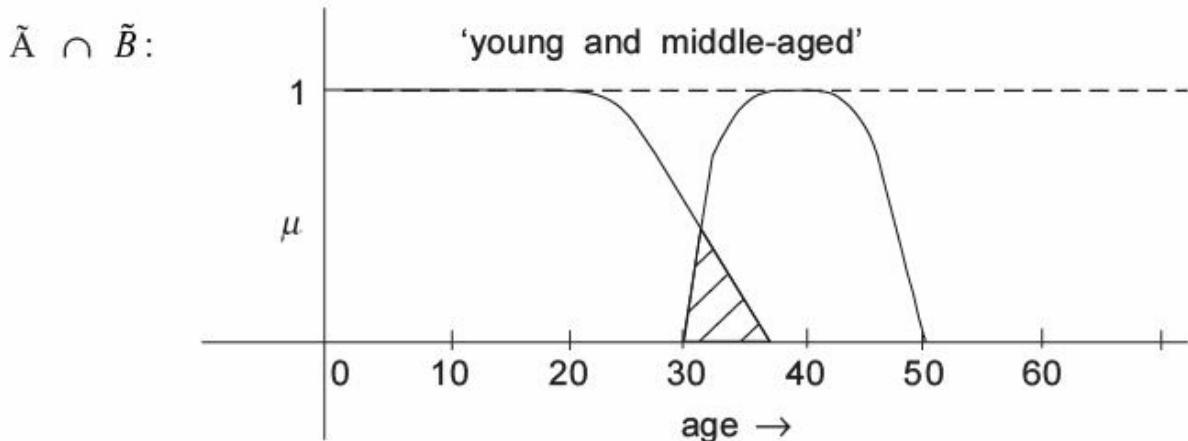
with

membership function defined as

(6.27)

Example

For \tilde{A} and \tilde{B} defined as “young” and “middle-aged” as illustrated in previous examples.



if

$$\tilde{A} = \{(x_1, 0.5), (x_2, 0.7), (x_3, 0)\} \text{ and } \tilde{B} = \{(x_1, 0.8), (x_2, 0.2), (x_3, 1)\}$$

$$\tilde{A} \cap \tilde{B} = \{(x_1, 0.5), (x_2, 0.2), (x_3, 0)\}$$

since,

$$\begin{aligned}\mu_{\tilde{A} \cap \tilde{B}}(x_1) &= \min(\mu_{\tilde{A}}(x_1), \mu_{\tilde{B}}(x_1)) \\ &= \min(0.5, 0.8) \\ &= 0.5\end{aligned}$$

$$\begin{aligned}\mu_{\tilde{A} \cap \tilde{B}}(x_2) &= \min(\mu_{\tilde{A}}(x_2), \mu_{\tilde{B}}(x_2)) \\ &= \min(0.7, 0.2) \\ &= 0.2\end{aligned}$$

$$\begin{aligned}\mu_{\tilde{A} \cap \tilde{B}}(x_3) &= \min(\mu_{\tilde{A}}(x_3), \mu_{\tilde{B}}(x_3)) \\ &= \min(0, 1) \\ &= 0\end{aligned}$$

$$\mu_{\tilde{A}}(x) = 1 - \mu_{\tilde{A}}(x)$$

In its discrete form, for x_1, x_2, x_3

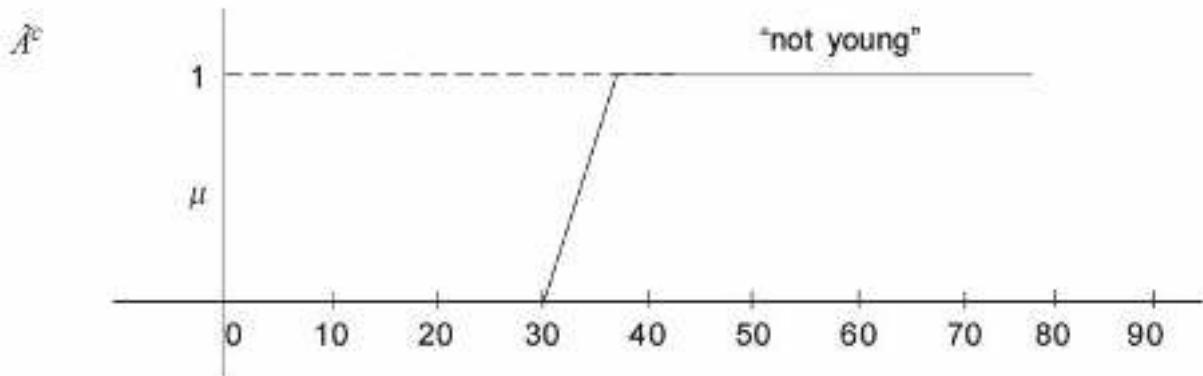
Complement

The complement of a fuzzy set \tilde{A} is a new fuzzy set \tilde{A}^c with a membership function

(6.28)

Example

For the fuzzy set \tilde{A} defined as “young” the complement “not young” is given by \tilde{A}^c . In its discrete form, for x_1, x_2 , and x_3



if

$$\tilde{A} = \{(x_1, 0.5) (x_2, 0.7) (x_3, 0)\}$$

then,

$$\tilde{A}^c = \{(x_1, 0.5) (x_2, 0.3) (x_3, 1)\}$$

since,

$$\mu_{\tilde{A}^c}(x_1) = 1 - \mu_{\tilde{A}}(x_1)$$

$$= 1 - 0.5$$

$$= 0.5$$

$$\mu_{\tilde{A}^c}(x_2) = 1 - \mu_{\tilde{A}}(x_2)$$

$$= 1 - 0.7$$

$$= 0.3$$

$$\mu_{\tilde{A}^c}(x_3) = 1 - \mu_{\tilde{A}}(x_3)$$

$$= 1 - 0$$

$$= 1$$

Other operations are,

\tilde{B}

\tilde{B}

$$\mu_{\tilde{A} \cdot \tilde{B}}(x) = \mu_{\tilde{A}}(x) \mu_{\tilde{B}}(x)$$

Product of two fuzzy sets

The product of two fuzzy sets \tilde{A} and \tilde{B} is a new fuzzy set $\tilde{A} \cdot \tilde{B}$ whose membership function is defined as

(6.29)

Example

$$\tilde{A} = \{(x_1, 0.2), (x_2, 0.8), (x_3, 0.4)\}$$

$$\tilde{B} = \{(x_1, 0.4), (x_2, 0), (x_3, 0.1)\}$$

$$\tilde{A} \cdot \tilde{B} = \{(x_1, 0.08), (x_2, 0), (x_3, 0.04)\}$$

Since

$$\begin{aligned}\mu_{\tilde{A} \cdot \tilde{B}}(x_1) &= \mu_{\tilde{A}}(x_1) \cdot \mu_{\tilde{B}}(x_1) \\ &= 0.2 \cdot 0.4 = 0.08\end{aligned}$$

$$\begin{aligned}\mu_{\tilde{A} \cdot \tilde{B}}(x_2) &= \mu_{\tilde{A}}(x_2) \cdot \mu_{\tilde{B}}(x_2) \\ &= 0.8 \cdot 0 = 0\end{aligned}$$

$$\mu_{\tilde{A} \cdot \tilde{B}}(x_3) = \mu_{\tilde{A}}(x_3) \cdot \mu_{\tilde{B}}(x_3)$$

$$\tilde{B}$$

$$\tilde{B}$$

$$\mu_{\tilde{A}}(x) = \mu_{\tilde{B}}(x)$$

$$\tilde{A} = \{(x_1, 0.2)(x_2, 0.8)\}$$

$$\tilde{B} = \{(x_1, 0.6)(x_2, 0.8)\}$$

$$\tilde{C} = \{(x_1, 0.2)(x_2, 0.8)\}$$

$$\tilde{A} \neq \tilde{B}$$

since

$$\mu_{\tilde{A}}(x_1) \neq \mu_{\tilde{B}}(x_1) \quad \text{although}$$

$$\mu_{\tilde{A}}(x_2) = \mu_{\tilde{B}}(x_2)$$

but

$$\tilde{A} = \tilde{C}$$

since

$$\mu_{\tilde{A}}(x_1) = \mu_{\tilde{C}}(x_1) = 0.2$$

and

$$\mu_{\tilde{A}}(x_2) = \mu_{\tilde{C}}(x_2) = 0.8$$

$$\mu_{a \cdot \tilde{A}}(x) = a \cdot \mu_{\tilde{A}}(x)$$

$$= 0.4 \cdot 0.1$$

$$= 0.04$$

Equality

Two fuzzy sets \tilde{A} and

are said to be equal ($\tilde{A} = \tilde{B}$) if

(6.30)

Example

Product of a fuzzy set with a crisp number

Multiplying a fuzzy set \tilde{A} by a crisp number a results in a new fuzzy set product $a \cdot \tilde{A}$ with the membership function

(6.31)

Example

$$\tilde{A} = \{(x_1, 0.4), (x_2, 0.6), (x_3, 0.8)\}$$

For

$$a = 0.3$$

$$a \cdot \tilde{A} = \{(x_1, 0.12), (x_2, 0.18), (x_3, 0.24)\}$$

since,

$$\begin{aligned}\mu_{a \cdot \tilde{A}}(x_1) &= a \cdot \mu_{\tilde{A}}(x_1) \\ &= 0.3 \cdot 0.4 \\ &= 0.12\end{aligned}$$

$$\begin{aligned}\mu_{a \cdot \tilde{A}}(x_2) &= a \cdot \mu_{\tilde{A}}(x_2) \\ &= 0.3 \cdot 0.6 \\ &= 0.18\end{aligned}$$

$$\begin{aligned}\mu_{a \cdot \tilde{A}}(x_3) &= a \cdot \mu_{\tilde{A}}(x_3) \\ &= 0.3 \cdot 0.8 \\ &= 0.24\end{aligned}$$

$$\mu_{A^a}(x) = (\mu_{\tilde{A}}(x))^{\alpha}$$

$$\tilde{A} = \{(x_1, 0.4), (x_2, 0.2), (x_3, 0.7)\}$$

For

$$\alpha = 2$$

$$\mu_{\tilde{A}^2}(x) = (\mu_{\tilde{A}}(x))^2$$

Hence,

$$(\tilde{A})^2 = \{(x_1, 0.16), (x_2, 0.04), (x_3, 0.49)\}$$

Since

$$\mu_{\tilde{A}^2}(x_1) = (\mu_{\tilde{A}}(x_1))^2 = (0.4)^2 = 0.16$$

$$\mu_{\tilde{A}^2}(x_2) = (\mu_{\tilde{A}}(x_2))^2 = (0.2)^2 = 0.04$$

$$\mu_{\tilde{A}^2}(x_3) = (\mu_{\tilde{A}}(x_3))^2 = (0.7)^2 = 0.49$$

\tilde{B}

\tilde{B}

Power of a fuzzy set

The a power of a fuzzy set \tilde{A} is a new fuzzy set Aa whose membership function is given by

(6.32)

Raising a fuzzy set to its second power is called *Concentration* (CON) and taking the square root is called *Dilation* (DIL).

Example

Difference

The difference of two fuzzy sets \tilde{A} and \tilde{B} is a new fuzzy set $\tilde{A} - \tilde{B}$ – defined as

$$\tilde{A} - \tilde{B} = (\tilde{A} \cap \tilde{B}^c)$$

$$\begin{aligned}\tilde{A} &= \{(x_1, 0.2), (x_2, 0.5), (x_3, 0.6)\}; \quad \tilde{B} = \{(x_1, 0.1), (x_2, 0.4), (x_3, 0.5)\} \\ \tilde{B}^c &= \{(x_1, 0.9), (x_2, 0.6), (x_3, 0.5)\} \\ \tilde{A} - \tilde{B} &= \tilde{A} \cap \tilde{B}^c \\ &= \{(x_1, 0.2)(x_2, 0.5)(x_3, 0.5)\}\end{aligned}$$

\tilde{B}

\tilde{B}

$$\tilde{A} \oplus \tilde{B} = (\tilde{A}^c \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{B}^c)$$

$$\begin{aligned}\tilde{A} &= \{(x_1, 0.4)(x_2, 0.8)(x_3, 0.6)\} \\ \tilde{B} &= \{(x_1, 0.2)(x_2, 0.6)(x_3, 0.9)\} \\ \text{Now, } \tilde{A}^c &= \{(x_1, 0.6)(x_2, 0.2)(x_3, 0.4)\} \\ \tilde{B}^c &= \{(x_1, 0.8)(x_2, 0.4)(x_3, 0.1)\} \\ \tilde{A}^c \cap \tilde{B} &= \{(x_1, 0.2)(x_2, 0.2)(x_3, 0.4)\} \\ \tilde{A} \cap \tilde{B}^c &= \{(x_1, 0.4)(x_2, 0.4)(x_3, 0.1)\} \\ \tilde{A} \oplus \tilde{B} &= \{(x_1, 0.4)(x_2, 0.4)(x_3, 0.4)\}\end{aligned}$$

(6.33)

Example

Disjunctive sum

The disjunctive sum of two fuzzy sets \tilde{A} and \tilde{B} is a new fuzzy set $\tilde{A} \approx \tilde{B}$ defined as

(6.34)

Example

6.3.3 Properties of Fuzzy Sets

Fuzzy sets follow some of the properties satisfied by crisp sets. In fact, crisp sets can be thought of as special instances of fuzzy sets. Any fuzzy set \tilde{A} is a subset of the reference set X . Also, the membership of any element belonging to the null set \emptyset is 0 and the membership of any element belonging to the reference set is 1.

The properties satisfied by fuzzy sets are

$$\begin{aligned} \text{Commutativity:} \quad & \tilde{A} \cup \tilde{B} = \tilde{B} \cup \tilde{A} \\ & \tilde{A} \cap \tilde{B} = \tilde{B} \cap \tilde{A} \end{aligned} \quad (6.35)$$

$$\begin{aligned} \text{Associativity:} \quad & \tilde{A} \cup (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cup \tilde{C} \\ & \tilde{A} \cap (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cap \tilde{C} \end{aligned} \quad (6.36)$$

$$\begin{aligned} \text{Distributivity:} \quad & \tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap (\tilde{A} \cup \tilde{C}) \\ & \tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{C}) \end{aligned} \quad (6.37)$$

$$\begin{aligned} \text{Idempotence:} \quad & \tilde{A} \cup \tilde{A} = \tilde{A} \\ & \tilde{A} \cap \tilde{A} = \tilde{A} \end{aligned} \quad (6.38)$$

$$\begin{aligned} \text{Identity:} \quad & \tilde{A} \cup \emptyset = \tilde{A} \\ & \tilde{A} \cup X = \tilde{A} \\ & \tilde{A} \cap \emptyset = \emptyset \\ & \tilde{A} \cup X = X \end{aligned} \quad (6.39)$$

$$\text{Transitivity: If } \tilde{A} \subseteq \tilde{B} \subseteq \tilde{C}, \text{ then } \tilde{A} \subseteq \tilde{C} \quad (6.40)$$

$$\text{Involution: } (\tilde{A}^c)^c = \tilde{A} \quad (6.41)$$

$$\begin{aligned} \text{De Morgan's laws:} \quad & (\tilde{A} \cap \tilde{B})^c = (\tilde{A}^c \cup \tilde{B}^c) \\ & (\tilde{A} \cup \tilde{B})^c = (\tilde{A}^c \cap \tilde{B}^c) \end{aligned} \quad (6.42)$$

\tilde{I}

\tilde{F}

\tilde{I}

\tilde{F}

$$\tilde{I}$$

$$\tilde{F}$$

$$(\tilde{I} - \tilde{F})$$

$$\tilde{F} \cup \tilde{F}^c$$

$$(\tilde{I} \cup \tilde{F})^c = \tilde{I}^c \cap \tilde{F}^c$$

Since fuzzy sets can overlap, the laws of excluded middle do not hold good.

Thus,

$$\tilde{A} \cup \tilde{A}^c \neq X \quad (6.43)$$

$$\tilde{A} \cap \tilde{A}^c \neq \emptyset \quad (6.44)$$

Example 6.5

The task is to recognize English alphabetical characters (F, E, X, Y, I, T) in an image processing system.

Define two fuzzy sets and to represent the identification of characters I and F .

$$= \{(F, 0.4), (E, 0.3), (X, 0.1), (Y, 0.1), (I, 0.9), (T, 0.8)\}$$

$$= \{(F, 0.99), (E, 0.8), (X, 0.1), (Y, 0.2), (I, 0.5), (T, 0.5)\}$$

Find the following.

(a) (i) \cup (ii)

(iii)

(b) Verify De Morgan's Law,

$$(a) \quad (i) \quad \tilde{I} \cup \tilde{F} = \{(F, 0.99), (E, 0.8), (X, 0.1), (Y, 0.2), (I, 0.9), (T, 0.8)\}$$

$$(ii) \quad \tilde{I} - \tilde{F} = (\tilde{I} \cap \tilde{F}^c) \\ = \{(F, 0.01), (E, 0.2), (X, 0.1), (Y, 0.1), (I, 0.5), (T, 0.5)\}$$

$$(iii) \quad \tilde{F} \cup \tilde{F}^c = \{(F, 0.99), (E, 0.8), (X, 0.9), (Y, 0.8), (I, 0.5), (T, 0.5)\}$$

$$(\tilde{I} \cup \tilde{F})^c = \tilde{I}^c \cap \tilde{F}^c$$

$$\tilde{I} \cup \tilde{F} = \{(F, 0.99), (E, 0.8), (X, 0.1), (Y, 0.2), (I, 0.9), (T, 0.8)\}$$

$$(\tilde{I} \cup \tilde{F})^c = \{(F, 0.01), (E, 0.2), (X, 0.9), (Y, 0.8), (I, 0.1), (T, 0.2)\}$$

$$\tilde{I}^c = \{(F, 0.6), (E, 0.7), (X, 0.9), (Y, 0.9), (I, 0.1), (T, 0.2)\}$$

$$\tilde{F}^c = \{(F, 0.01), (E, 0.2), (X, 0.9), (Y, 0.8), (I, 0.5), (T, 0.5)\}$$

and

$$\tilde{I}^c \cap \tilde{F}^c = \{(F, 0.01), (E, 0.2), (X, 0.9), (Y, 0.8), (I, 0.1), (T, 0.2)\}$$

$$\text{Hence, } (\tilde{I} \cup \tilde{F})^c = \tilde{I}^c \cap \tilde{F}^c$$

\tilde{B}

$$\mu_{\tilde{A}}(x) = \frac{x}{x+1}, \quad \mu_{\tilde{B}}(x) = 2^{-x}$$

Solution

(b) De Morgan's Law

Example 6.6

Consider the fuzzy sets \tilde{A} and \tilde{B} defined on the interval $X = [0, 5]$ of real numbers, by the membership grade functions

Determine the mathematical formulae and graphs of the membership grade functions of each of the following sets

(a) A_c, B_c

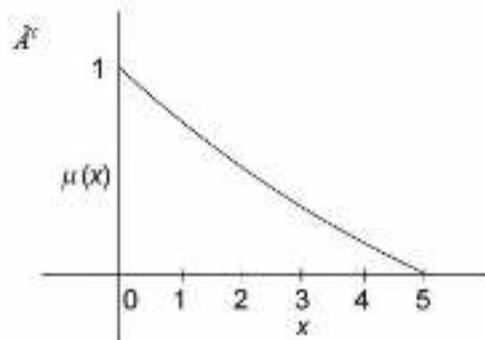
(b) $A \cup B$

(c) $A \cap B$

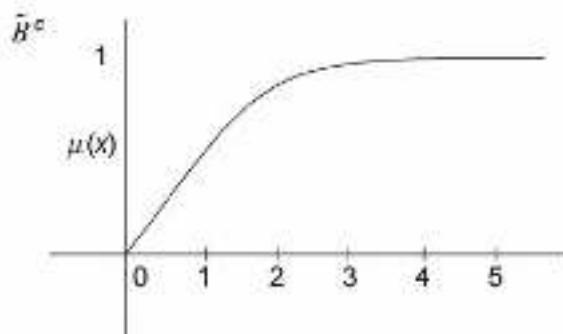
(d) $(A \cup B)^c$

Solution

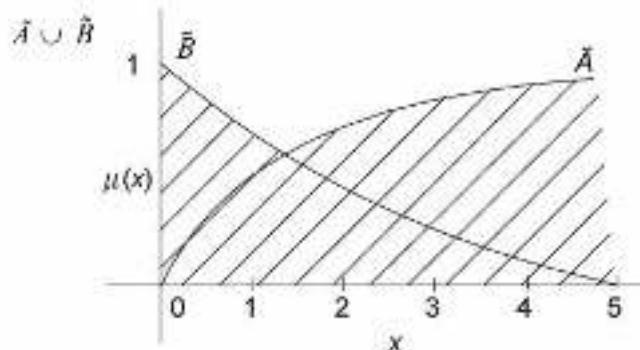
(a) $\mu_{\tilde{A}^c}(x) = 1 - \mu_{\tilde{A}}(x) = 1 - \frac{x}{x+1}$
 $= \frac{1}{x+1}$



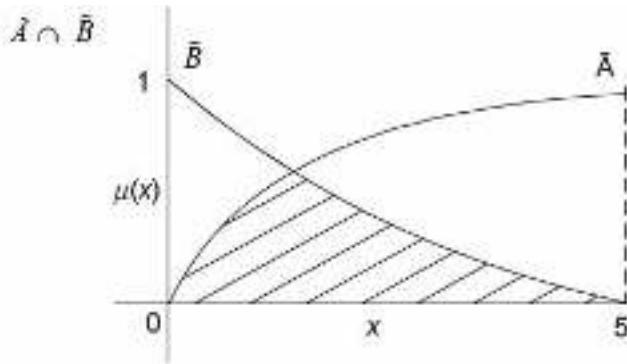
$$\begin{aligned}\mu_{\tilde{B}^c}(x) &= 1 - \mu_{\tilde{B}}(x) \\ &= 1 - 2^{-x} \\ &= \frac{2^x - 1}{2^x}\end{aligned}$$



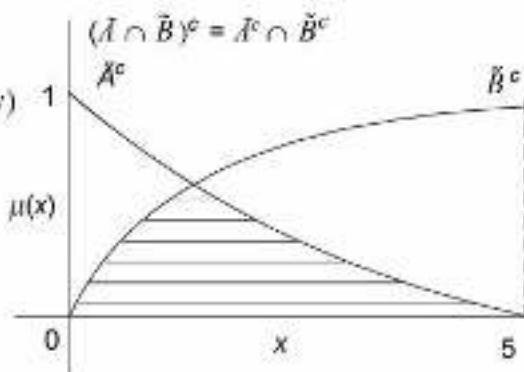
(b) $\mu_{\tilde{A} \cup \tilde{B}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$
 $= \max\left(\frac{x}{x+1}, 2^{-x}\right)$



$$(c) \quad \mu_{\tilde{A} \cap \tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) \\ = \min\left(\frac{x}{x+1}, 2^{-x}\right)$$



$$(d) \quad \mu_{(\tilde{A} \cup \tilde{B})^c}(x) = \mu_{\tilde{A}^c \cap \tilde{B}^c}(x) \quad (\because \text{De Morgan's law}) \\ = \min(\mu_{\tilde{A}^c}(x), \mu_{\tilde{B}^c}(x)) \\ = \min\left(\frac{1}{x+1}, \frac{2^x-1}{2^x}\right)$$



$$A \times B = \{(a, b) / a \in A, b \in B\}$$

$$\bigtimes_{i=1}^n A_i = \{(a_1, a_2, a_3, \dots, a_n) / a_i \in A_i \text{ for every } i = 1, 2, \dots, n\}$$

$$\left| \bigtimes_{i=1}^n A_i \right| = \prod_{i=1}^n |A_i|$$

$$A_1 \times A_2 = \{(a, 1), (b, 1), (a, 2), (b, 2)\}, |A_1 \times A_2| = 4, \text{ and } |A_1| = |A_2| = 2$$

$$\text{Hence, } |A_1 \times A_2| = |A_1| \cdot |A_2|$$

$$\text{Also, } A_1 \times A_2 \times A_3 = \{(a, 1, \alpha), (a, 2, \alpha), (b, 1, \alpha), (b, 2, \alpha)\}$$

$$|A_1 \times A_2 \times A_3| = 4 = |A_1| \cdot |A_2| \cdot |A_3|$$

$$\bigtimes_{i=1}^n X_i$$

6.4 CRISP RELATIONS

In this section, we review crisp relations as a prelude to fuzzy relations. The concept of relations between sets is built on the Cartesian product operator of sets.

6.4.1 Cartesian Product

The *Cartesian product* of two sets A and B denoted by $A \times B$ is the set of all ordered pairs such that the first element in the pair belongs to A and the second element belongs to B .

i.e.

If $A \neq B$ and A and B are non-empty then $A \times B \neq B \times A$.

The Cartesian product could be extended to n number of sets (6.45)

Observe that

(6.46)

Example

Given $A_1 = \{a, b\}$, $A_2 = \{1, 2\}$, $A_3 = \{a\}$, **6.4.2 Other Crisp Relations**

An n -ary relation denoted as $R(X_1, X_2, \dots, X_n)$ among crisp sets X_1, X_2, \dots, X_n is a subset of the Cartesian product

and is indicative of an association or

relation among the tuple elements.

For $n = 2$, the relation $R(X_1, X_2)$ is termed as a *binary* relation; for $n = 3$, the relation is termed *ternary*; for $n = 4$, *quaternary*; for $n = 5$, *quinary* and so on.

$$\in R \text{ and } R(i, j) = 0 \text{ if } (x_i, y_j) \notin R$$

$$X \times X = \left\{ \begin{array}{l} (1,1)(1,2)(1,3)(1,4)(2,1)(2,2)(2,3)(2,4) \\ (3,1)(3,2)(3,3)(3,4)(4,1)(4,2)(4,3)(4,4) \end{array} \right\}$$

$$\{(x,y)/y = x + 1, x, y \in X\}$$

$$\begin{matrix} & 1 & 2 & 3 & 4 \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

$$R \cup S(x, y) = \max (R(x, y), S(x, y))$$

$$R \cap S(x, y) = \min (R(x, y), S(x, y))$$

$$\bar{R}$$

$$\bar{R}(x, y) = 1 - R(x, y)$$

If the universe of discourse or sets are finite, the n -ary relation can be expressed as an

n -dimensional *relation matrix*. Thus, for a binary relation $R(X, Y)$ where $X = \{ x_1, x_2, \dots, x_n \}$ and

$Y = \{ y_1, y_2, \dots, y_m \}$, the relation matrix R is a two dimensional matrix where X represents the rows, Y represents the columns and $R(i, j) = 1$ if (x_i, y_j)

.

Example

Given $X = \{1, 2, 3, 4\}$,

Let the relation R be defined as

$$R =$$

$$R = \{(1, 2)(2, 3)(3, 4)\}$$

The relation matrix R is given by

$$R =$$

6.4.3 Operations on Relations

Given two relations R and S defined on $X \times Y$ and represented by relation matrices, the following operations are supported by R and S

Union: $R \cup S$

(6.47)

Intersection: $R \cap S$

(6.48)

Complement:

(6.49)

Composition of relations: $R \circ S$

$$R \circ S = \{(x, z) / (x, z) \in X \times Z, \exists y \in Y \text{ such that } (x, y) \in R \text{ and } (y, z) \in S\}$$

$$\max_{y \in Y} (\min(R(x, y), S(y, z)))$$

$$R: \begin{matrix} & 1 & 3 & 5 \\ \begin{matrix} 1 \\ 3 \\ 5 \end{matrix} & \left[\begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{matrix} \right] & S: \begin{matrix} & 1 & 3 & 5 \\ \begin{matrix} 1 \\ 3 \\ 5 \end{matrix} & \left[\begin{matrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{matrix} \right] \end{matrix} \end{matrix}$$

$$\begin{matrix} & 1 & 3 & 5 \\ \begin{matrix} 1 \\ 3 \\ 5 \end{matrix} & \left[\begin{matrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix} \right] \end{matrix}$$

Given R to be a relation on X, Y and S to be a relation on Y, Z then $R o S$ is a composition of relation on X, Z defined as

(6.50)

A common form of the composition relation is the *max-min composition*.

Max-min composition:

Given the relation matrices of the relation R and S , the max-min composition is defined as

For $T = R o S$

$$T(x, z) =$$

(6.51)

Example

Let R, S be defined on the sets $\{1, 3, 5\} \times \{1, 3, 5\}$

Let $R : \{(x, y) \mid y = x + 2\}, S : \{(x, y) \mid x < y\}$

$R = \{(1, 3)(3, 5)\}, S = \{(1, 3)(1, 5)(3, 5)\}$

The relation matrices are

Using max-min composition

$$R \circ S =$$

$$\text{since } R \circ S (1, 1) = \max\{\min(0, 0), \min(1, 0), \min(0, 0)\}$$

$$= \max(0, 0, 0) = 0.$$

$$R \circ S (1, 3) = \max\{0, 0, 0\} = 0$$

$$R \circ S (1, 5) = \max\{0, 1, 0\} = 1.$$

Similarly, $R \circ S (3, 1) = 0$.

$$R \circ S (3, 3) = R \circ S (3, 5) = R \circ S (5, 1) = R \circ S (5, 3) = R \circ S (5, 5) =$$

$$0$$

$R \circ S$ from the relation matrix is $\{(1, 5)\}$.

$$\begin{matrix} & 1 & 3 & 5 \\ 1 & \left[\begin{matrix} 0 & 0 & 1 \end{matrix} \right] \\ 3 & \left[\begin{matrix} 0 & 0 & 0 \end{matrix} \right] \\ 5 & \left[\begin{matrix} 0 & 0 & 0 \end{matrix} \right] \end{matrix}$$

Also, $S \circ R =$

$$\bar{A}$$

$$\tilde{B}$$

$$\bar{A}$$

$$\tilde{B}$$

$$\tilde{A} \times \tilde{B}$$

$$\tilde{R}$$

$$\tilde{R} = \tilde{A} \times \tilde{B} \subset X \times Y$$

$$\tilde{R}$$

$$\mu_{\tilde{R}}(x,y) = \mu_{\tilde{A} \times \tilde{B}}(x,y)$$

$$\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y))$$

$$\tilde{A}$$

$$\tilde{B}$$

6.5 FUZZY RELATIONS

Fuzzy relation is a fuzzy set defined on the Cartesian product of crisp sets X_1, X_2, \dots, X_n where the n -tuples (x_1, x_2, \dots, x_n) may have varying degrees of membership within the relation. The membership values indicate the strength of the relation between the tuples.

Example

Let R be the fuzzy relation between two sets X_1 and X_2 where X_1 is the set of diseases and X_2 is the set of symptoms.

$$X_1 = \{\text{typhoid, viral fever, common cold}\}$$

$$X_2 = \{\text{running nose, high temperature, shivering}\}$$

The fuzzy relation R may be defined as

Running nose

High temperature

Shivering

Typhoid

0.1

0.9

0.8

Viral fever

0.2

0.9

0.7

Common cold

0.9

0.4

0.6

6.5.1 Fuzzy Cartesian Product

Let be a fuzzy set defined on the universe X and be a fuzzy set defined on the universe Y , the Cartesian product between the fuzzy sets and

indicated as

and resulting in a fuzzy relation is given by

(6.52)

where has its membership function given by

=

(6.53)

Example

Let $A = \{(x_1, 0.2), (x_2, 0.7), (x_3, 0.4)\}$ and $B = \{(y_1, 0.5), (y_2, 0.6)\}$ be two fuzzy sets defined on the universes of discourse $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2\}$.

\tilde{R}

$\tilde{A} \times \tilde{B}$

$$\tilde{R} = \tilde{A} \times \tilde{B} = \begin{matrix} & y_1 & y_2 \\ x_1 & \begin{bmatrix} 0.2 & 0.2 \end{bmatrix} \\ x_2 & \begin{bmatrix} 0.5 & 0.6 \end{bmatrix} \\ x_3 & \begin{bmatrix} 0.4 & 0.4 \end{bmatrix} \end{matrix}$$

$$\tilde{R}(x_1, y_1) = \min(\mu_{\tilde{A}}(x_1), \mu_{\tilde{B}}(y_1)) = \min(0.2, 0.5) = 0.2$$

$$\tilde{R}(x_1, y_2) = \min(0.2, 0.6) = 0.2$$

$$\tilde{R}(x_2, y_1) = \min(0.7, 0.5) = 0.5$$

$$\tilde{R}(x_2, y_2) = \min(0.7, 0.6) = 0.6$$

$$\tilde{R}(x_3, y_1) = \min(0.4, 0.5) = 0.4$$

$$\tilde{R}(x_3, y_2) = \min(0.4, 0.6) = 0.4$$

\tilde{R}

\tilde{S}

$$\mu_{\tilde{R} \cup \tilde{S}}(x, y) = \max(\mu_{\tilde{R}}(x, y), \mu_{\tilde{S}}(x, y))$$

$$\mu_{\tilde{R} \cap \tilde{S}}(x, y) = \min(\mu_{\tilde{R}}(x, y), \mu_{\tilde{S}}(x, y))$$

$$\mu_{\tilde{R}^c}(x, y) = 1 - \mu_{\tilde{R}}(x, y)$$

\tilde{R}

\tilde{S}

\tilde{R}

\circ

\tilde{S}

$$\mu_{\tilde{R} \circ \tilde{S}}(x, z) = \max_{y \in Y} (\min(\mu_{\tilde{R}}(x, y), \mu_{\tilde{S}}(y, z)))$$

$y 2\}$ respectively. Then the fuzzy relation resulting out of the fuzzy Cartesian product

is given by

since,

6.5.2 Operations on Fuzzy Relations

Let and be fuzzy relations on $X \times Y$.

Union

(6.54)

Intersection

(6.55)

Complement

(6.56)

Composition of relations

The definition is similar to that of crisp relation. Suppose is a fuzzy relation defined on $X \times Y$, and is a fuzzy relation defined on $Y \times Z$, then is a

fuzzy relation on $X \times Z$. The fuzzy max-min composition is defined as

.

(6.57)

Example

$$X = \{x_1, x_2, x_3\} \quad Y = \{y_1, y_2\} \quad Z = \{z_1, z_2, z_3\} \quad (6.58)$$

\tilde{R}

$$\begin{array}{cc} & \begin{matrix} y_1 & y_2 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \left[\begin{matrix} 0.5 & 0.1 \\ 0.2 & 0.9 \\ 0.8 & 0.6 \end{matrix} \right] \end{array}$$

\tilde{S}

$$\begin{array}{ccc} & \begin{matrix} z_1 & z_2 & z_3 \end{matrix} \\ \begin{matrix} y_1 \\ y_2 \end{matrix} & \left[\begin{matrix} 0.6 & 0.4 & 0.7 \\ 0.5 & 0.8 & 0.9 \end{matrix} \right] \end{array}$$

$$R \circ S = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} z_1 & z_2 & z_3 \\ 0.5 & 0.4 & 0.5 \\ 0.5 & 0.8 & 0.9 \\ 0.6 & 0.6 & 0.7 \end{bmatrix}$$

$$\begin{aligned}\mu_{R \circ S}(x_1, z_1) &= \max(\min(0.5, 0.6), \min(0.1, 0.5)) \\ &= \max(0.5, 0.1) \\ &= 0.5\end{aligned}$$

$$\begin{aligned}\mu_{R \circ S}(x_1, z_2) &= \max(\min(0.5, 0.4), \min(0.1, 0.8)) \\ &= \max(0.4, 0.1) \\ &= 0.4\end{aligned}$$

Similarly,

$$\mu_{R \circ S}(x_1, z_3) = \max(0.5, 0.1) = 0.5$$

$$\mu_{R \circ S}(x_2, z_1) = \max(0.2, 0.5) = 0.5$$

$$\mu_{R \circ S}(x_2, z_2) = \max(0.2, 0.8) = 0.8$$

$$\mu_{R \circ S}(x_2, z_3) = \max(0.2, 0.9) = 0.9$$

$$\mu_{R \circ S}(x_3, z_1) = \max(0.6, 0.5) = 0.6$$

$$\mu_{R \circ S}(x_3, z_2) = \max(0.4, 0.6) = 0.6$$

$$\mu_{R \circ S}(x_3, z_3) = \max(0.7, 0.6) = 0.7$$

Let be a fuzzy relation

Let be a fuzzy relation

Then $R \circ S$, by max-min composition yields, **Example 6.7**

Consider a set $P = \{P 1, P 2, P 3, P 4\}$ of four varieties of paddy plants, set $D =$

$\{D 1, D 2, D 3, D 4\}$ of the various diseases affecting the plants and $S = \{S 1, S 2, S 3, S 4\}$ be the common symptoms of the diseases.

\tilde{R}

\tilde{S}

$$\tilde{R} = P \begin{bmatrix} D_1 & D_2 & D_3 & D_4 \\ P_1 & 0.6 & 0.6 & 0.9 & 0.8 \\ P_2 & 0.1 & 0.2 & 0.9 & 0.8 \\ P_3 & 0.9 & 0.3 & 0.4 & 0.8 \\ P_4 & 0.9 & 0.8 & 0.1 & 0.2 \end{bmatrix} \tilde{S} = D \begin{bmatrix} S_1 & S_2 & S_3 & S_4 \\ D_1 & 0.1 & 0.2 & 0.7 & 0.9 \\ D_2 & 1 & 1 & 0.4 & 0.6 \\ D_3 & 0 & 0 & 0.5 & 0.9 \\ D_4 & 0.9 & 1 & 0.8 & 0.2 \end{bmatrix}$$

\circ

$$R \circ S = P \begin{bmatrix} S_1 & S_2 & S_3 & S_4 \\ P_1 & 0.8 & 0.8 & 0.8 & 0.9 \\ P_2 & 0.8 & 0.8 & 0.8 & 0.9 \\ P_3 & 0.8 & 0.8 & 0.8 & 0.9 \\ P_4 & 0.8 & 0.8 & 0.7 & 0.9 \end{bmatrix}$$

Let R be a relation on $P \times D$ and S be a relation on $D \times S$

For,

Obtain the association of the plants with the different symptoms of the diseases using max-min composition.

Solution

To obtain the association of the plants with the symptoms, $R \circ S$ which is a relation on the sets P and S is to be computed.

Using max-min composition,

$$\tilde{A} \cup \tilde{B} = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$$

$$\tilde{A} \cap \tilde{B} = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$$

$$\tilde{A}^c = 1 - \mu_{\tilde{A}}(x)$$

SUMMARY

Fuzzy set theory is an effective tool to tackle the problem of uncertainty.

In crisp logic, an event can take on only two values, either a 1 or 0

depending on whether its occurrence is true or false respectively.

However, in fuzzy logic, the event may take a range of values between 0 and 1.

Crisp sets are fundamental to the study of fuzzy sets. The basic concepts include universal set, membership, cardinality of a set, family of sets, Venn diagrams, null set, singleton set, power set, subset, and super set. The basic operations on crisp sets are union, intersection, complement, and difference. A set of properties are satisfied by crisp sets. Also, the concept of partition and covering result in the two important rules, namely rule of addition and principle of inclusion and exclusion.

Fuzzy sets support a flexible sense of membership and is defined to be the pair $(x, \mu_{\tilde{A}}(x))$ where $\mu_{\tilde{A}}(x)$ could be discrete or could be described by a continuous function. The membership functions could be triangular, trapezoidal, curved or its variations.

The basic fuzzy operations used often are,

Fuzzy sets, similar to crisp sets satisfy properties such as commutativity, associativity, distributivity, De Morgan's laws and so on.

Crisp relations on sets are subsets of the Cartesian product of the given sets. A crisp relation associates the tuples by means of a relation. A Cartesian relation could be represented by a relation matrix.

Fuzzy relations also associate tuples but to a varying degree of membership. Some of the fuzzy relation operations are,

$$\max_{y \in Y} (\min(R(x,y), S(y,z)))$$

$$R \cup S(x, y) = \max (R(x, y), S(x, y)) \quad R \cap S(x, y) = \min (R(x, y), S(x, y))$$
$$R_c(x, y) = 1 - R(x, y) \quad R \circ S(x, y) =$$

(using the max-min composition)

PROGRAMMING ASSIGNMENT

P6.1 (a) Design and implement a fuzzy library FUZZYLIB.H comprising the basic fuzzy set operations such as union, intersection, complement etc.

(b) Also provide routines to implement fuzzy relations and their operations, namely union, intersection, complement, and max-min composition.

Note: Make use of relation matrix representation for the relations.

(c) Define an appropriate fuzzy problem and apply FUZZYLIB.H to solve the problem.

SUGGESTED FURTHER READING

Fuzzy Logic with Engineering Applications (Ross, 1997) is a lucid treatise on fuzzy logic. *Introduction to the Theory of Fuzzy Subsets*, Vol. 1, (Kaufmann, 1975), *Fuzzy Sets and Systems: Theory and Applications* (Dubois and Prade, 1980), *Fuzzy Set Theory and its Applications* (Zimmerman, 1987) and *Fuzzy Mathematical Techniques with Applications* (Kandel, 1986) are some of the early literature in this field. *Fuzzy Sets and Fuzzy Logic* (Klir and Yuan Bo, 1997) provides good material on fuzzy systems and its applications.

REFERENCE

Zadeh, Lotfi A. (1965), *Fuzzy Sets, Inf. Control*, Vol. 8, pp. 338–353.

Chapter 7

Fuzzy Systems

Logic is the science of reasoning. Symbolic or mathematical logic has turned out to be a powerful computational paradigm. Not only does symbolic logic help in the description of events in the real world but has also turned out to be an effective tool for inferring or deducing information from a given set of facts.

Just as mathematical sets have been classified into crisp sets and fuzzy sets (Refer Chapter 6), logic can also be broadly viewed as *crisp logic* and *fuzzy logic*. Just as crisp sets survive on a 2-state membership (0/1) and fuzzy sets on a multistate membership [0–1], crisp logic is built on a 2-state truth value (True/False) and fuzzy logic on a multistate truth value (True/False/very True/partly False and so on.)

We now briefly discuss crisp logic as a prelude to fuzzy logic.

7.1 CRISP LOGIC

Consider the statements “Water boils at 90°C” and “Sky is blue”. An agreement or disagreement with these statements is indicated by a “True” or “False” value accorded to the statements. While the first statement takes on a value *false*, the second takes on a value *true*.

Thus, a statement which is either ‘True’ or ‘False’ but not both is called a *proposition*. A proposition is indicated by upper case letters such as *P, Q, R* and so on.

Example: *P*: Water boils at 90°C.

Q: Sky is blue.

are propositions.

A simple proposition is also known as an *atom*. Propositions alone are insufficient to represent phenomena in the real world. In order to represent complex information, one has to build a sequence of propositions linked using *c onnectives* or *o perators*. Propositional logic recognizes five major operators as shown in Table 7.1.

Table 7.1 Propositional logic connectives

Symbol

Connective

Usage

Description

\wedge

and

$P \wedge Q$

P and Q are true.

\vee

or

$P \vee Q$

Either P or Q is true.

\wedge or \sim

not

$\sim P$ or $\wedge P$

P is not true.

\Rightarrow

implication

$P \Rightarrow Q$

P implies Q is true.

$=$

equality

$P = Q$

P and Q are equal (in truth values) is true.

Observe that \wedge , \vee , \Rightarrow , and $=$ are ‘binary’ operators requiring two propositions while \sim is a ‘unary’ operator requiring a single proposition. \wedge and \vee operations are referred to as *conjunction* and *disjunction* respectively.

In the case of \Rightarrow operator, the proposition occurring before the ‘ \Rightarrow ’ symbol is called as the *antecedent* and the one occurring after is called as the *consequent*.

The semantics or meaning of the logical connectives are explained using a *truth table*. A truth table comprises rows known as *interpretations*, each of which evaluates the logical formula for the given set of truth values. Table 7.2 illustrates the truth table for the five connectives.

Table 7.2 Truth table for the connectives \wedge , \vee , \sim , \Rightarrow , $=$

P

Q

$P \wedge Q$

$P \vee Q$

$\sim P$

$P \Rightarrow Q$

$P = Q$

T

T

T

T

F

T

T

T

F

F

T

F

F

F

F

F

F

F

T

T

T

F

T

F

T

T

T

F

T : True, F : False

A logical formula comprising n propositions will have $2n$ interpretations in its truth table. A formula which has all its interpretations recording true is known as a *tautology* and the one which records false for all its interpretations is known as *contradiction*.

Example 7.1

Obtain a truth table for the formula $(P \vee Q) \Rightarrow (\sim P)$. Is it a tautology?

Solution

The truth table for the given formula is

P

Q

$P \vee Q$

$\sim P$

$P \vee Q \Rightarrow \sim P$

T

F

T

F

F

F

T

T

T

T

T

T

T

F

F

F

F

F

T

T

No, it is not a tautology since all interpretations do not record ‘True’ in its last column.

Example 7.2

Is $((P \Rightarrow Q) \wedge (Q \Rightarrow P)) = (P = Q)$ a tautology?

Solution

A:

B:

P

Q

$P \Rightarrow Q$

$Q \Rightarrow P$

$A = B$

$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$

$P = Q$

T

F

F

T

F

F

T

F

T

T

F

F

F

T

T

T

T

T

T

T

T

F

F

T

T

T

T

T

Yes, the given formula is a tautology.

Example 7.3

Show that $(P \Rightarrow Q) = (\sim P \vee Q)$

Solution

The truth table for the given formula is

P

Q

A: $P \Rightarrow Q$

$\sim P$

B: $\sim P \vee Q$

$A = B$

T

T

T

F

T

T

T

F

F

F

T

F

F

T

T

T

T

T

T

T

T

T

T

T

Since the last column yields ‘True’ for all interpretations, it is a tautology.

The logical formula presented in Example 7.3 is of practical importance since $(P \Rightarrow Q)$ is shown to be equivalent to $(\sim P \vee Q)$, a formula devoid of ' \Rightarrow ' connective. This equivalence can therefore be utilised to eliminate ' \Rightarrow ' in logical formulae.

It is useful to view the ' \Rightarrow ' operator from a set oriented perspective. If X is the universe of discourse and A, B are sets defined in X , then propositions P and Q could be defined based on an element $x \in X$ belonging to A or B . That is,

$$P: x \in A$$

$$Q: x \in B \quad (7.1)$$

Here, P, Q are true if $x \in A$ and $x \in B$ respectively, and $\sim P, \sim Q$ are true if

$$\overline{A}$$

$x \in A$ and $x \in B$ respectively. In such a background, $P \Rightarrow Q$ which is equivalent to $(\sim P \vee Q)$ could be interpreted as $(P \Rightarrow Q) : x \in A \text{ or } x \in B$ (7.2) However, if the ' \Rightarrow ' connective deals with two different universes of discourse, that is,

$A \subset X$ and $B \subset Y$ where X and Y are two universes of discourse then the ' \Rightarrow ' connective is represented by the relation R such that

$R = (A \times B) \cup (\emptyset \times Y)$ (7.3) In such a case, $P \Rightarrow Q$ is linguistically referred to as IF A THEN B . The compound proposition ($P \Rightarrow Q$)

$Q) \vee (\sim P \Rightarrow S)$ linguistically referred to as IF A THEN B ELSE C is equivalent to

IF A THEN B ($P \Rightarrow Q$)

IF $\sim A$ THEN C ($\sim P \Rightarrow S$) (7.4) where P, Q , and S are defined by sets $A, B, C, A \subset X$, and $B, C \subset Y$.

7.1.1 Laws of Propositional Logic

Crisp sets as discussed in Section 6.2.2. exhibit properties which help in their simplification. Similarly, propositional logic also supports the following laws which can be effectively used for their simplification. Given P, Q, R to be the propositions,

(i) *Commutativity*

$$(P \vee Q) = (Q \vee P)$$

$$(P \wedge Q) = (Q \wedge P) \quad (7.5)$$

(ii) *Associativity*

$$(P \vee Q) \vee R = P \vee (Q \vee R) \quad (P \wedge Q) \wedge R = P \wedge (Q \wedge R) \quad (7.6)$$

(iii) *Distributivity*

$$(P \vee Q) \wedge R = (P \wedge R) \vee (Q \wedge R) \quad (P \wedge Q) \vee R = (P \vee R) \wedge (Q \vee R) \quad (7.7)$$

(iv) *Identity*

$$P \vee \text{false} = P$$

$$P \wedge \text{True} = P$$

$$P \wedge \text{False} = \text{False}$$

$$P \vee \text{True} = \text{True} \quad (7.8)$$

(v) *Negation*

$$P \wedge \sim P = \text{False}$$

$$P \vee \sim P = \text{True} \quad (7.9)$$

(vi) *Idempotence*

$$P \vee P = P$$

$$P \wedge P = P \quad (7.10)$$

(vii) *Absorption*

$$P \wedge (P \vee Q) = P$$

$$P \vee (P \wedge Q) = P \quad (7.11)$$

(viii) *De Morgan's laws*

$$\sim(P \vee Q) = (\sim P \wedge \sim Q)$$

$$\sim(P \wedge Q) = (\sim P \vee \sim Q) \quad (7.12) \quad (\text{ix}) \text{ } \textit{Involution}$$

$$\sim(\sim P) = P \quad (7.13)$$

Each of these laws can be tested to be a tautology using truth tables.

Example 7.4

Verify De Morgan's laws.

$$(a) \sim(P \vee Q) = (\sim P \wedge \sim Q)$$

$$(b) \sim(P \wedge Q) = (\sim P \vee \sim Q)$$

Solution

P

Q

$P \vee Q$

$A: \sim(P \vee Q)$

$\sim P$

$\sim Q$

$B: \sim P \wedge \sim Q$

$A = B$

T

T

T

F

F

F

F

T

T

F

T

F

F

T

F

T

F

T

T

F

T

F

F

T

F

F

F

T

T

T

T

T

Therefore, $\sim(P \vee Q) = (\sim P \wedge \sim Q)$

P

$P \wedge Q$

$A: \sim(P \wedge Q)$

$\sim P$

$\sim Q$

$B: \sim P \vee \sim Q$

$A = B$

T

T

T

F

F

F

F

T

T

F

F

T

F

T

T

T

F

T

F

T

T

T

T

F

F

F

T

T

F

T

T

Therefore $\sim(P \wedge Q) = (\sim P \vee \sim Q)$

Example 7.5

Simplify $(\sim(P \wedge Q) \Rightarrow R) \wedge P \wedge Q$

Solution

Consider $(\sim(P \wedge Q) \Rightarrow R) \wedge P \wedge Q$

$$= (\sim(\sim(P \wedge Q) \vee R) \wedge P \wedge Q)$$

(by eliminating ' \Rightarrow ' using $(P \Rightarrow Q) = (\sim P \vee Q)$)

$$= ((P \wedge Q) \vee R) \wedge P \wedge Q \text{ (by the law of involution)}$$

$$= (P \wedge Q) \text{ (by the law of absorption)}$$

7.1.2 Inference in Propositional Logic

Inference is a technique by which, given a set of *facts* or *postulates* or *axioms* or *premises* F_1 ,

F_2, \dots, F_n , a *goal* G is to be derived. For example, from the facts “Where there is smoke there is fire”, and “There is smoke in the hill”, the statement

“Then the hill is on fire” can be easily deduced.

In propositional logic, three rules are widely used for inferring facts, namely

(i) *Modus Ponens*

(ii) *Modus Tollens*, and

(iii) *Chain rule*

Modus ponens (mod pons)

$$\frac{P \Rightarrow Q}{\frac{P}{Q}}$$

$$P \Rightarrow Q$$

$$\frac{\sim Q}{\sim P}$$

$$P \Rightarrow Q$$

$$\frac{Q \Rightarrow R}{P \Rightarrow R}$$

(i)

$$C \vee D$$

(iii)

$$\frac{(C \vee D) \Rightarrow \sim H}{\sim H} \quad (\text{v})$$

Given $P \Rightarrow Q$ and P to be true, Q is true.

(7.14)

Here, the formulae above the line are the *premises* and the one below is the *goal* which can be inferred from the premises.

Modus tollens

Given $P \Rightarrow Q$ and $\sim Q$ to be true, $\sim P$ is true.

(7.15)

Chain rule

Given $P \Rightarrow Q$ and $Q \Rightarrow R$ to be true, $P \Rightarrow R$ is true.

(7.16)

Note that the chain rule is a representation of the *transitivity* relation with respect to the ' \Rightarrow ' connective.

Example 7.6

Given

$$(i) C \vee D$$

$$(ii) \sim H \Rightarrow (A \wedge \sim B)$$

$$(iii) (C \vee D) \Rightarrow \sim H$$

$$(iv) (A \wedge \sim B) \Rightarrow (R \vee S)$$

Can $(R \vee S)$ be inferred from the above?

Solution

From (i) and (iii) using the rule of Modus Ponens , $\sim H$ can be inferred.

From (ii) and (iv) using the chain rule, $\sim H \Rightarrow (R \vee S)$ can be inferred.

$$\begin{array}{c} (ii) \quad \sim H \Rightarrow (A \wedge \sim B) \\ (iv) \quad \frac{(A \wedge \sim B) \Rightarrow (R \vee S)}{\sim H \Rightarrow (R \vee S)} \quad (vi) \\ \\ (vi) \quad \sim H \Rightarrow (R \vee S) \\ (v) \quad \frac{\sim H}{R \vee S} \end{array}$$

From (v) and (vi) using the rule of Modus Ponens $(R \vee S)$ can be inferred.

Hence, the result.

7.2 PREDICATE LOGIC

In propositional logic, events are symbolised as propositions which acquire either ‘True/False’ values. However, there are situations in the real world

where propositional logic falls short of its expectation. For example, consider the following statements:

P : All men are mortal.

Q : Socrates is a man.

From the given statements it is possible to infer that Socrates is mortal.

However, from the propositions P , Q which symbolise these statements nothing can be made out. The reason being, propositional logic lacks the ability to symbolise *quantification*. Thus, in this example, the quantifier “All”

which represents the entire class of men encompasses Socrates as well, who is declared to be a man, in proposition Q . Therefore, by virtue of the first proposition P , Socrates who is a man also becomes a mortal, giving rise to the deduction Socrates is mortal. However, the deduction is not directly perceivable owing to the shortcomings in propositional logic. Therefore, propositional logic needs to be augmented with more tools to enhance its logical abilities.

Predicate logic comprises the following apart from the connectives and propositions recognized by propositional logic.

(i) Constants

(ii) Variables

(ii) Predicates

(iv) Quantifiers

(v) Functions

Constants represent objects that do not change values.

Example Pencil, Ram, Shaft, 100°C.

Variables are symbols which represent values acquired by the objects as qualified by the quantifier with which they are associated with.

Example x, y, z .

Predicates are representative of associations between objects that are constants or variables and acquire truth values ‘True’ or ‘False’. A *predicate* carries a name representing the association followed by its arguments representing the objects it is to associate.

Example

likes (Ram, tea).....(Ram likes tea)

plays (Sita, x).....(Sita plays anything)

Here, likes and plays are predicate names and Ram, tea and Sita, x are the associated objects. Also, the predicates acquire truth values. If Ram disliked tea, likes (Ram, tea) acquires the value *false* and if Sita played any sport, plays (Sita, x) would acquire the value *true* provided x is suitably qualified by a quantifier.

Quantifiers are symbols which indicate the two types of quantification, namely, *All* (\forall) and *Some* (\exists). ‘ \forall ’ is termed *universal quantifier* and ‘ \exists ’ is termed *existential quantifier*.

Example Let,

man (x) : x is a man.

mortal (x) : x is mortal.

mushroom (x) : x is a mushroom.

poisonous (x) : x is poisonous.

Then, the statements

All men are mortal.

Some mushrooms are poisonous.

are represented as

$$\forall x (\text{man} (x) \Rightarrow \text{mortal} (x))$$

$$\exists x (\text{mushroom} (x) \wedge \text{poisonous} (x))$$

Here, a useful rule to follow is that a universal quantifier goes with implication and an existential quantifier with conjunction. Also, it is possible for logical formula to be quantified by multiple quantifiers.

Example Every ship has a captain.

$$\forall x \exists y (\text{ship} (x) \Rightarrow \text{captain} (x, y)) \text{ where, } \text{ship} (x) : x \text{ is a ship}$$

$\text{captain} (x, y) : y \text{ is the captain of } x$.

Functions are similar to predicates in form and in their representation of association between objects but unlike predicates which acquire truth values alone, functions acquire values other than truth values. Thus, functions only serve as object descriptors.

Example

plus (2, 3) (2 plus 3 which is 5)

mother (Krishna) (Krishna's mother)

Observe that plus () and mother () indirectly describe “5” and “Krishna’s mother” respectively.

Example 7.7

Write predicate logic statements for

- (i) Ram likes all kinds of food.
- (ii) Sita likes anything which Ram likes.
- (iii) Raj likes those which Sita and Ram both like.
- (iv) Ali likes some of which Ram likes.

Solution

Let food (x) : x is food.

likes (x, y) : x likes y

Then the above statements are translated as

- (i) $\forall x \text{ food } (x) \Rightarrow \text{likes} (\text{Ram}, x)$
- (ii) $\forall x (\text{likes} (\text{Ram}, x) \Rightarrow \text{likes} (\text{Sita}, x))$ (iii) $\forall x (\text{likes} (\text{Sita}, x) \wedge \text{likes} (\text{Ram}, x)) \Rightarrow \text{likes} (\text{Raj}, x)$ (iv) $\exists x (\text{likes} (\text{Ram}, x) \wedge \text{likes} (\text{Ali}, x))$ The application of the rule of universal quantifier and rule of existential quantifier can be observed in the translations given above.

7.2.1 Interpretations of Predicate Logic Formula

For a formula in propositional logic, depending on the truth values acquired by the propositions, the truth table interprets the formula. But in the case of predicate logic, depending on the truth values acquired by the predicates, the nature of the quantifiers, and the values taken by the constants and functions over a domain D, the formula is interpreted.

Example

Interpret the formulae

$$(i) \forall x p(x)$$

$$(ii) \exists x p(x)$$

where the domain $D = \{1, 2\}$ and

$$p(1) \qquad \qquad p(2)$$

True False

Solution

(i) $\forall x p(x)$ is true only if $p(x)$ is true for all values of x in the domain D , otherwise it is false.

Here, for $x = 1$ and $x = 2$, the two possible values for x chosen from D , namely $p(1) = \text{true}$ and $p(2) = \text{false}$ respectively, yields (i) to be false since $p(x)$ is not true for $x = 2$. Hence, $\forall x p(x)$ is false.

(ii) $\exists x p(x)$ is true only if there is atleast one value of x for which $p(x)$ is true.

Here, for $x = 1$, $p(x)$ is true resulting in (ii) to be true. Hence, $\exists x p(x)$ is true.

Example 7.8

Interpret $\forall x \exists y P(x, y)$ for $D = \{1, 2\}$ and $P(1, 1)$

$$P(1, 2)$$

$$P(2, 1)$$

$$P(2, 2)$$

True

False

False

True

Solution

For $x = 1$, there exists a y , ($y = 1$) for which $P(x, y)$, i.e. ($P(1, 1)$) is true.

For $x = 2$, there exists a y , ($y = 2$) for which $P(x, y)$ ($P(2, 2)$) is true.

Thus, for all values of x there exists a y for which $P(x, y)$ is true.

Hence, $\forall x \exists y P(x, y)$ is true.

7.2.2 Inference in Predicate Logic

The rules of inference such as Modus Ponens, Modus Tollens and Chain rule, and the laws of propositional logic are applicable for inferring predicate logic but not before the quantifiers have been appropriately eliminated (refer Chang & Lee, 1973).

Example

Given (i) All men are mortal.

(ii) Confucius is a man.

Prove: Confucius is mortal.

Translating the above into predicate logic statements

(i) $\forall x (\text{man}(x) \Rightarrow \text{mortal}(x))$

(ii) $\text{man}(\text{Confucius})$

(iii) $\text{mortal}(\text{Confucius})$

Since (i) is a tautology qualified by the universal quantifier for $x = \text{Confucius}$, the statement is true, i.e.

$\text{man}(\text{Confucius}) \Rightarrow \text{mortal}(\text{Confucius})$

$\Rightarrow. \neg\text{man}(\text{Confucius}) \vee \text{mortal}(\text{Confucius})$

But from (ii), man (Confucius) is true.

Hence (iv) simplifies to

False \vee mortal (Confucius)

= mortal (Confucius)

Hence, Confucius is mortal has been proved.

Example 7.9

Given (i) Every soldier is strong-willed.

(ii) All who are strong-willed and sincere will succeed in their career.

(iii) Indira is a soldier.

(iv) Indira is sincere.

Prove: Will Indira succeed in her career?

Solution

Let

soldier (x): x is a soldier.

strong-willed (x): x is a strong-willed.

sincere (x) : x is sincere.

succeed career (x) : x succeeds in career.

Now (i) to (iv) are translated as

$\forall x (\text{soldier} (x) \Rightarrow \text{strong-willed} (x)) \dots \dots \dots \text{(i)}$

$\forall x ((\text{strong-willed}(x) \wedge \text{sincere}(x)) \Rightarrow \text{succeed_career}(x))$

(ii)

soldier (Indira).....(iii)

sincere (Indira).....(iv)

To show whether Indira will succeed in her career, we need to show
succeed_career(Indira) is true.....(v)

Since (i) and (ii) are quantified by \forall , they should be true for $x = \text{Indira}$.

Substituting $x = \text{Indira}$ in (i) results in (soldier (Indira) \Rightarrow strong-willed (Indira)),

i.e. $\sim\text{soldier}(\text{Indira}) \vee \text{strong-willed}(\text{Indira})$(vi)

Since from (iii) soldier (Indira) is true, (vi) simplifies to

strong-willed (Indira).....(vii)

Substituting $x = \text{Indira}$ in (ii),

$(\text{strong-willed}(\text{Indira}) \wedge \text{sincere}(\text{Indira})) \Rightarrow \text{succeed_career}(\text{Indira})$ i.e. $\sim(\text{strong-willed}(\text{Indira}) \wedge \text{sincere}(\text{Indira})) \vee \text{succeed_career}(\text{Indira})$

($\because P \Rightarrow Q = \sim P \vee Q$)

i.e. $\sim(\text{strong-willed}(\text{Indira}) \vee \sim\text{sincere}(\text{Indira})) \vee \text{succeed_career}(\text{Indira})$

(De Morgan's law) (viii)

From (vii), strong-willed (Indira) is true and from (iv) sincere (Indira) is true.

Substituting these in (viii),

False \vee False \vee succeed_career (Indira)

i.e. succeed_career (Indira) (using law of identity)

Hence, Indira will succeed in her career is true.

\tilde{P}

\tilde{P}

\tilde{P}

\tilde{P}

$$T(\tilde{P}) = \mu_{\tilde{A}}(x) \text{ where } 0 \leq \mu_{\tilde{A}}(x) \leq 1$$

\tilde{P}

\tilde{P}

\tilde{P}

\tilde{P}

\tilde{P}

\tilde{P}

\tilde{Q}

\tilde{P}

\tilde{Q}

7.3 FUZZY LOGIC

In crisp logic, the truth values acquired by propositions or predicates are 2-valued, namely *True*, *False* which may be treated numerically equivalent to

(0, 1). However, in fuzzy logic, truth values are multivalued such as *absolutely true, partly true, absolutely false, very true*, and so on and are numerically equivalent to (0–1).

Fuzzy propositions

A *fuzzy proposition* is a statement which acquires a fuzzy truth value. Thus, given to be a fuzzy proposition, $T()$ represents the truth value (0–1) attached to . In its simplest form, fuzzy propositions are associated with fuzzy sets. The fuzzy membership value associated with the fuzzy set \tilde{A} for is treated as the fuzzy truth value $T()$.

i.e.

.....(7.17)

Example

: Ram is honest.

$T() = 0.8$, if is partly true.

$T() = 1$, if is absolutely true.

Fuzzy connectives

Fuzzy logic similar to crisp logic supports the following connectives: (i)
Negation : –

(ii) *Disjunction* : \vee

(iii) *Conjunction* : \wedge

(iv) *Implication* : \Rightarrow

Table 7.3 illustrates the definition of the connectives. Here , are fuzzy propositions and $T(), T()$, are their truth values.

Table 7.3 Fuzzy connectives

Symbol	Connective	Usage	Definition
-	Negation	\bar{P}	$1 - T(\bar{P})$
\vee	Disjunction	$\bar{P} \vee \bar{Q}$	$\max(T(\bar{P}), T(\bar{Q}))$
\wedge	Conjunction	$\bar{P} \wedge \bar{Q}$	$\min(T(\bar{P}), T(\bar{Q}))$
\Rightarrow	Implication	$\bar{P} \Rightarrow \bar{Q}$	$\sim \bar{P} \vee \bar{Q} = \max(1 - T(\bar{P}), T(\bar{Q}))$

 \tilde{P} \tilde{Q} \tilde{A} \tilde{B}

$$\tilde{R} = (\tilde{A} \times \tilde{B}) \cup (\tilde{A} \times \tilde{Y})$$

 \tilde{R}

$$\mu_{\tilde{R}}(x, y) = \max(\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)), 1 - \mu_{\tilde{A}}(x))$$

 \tilde{B} \tilde{C}

$$\tilde{R} = (\tilde{A} \times \tilde{B}) \cup (\tilde{A} \times C)$$

 \tilde{R}

$$\mu_{\tilde{R}}(x, y) = \max(\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)), \min(1 - \mu_{\tilde{A}}(x), \mu_{\tilde{C}}(y)))$$

 \tilde{P}

\tilde{P} \tilde{Q} \tilde{Q} $\bar{\tilde{P}}$ $\bar{\tilde{P}}$ \tilde{P} $\tilde{P} \wedge \tilde{Q}$ $T(\tilde{P} \wedge \tilde{Q})$ \tilde{P} \tilde{Q} $T(\tilde{P} \vee \tilde{Q})$ $T(\tilde{P} \vee \tilde{Q})$ \tilde{P} \tilde{Q}

and related by the ' \Rightarrow ' operator are known as antecedent and consequent respectively. Also, just as in crisp logic, here too, ' \Rightarrow ' represents the IF-THEN statement as

IF x is THEN y is , and is equivalent to

.....(7.18)

The membership function of is given by

.....(7.19)

Also, for the compound implication IF x is \tilde{A} THEN y is ELSE y is the relation R is equivalent to

.....(7.20)

The membership function of is given by

.....(7.21)

Example

: Mary is efficient, $T(\) = 0.8$

: Ram is efficient, $T(\) = 0.65$

(i) : Mary is not efficient.

$$T(\) = 1 - T(\) = 1 - 0.8 = 0.2$$

(ii)

: Mary is efficient and so is Ram.

$$= \min (T(\), T(\))$$

$$= \min (0.8 , 0.65)$$

$$= 0.65$$

(iii)

: Either Mary or Ram is efficient.

$$= \max\left(\mathcal{T}(\mathbf{\Lambda}),\mathcal{T}(\mathbf{\Lambda})\right)$$

$$= \max\left(0.8,0.65\right)$$

$$\tilde{P}\Rightarrow \tilde{Q}$$

$$\mathcal{T}(\tilde{P}\Rightarrow \tilde{Q})$$

$$\tilde P$$

$$\tilde Q$$

$$\tilde{B}$$

$$\tilde C$$

$$\tilde{B}$$

$$\tilde{B}$$

$$\tilde C$$

$$\tilde R$$

$$(\tilde A\times \tilde B)\cup (\overline{\tilde A}\times Y)$$

$$\mu_{\tilde{R}}(x,y)$$

$$\max\left(\min\left(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(y)\right),1-\mu_{\tilde{A}}(x)\right)$$

$$\tilde{A}\times \tilde{B}$$

$$\begin{array}{cccc}
 1 & 2 & 3 & 4 \\
 \hline
 a & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\
 b & \begin{bmatrix} 0.2 & 0.8 & 0.8 & 0 \end{bmatrix} \\
 c & \begin{bmatrix} 0.2 & 0.6 & 0.6 & 0 \end{bmatrix} \\
 d & \begin{bmatrix} 0.2 & 1 & 0.8 & 0 \end{bmatrix}
 \end{array}$$

$$\bar{\bar{A}} \times Y$$

$$\begin{array}{cccc}
 1 & 2 & 3 & 4 \\
 \hline
 a & \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \\
 b & \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix} \\
 c & \begin{bmatrix} 0.4 & 0.4 & 0.4 & 0.4 \end{bmatrix} \\
 d & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}
 \end{array}$$

$$= 0.8$$

(iv)

: If Mary is efficient then so is Ram.

$$= \max(1 - T(\), T(\))$$

$$= \max(0.2, 0.65)$$

$$= 0.65$$

Example 7.10

Let $X = \{a, b, c, d\}$ $Y = \{1, 2, 3, 4\}$

and

$$\tilde{A} = \{(a, 0)(b, 0.8)(c, 0.6)(d, 1)\}$$

$$= \{(1, 0.2)(2, 1)(3, 0.8)(4, 0)\}$$

$$= \{(1, 0)(2, 0.4)(3, 1)(4, 0.8)\}$$

Determine the implication relations

(i) IF x is \tilde{A} THEN y is .

(ii) IF x is \tilde{A} THEN y is ELSE y is .

Solution

To determine (i) compute

=

where

=

=

=

Here, Y the universe of discourse could be viewed as $\{(1, 1)(2, 1)(3, 1)(4, 1)\}$ a fuzzy set all of whose elements x have $\mu(x) = 1$.

Therefore,

\tilde{R}

$$\begin{array}{cccc} & 1 & 2 & 3 & 4 \\ a & \left[\begin{matrix} 1 & 1 & 1 & 1 \\ 0.2 & 0.8 & 0.8 & 0.2 \end{matrix} \right] \\ b & \left[\begin{matrix} 0.4 & 0.6 & 0.6 & 0.4 \end{matrix} \right] \\ c & \left[\begin{matrix} 0.2 & 0.1 & 0.8 & 0 \end{matrix} \right] \end{array}$$

$$\tilde{B}$$

$$\tilde R$$

$$(\tilde A\times \tilde B)\cup (\overline{\tilde A}\times \tilde C)$$

$$\mu_{\tilde R}(x,y)$$

$$\max\left(\min\left(\mu_{\tilde A}(x),\mu_{\tilde B}(y)\right),\,\min\left(1-\mu_{\tilde A}(x),\mu_{\tilde C}(y)\right)\right)$$

$$\tilde A\times \tilde B$$

$$\begin{array}{cccc}1&2&3&4\\a\left[\begin{matrix}0&0&0&0\\b&0.2&0.8&0.8&0\\c&0.2&0.6&0.6&0\\d&0.2&1&0.8&0\end{matrix}\right]\end{array}$$

$$\overline{\tilde A}\times \bar C$$

$$\begin{array}{cccc}1&2&3&4\\a\left[\begin{matrix}0&0.4&1&0.8\\b&0&0.2&0.2&0.2\\c&0&0.4&0.4&0.4\\d&0&0&0&0\end{matrix}\right]\end{array}$$

$$\tilde R$$

$$((\tilde A\times \tilde B),(\overline{\tilde A}\times \tilde C))$$

\tilde{R}

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline a & \left[\begin{matrix} 0 & 0.4 & 1 & 0.8 \\ 0.2 & 0.8 & 0.8 & 0.2 \end{matrix} \right] \\ b & \left[\begin{matrix} 0.2 & 0.6 & 0.6 & 0.4 \\ 0.2 & 1 & 0.8 & 0 \end{matrix} \right] \\ c & \\ d & \end{array}$$

\tilde{R}

\tilde{B}

\tilde{C}

=

which represents IF x is \tilde{A} THEN y is .

To determine (ii) compute

=

where

=

=

=

Therefore,

= max

gives

=

The above represents IF x is \tilde{A} THEN y is \tilde{B} ELSE y is .

7.3.1 Fuzzy Quantifiers

Just as in crisp logic where predicates are quantified by quantifiers, fuzzy logic propositions are also quantified by *fuzzy quantifiers*. There are two classes of fuzzy quantifiers such as

(i) Absolute quantifiers and

(ii) Relative quantifiers

$$\text{IF } x \text{ is } \tilde{A} \text{ THEN } y \text{ is } \tilde{B}$$
$$\frac{x \text{ is } \tilde{A}'}{y \text{ is } \tilde{B}'} \quad (7.22)$$

\tilde{B}

\tilde{A}'

\tilde{B}'

\tilde{R}

$$\tilde{B}' = \tilde{A}' \circ \tilde{R}(x, y)$$

$$\mu_{\tilde{B}'}(y) = \max(\min(\mu_{\tilde{A}'}(x), \mu_{\tilde{R}}(x, y)))$$

$$\mu_{\tilde{A}'}(x)$$

\tilde{A}'

$$\mu_{\tilde{R}}(x, y)$$

$$\mu_{\tilde{B}'}(y)$$

$$\tilde{B}'$$

While absolute quantifiers are defined over R, relative quantifiers are defined over [0–1].

Example

Absolute quantifier

Relative quantifier

round about 250

almost

much greater than 6

about

some where around 20

most

7.3.2 Fuzzy Inference

Fuzzy inference also referred to as *approximate reasoning* refers to computational procedures used for evaluating linguistic descriptions. The two important inferring procedures are

- (i) Generalized Modus Ponens (GMP)
- (ii) Generalized Modus Tollens (GMT)

GMP is formally stated as

Here, \tilde{A} , ,

and

are fuzzy terms. Every fuzzy linguistic statement

above the line is analytically known and what is below is analytically unknown.

To compute the membership function of

, the max-min composition of

fuzzy set A' with (x, y) which is the known implication relation (IF-THEN relation) is used. That is,

(7.23)

In terms of membership function,

(7.24)

where

is the membership function of

,

is the membership

function of the implication relation and

is the membership function of

.

IF x is \tilde{A} THEN y is \tilde{B}

$$\frac{y \text{ is } \tilde{B}'}{x \text{ is } \tilde{A}'}$$

\tilde{A}'

$$\tilde{B}' = \tilde{A}' \circ \tilde{R}(x, y)$$

$$\mu_{\tilde{B}'}(y) = \max(\min(\mu_{\tilde{A}'}(x), \mu_{\tilde{R}}(x, y)))$$

\bar{H}

$\tilde{V}\bar{H}$

\tilde{S}

$\tilde{\underline{Q}}S$

\bar{H}

$\tilde{V}\bar{H}$

$\tilde{\underline{Q}}S$

\tilde{S}

\tilde{R}

\tilde{R}

$$\max(\tilde{H} \times \tilde{S}, \bar{\tilde{H}} \times Y)$$

$$\tilde{H} \times \tilde{S}$$

	10	20	30	40	50	60
30	0	0	0	0	0	0
40	0	0	0	0	0	0
50	0	0	0	0	0	0
60	0	0	0	0	0	0
70	0	0.8	0.8	1	0.6	0
80	0	0	0.8	1	0.6	0
90	0	0.3	0.3	0.3	0.3	0
100	0	0	0	0	0	0

On the other hand, GMT has the form

The membership of

is computed on similar lines as

In terms of membership function,

.....(7.25)

Example

Apply the fuzzy Modus Ponens rule to deduce Rotation is quite slow given

(i) If the temperature is high then the rotation is slow.

(ii) The temperature is very high.

Let (High),

(Very High), (Slow) and
(Quite Slow) indicate the
associated fuzzy sets as follows:

For $X = \{30, 40, 50, 60, 70, 80, 90, 100\}$, the set of temperatures and $Y = \{10, 20, 30, 40, 50, 60\}$, the set of rotations per minute,

$$= \{(70, 1) (80, 1) (90, 0.3)\}$$

$$= \{(90, 0.9) (100, 1)\}$$

$$= \{(10, 1) (20, 0.8)\}$$

$$= \{(30, 0.8) (40, 1) (50, 0.6)\}$$

To derive

(x, y) representing the implication relation (i), we need to compute

$$(x, y) =$$

=

$$\bar{\tilde{H}} \times Y$$

10 20 30 40 50 60

$$\begin{array}{c} 30 \\ 40 \\ 50 \\ 60 \\ 70 \\ 80 \\ 90 \\ 100 \end{array} \left[\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right]$$

$\tilde{R}(x, y)$

10 20 30 40 50 60

$$\begin{array}{c} 30 \\ 40 \\ 50 \\ 60 \\ 70 \\ 80 \\ 90 \\ 100 \end{array} \left[\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0.8 & 0.8 & 1 & 0.6 & 0 \\ 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right]$$

\tilde{Q}_S

$V\tilde{H} \circ \tilde{R}(x, y)$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0.8 & 0.8 & 1 & 0.6 & 0 \\ 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

=

=

To deduce Rotation is quite slow we make use of the composition rule

=

= [0 0 0 0 0 0 0.9 1] ×

= [1 1 1 1 1 1]

(x_1 is \tilde{A}_1, x_2 is \tilde{A}_2, \dots, x_n is \tilde{A}_n)

(y_1 is \tilde{B}_1, y_2 is \tilde{B}_2, \dots, y_n is \tilde{B}_n)

$\mu_C(y)$

$\min(\mu_{C_1}(y), \mu_{C_2}(y), \dots, \mu_{C_n}(y)), \forall y \in Y$

$\mu_C(y) = \max(\mu_{C_1}(y), \mu_{C_2}(y), \dots, \mu_{C_n}(y)), \forall y \in Y$

7.4 FUZZY RULE BASED SYSTEM

Fuzzy linguistic descriptions are formal representations of systems made through fuzzy IF-THEN rules. They encode knowledge about a system in statements of the form—

IF (a set of conditions) are satisfied THEN (a set of consequents) can be inferred.

Fuzzy IF-THEN rules are coded in the form—

IF

THEN

.

where linguistic variables x_i, y_j take the values of fuzzy sets A_i and B_j respectively.

Example

If there is heavy rain and strong winds

then there must be severe flood warning.

Here, heavy, strong, and severe are fuzzy sets qualifying the variables rain, wind, and flood warning respectively.

A collection of rules referring to a particular system is known as a *fuzzy rule base*. If the conclusion C to be drawn from a rule base R is the conjunction of all the individual consequents C_i of each rule, then $C = C_1 \cap C_2 \cap \dots \cap C_n$ (7.26) where

=

(7.27)

where Y is the universe of discourse.

On the other hand, if the conclusion C to be drawn from a rule base R is the disjunction of the individual consequents of each rule, then

$$C = C_1 \cup C_2 \cup C_3 \dots \cup C_n \quad (7.28) \text{ where}$$

(7.29)

$$\frac{\int \mu(x) x d x}{\int \mu(x) d x}$$

$$\frac{\sum_{i=1}^n x_i \cdot \mu(x_i)}{\sum_{i=1}^n \mu(x_i)}$$

$$\frac{\sum_{i=1}^N x_i \cdot \sum_{k=1}^n \mu_{\tilde{A}_k}(x_i)}{\sum_{i=1}^N \sum_{k=1}^n \mu_{\tilde{A}_k}(x_i)}$$

7.5 DEFUZZIFICATION

In many situations, for a system whose output is fuzzy, it is easier to take a crisp decision if the output is represented as a single scalar quantity. This conversion of a fuzzy set to single crisp value is called *defuzzification* and is the reverse process of *fuzzification*.

Several methods are available in the literature (Hellendoorn and Thomas, 1993) of which we illustrate a few of the widely used methods, namely *centroid method*, *centre of sums*, and *mean of maxima*.

Centroid method

Also known as the *centre of gravity* or the *centre of area* method, it obtains the centre of area (x^*) occupied by the fuzzy set. It is given by the expression

$$x^* =$$

(7.30)

for a continuous membership function, and

$$x^* =$$

(7.31)

for a discrete membership function.

Here, n represents the number of elements in the sample , xi 's are the elements, and $\mu (xi)$ is its membership function.

Centre of sums (COS) method

In the centroid method, the overlapping area is counted once whereas in *centre of sums*, the overlapping area is counted twice. COS builds the resultant membership function by taking the algebraic sum of outputs from each of the contributing fuzzy sets $\tilde{A} 1$, $\tilde{A} 2$, ..., etc. The defuzzified value x^* is given by

$$x^* =$$

(7.32)

$$\frac{\sum_{x_i \in M} x_i}{|M|}$$

COS is actually the most commonly used defuzzification method. It can be implemented easily and leads to rather fast inference cycles.

Mean of maxima (MOM) defuzzification

One simple way of defuzzifying the output is to take the crisp value with the highest degree of membership. In cases with more than one element having the maximum value, the mean value of the maxima is taken. The equation of the defuzzified value x^* is given by

$$x^* =$$

(7.33)

where $M = \{ x_i | \mu(x_i) \text{ is equal to the height of fuzzy set} \}$

$|M|$ is the cardinality of the set M . In the continuous case, M could be defined as

$$M = \{ x \in [-c, c] | \mu(x) \text{ is equal to the height of the fuzzy set} \} \quad (7.34)$$

In such a case, the *mean of maxima* is the arithmetic average of mean values of all intervals contained in M including zero length intervals.

The *height* of a fuzzy set A , i.e. $h(A)$ is the largest membership grade obtained by any element in that set.

Example

\tilde{A}_1 , \tilde{A}_2 , and \tilde{A}_3 are three fuzzy sets as shown in Fig. 7.1(a), (b), and (c).

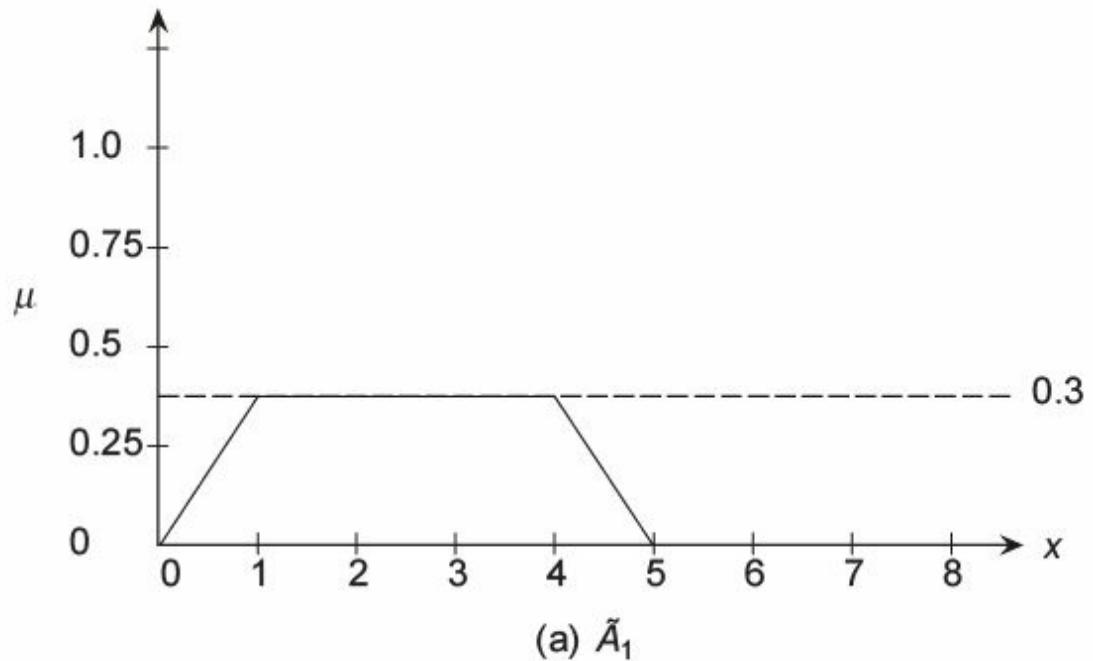
Figure 7.2 illustrates the aggregate of the fuzzy sets.

The defuzzification using (i) centroid method, (ii) centre of sums method, and (iii) mean of maxima method is illustrated as follows.

Centroid method

To compute x^* , the centroid, we view the aggregated fuzzy sets as shown in Figs. 7.2 and 7.3. Note that in Fig. 7.3 the aggregated output has been divided

into areas for better understanding.



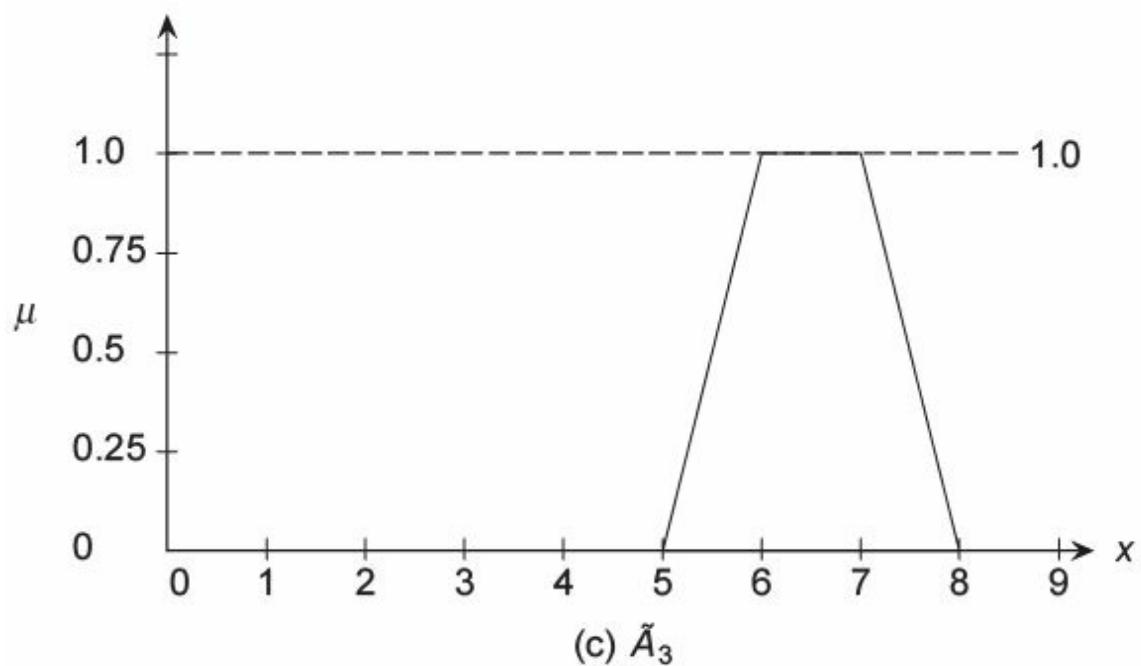
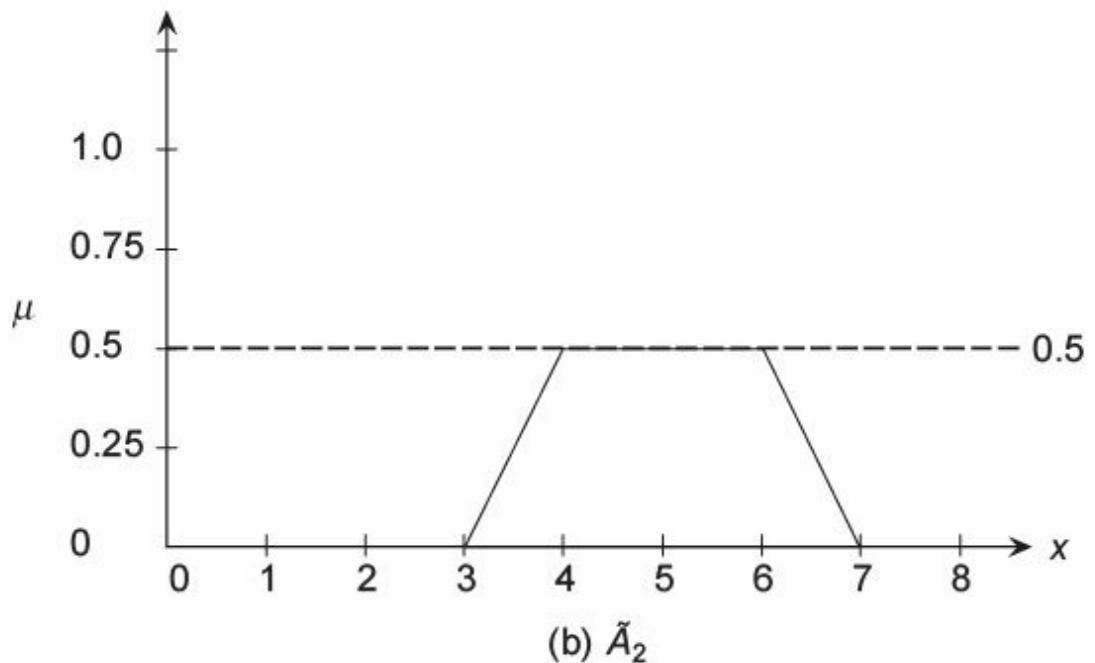


Fig. 7.1 Fuzzy sets \tilde{A}_1 , \tilde{A}_2 , \tilde{A}_3 .

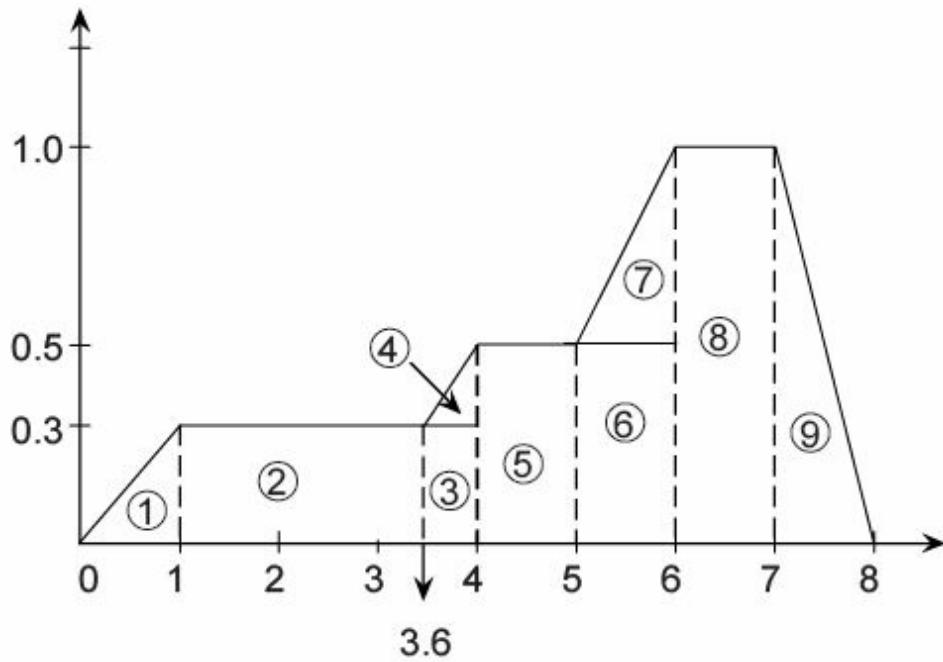
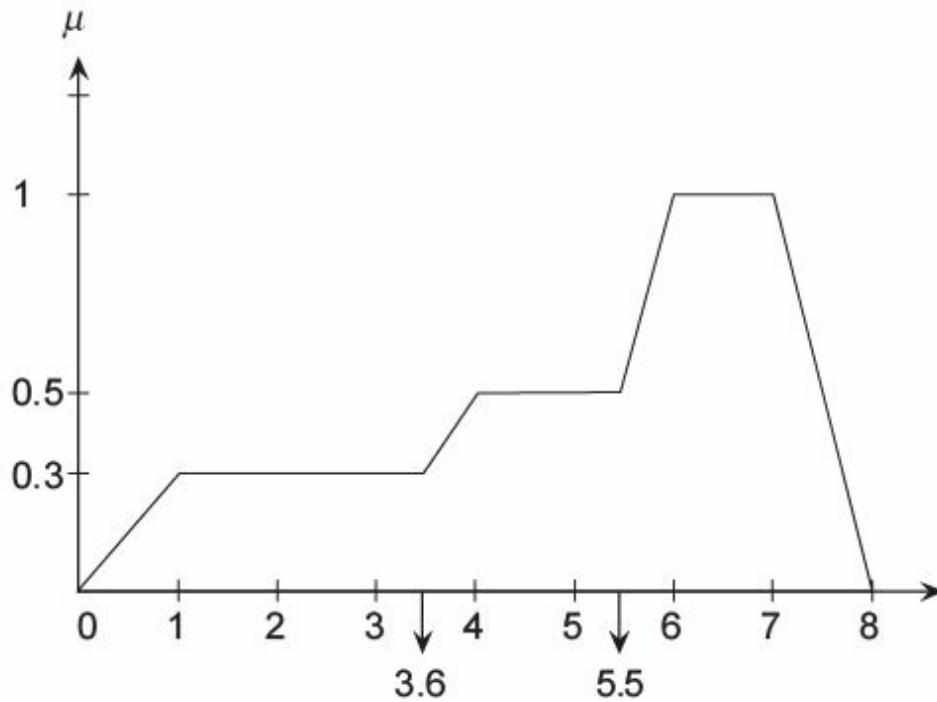


Fig. 7.2 Aggregated fuzzy set of \tilde{A}_1 , \tilde{A}_2 , and \tilde{A}_3 .

Fig. 7.3 Aggregated fuzzy set of \tilde{A}_1 , \tilde{A}_2 , and \tilde{A}_3 viewed as area segments.

Table 7.4 illustrates the computations for obtaining x^* .

Table 7.4 Computation of x^*

Area segment no.	Area (A)	\bar{x}	$A\bar{x}$
1	$\frac{1}{2} \times 0.3 \times 1 = 0.15$	0.67	0.1005
2	$2.6 \times 0.3 = 0.78$	2.3	1.748
3	$0.3 \times 0.4 = 0.12$	3.8	0.456
4	$\frac{1}{2} \times 0.4 \times 0.2 = 0.04$	3.8667	0.1546
5	$1.5 \times 0.5 = 0.75$	4.75	3.5625
6	$1.5 \times 0.5 = 0.75$	5.75	1.4375
7	$\frac{1}{2} \times 0.5 \times 0.5 = 0.125$	5.833	0.729
8	$1 \times 1 = 1$	6.5	6.5
9	$\frac{1}{2} \times 1 \times 1 = 0.5$	7.33	3.665

$$\bar{x}$$

$$\frac{\sum A\bar{x}}{\sum \bar{x}}$$

$$x^* = \frac{\frac{1}{2} \times 0.3 \times (3+5) \times 2.5 + \frac{1}{2} \times 0.5 \times (4+2) \times 5 + \frac{1}{2} \times 1 \times (3+1) \times 6.5}{\frac{1}{2} \times 0.3 \times (3+5) + \frac{1}{2} \times 0.5 \times (4+2) + \frac{1}{2} \times 1 \times (3+5)} \\ = 2.3$$

$$\frac{1}{2} \times 0.3 \times (3+5),$$

$$\frac{1}{2} \times 0.5 \times (4+2)$$

$$\frac{1}{2} \times 1 \times (3+1)$$

In Table 7.4, Area (A) shows the area of the segments of the aggregated fuzzy set and shows the corresponding centroid. Now,

$$x^* =$$

$$\text{i.e. } x^* = 18.353/3.695$$

$$= 4.9$$

Centre of sums method

Here, unlike centroid method the overlapping area is counted not once but twice. Making use of the aggregated fuzzy set shown in Fig.7.2, the centre of sums, x^* is given by

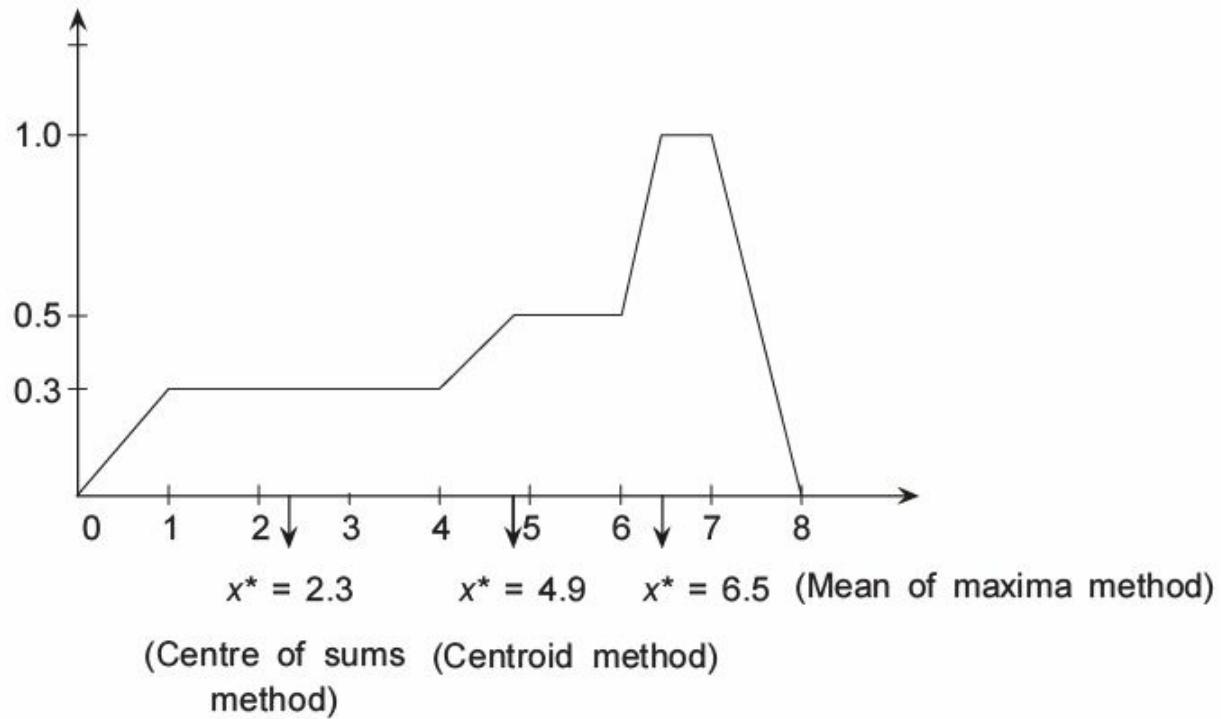
Here, the areas covered by the fuzzy sets $\tilde{A}_1, \tilde{A}_2, \tilde{A}_3$ (Refer Figs. 7.1(a), (b), and (c)) are given by

, and

respectively.

Mean of maxima method

Since the aggregated fuzzy set shown in Fig. 7.2 is a continuous set, x^* the

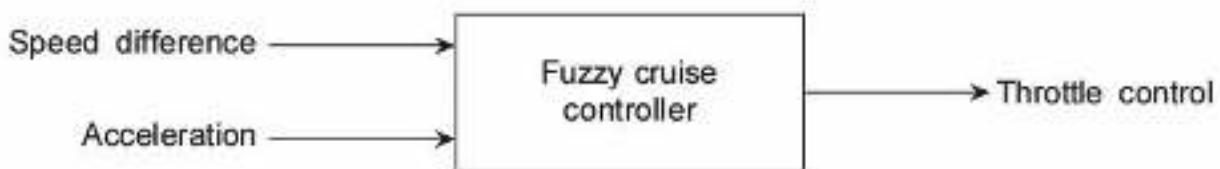


mean of maxima is computed as $x^* = 6.5$.

Here, $M = \{ X \in [6, 7] \mid \mu(x) = 1 \}$ and the height of the aggregated fuzzy set is 1.

Figure 7.4 shows the defuzzified outputs using the above three methods.

Fig. 7.4 Defuzzified outputs of the aggregate of \tilde{A}_1 , \tilde{A}_2 , and \tilde{A}_3 .



7.6 APPLICATIONS

In this section we illustrate two examples of Fuzzy systems, namely (i) *Greg Viot's (Greg Viot, 1993) Fuzzy Cruise Control System* (ii) *Yamakawa's (Yamakawa, 1993) Air Conditioner Controller*

7.6.1 Greg Viot's Fuzzy Cruise Controller

This controller is used to maintain a vehicle at a desired speed. The system consists of two fuzzy inputs, namely speed difference and acceleration, and one fuzzy output, namely throttle control as illustrated in Fig. 7.5.

Fig. 7.5 Fuzzy cruise controller.

Fuzzy rule base

A sample fuzzy rule base R governing the cruise control is as given in Table 7.5.

Table 7.5 Sample cruise control rule base

Rule 1

If (speed difference is NL) and (acceleration is ZE) then (throttle control is PL).

Rule 2

If (speed difference is ZE) and (acceleration is NL) then (throttle control is PL).

Rule 3

If (speed difference is NM) and (acceleration is ZE) then (throttle control is PM).

Rule 4

If (speed difference is NS) and (acceleration is PS) then (throttle control is PS).

Rule 5

If (speed difference is PS) and (acceleration is NS) then (throttle control is NS).

Rule 6

If (speed difference is PL) and (acceleration is ZE) then (throttle control is NL).

Rule 7

If (speed difference is ZE) and (acceleration is NS) then (throttle control is PS).

Rule 8

If (speed difference is ZE) and (acceleration is NM) then (throttle control is PM).

Key

NL – Negative Large

PM – Positive Medium

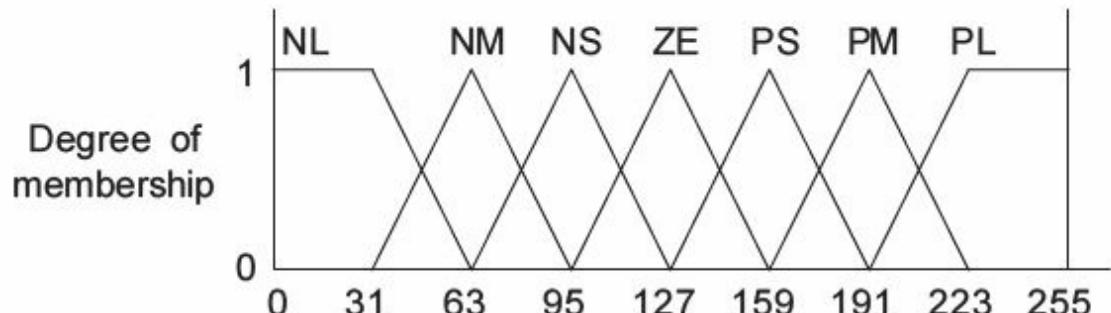
ZE – Zero

NS – Negative Small

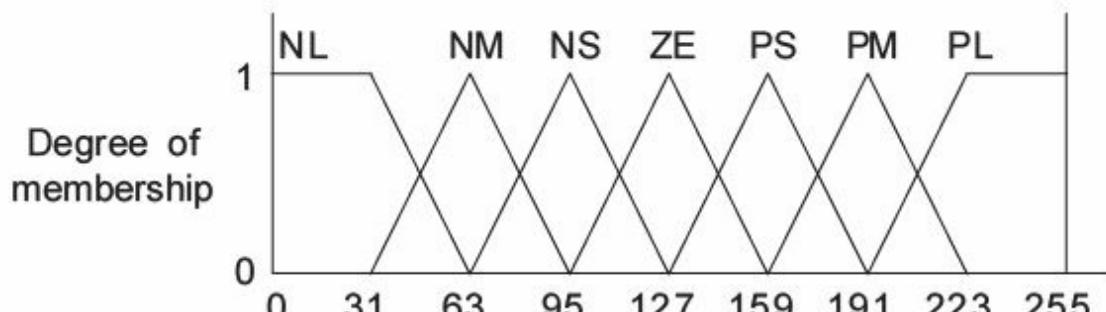
PL – Positive Large

PS – Positive Small

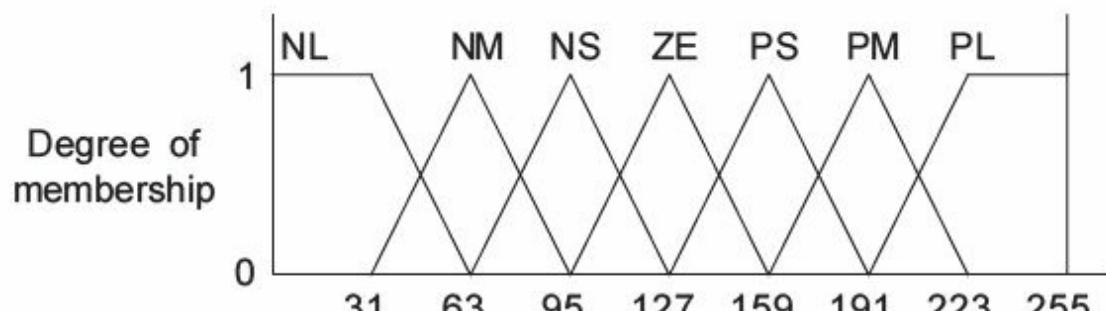
NM – Negative Medium



(a) Speed difference (normalized)



(b) Acceleration (normalized)



(c) Throttle control (normalized)

Fuzzy sets

The fuzzy sets which characterize the inputs and output are as given in Fig. 7.6.

Fig. 7.6 Fuzzy sets characterising fuzzy cruise control.

Fuzzification of inputs

For the *fuzzification* of inputs, that is, to compute the membership for the antecedents, the formula illustrated in Fig. 7.7 is used.

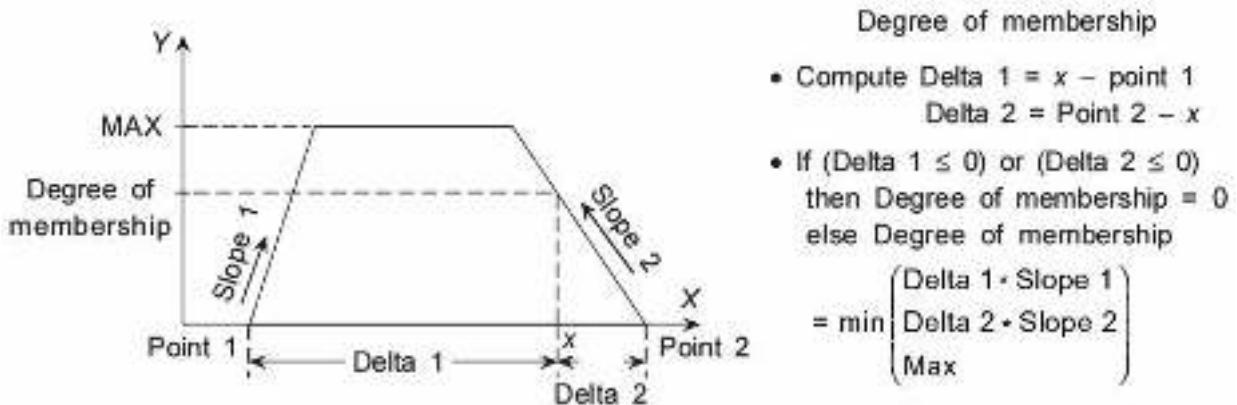


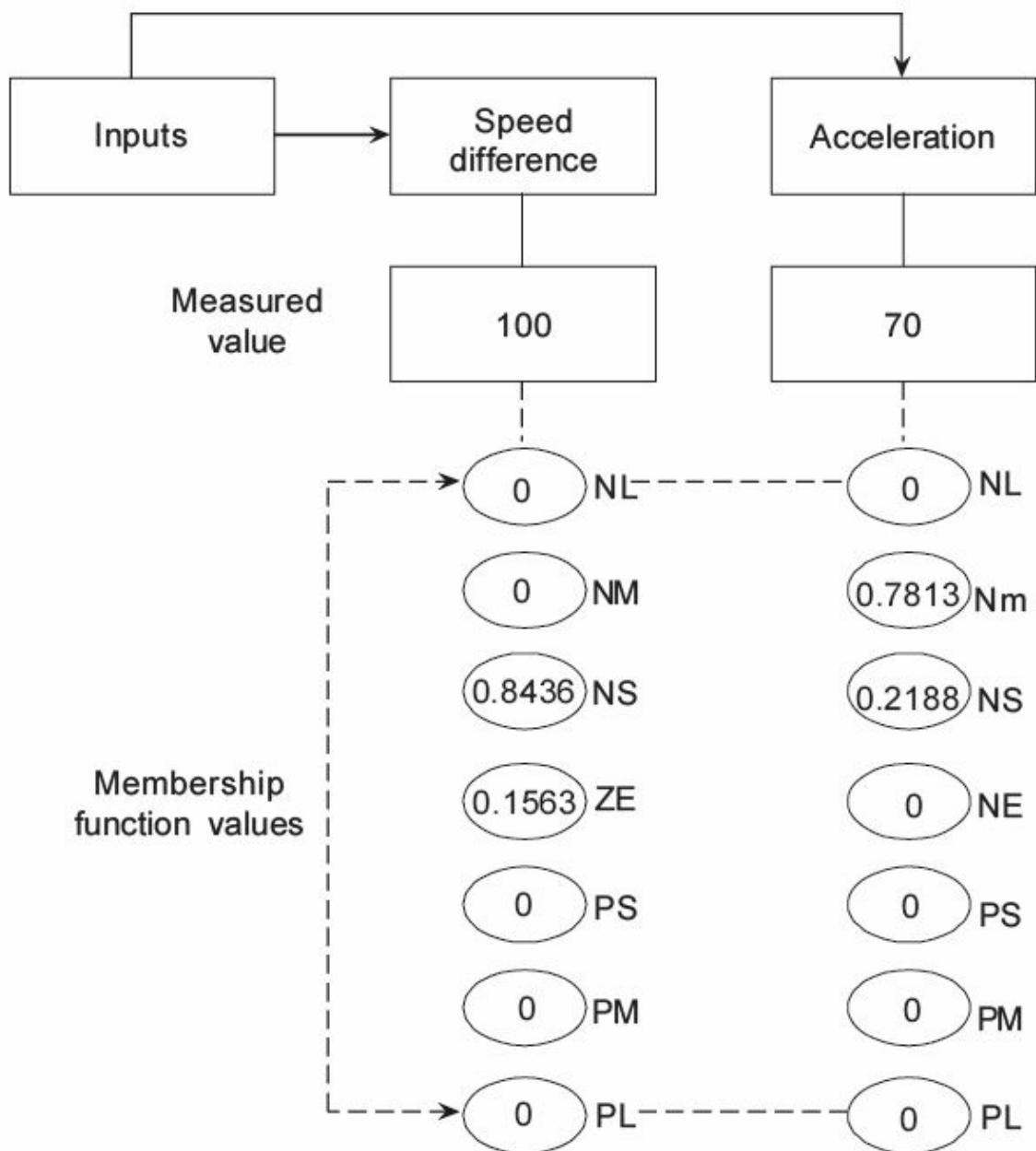
Fig. 7.7 Computation of fuzzy membership value.

Here, x which is the system input has its membership function values computed for all fuzzy sets. For example, the system input speed difference deals with 7 fuzzy sets, namely NL, NM, NS, ZE, PS, PM, and PL. For a measured value of the speed difference x' , the membership function of x' in each of the seven sets is computed using the formula shown in Fig. 7.7. Let $\mu_1', \mu_2', \dots, \mu_7'$ be the seven membership values. Then, all these values are recorded for the input x' in an appropriate data structure.

Similarly, for each of the other system inputs (acceleration in this case), the fuzzy membership function values are recorded.

Example

Let the measured normalized speed difference be 100 and the normalized acceleration be 70, then the fuzzified inputs after computation of the fuzzy membership values are shown in Fig. 7.8.



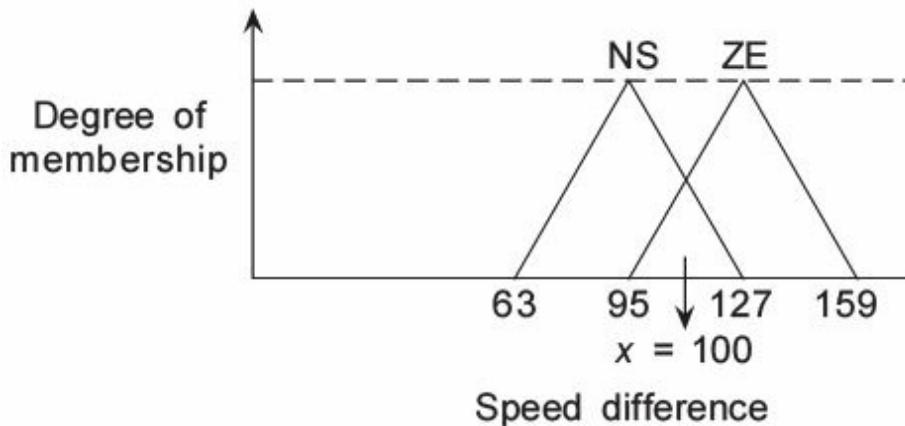


Fig. 7.8 Fuzzy membership values for speed difference = 100 and acceleration = 70.

The computations of the fuzzy membership values for the given inputs have been shown in

Fig. 7.9.

Fig. 7.9 Fuzzification of speed difference = 100.

For speed difference ($x = 100$), the qualifying fuzzy sets are as shown in

$$\begin{pmatrix} 37 \times 0.03125 \\ 27 \times 0.03125 \\ 1 \end{pmatrix}$$

$$\frac{1}{32}$$

$$\begin{pmatrix} 5 \times 0.03125 \\ 59 \times 0.03125 \\ 1 \end{pmatrix}$$

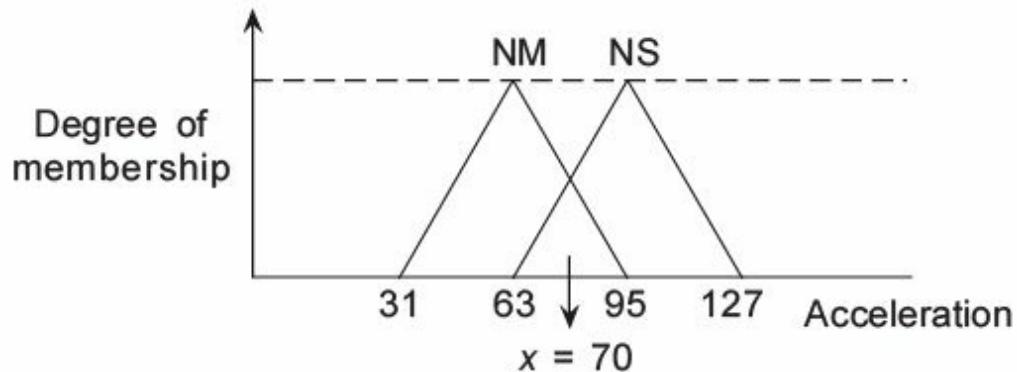


Fig. 7.9.

Fuzzy membership function of x for NS where

$$\text{Delta } 1 = 100 - 63 = 37$$

$$\text{Delta } 2 = 127 - 100 = 27$$

$$\text{Slope } 1 = 1/32 = 0.03125$$

$$\text{Slope } 2 = 1/32 = 0.03125$$

Degree of membership function

$$\mu_{NS}(x) = \min$$

$$= 0.8438$$

Fuzzy membership function of x for ZE where

$$\text{Delta } 1 = 100 - 95 = 5$$

$$\text{Delta } 2 = 159 - 100 = 59$$

$$\text{Slope } 1 = 0.03125$$

$$\text{Slope } 2 = 0.03125$$

Degree of membership function

$$\begin{aligned}\mu ZE(x) &= \min \\ &= 0.1563\end{aligned}$$

The membership function of x with the remaining fuzzy sets, namely NL, NM, PS, PM, PL is zero.

Similarly for acceleration ($x = 70$), the qualifying fuzzy sets are as shown in Fig. 7.10.

Fig. 7.10 Fuzzification of acceleration = 70.

The fuzzy membership function of $x = 70$ for NM is $\mu NM(x) = 0.7813$ and for NS is

$$\mu NS(x) = 0.2188.$$

Rule strength computation

The *rule strengths* are obtained by computing the minimum of the membership functions of the antecedents.

Example

For the sample rule base R given in Table 7.5, the rule strengths using the fuzzy membership values illustrated in Fig. 7.8 are

$$\text{Rule 1: } \min(0, 0) = 0$$

$$\text{Rule 2: } \min(0.1563, 0) = 0$$

$$\text{Rule 3: } \min(0, 0) = 0$$

$$\text{Rule 4: } \min(0.8438, 0) = 0$$

$$\text{Rule 5: } \min(0, 0.2188) = 0$$

$$\text{Rule 6: } \min(0, 0) = 0$$

Rule 7: $\min(0.1563, 0.2188) = 0.1563$

Rule 8: $\min(0.1563, 0.7813) = 0.1563$

Fuzzy output

The *fuzzy output* of the system is the ‘fuzzy OR’ of all the fuzzy outputs of the rules with

non-zero rule strengths. In the event of more than one rule qualifying for the same fuzzy output, the stronger among them is chosen.

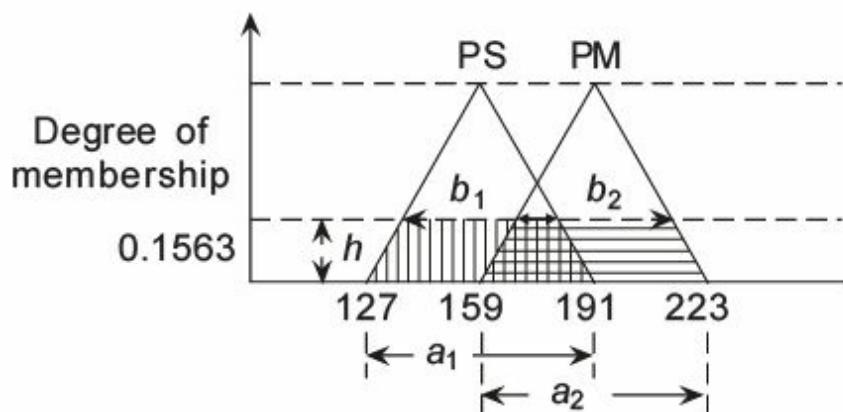
Example

In the given rule base R , the competing fuzzy outputs are those of Rules 7 and 8 with strengths of 0.1563 each.

However, the fuzzy outputs computed here do not aid a clear-cut decision on the throttle control. Hence, the need for defuzzification arises.

Defuzzification

The centre of gravity method is applied to defuzzify the output. Initially, the centroids are computed for each of the competing output membership functions. Then, the new output membership areas are determined by



$$\frac{1}{2} h \cdot (a_1 + b_1)$$

$$\frac{1}{2} (0.1563)(64 + 63.82)$$

$$\frac{1}{2} h \cdot (a_1 + b_1)$$

$$\frac{1}{2} (0.1563)(64 + 63.82)$$

$$\frac{9.99 \times 159 + 9.99 \times 191}{19.98}$$

shortening the height of the membership value on the Y axis as dictated by the rule strength value. Finally, the Centre of Gravity (CG) is computed using the weighted average of the X -axis centroid points with the newly computed output areas, the latter serving as weights.

Example

Figure 7.11 illustrates the computation of CG for the two competing outputs of rules 7 and 8 with strength of 0.1563 each.

Fig. 7.11 Computation of CG for fuzzy cruise control system.

For the fuzzy set PS,

X -axis centroid point = 159

Rule strength applied to determine output area = 0.1563

Shaded area =

=

$$= 9.99$$

For the fuzzy set PM,

$$X\text{-axis centroid point} = 191$$

$$\text{Rule strength applied to determine output area} = 0.1563$$

Shaded area =

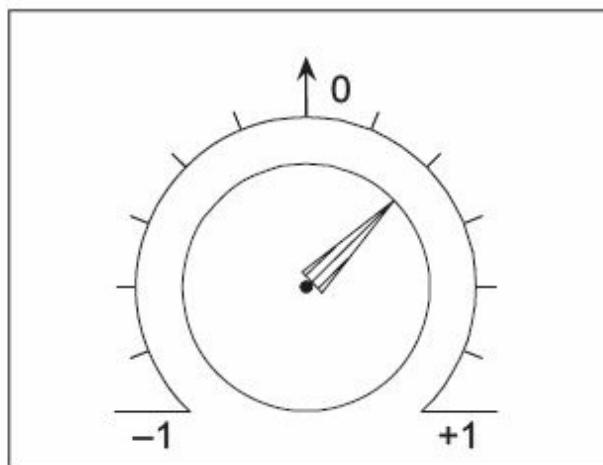
=

$$= 9.99$$

Therefore,

Weighted average, (CG) =

$$= 175$$



Air conditioner

$$\Delta T = T - T_0$$

T_0 : Set-point

In crisp terms, the throttle control (normalized) is to be set as 175.

7.6.2 Air Conditioner Controller

The system as illustrated in Fig. 7.12 comprises a dial to control the flow of warm/hot or cool/cold air and a thermometer to measure the room temperature (T oC). When the dial is turned positive, warm/hot air is supplied from the air conditioner and if it is turned negative, cool/cold air is supplied.

If set to zero, no air is supplied.

Fig. 7.12 Air conditioner control system.

A person now notices the difference in temperature (ΔT oC) between the room temperature (T oC) as measured by the thermometer and the desired temperature (T_0

oC) at which the room is desired to be kept (set-point). The problem now is to determine to what extent the dial should be turned so that the appropriate supply of air (hot/warm/cool/cold) will nullify the change in temperature.

For the above problem the rule base is as shown in Table 7.6.

Table 7.6 Fuzzy rule base for the air conditioner control

S.no.	Fuzzy rule (Descriptive)	Fuzzy rule (Notational)
1	<p><i>If</i> the room temperature is approximately equal to the set point T_0 oC, ΔT is approximately Zero (ZE) <i>and</i> the temperature is rapidly changing higher, i.e. $\frac{d\Delta T}{dt}$ is positively large (PL)</p> <p><i>then</i> blow cold air rapidly, i.e. turn the dial negative large (NL).</p>	<p><i>If</i> ΔT is ZE <i>and</i> $\frac{d\Delta T}{dt}$ is PL <i>then</i> dial should be NL</p>

2	<p><i>If</i> the room temperature is high <i>and</i> there is no change in temperature, i.e. ΔT is positive large (PL) and $\frac{d\Delta T}{dt}$ is approximately zero (ZE)</p>	<p><i>If</i> ΔT is PL and $\frac{d\Delta T}{dt}$ is ZE then dial should be NM.</p>
	<p><i>then</i> blow cold air at an intermediate level, i.e. turn the dial negative medium (NM).</p>	
3	<p><i>If</i> the room temperature is a little bit higher than the set-point <i>and</i> the temperature is gradually decreasing, i.e. ΔT is positively small (PS) and $\frac{d\Delta T}{dt}$ is negatively small (NS)</p>	<p><i>If</i> ΔT is PS and $\frac{d\Delta T}{dt}$ is NS <i>then</i> dial should be ZE.</p>
	<p><i>then</i> there is no need to blow hot or cold air, i.e. turn the dial to approximately zero (ZE).</p>	

$$\frac{d\Delta T}{dt}$$

The fuzzy sets for the system inputs, namely ΔT and

, and the system

output, namely turn of the dial are as shown in Fig. 7.13.

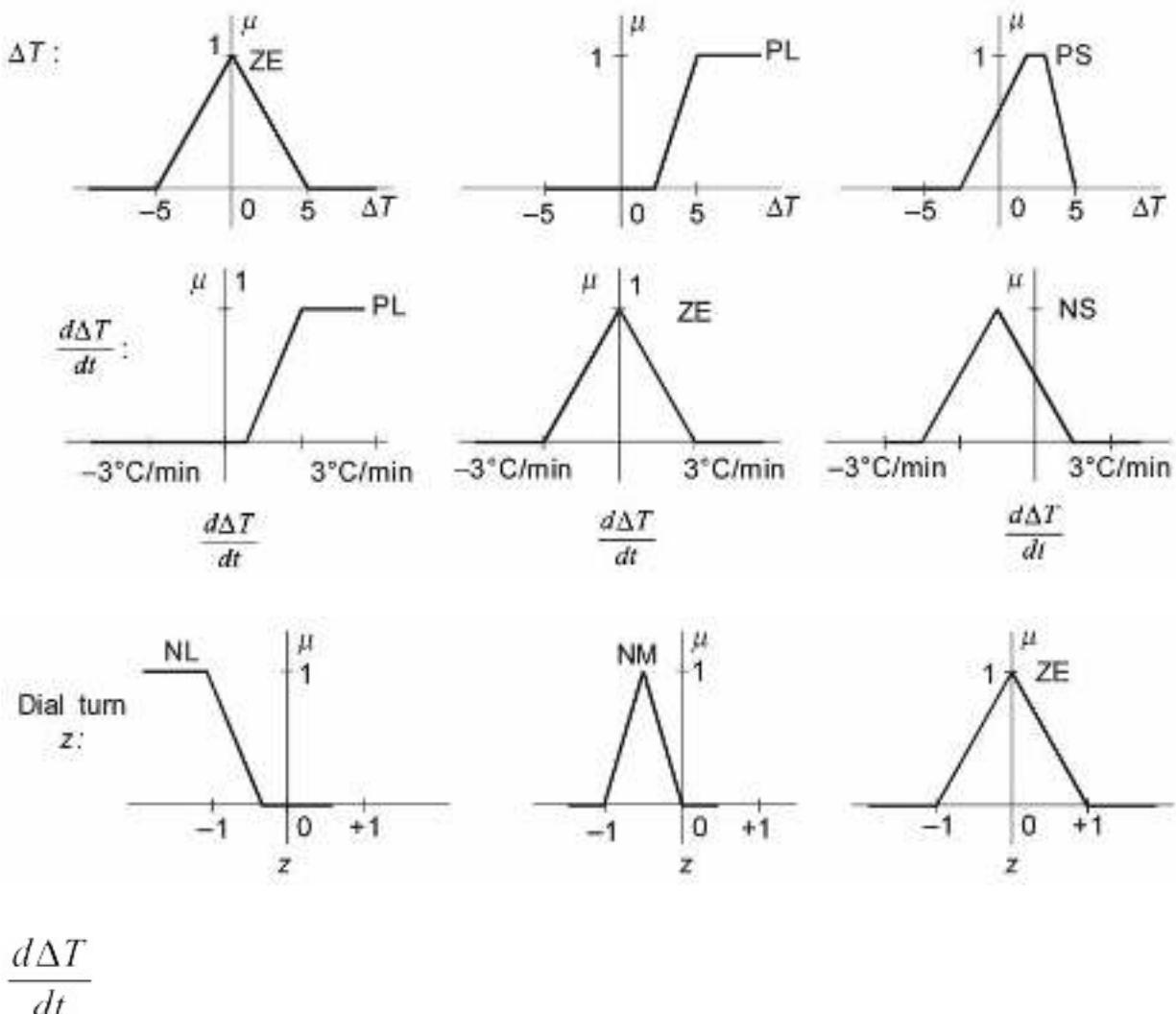


Fig. 7.13 Fuzzy sets for the air conditioner control system.

Consider the system inputs, $\Delta T = 2.5^\circ\text{C}$ and

$= -1^\circ\text{C/min}$. Here the

fuzzification of system inputs has been directly done by noting the membership value corresponding to the system inputs as shown in Fig. 7.14.

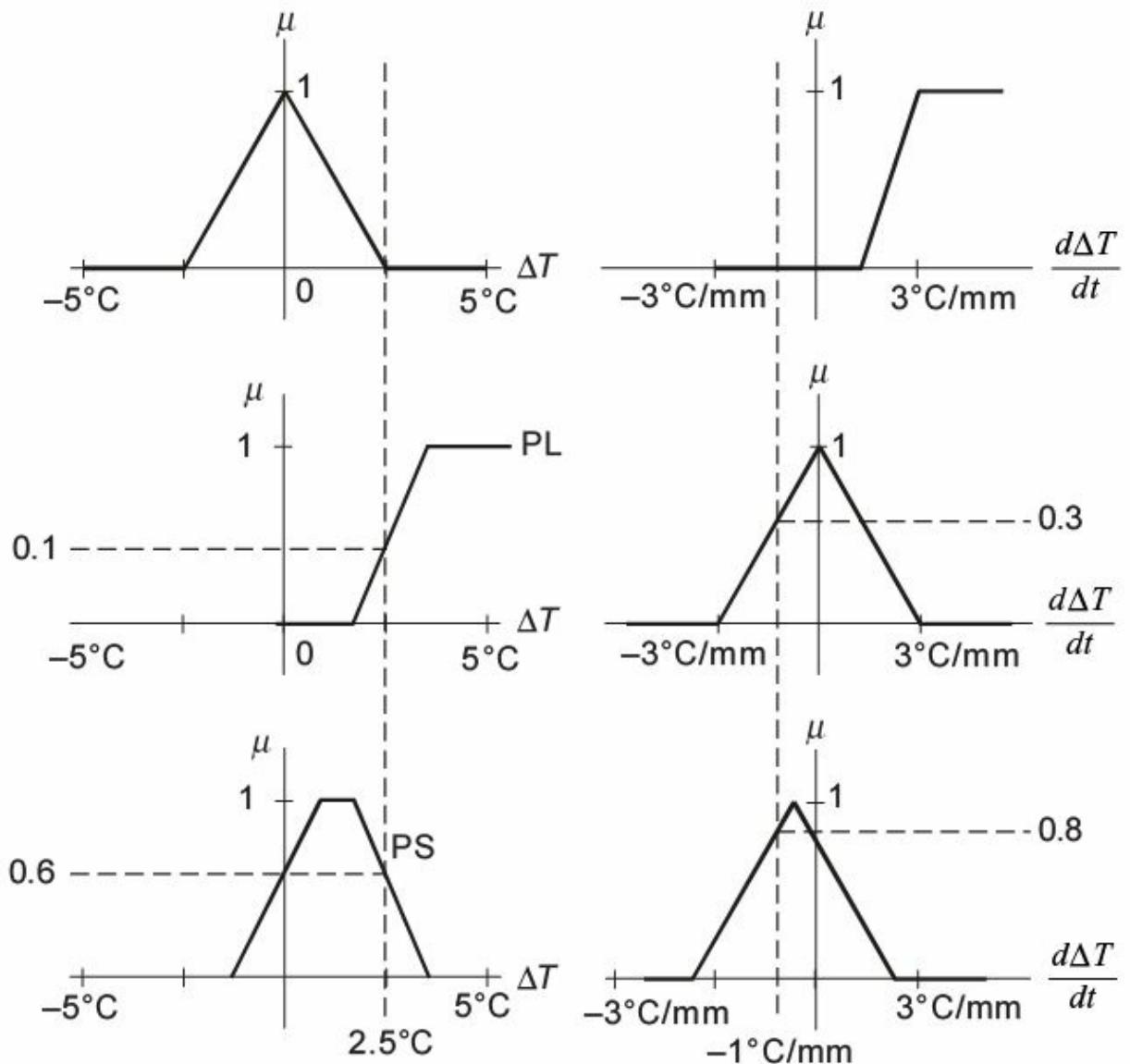


Fig. 7.14 Fuzzification of inputs $\Delta T = 2.5^{\circ}\text{C}$, $d\Delta T/dt = -1^{\circ}\text{C/min}$.

The rule strengths of rules 1, 2, 3 choosing the minimum of the fuzzy membership value of the antecedents are 0, 0.1 and 0.6 respectively. The fuzzy output is as shown in Fig 7.15.

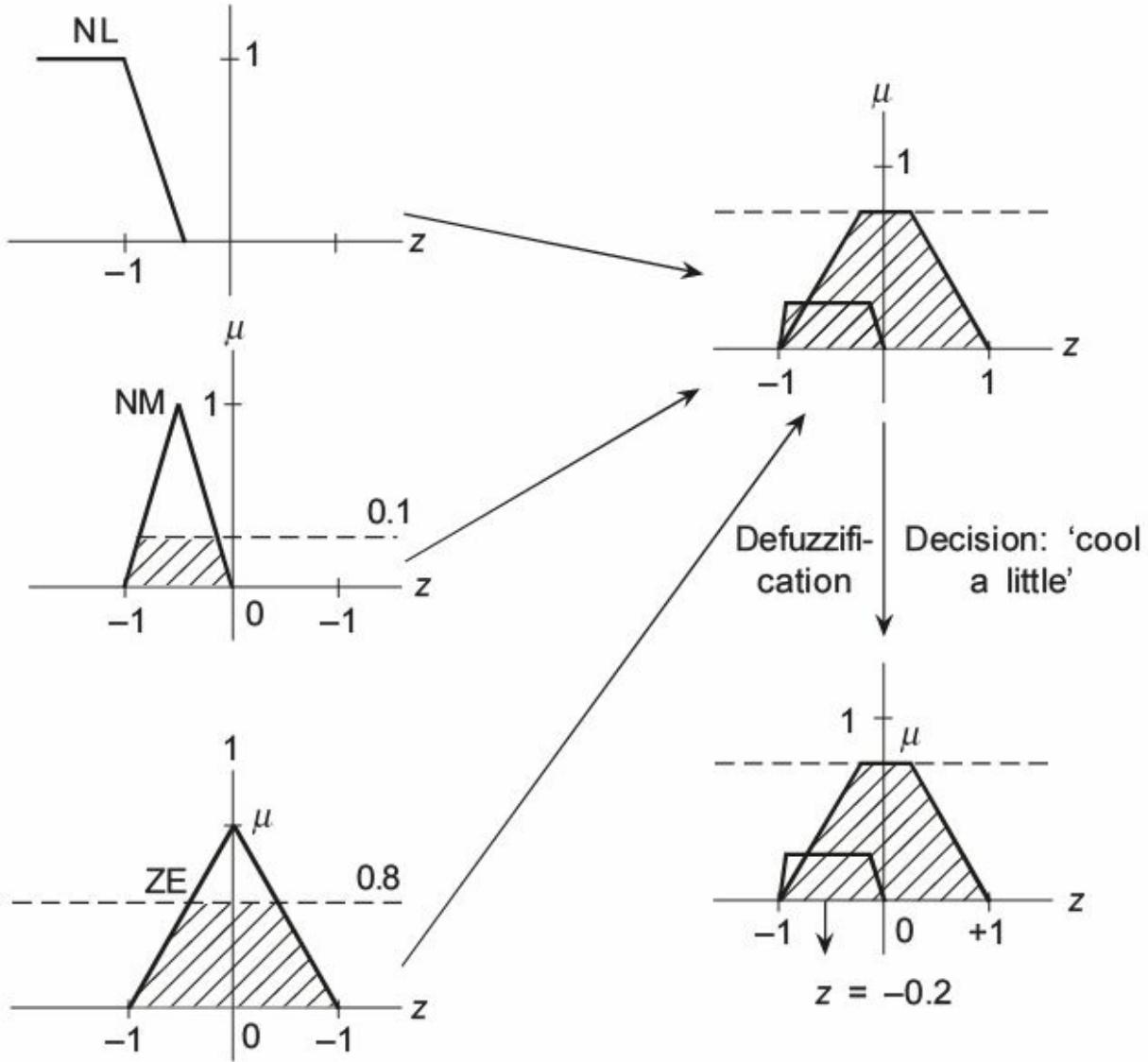


Fig. 7.15 Defuzzification of fuzzy outputs for z (turn of the dial).

The defuzzification of the fuzzy output yields $Z = -0.2$ for $\Delta T = 2.5^{\circ}\text{C}$ and $y = -1^{\circ}\text{C}/\text{min}$.

Hence, the dial needs to be turned in the negative direction, i.e. -0.2 to achieve the desired temperature effect in the room.

∴

SUMMARY

Crisp logic is classified into *propositional logic* and *predicate logic*.

Propositions are statements which are either true or false but not both.

Propositional logic supports the five major *connectives* \wedge , \vee , \sim , \Rightarrow , $=$.

Truth tables describe the semantics of these connectives.

The laws of propositional logic help in the simplification of formulae.

Modus Ponens ($P \Rightarrow Q$ and P , infers Q), *Modus Tollens* ($P \Rightarrow Q$ and $\sim Q$, infers $\sim P$), and *Chain rule* ($P \Rightarrow Q$ and $Q \Rightarrow R$ infers $P \Rightarrow R$) are useful rules of inference in propositional logic.

Propositional logic is handicapped owing to its inability to quantify.

Hence, the need for *predicate logic arises*. Besides propositions and connectives, predicate logic supports *predicates, functions, variables, constants and quantifiers* (\forall, \exists). The interpretation of predicate logic formula is done over a domain D . The three rules of inference of propositional logic are applicable here as well.

Fuzzy logic on the other hand accords multivalued truth values such as *absolutely true, partly true, partly false* etc. to fuzzy propositions.

While crisp logic is two valued, fuzzy logic is multivalued [0–1].

Fuzzy logic also supports fuzzy quantifiers classified as relative and absolute quantifiers and the Fuzzy rules of inference *Generalized Modus Ponens* (GMP) and *Generalized Modus Tollens* (GMT).

A set of fuzzy *if-then* rules known as a *fuzzy rule base* describes a *fuzzy rule based system*. However, for effective decision making, defuzzification techniques such as center of gravity method are employed which render the fuzzy outputs of a system in crisp terms.

Fuzzy systems have been illustrated using two examples, namely Greg Viot's fuzzy cruise control system and Yamakawa's air conditioner control system.

PROGRAMMING ASSIGNMENT

P7.1 Solve the Air conditioner controller problem (Sec. 7.6.2) using MATLAB®'s fuzzy logic tool box.

- (a) Make use of the FIS (Fuzzy Inference System) editor to frame the rule base and infer from it. Employ the centroid method of defuzzification.
- (b) Download Robert Babuska's fuzzy logic tool box.

(<http://lcewww.et.tudelft.nl/~babuska/>) and implement the same problem.

SUGGESTED FURTHER READING

Fuzzy logic concepts are discussed in *A First Course in Fuzzy Logic* (Nguyen and Walker, 1999). The design and properties of fuzzy systems and fuzzy control systems could be found in *A Course in Fuzzy Systems and Control* (Wang, 1997). Several fuzzy system case studies have been discussed in *The Fuzzy Systems Handbook* (Earl Cox, 1998). The book is also supplemented by a CD-ROM containing Windows 95 fuzzy logic library with code to generate 32 bit DLLs for Visual BASIC and Visual C++. The applications of fuzzy systems for neural networks, knowledge engineering and chaos are discussed in *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering* (Kasabov, 1996).

REFERENCES

Chang, C.L. and R.C. Lee (1973), Symbolic Logic and Mechanical Theorem Proving, Academic Press, NY.

Earl Cox, (1998), *The Fuzzy Systems Handbook*, Morgan Kaufmann Publishers.

Greg Viot (1993), Fuzzy Logic Concepts to Constructs, *AI Expert*, pp. 26–33, November.

Hellendoorn, H. and C. Thomas (1993), Defuzzification in Fuzzy Controllers , *Intelligent and Fuzzy Systems*, Vol. 1, pp. 109–123.

Hung T. Nguyen, Elbert A. Walker (1999), *A First Course in Fuzzy Logic*, CRC Press.

Li-Xin Wang (1997), *A Course in Fuzzy Systems and Control*, Prentice Hall, June.

Nikola K. Kasabov (1996), *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering*, MIT Press, 1996.

Yamakawa Takeshi (1993), A Fuzzy inference Engine in Nonlinear Analog mode and its application to a Fuzzy Logic Control, *IEEE Trans On Neural Networks*, Vol. 4, No. 3.

pp. 496–522.

PART 3

GENETIC ALGORITHMS

- Fundamentals of Genetic Algorithms
- Genetic Modelling

Chapter 8

Fundamentals of Genetic Algorithms

Decision-making features occur in all fields of human activities such as scientific and technological and affect every sphere of our life. Engineering design, which entails sizing, dimensioning, and detailed element planning is also not exempt from its influence.

For example an aircraft wing can be made from aluminium or steel and once material and shape are chosen, there are many methods of devising the

required internal structure. In civil engineering also, designing a roof to cover large area devoid of intermediate columns requires optimal designing.

The aim is to make objective function a maximum or minimum, that is, it is required to find an element X_0 in A if it exists such that $F(X_0) \leq F(X)$ for minimization

$F(X) \leq F(X_0)$ for maximization.....(8.1) The following major questions arise in this process

Does an optimal solution exist?

Is it unique?

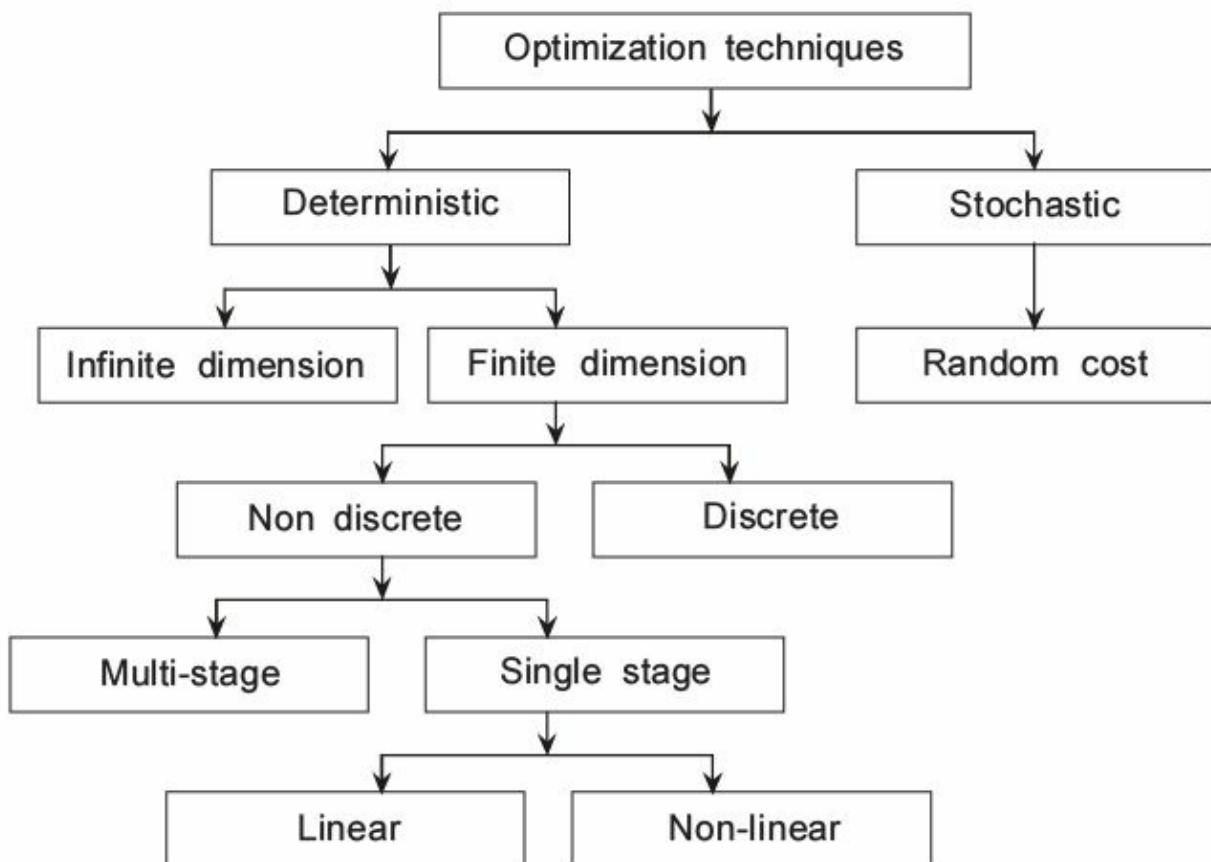
What is the procedure?

How sensitive the optimal solution is?

How the solution behaves for small changes in parameters?

Since 1940, several optimization problems have not been tackled by classical procedures including:

1. Linear programming
2. Transportation
3. Assignment
4. Nonlinear programming
5. Dynamic programming
6. Inventory
7. Queuing
8. Replacement



9. Scheduling

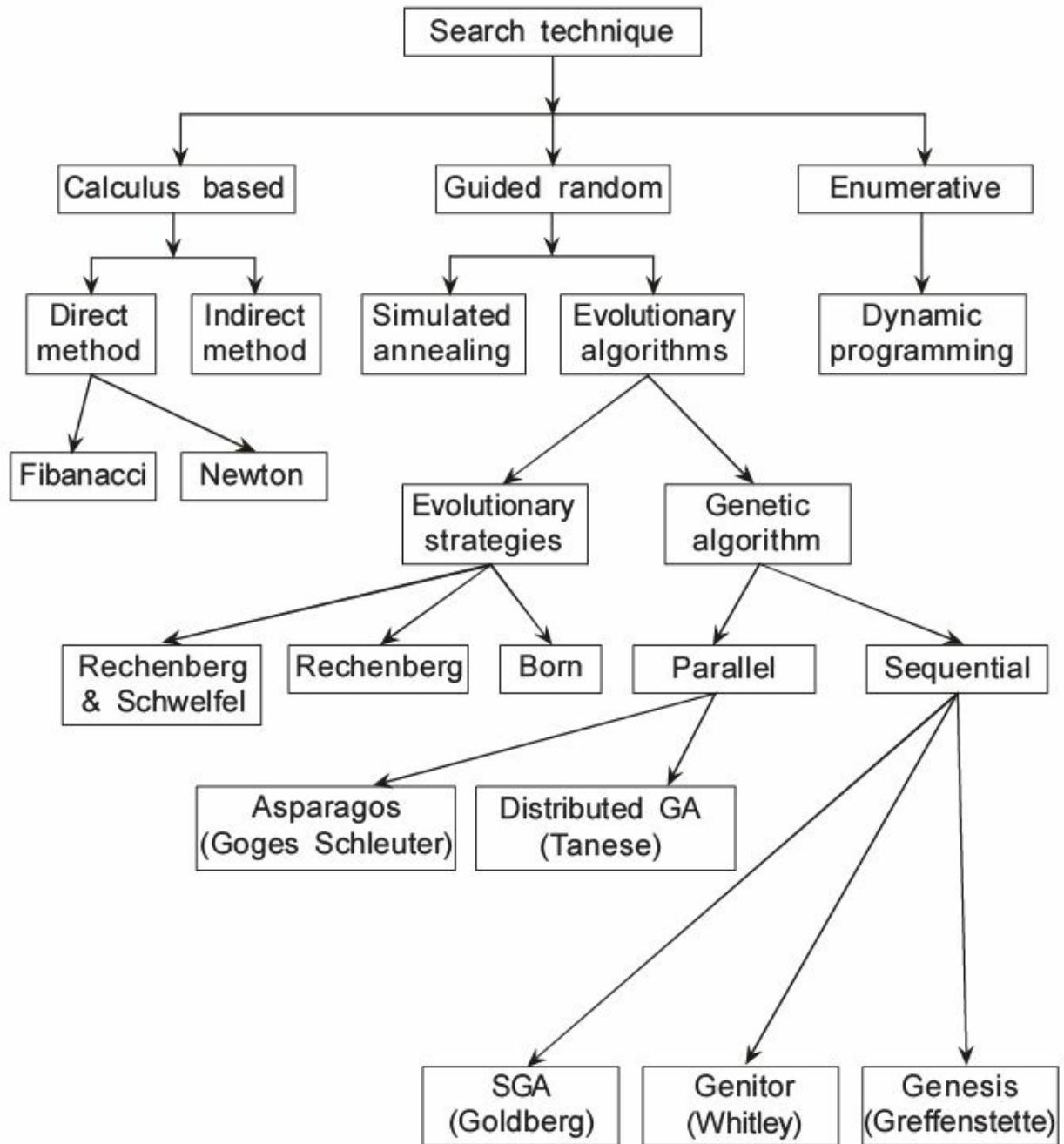
The classification of optimization techniques is shown in Fig. 8.1.

Basically, we have been following traditional search technique for solving nonlinear equations. Figure 8.2 shows the classes of both traditional and nontraditional search techniques. Normally, any engineering problem will have a large number of solutions out of which some are feasible and some are infeasible. The designer's task is to get the best solution out of the feasible solutions. The complete set of feasible solutions constitutes feasible design space and the progress towards the optimal design involves some kind of search within the space (combinatorial optimization). The search is of two kinds, namely deterministic and stochastic.

Fig. 8.1 Classification of optimization techniques.

In the case of *deterministic search*, algorithm methods such as steepest gradient methods are employed (using gradient concept), whereas in *stochastic approach*, random variables are introduced. Whether the search is deterministic or stochastic, it is possible to improve the reliability of the results where reliability means getting the result near optimum. A transition rule must be used to improve the reliability. Algorithms vary according to the transition rule used to improve the result.

Nontraditional search and optimization methods have become popular in



engineering optimization problems in recent past. These algorithms include:

1. Simulated annealing (Kirkpatrick, et al. 1983)

2. Ant colony optimization (Dorigo and Caro, 1999)

3. Random cost (Kost and Baumann, 1999)

4. Evolution strategy (Kost, 1995)
5. Genetic algorithms (Holland, 1975)
6. Cellular automata (Wolfram, 1994)

Fig. 8.2 Classes of search techniques.

Simulated annealing mimics the cooling phenomenon of molten metals to constitute a search procedure. *Genetic algorithm* and *evolutionary strategies* mimic the principle of natural genetics and natural selection to construct search and optimization procedures. The collective behaviour that emerges

from a group of social insects such as ants, bees, wasps, and termites has been dubbed as *Swarm intelligence*. The foraging of ants has led to a novel algorithm called *Ant colony optimization* for rerouting network traffic in busy telecommunication systems. This method was originally developed by Deneubourg and extended by Dorigo (1999) of Brussels. Random cost method is a stochastic algorithm which moves as enthusiastically uphill as down-hill. The method has no severe problems in escaping from a dead end and is able to find the optima. In this chapter, we discuss the fundamentals of genetic algorithms.

8.1 GENETIC ALGORITHMS: HISTORY

The idea of evolutionary computing was introduced in 1960 by I. Rechenberg in his work *Evolutionary strategies*. *Genetic algorithms* are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection. Prof. Holland of University of Michigan, Ann Arbor, envisaged the concept of these algorithms in the mid-sixties and published his seminal work (Holland, 1975). Thereafter, a number of students and other researchers have contributed to the development of this field.

To date, most of the GA studies are available through some books by Davis (1991), Goldberg (1989), Holland (1975), Michalewicz (1992) and Deb (1995) and through a number of conference proceedings. The first

application towards structural engineering was carried by Goldberg and Samtani (1986).

They applied genetic algorithm to the optimization of a

ten-member plane truss. Jenkins (1991) applied genetic algorithm to a trussed beam structure.

Deb (1991) and Rajeev and Krishnamoorthy (1992) have also applied GA to structural engineering problems. Apart from structural engineering there are many other fields in which GAs have been applied successfully. It includes biology, computer science, image processing and pattern recognition, physical science, social sciences and neural networks. In this chapter, we will discuss the basic concepts, representatives of chromosomes, fitness functions, and genetic inheritance operators with example. In Chapter 9, genetic modelling for real life problems will be discussed.

8.2 BASIC CONCEPTS

Genetic algorithms are good at taking larger, potentially huge, search spaces and navigating them looking for optimal combinations of things and solutions which we might not find in a life time.

Genetic algorithms are very different from most of the traditional optimization methods. Genetic algorithms need design space to be converted into genetic space. So, genetic algorithms work with a coding of variables.

The advantage of working with a coding of variable space is that coding discretizes the search space even though the function may be continuous. A more striking difference between genetic algorithms and most of the traditional optimization methods is that GA uses a population of points at one time in contrast to the single point approach by traditional optimization methods. This means that GA processes a number of designs at the same time. As we have seen earlier, to improve the search direction in traditional optimization methods, transition rules are used and they are deterministic in nature but GA uses randomized operators. Random operators improve the search space in an adaptive manner.

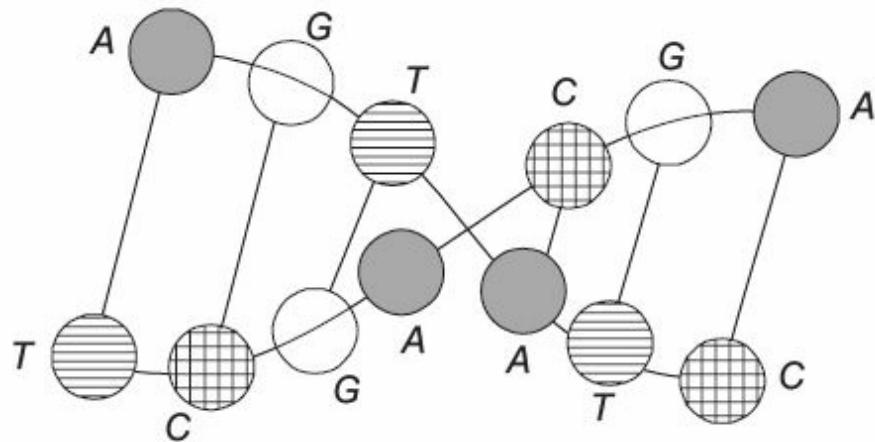
Three most important aspects of using GA are:

1. definition of objective function
2. definition and implementation of genetic representation
3. definition and implementation of genetic operators.

Once these three have been defined, the GA should work fairly well beyond doubt. We can, by different variations, improve the performance, find multiple optima (species if they exist) or parallelize the algorithms.

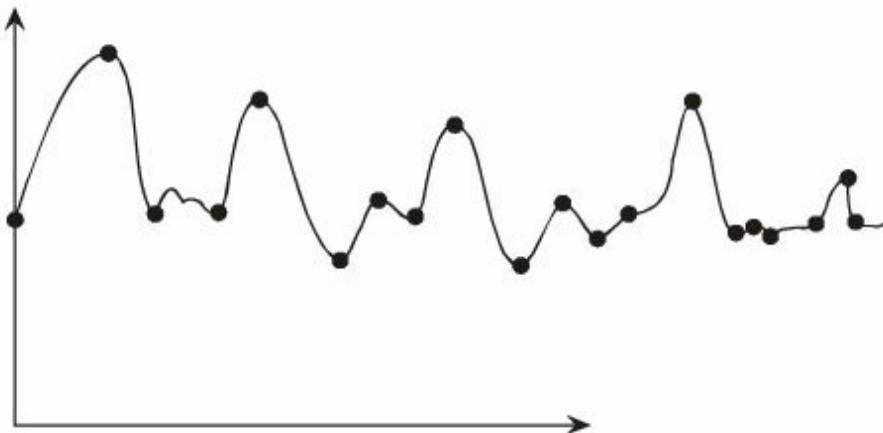
8.2.1 Biological Background

All living organisms consist of cells. In each cell, there is a set of chromosomes which are strings of DNA and serve as a model for the whole organism. A chromosome consists of genes on blocks of DNA as shown in Fig. 8.3. Each gene encodes a particular pattern. Basically, it can be said that each gene encodes a trait, e.g. colour of eyes. Possible settings of traits (bluish brown eyes) are called *alleles*. Each gene has its own position in the chromosome search space. This position is called *locus*. Complete set of



genetic material is called *genome* and a particular set of genes in genome is called *genotype*. The genotype is based on organism's phenotype (development after birth), its physical and mental characteristics such as eye colour, intelligence and so on.

Fig. 8.3 Genome consisting of chromosomes.



8.3 CREATION OF OFFSPRINGS

During the creation of offspring, recombination occurs (due to cross over) and in that process genes from parents form a whole new chromosome in some way. The new created offspring can then be mutated. *Mutation* means that the element of DNA is modified. These changes are mainly caused by errors in copying genes from parents. The *fitness* of an organism is measured by means of success of organism in life.

8.3.1 Search Space

If we are solving some problems, we work towards some solution which is the best among others. The space for all possible feasible solutions is called *search space*. Each solution can be marked by its value of the fitness of the problem. ‘Looking for a solution’ means looking for extrema (either maximum or minimum) in search space. The search space can be known by the time of solving a problem and we generate other points as the process of finding the solution continues (shown in Fig. 8.4).

Fig. 8.4 Examples of search space.

The problem is that, search space is complicated and one does not know where to look for the solution or where to start from and this is where genetic algorithm is useful. GAs are inspired by *Darwinian theory of the*

survival of the fittest. Algorithm is started with a set of solutions (represented by chromosomes) called *populations*. Solutions for one population are taken and used to form a new population. This is motivated by a hope that new population will be better than the old one. Solutions, which are selected to form new population (offspring), are selected according to their fitness. The more suitable they are, the more chances they have to reproduce. This is repeated until some conditions (number of populations) for improvement of best solution are satisfied.

$$X_i^{(L)} \leq X_i \leq X_i^{(U)} \text{ for } i = 1, 2, \dots, N$$

$$\frac{1}{1 + f(X)}$$

8.4 WORKING PRINCIPLE

To illustrate the working principle of GA, we first consider unconstrained optimization problem. Later, we shall discuss how GA can be used to solve a constrained optimization problem. Let us consider the following maximization problem.

$$\text{maximize } f(X) \quad (8.2)$$

If we want to minimize $f(X)$, for $f(X) > 0$, then we can write the objective function as

$$\begin{aligned} &\text{maximize} \\ &(8.3) \end{aligned}$$

If $f(X) < 0$ instead of minimizing $f(X)$, maximize $\{-f(X)\}$. Hence, both maximization and minimization problems can be handled by GA.

If the same problem is solved by multiple regression analysis, given k independent variables, for regressing the dependent variable $2(k+1)-1$

including the intercept which are given in Table 8.1.

Table 8.1 Subsets for regression analysis

Variable

Subsets

2

7

3

15

—

—

—

—

9

1023

—

—

19

10,48,578

On the other hand, in GA the variables are coded.

8.5 ENCODING

There are many ways of representing individual genes. Holland (1975) worked mainly with string bits but we can use arrays, trees, lists or any other object. Here, we consider only bit strings.

8.5.1 Binary Encoding

Example Problem (Knapsack Problem)

There are things with given values and size. The knapsack has a given capacity. Select things to minimize their value in knapsack not exceeding the capacity of the knapsack.

Encoding

Each bit says if the thing is in knapsack or not. Binary coding is the most commonly used in GA as shown in Table 8.2.

Table 8.2 Chromosomes

Chromosome A

101101100011

Chromosome B

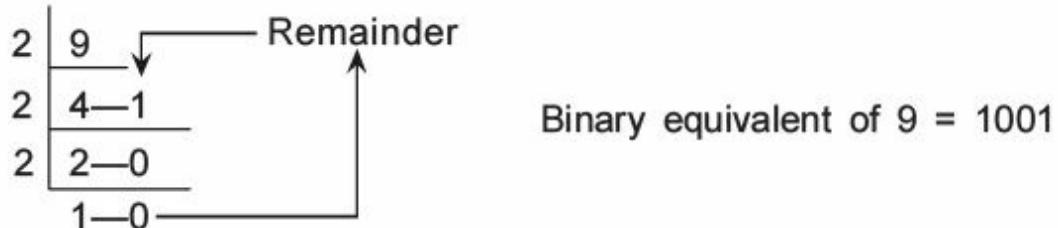
010011001100

Binary encoding gives many possible chromosomes even with small number of alleles. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after genetic operator corrections.

In order to use GA to solve the maximization or minimization problem, unknown variables X_i are first coded in some string structures. It is important to mention that coding of the variable is not absolutely necessary. There exist some studies where GAs are directly used on the variables themselves, but here we shall ignore the exceptions and discuss the encoding for simple genetic algorithm. Binary-coded strings having 1s and 0s are mostly used.

The length of the string is usually determined according to the desired solution accuracy. For example, 4-bit binary string can be used to represent 16 numbers as shown in Table 8.3.

Table 8.3 Four-bit string



$$\begin{array}{r}
 1 \quad 0 \quad 0 \quad 1 \\
 \times 2^0 = 1 \\
 \times 2^1 = 0 \\
 \times 2^2 = 0 \\
 \times 2^3 = \frac{8}{9}
 \end{array}$$

$$X_i^L \leq X_i \leq X_i^U$$

$$(X_1^L, X_2^L);$$

$$(X_1^U, X_2^U)$$

4-bit

Numeric

4-bit

Numeric

4-bit

Numeric

string

value

string

value

string

value

0000

0

0110

6

1100

12

0001

1

0111

7

1101

13

0010

2

1000

8

1110

14

0011

3

1001

9

1111

15

0100

4

1010

10

0101

5

1011

11

To convert any integer to a binary string, go on dividing the integer by 2 as shown in

Fig. 8.5. We get equivalent integer for the binary code by decoding it as shown in Fig. 8.6.

Fig. 8.5 Binary coding.

Fig. 8.6 Equivalent integer for a binary code.

For example, if we want to code a two variable function assuming four bits are used for each variable, we represent the two variables X_1, X_2 as (1011

0110). As given in Eq. (8.2), every variable will have both upper and lower limits as

(8.4)

As shown in Table 8.3 a four-bit string can represent the integers from 0 to 15 (16 elements) and hence, (0000 0000) and (1111 1111) represent the points for X_1, X_2 as

respectively because the substrings

$$\sum_{k=0}^{k=n_i-1} 2^k s_k$$

$$X_i^L$$

$$X_i^U$$

$$X_i^L + \frac{(X_i^U - X_i^L)}{(2^{n_i} - 1)}$$

X_i^L

X_i^U

$$2 + \frac{(17 - 2)}{(2^4 - 1)} \times 10$$

$$(X_i^U - X_i^L)/2^{n_i}$$

(0000) and (1111) have the minimum and the maximum decoded values.

Hence, an n -bit string can represent integers from 0 to $2^n - 1$, i.e. 2^n integers.

Assume that X_i is coded as a substring S_i of length n_i . The decoded value of a binary substring S_i is calculated as shown in Fig. 8.6 as (8.5)

where s_i can be either zero or 1 and the string S is represented as $s_{n-1} \dots s_3 s_2 s_1 s_0$ (8.6) For example, a four-bit string (0111) has a decoded value equal to $23 \times 0 + 22 \times 1 + 21 \times 1 + 20 \times 1 = 7$

Knowing

and

corresponding to (0000) and (1111), the equivalent

value for any

4-bit string can be obtained as

$X_i =$

$$\times (\text{decoded value of string}) \quad (8.7)$$

Assume for a variable $X_i = 2$, and

= 17, to find what value of 4-bit

string of $X_i = (1010)$ would represent. First we get the decoded value for S_i as
 $S_i = 1010 = 23 \times 1 + 22 \times 0 + 21 \times 1 + 20 \times 0 =$

$$10 \quad (8.8a)$$

$X_i =$

$$= 12 \quad (8.8b)$$

Hence, the accuracy that can be obtained with a four-bit code is 1/16th of search space. But as the string length is increased by one, the obtained accuracy increases exponentially to 1/32th of the search space. It is not necessary to code all variables in equal substring length. The length of substring representing a variable depends on the desired accuracy in that variable. Generalizing the concept, we may say that with n_i bit-length coding for a variable, the obtainable accuracy in that variable approximation is

. Once the coding of the variables is done, the corresponding point ($X_1 \dots X_n$) T can be found out using Eq. (8.7). For continuous design variable,

$$\log_2 \left(\frac{X^U - X^L}{\varepsilon} \right)$$

if ε is the precision representation required then string length ‘ S ’ should be equal to

$S =$

$$(8.9)$$

In some cases, X_i need not be equally distributed so as to apply the linear mapping rule. Hence, X_i can be given in the form of a table as shown in Table 8.4.

Table 8.4 Binary representation of fibre angles

S.No.

Binary coding

Decoded value

Fibre angle

1

0000

0

0

2

0001

1

10

3

0010

2

20

4

0011

3

30

5

0100

4

45

6

0101

5

60

7

0110

6

70

8

0111

7

80

9

1000

8

90

10

1001

9

-10

11

1010

10

-20

12

1011

11

-30

13

1100

12

-45

14

1101

13

-60

15

1110

14

-70

16

1111

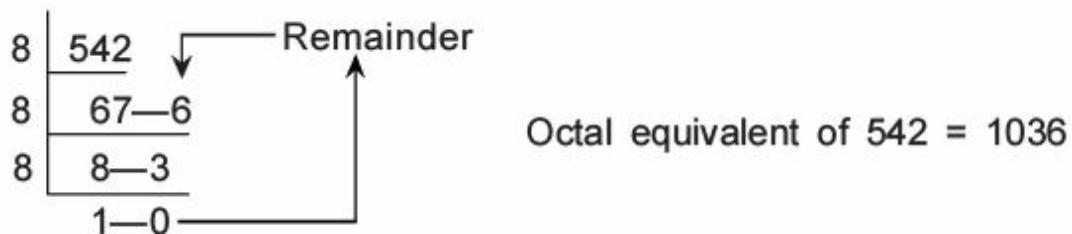
15

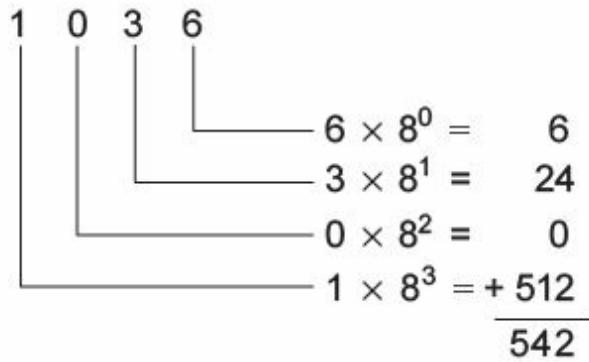
-80

Hence, when the values are not uniformly distributed, tabulated values can be used to find the corresponding point $X = (X_1, X_2, \dots, X_n)^T$. Thereafter, the function value at that point X can also be calculated by substituting X in the given objective function.

8.5.2 Octal Encoding (0 to 7)

To convert any integer to an octal string, go on diving the integer by 8 as





(X_1^L, X_2^L) ;

(X_1^U, X_2^U)

$$\sum_{k=0}^{k=n_i-1} 8^k s_k$$

$$(X_i^U - X_i^L) / 8^{n_i}$$

shown in Fig. 8.7. For example, 542 is given in octal form as 1036.

Fig. 8.7 Octal encoding.

For the octal code, we can get the equivalent integer by decoding it as shown in Fig. 8.8. The integer value for the octal code 1036 is 542.

Fig. 8.8 Equivalent integer for an octal code.

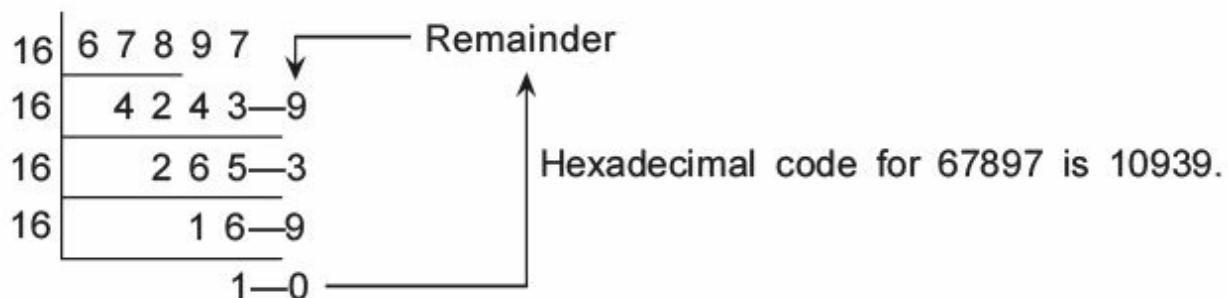
A four-bit octal string can represent the integers from 0 to 4095 and hence, (0000 0000) and (7777 7777) would represent the points for X_1 and X_2 respectively. The decoded value of a binary substring S_i is calculated as

(8.10)

and hence, the obtainable accuracy in that variable approximation is

8.5.3 Hexadecimal Encoding (0123456789ABCDEF)

To convert any number to hexadecimal form, we go on dividing the number by 16 as shown in Fig. 8.9. The hexadecimal code for 67897 is shown to be 10939. We get equivalent integer for the hexadecimal code by decoding it as shown in Fig. 8.10. The decoded value for the hexadecimal number BO79E6 is 11565542.



B	0	7	9	E	6	
						$6 \times 16^0 = 6$
						$14 \times 16^1 = 24$
						$9 \times 16^2 = 2304$
						$7 \times 16^3 = 28672$
						$0 \times 16^4 = 0$
						$11 \times 16^5 = +11534336$
						<hr/>
						11565542

(X_1^L, X_2^L) ;

(X_1^U, X_2^U)

$$\sum_{k=0}^{k=n_i-1} 16^k s_k$$

$$(X_i^U - X_i^L) / 16^{n_i}$$

$$(X_1^L, X_2^L);$$

$$(X_1^U, X_2^U)$$

$$\sum_{k=0}^{k=n_i-1} b^k s_k$$

$$(X_i^U - X_i^L) / b^{n_i}$$

Fig. 8.9 Hexadecimal coding.

Fig. 8.10 Equivalent integer for hexadecimal code.

A four-bit hexadecimal can represent the integers from 0 to 65535 and hence, (0000 0000) and

(FFFF FFFF) would represent the points for X_1 and X_2 as respectively. The decoded value of a hexadecimal string S_i is calculated as (8.11)

And hence, the obtainable accuracy in that variable approximation is

. From the above discussion it is clear that encoding can be given to any base ‘ b ’, bits of n_i length can represent the integers from 0 to ($b^{n_i} - 1$) and hence (0000 0000), and (($b - 1$)($b - 1$)($b - 1$)($b - 1$)), and ($b - 1$)($b - 1$)($b - 1$)($b - 1$)) would represent the points X_1 and X_2 as respectively. The decoded value of ‘ b ’ bit-string S_i is calculated as (8.12a)

And hence, obtainable accuracy in that variable approximation is (8.12b)

8.5.4 Permutation Encoding

This can be used in ordering problems such as travelling salesman or task ordering. In a *permutation encoding*, every chromosome is a string of numbers which represents the number in the sequence as shown in Table 8.5.

Table 8.5 Permuation encoding

Chromosome- A

1

5

3

2

4

7

9

8

6

Chromosome- B

8

5

6

7

2

3

1

4

9

Even for ordering problems after applying for sometimes, the genetic operators corrections must be made to leave the chromosome consistent.

Example Problem Travelling Salesman Problem

The problem: There are cities and given distances between them. Travelling salesman has to visit all of them. Find the sequence of cities to minimize the travelling distance.

Encoding

Chromosome illustrates the order of cities in which the salesman would visit them.

8.5.5 Value Encoding

In this, every chromosome is a string of some values and the values can be any thing connected to the problem. From numbers, real numbers characterize some complicated objects as shown in Table 8.6.

Table 8.6 Value encoding

Chromosome– A

1.234

5.3243

0.4556

2.0253

Chromosome- *B*

abdjetijdhj...

Chromosome- *C*

(Back),

(Right),

(Forward),

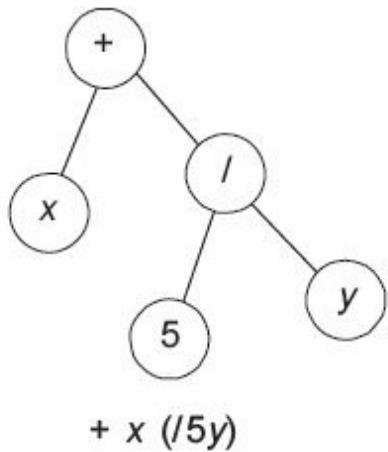
(Left)

Value encoding is very good for some special problems. On the other hand, this encoding is often necessary to develop new genetic operators specific to the problem.

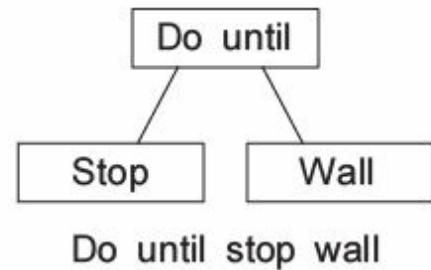
Example Find the weights of neural network.

The problem: To find the weights of synapses connecting input to hidden layer and hidden layer to output layer.

Chromosome-A



Chromosome-B



Encoding

Each value in chromosome represents the corresponding weights.

8.5.6 Tree Encoding

This is mainly used for evolving program expressions for genetic programming. In a tree encoding, every chromosome is a tree of some objects such as functions and commands, in a programming language as shown in Fig. 8.11. Tree encoding is good for evolving programs in a programming language. LISP is often used because programs in it are represented in this form and can easily be parsed as a tree so that functions and genetic operators can be applied rather easily.

Fig. 8.11 Tree encoding.

Example Find the function for a given value.

Problem: Some input and output values are given. The task is to find the function which will give the best relation to satisfy all values.

Encoding

Chromosomes are functions represented in a tree.

8.6 FITNESS FUNCTION

As pointed out earlier GAs mimic the Darwinian theory of survival of the fittest and principle of nature to make a search process. Therefore, GAs are usually suitable for solving maximization problems. Minimization problems are usually transformed into maximization problems by some suitable transformation. In general, fitness function $F(X)$ is first derived from the objective function and used in successive genetic operations.

Certain genetic operators require that fitness function be non-negative, although certain operators do not have this requirement. Consider the following transformations

$F(X) = f(X)$ for maximization problem

$F(X) = 1/f(X)$ for minimization problem, if $f(X) \neq 0$

$F(X)$

=

$1/(1$

+

$f(X)),$

if

$f(X)$

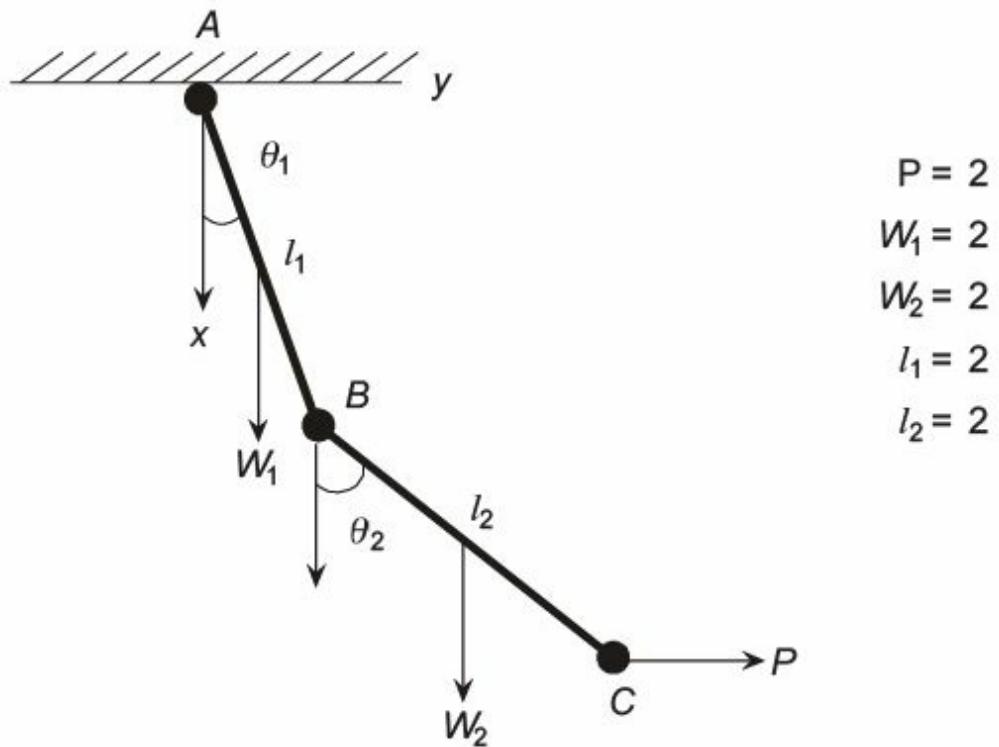
=

0 (8.13)

A number of such transformations are possible. The fitness function value of the string is known as *string's fitness*.

Example 8.1

Two uniform bars are connected by pins at A and B and supported at A . A horizontal force P acts at C . Knowing the force, length of bars and its weight determine the equilibrium configuration of the system if friction at all joints are neglected (see Fig. 8.12).



$$-P[(l_1 \sin \theta_1 + l_2 \sin \theta_2)] - \frac{W_1 l_1}{2} \cos \theta_1 - W_2 \left[\frac{l_2}{2} \cos \theta_2 + l_1 \cos \theta_1 \right]$$

$$\frac{\partial \Pi}{\partial \theta_1} \delta \theta_1 + \frac{\partial \Pi}{\partial \theta_2} \delta \theta_2$$

$$\frac{\partial \Pi}{\partial \theta_1}$$

$$\frac{\partial \Pi}{\partial \theta_2}$$

$$\frac{2}{3}$$

Fig. 8.12 Two bar pendulum.

The total potential for the two bar pendulum is written as

$$\Pi =$$

$$(8.14)$$

Substituting the values for P , W_1 , W_2 , and for the lengths as 2 we get,

$$\Pi(\theta_1,$$

$$\theta_2)$$

$$=$$

$$-4\sin\theta_1$$

$$-$$

$$6\cos\theta_1$$

$$-$$

$$4\sin\theta_2$$

$$-$$

$$2\cos\theta_2 \quad (8.15a)$$

$$0 \leq \theta_1, \theta_2 \leq 90 \dots \dots \dots (8.15b)$$

Equilibrium configuration is the one which makes Π a minimum.

Theoretical solution

$\Delta \Pi = 0$, for Π to be maximum or minimum

$$\Delta \Pi =$$

$$= 0 \quad (8.16)$$

$\Delta\theta_1, \Delta\theta_2$ are arbitrary. Therefore we get,

$$= 4\cos\theta_1 - 6\sin\theta_1 = 0 \quad (8.17a)$$

$$= 4\cos\theta_2 - 2\sin\theta_2 = 0 \quad (8.17b)$$

From Eq. (8.17(a)) and (b) we get,

$$\tan\theta_1 = , \theta_1 = 33.7^\circ \text{ (0.558 radians)}$$

$$\frac{X^U - X^L}{2^4 - 1} = \frac{90}{15} = 6^\circ$$

$$X_i^L + \frac{X_i^U - X_i^L}{2^4 - 1} S_i$$

$$\tan\theta_2 = 2, \theta_2 = 63.43^\circ \text{ (1.107 radians)} \quad (8.18)$$

For which $\prod = -11.68$

Since there are two unknowns θ_1 and θ_2 in this problem, we will use 4-bit binary string for each unknown.

Accuracy =

(8.19)

Hence, the binary coding and the corresponding angles are given as $X_i =$

(8.20)

where S_i is the decoded value of the i th chromosome. The binary coding and the corresponding angles are given in Table 8.7.

Table 8.7 Binary coding and the corresponding angles

S. no.

Binary coding

Angle

S. no.

Binary coding

Angle

1

0000

0

9

1000

48

2

0001

6

10

1001

54

3

0010

12

11

1010

60

4

0011

18

12

1011

66

5

0100

24

13

1100

72

6

0101

30

14

1101

78

7

0110

36

15

1110

84

8

0111

42

16

1111

90

The objective function of the problem is given in Eq. (8.15). The contours of the objective function as well as the 3D plot are shown in Figs. 8.13(a) and (b) respectively.

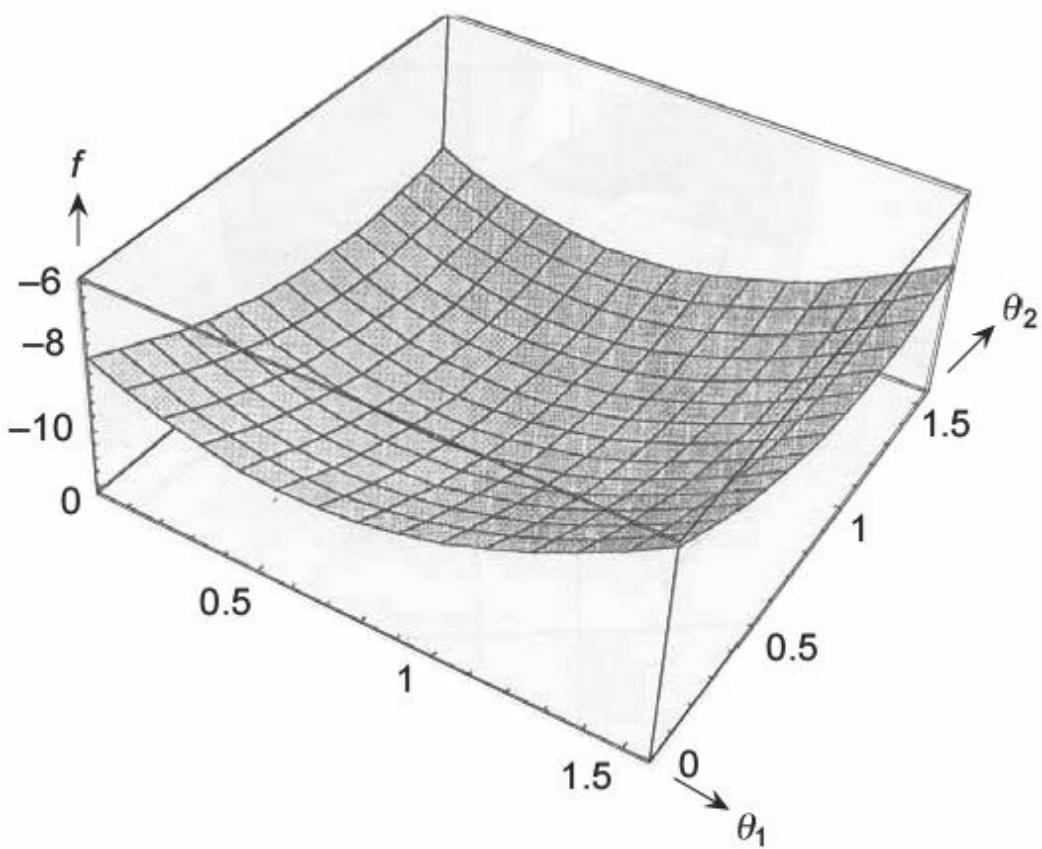
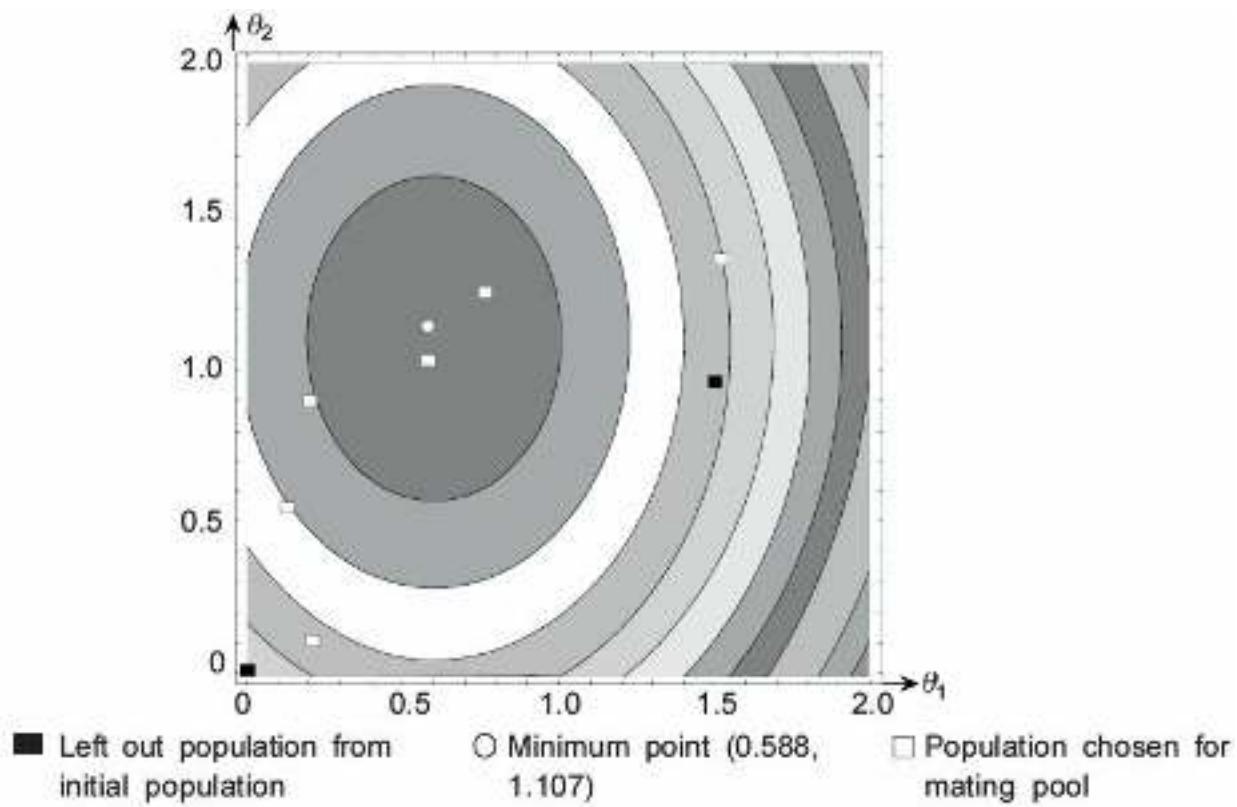


Fig. 8.13(a) Contours of equal objective functions.

Fig. 8.13(b) Three-dimensional plot of the objective function.

Since the objective function is negative, instead of minimizing the function

‘ f ’ let us maximize

$-f = f$. The maximum value of $f = 8$ when θ_1, θ_2 are zero. Hence, the fitness function F is given as

$$F = f - 7 = -f + 7 \quad (8.21)$$

First randomly generate eight populations with 8-bit strings as shown in Table 8.8.

Table 8.8 Computation of fitness function

Angles

Population

Population

No.

θ_1

θ_2

$$F = -f + 7$$

1

0000 0000

0

0

1

2

0010 0001

12

6

2.1

3

0001 0101

6

30

3.11

4

0010 1000

12

48

4.01

5

0110 1010

36

60	
4.66	
6	
1110 1000	
84	
48	
1.91	
7	
1110 1101	
84	
78	
1.93	
8	
0111 1100	
42	
72	
4.55	

As shown in Table 8.8 and Fig. 8.13(c), GA begins with a population of random strings representing design or decision variables. Thereafter, each string is evaluated to find the fitness value. The population is then operated by three main operators, namely reproduction, cross over, and mutation, to create a new population of points. The new population is further evaluated

and tested for termination. If the termination criteria are not met, the population is iteratively operated by the three operators and evaluated until the termination criteria are met. One cycle of these operations and the subsequent evaluation procedure is known as a *generation* in GA terminology.

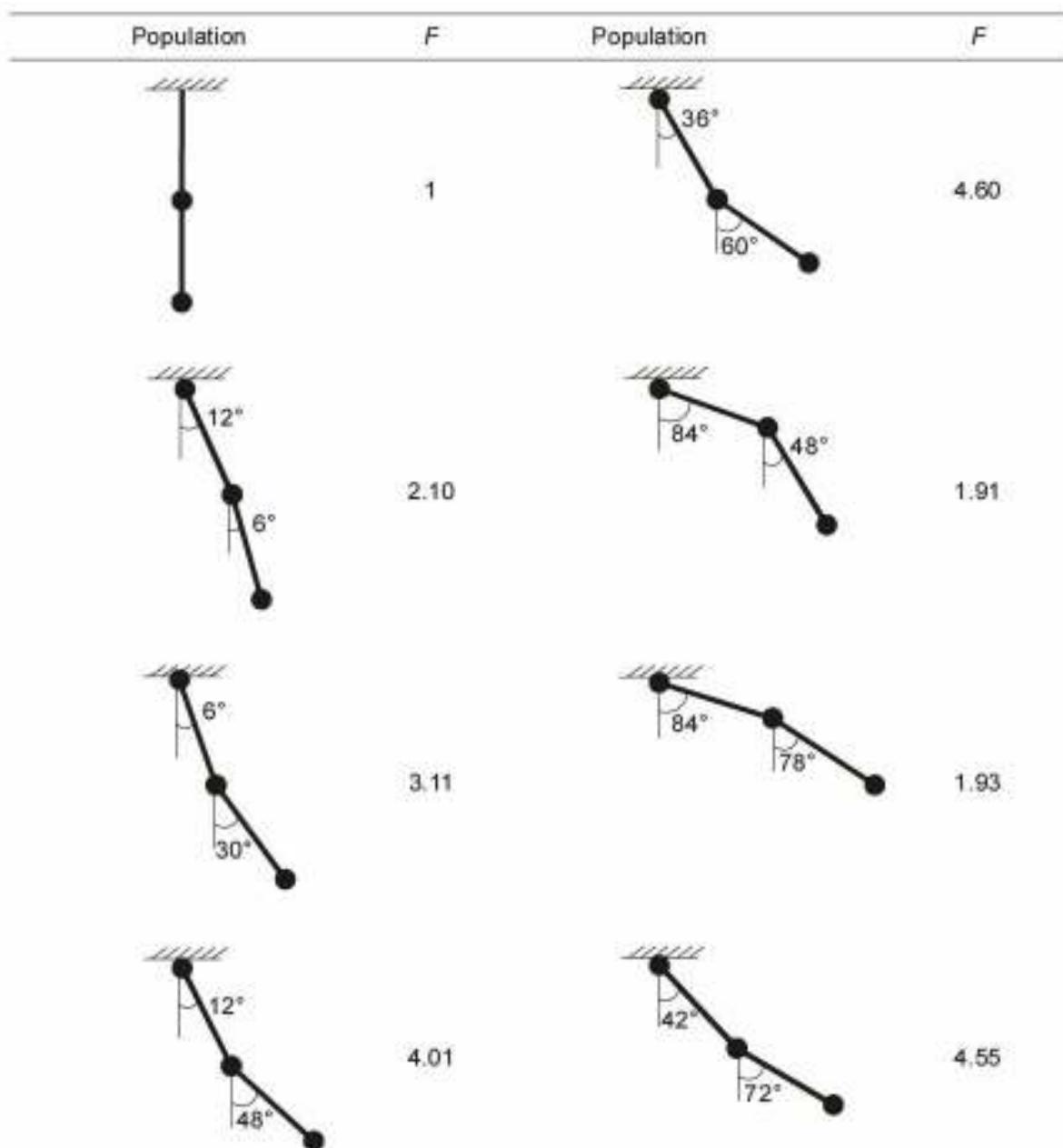


Fig. 8.13(c) ‘*F*’ for various population.

$$\frac{F_i}{\sum_{j=1}^n F_j}$$

8.7 REPRODUCTION

Reproduction is usually the first operator applied on population.

Chromosomes are selected from the population to be parents to cross over and produce offspring. According to Darwin's evolution theory of survival of the fittest, the best ones should survive and create new offspring. That is why reproduction operator is sometimes known as the *selection operator*. There exists a number of reproduction operators in GA literature but the essential idea in all of them is that the above average strings are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner. The various methods of selecting chromosomes for parents to cross over are:

1. Roulette-wheel selection
2. Boltzmann selection
3. Tournament selection
4. Rank selection
5. Steady-state selection

8.7.1 Roulette-wheel Selection

The commonly used reproduction operator is the proportionate reproductive operator where a string is selected from the mating pool with a probability proportional to the fitness. Thus, i th string in the population is selected with a probability proportional to F_i where F_i is the fitness value for that string.

Since the population size is usually kept fixed in a simple GA, the sum of the probabilities of each string being selected for the mating pool must be one.

The probability of the i th selected string is

$$p_i =$$

(8.22)

where ' n ' is the population size. For the example problem discussed in Example 8.1 the probability values of each string are given in Table 8.9.

Table 8.9 Probability of an individual string

Population No.

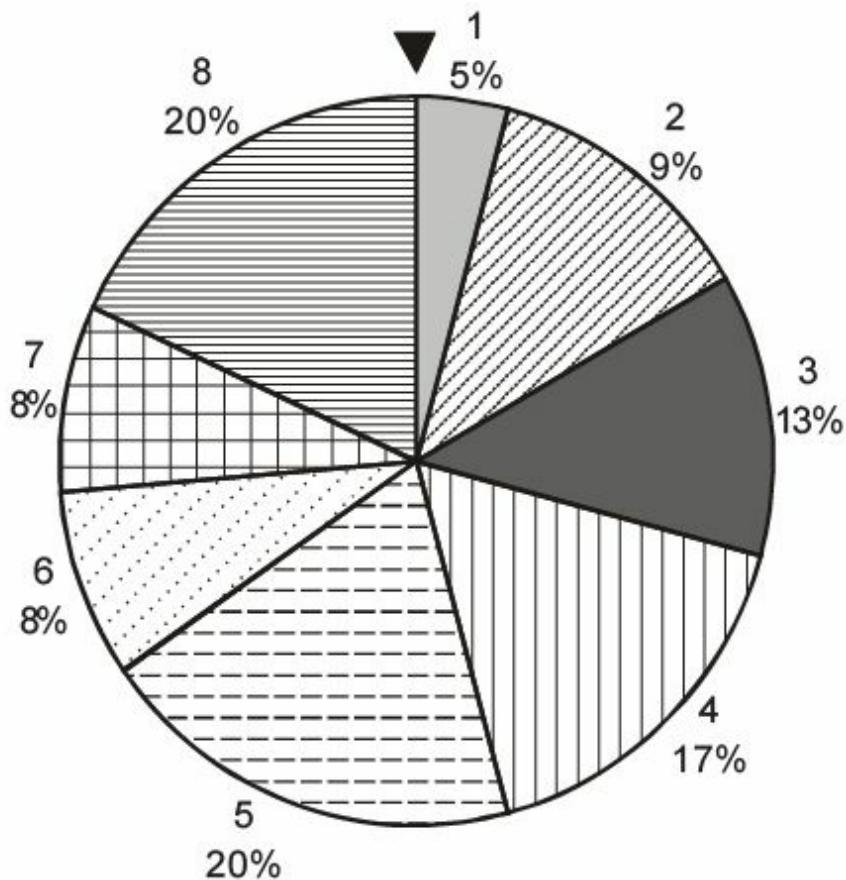
Population

$$F = -f - 7$$

$$\beta_i$$

$$\bar{F}$$

$$\bar{F}$$



1

0000 0000

1

0.0429

2

0010 0001

2.1

0.090

3

0001 0101

3.11

0.1336

4

0010 1000

4.01

0.1723

5

0110 1010

4.66

0.200

6

1110 1000

1.91

0.082

7

1110 1101

1.93

0.0829

8

0111 1100

4.55

0.1955

= 2.908

One way to implement this selection scheme is to imagine a Roulette-wheel with its circumference for each string marked proportionate to string's fitness (see Fig. 8.14). The fitness of the population is calculated as Roulette-wheel is spun ' n ' times (in this example eight times), each time selecting an instance of the string chosen by the Roulette-wheel pointer. Since the circumference of the wheel is marked according to a string's fitness, the Roulette-wheel mechanism is expected to make F_i/n copies of the i th string of the mating pool.

Fig. 8.14 Roulette-wheel marked for eight individuals according to fitness.

The average fitness

\bar{F}

$$\sum_{j=1}^n F_j / n$$

=

(8.23)

Figure 8.14 shows a Roulette-wheel for eight individuals having different fitness values. Since the fifth individual has a higher fitness than any other, it is expected that the Roulette-wheel selection will choose the fifth individual more than any other individual.

This Roulette-wheel selection scheme can be simulated easily. Using the fitness value F_i of all strings, the probability of selecting a string p_i can be

calculated. Thereafter, cumulative probability P_i of each string being copied, can be calculated by adding the individual probabilities from the top of the list. Thus, the bottom most string in the population should have a cumulative probability of $P_8 = 1$. The Roulette-wheel concept can be simulated by realizing that the i th string in the population represents the cumulative probability from P_{i-1} to P_i . Thus, the first string represents the cumulative values from 0 to P_1 .

Hence, cumulative probability of any string lies between 0–1. In order to choose n strings, n random numbers between zero and one are created at random. Thus, the string that represents the chosen random number in the cumulative probability range (calculated from fitness value) for the string, is copied to the mating pool. This way, the string with a higher fitness value will represent a larger range in the cumulative probability values and therefore, has a higher probability of being copied into the mating pool. On the other hand, a string with a smaller fitness value represents a smaller range in cumulative probability values and has a smaller probability of being copied into the mating pool. Now, we illustrate the working of Roulette-wheel simulation for an example.

Referring to Table 8.10, once probability of the individual strings are known we can find the *expected count* of each string as

$$\text{Expected count} = (n = 8) \times p_i \quad (8.24)$$

These values are calculated and shown in column A of Table 8.10. From the probability p_i , the cumulative probability can be computed. For example, P_5 is given by

$$P_5 = 0.0429 + 0.090 + 0.1336 + 0.1723 + 0.2 =$$

$$0.6388 \quad (8.25)$$

These distributions are shown in column B of Table 8.10. In order to form the mating pool, we create random numbers between zero and one (given in column C) and identify the particular string which is specified by each of these random numbers. For example, if a random number of 0.428 is created,

the fourth string gets a copy in the mating pool because the string occupies the interval 0.266–0.438, as shown in column B. Column D refers to the selected string. Similarly, other strings are selected according to random numbers shown in column C. After this selection procedure is repeated $n = 8$

times, where ‘ n ’ is the population size, the number of selected copies for each string is counted. This number is shown in column E. For example, the strings 4 and 5 get 2 copies, 6 and 7 get no copies, and the remaining strings get one copy each. Comparing to column A, the expected counts are that strings 5 and 8 get 2 copies, 1 and 6 get no copies, and the remaining get one copy. Column A and E reveal that the theoretical expected count and the true count of each string more or less agree with each other.

Table 8.10 Roulette-wheel selection

Population

β

Population

Population

i

A

B

C

D

E

No.

$= p$

θ

i

1

$\theta 2$

$\theta 1$

$\theta 2$

1

0000

0000

0.0429

0.33

0.0429

0.259

3

1

0000

0000

2

0010

0001

0.090

0.72

0.1329

0.038

1

1

0010

0001

3

0001

0101

0.1336

1.064

0.266

0.486

5

1

0001

0101

4

0010

1000

0.1723

1.368

0.438

0.428

4

2

0010

1000

5

0110

1010

0.200

1.6

0.638

0.095

2

2

0010

1000

6

1110

1000

0.082

0.656

0.720

0.3

4

0

0110

1010

7

1110

1101

0.0829

0.664

0.809

0.616

5

0

0110

1010

8

0111

1100

0.1955

1.56

1.0

0.897

8

1

0111

1100

PI = Probability

D = String number

$$\left(-\frac{E}{kT} \right)$$

A = Expected count

E = The count in the mating pool

B = Cumulative probability

C = Random number between 0–1

Figure 8.13(a) shows the initial random population and the mating pool after reproduction. The points marked with enclosed box are the points in the mating pool and the points marked with a filled box show the population left out in the pool. The action of the reproduction operator is clear from this point. The inferior points have been probabilistically eliminated from further consideration. It should also be noted that not all selected points are better than rejected points. For example, first individual is selected whereas the sixth individual is not selected. Although the above Roulette-wheel selection is easier to implement, it is noisy. A better stable version of the selection operator is sometimes used. After the expected count for each individual string is calculated, the strings are first assigned value exactly equal to the mantissa of the expected count. Thereafter, the regular Roulette-wheel selection is implemented using decimal part of the expected count of the probability distribution. This selection method is less noisy and is known as *stochastic remainder selection*.

8.7.2 Boltzmann Selection

Simulated annealing is a method of functional minimization or maximization.

This method simulates the process of slow cooling of molten metal to achieve the minimum function value in a minimization problem. The cooling phenomenon is simulated by controlling a temperature like parameter introduced with the concept of Boltzmann probability distribution so that a system in thermal equilibrium at a temperature T has its energy distributed probabilistically according to

$$P(E) = \exp$$

(8.26)

where ‘ k ’ is Boltzmann constant. This expression suggests that a system at a high temperature has almost uniform probability of being at any energy state, but at a low temperature it has a small probability of being at a high energy state. Therefore, by controlling the temperature T and assuming search process follows Boltzmann probability distribution, the convergence of the algorithm is controlled. This is beyond the scope of this book and the reader is advised to refer to the book by Deb (1995).

8.7.3 Tournament Selection

GA uses a strategy to select the individuals from population and insert them into a mating pool. Individuals from the mating pool are used to generate new offspring, which are the basis for the next generation. As the individuals in the mating pool are the ones whose genes will be inherited by the next generation, it is desirable that the mating pool consists of good individuals. A selection strategy in GA is simply a process that favours the selection of better individuals in the population for the mating pool.

There are two important issues in the evolution process of genetic search, population diversity and selective pressure, as given by Whitley (1989).

Population diversity means that the genes from the already discovered good individuals are exploited while promising the new areas of the search space continue to be explored.

Selective pressure is the degree to which the better individuals are favoured.

The higher the selective pressure the more, the better individuals are favoured. The selective pressure drives GA to improve population fitness over succeeding generations. The convergence rate of GA is largely determined by the selective pressure and population diversity. In general, higher selective pressure results in higher convergence rates. However, if the selective pressure is too high, there is an increased chance of GA prematurely converging to local optimal solution because the population diversity of the search space to be exploited is lost.

If the selective pressure is too low, the convergence rate will be slow and the GA will take unnecessarily long time to find the optimal solution because more genes are explored in the search. An ideal selection strategy should be such that it is able to adjust its selective pressure and population diversity so as to fine-tune GA search performance.

Whitley (1989) pointed out that the fitness proportional selection (e.g.

Roulette-wheel selection) is likely to lead to two problems, namely Stagnation of search because it lacks selection pressure, and

Premature convergence of the search because it causes the search to narrow down too quickly.

Unlike the Roulette-wheel selection, the tournament selection strategy provides selective pressure by holding a tournament competition among NU individuals (Frequency of $NU = 2$) (Goldberg and Deb, 1991).

The best individual (the winner) from the tournament is the one with highest fitness φ which is the winner of NU . Tournament competitors and the winner are then inserted into the mating pool. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising of tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of succeeding genes. The following steps illustrate the tournament selection strategy (see

Table 8.11) and the fitness values are taken from Table 8.8.

Table 8.11 Fitness values for individuals

Individuals

1

2

3

4

5

6

7

8

Fitness

1

2.10

3.11

4.01

4.66

1.91

1.93

4.55

Step 1: First select individuals 2 and 4 at random.

φ_2 φ_4

2.10 4.01

4 is the winner and hence, select the string as 0010 1000.

Step 2: Select individuals 3 and 8 at random.

φ_3 φ_8

3.11 4.55

8 is the winner and hence, select the string as 0111 1100.

Step 3: Next select 1 and 3.

φ_1 φ_3

1.0 3.11

3 is the winner and thus, select the third string as 0001 0101.

Similarly, other populations are selected from the mating pool as Individuals

Selected

4 and 5

5

1 and 6

6

1 and 2

2

4 and 2

4

8 and 3

8

From the above, it is clear that 2, 3, 5 and 6 are chosen only once 4, 8 are chosen twice, and 1 and 7 are not chosen at all . Table 8.12 gives the new mating pool.

Table 8.12 Population for mating pool

Population no.

Population

1

0010

1000

2

0111

1100

3

0001

0101

4

0110

1010

5

1110

1000

6

0010

0001

7

0010

1000

8

0111

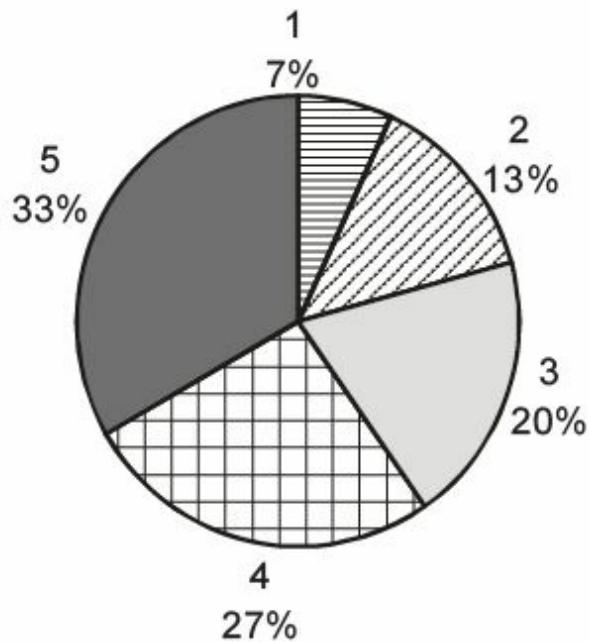
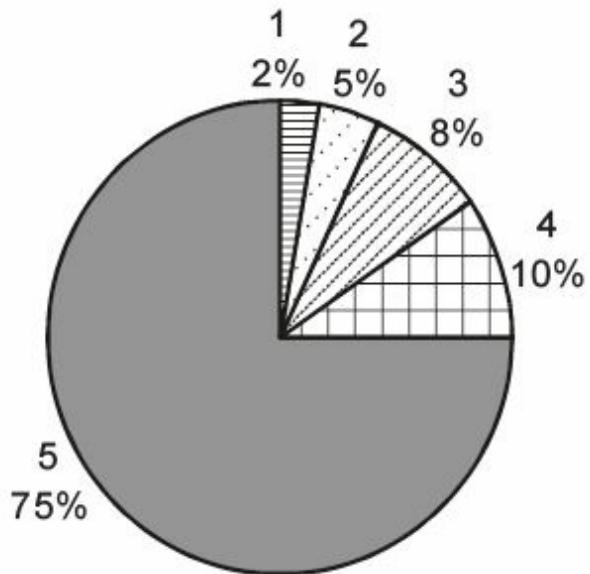
1100

Roulette-wheel selection omitted populations 6 and 7, two copies of 4 and 5, and single copy for the others whereas tournament selection omitted 1 and 7, two copies for 4 and 8, and single copy for the others.

During the early genetic evolution process, there are a large number of individuals or chromosomes that almost satisfy all constraints except one or two. A change in one or two design variable (strings) may produce a solution with a higher fitness value. This means throwing out these solutions may result in a loss of some important information which might eventually lead to optimal solution.

8.7.4 Rank Selection

The Roulette-wheel will have problem when the fitness values differ very much. For example, if the best chromosome fitness is 90%, its circumference occupies 90% of Roulette-wheel, then other chromosomes will have very few



chances to be selected. *Rank selection* first ranks the population and taken every chromosome, receives fitness from the ranking. The worst will have fitness 1, the next 2, ..., and the best will have fitness N (N is the number of chromosomes in the population). The Roulette-wheel selection is applied to the modified wheel as shown in Figs. 8.15 and 8.16. Figure 8.15 is according to fitness and Fig. 8.16 is according to rank. The method can lead to slow convergence because the best chromosome does not differ so much from the other.

Fig. 8.15 Roulette-wheel according to fitness.

Fig. 8.16 Roulette-wheel according to rank.

8.7.5 Steady-state Selection

This is not a particular method of selecting the parents. The main idea of the selection is that bigger part of chromosome should survive to next generation.

Here, GA works in the following way. In every generation are selected, a few

\bar{F}

\bar{F}

(good individuals with high fitness for maximization problem) chromosomes, for creating new off springs. Then, some (bad with low fitness) chromosomes are removed and new offspring is placed in that place. The rest of population survives a new generation.

8.7.6 Elitism

In this method, first the best chromosome or few best chromosomes are copied to new population. The rest is done in a classical way. Elitism can very rapidly increase the performance of GA because it prevents loosing the best-found solutions. From practical consideration point of view, if F fitness functions are positive and for minimization problem, Goldberg (1989), suggest that the fitness of any i th individual must be subtracted from a large constant, so that all fitness values are non-negative and individuals get fitness values according to their actual merit.

Now, the new expression for fitness becomes

$$\varphi_i = (F_{\max} - F_{\min}) - F_i(X) \quad (8.27) \text{ for minimization problem.}$$

If F_i are positive for maximization problem then $\varphi_i = F_i$. For the example problem it is shown in Table. 8.13.

Table 8.13 Mating pool as per rank selection

(= 2.908)

Population no.

Population

$F = \varphi$

$F/$

Count

Mating pool

1

0000 0000

1

0.38

0

0010 0001

2

0010 0001

2.1

0.812

1

0001 0101

3

0001 0101

3.11

1.203

1

0010 1000

4

0010 1000

4.01

1.55

1

0110 1010

5

0110 1010

4.66

1.802

2

0110 1010

6

1110 1000

1.91

0.738

0

1110 1101

7

1110 1101

1.93

0.746

1

0111 1100

8

0111 1100

4.55

1.760

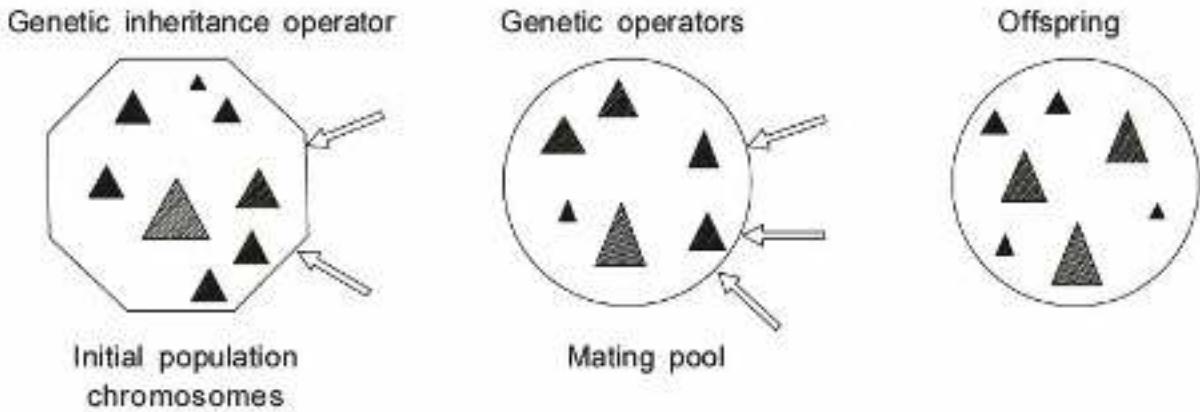
2

0111 1100

The reproduction operator selects fit individuals from the current population and places them in a mating pool. Highly fit individuals get more copies in the mating pool, whereas the less fit ones get fewer copies. As the

$\overline{\phi}$

$\overline{\phi}$



number of individuals in the next generation is also same, the worst fit individuals die off. The reproduction operator can be implemented in the following manner.

The factor ϕ_i for all individuals is calculated, where \bar{f} is the average fitness. This factor is the expected count of individuals in the mating pool, and shown in column 4 of Table 8.13. It is then converted to an actual count by appropriately rounding off so that individuals get copies in the mating pool proportional to their fitness, as shown in Column 5 of Table 8.13. A mating pool is created where individuals 1 and 6 die off. This process of reproduction confirms the Darwinian principle of survival of the fittest.

Figure 8.17 explains how the mating pool is created.

Fig. 8.17 Population for the mating pool.

8.7.7 Generation Gap and Steady-state Replacement

The *generation gap* is defined as the proportion of individuals in the population, which are replaced in each generation. So far, we have been doing reproduction with a generation gap of 1, i.e. population is replaced in each generation. However, a more recent trend has favoured steady-state replacement which is given by Whitley (1987, 1989). This operates at the other extreme and in each generation only a few (typically two) individuals are replaced. This may be a better model of what happens in nature. In

shortlived species including some insects, parents lay eggs and then die before their offsprings hatch. But in longer-lived species including mammal's, offspring and parents live concurrently. This allows parents to nurture and teach their offspring but also gives rise to competition among them.

Generation gap can be classified as

$$\frac{P}{N_p}$$

$$Gp = \quad (8.28)$$

where N_p is the population size and p is the number of individuals that will be replaced. Several schemes are possible. Some of which are:

Selection of parents according to fitness and selection of replacement at random,

Selection of parents at random and selection of replacement by inverse fitness,

Selection of both parents and replacements according to fitness/inverse fitness.

Generation gap can be gradually increased as the evolution takes place to widen exploration space and may lead to better results.

SUMMARY

In this chapter, we have seen that genetic algorithm comprises a set of individuals, elements (the populations) and a set of biologically inspired operators defining the population itself. According to evolutionary theory, only the most suited element in a population is likely to survive and generate offspring, thus transmitting the biological heredity to the new generation. In computing, GA maps problem on to a set of (typically binary) strings, each string representing a potential solution. Table 8.14 gives the comparison between biological terms and the corresponding terms in GA.

Table 8.14 Comparison of biological terms with GA terms

Biological term

GA term

Chromosome

Coded design vector

Substring

Coded design variable

Gene

Every bit

Population

A number of coded design variable

Generation

Population of design vector which are

obtained after one computation

In the next chapter we will discuss inheritance operators, their performance, and the application of GA to real life problems.

Various optimization techniques are illustrated.

Non-traditional search and optimization methods are discussed.

Encoding of variables in GA are given.

Evaluation of fitness functions for an example of two bar pendulum bar is described.

Various selection methods such as Roulette-wheel selection, Boltzmann selection, Tournament selection, Rank selection, Steady-state selection are discussed.

PROGRAMMING ASSIGNMENT

P8.1 In a three variable problem the following variable bounds are specified.

$$-6 < x < 12$$

$$0.002 \leq y \leq 0.004$$

$$104 \leq z \leq 105$$

What should be the minimum string length of any point (x, y, z) coded in binary string to achieve the following accuracy in the solution 1. two significant digits.

2. three significant digits.

P8.2 Repeat the above problem when ternary strings (with three alleles 0, 1, 2) are used instead of binary string.

P8.3 We want to use GA to solve the following nonlinear programming problem.

$$\text{minimize } (x_1 - 2.5)^2 + (x_2 - 5)^2$$

subject to

$$5.5 x$$

$$2$$

$$1 + 2x_2 - 18 \leq 0$$

$$0 \leq x_1, x_2 \leq 5$$

We decide to give three and two decimal places of accuracy to variables x_1 , x_2 respectively.

1. How many bits are required for coding the variables?
2. Write down the fitness function which you would be using in reproduction.

P8.4 Consider the following population of binary strings for a maximization problem.

String

Fitness

01101

5

11000

2

10110

1

00111

10

10101

3

00010

100

Find out the expected number of copies of the best string in the above population of the mating pool under

1. Roulette wheel selection.
2. Tournament selection.

If only the reproduction operator is used, how many generations are required before the best individual occupies the complete population under each selection operator.

P8.5 Write a program in “C” or in FORTRAN for creating initial population (for n variables) with n -bits string for each variable. The values of each variable can be selected from a table of data. Assume an objective function, find fitness value, and get the offspring using Roulette-wheel selection.

REFERENCES

Davis, L. (1991), *Handbook of Genetic Algorithms*, New York, Van Nostrand, Reinhold.

Deb, K. (1991), Optimal Design of a Welded Beam Structure via Genetic Algorithms, *AIAA Journal*, Vol. 29, No. 11, pp. 2013–2015.

Deb, K. (1995), Optimization for Engineering Design—Algorithms and Examples, Prentice-Hall of India, New Delhi.

Dorigo, M. and G.D. Caro (1999), Ant Algorithm for Discrete Optimization, *Artificial Life*, Vol. 5, pp. 137–172.

Goldberg, D.E. and M.P. Samatini (1986), Engineering Optimization via Genetic Algorithm, *Proc. of the Ninth Conference on Electronic Computation*, pp. 471–482.

Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Mass., Addison-Wesley.

Goldberg, D.E. and K. Deb (1991), A Comparative Analysis of Selection Schemes Used in GA, *Foundations of Genetic Algorithms*, I, pp. 53–69.

Holland, J.H. (1975), *Adaptation of Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Jenkins, W.M. (1991), Towards Structural Optimization via the Genetic Algorithms, *Computers and Structures*, Vol. 40, No. 5, pp. 1321–1327.

Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi (1983), Optimization by Simulated Annealing, *Science* 220, pp. 671–680.

Kost, B. and Baumann (1999), Structural Topology Optimization by Stochastic Algorithms, In *Computer Aided Optimum Design of Structures*, WIT Press. Eds., S. Hernandez, A.J. Kassab and C.A. Brebbia, pp. 13–22.

Kost, B. (1995), Evolution Strategies in Structural Topology Optimization of Trusses, Eds.

P.J. Pahl and H. Werner, *Proc of 6th Int. Conf. on Computer in Civil and Building Engg*, A.A. Balkimy, Rotter Dama, pp. 675–681.

Michalewicz, Z. (1992), *Genetic Algorithm + Data Structures = Evolution Program*, Berlin: Springer Verlag.

Rajeev, S. and G.S. Krishnamoorthy (1992), Discrete Optimization of Structures Using Genetic Algorithms, *Jol. of Structural Engineering*, ASCE, Vol. 118, No. 5, pp. 1223–1250.

Whitley, D. (1987), Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery, J.J. Greffenstette (Ed.), *Proc. of Second Intl. Conf. on Genetic Algorithms*, Lawrence Erlbaum Associates, pp.

108–115.

Whitley, D. (1989), The GENITUR Algorithm and Selection Pressure—Why Rank-based Allocation of Reproduction Trials is Best, *Proc. of Int. Conf.*

on Genetic Algorithms, Schaffer, . Morgan Kaufmann Publishers, Los Altos, Cal, pp. 10–19.

Wolfram, S. (1994), *Cellular Automata and Complexity*, First ed., Addison-Wesley,

Chapter 9

Genetic Modelling

In the last Chapter, we discussed how variables in GA are encoded and how fitness function is calculated. Various selection methods such as Roulette-wheel selection, Boltzmann selection, tournament selection, and steady-state selection have been used to produce the population of the mating pool i.e. in reproduction, good strings in a population are probabilistically assigned a large number of copies and a mating pool is formed. It is important to note that no new strings are formed in the reproduction phase. There are many inheritance operators applied to the mating pool with a hope that it would create a better string. The aim of inheritance operators is to search the parameter space. In addition, search is to be in a way that the information stored in the present strings are maximally preserved, because these parent strings are instances of good strings selected using the reproduction operator.

9.1 INHERITANCE OPERATORS

As already seen, genetic algorithm makes use of the Darwinian survival of the fittest procedure. Genetic algorithms are search procedures based on mechanics of natural genetics and natural selection.

Genetic algorithm derives power from the genetic operators listed here.

Low-level operators, namely

1. Inversion
2. Dominance
3. Deletion

4. Intrachromosomal duplication

5. Translocation

6. Segregation

7. Speciation

8. Migration

9. Sharing

10. Mating

A simple genetic algorithm largely uses three basic operators which are 1. Reproduction

2. Cross over

3. Mutation

First we will discuss the basic operators and then the low-level operators.

9.2 CROSS OVER

After the reproduction phase is over, the population is enriched with better individuals. Reproduction makes clones of good strings, but does not create new ones. Cross over operator is applied to the mating pool with a hope that it would create a better string. The aim of the cross over operator is to search the parameter space. In addition, search is to be made in a way that the information stored in the present string is maximally preserved because these parent strings are instances of good strings selected during reproduction.

Cross over is a recombination operator, which proceeds in three steps.

First, the reproduction operator selects at random a pair of two individual strings for mating, then a cross-site is selected at random along the string

length and the position values are swapped between two strings following the cross site. For instance, let the two selected strings in a mating pair be A =

11111 and B = 00000. If the random selection of a cross-site is two, then the new strings following cross over would be $A^* = 11000$ and $B^* = 00111$. This is a single-site cross over. Though these operators look very simple, their combined action is responsible for much of GA's power. From a computer implementation point of view, they involve only random number of generations, string copying, and partial string swapping. There exist many types of cross over operations in genetic algorithm which are discussed in the following sections.

9.2.1 Single-site Cross Over

In a single-site cross over, a cross-site is selected randomly along the length of the mated strings and bits next to the cross-sites are exchanged as shown in Fig. 9.1. If an appropriate site is chosen, better children can be obtained by combining good substances of parents. Since the knowledge of the appropriate site is not known and it is selected randomly, this random selection of cross-sites may produce enhanced children if the selected site is appropriate. If not, it may severely hamper the string quality. Anyway, because of the crossing of parents better children are produced and that will continue in the next generation also. But if good strings are not created by cross over, they will not survive beyond next generation because reproduction will not select those strings for the next mating pool.

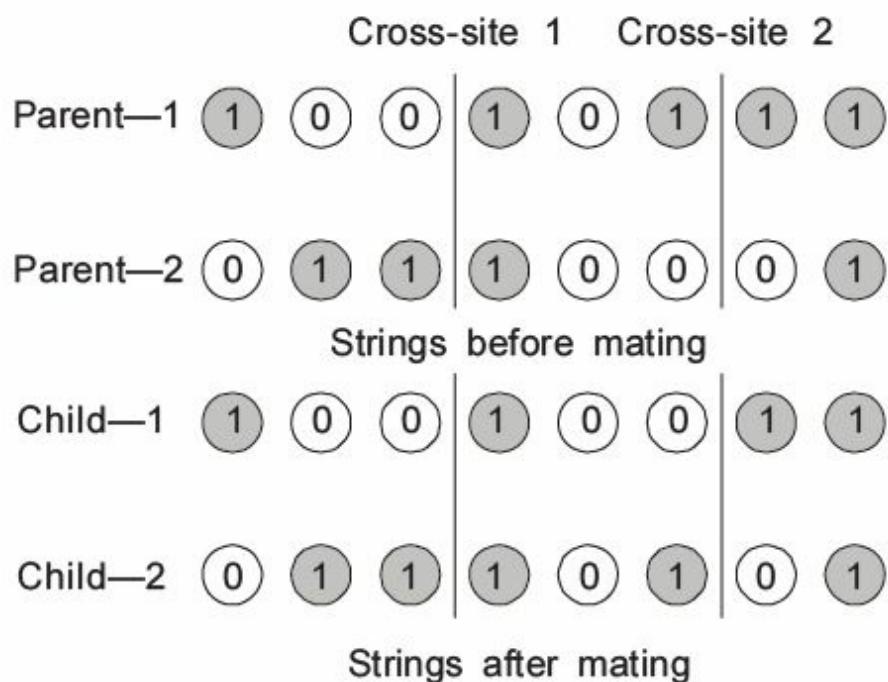
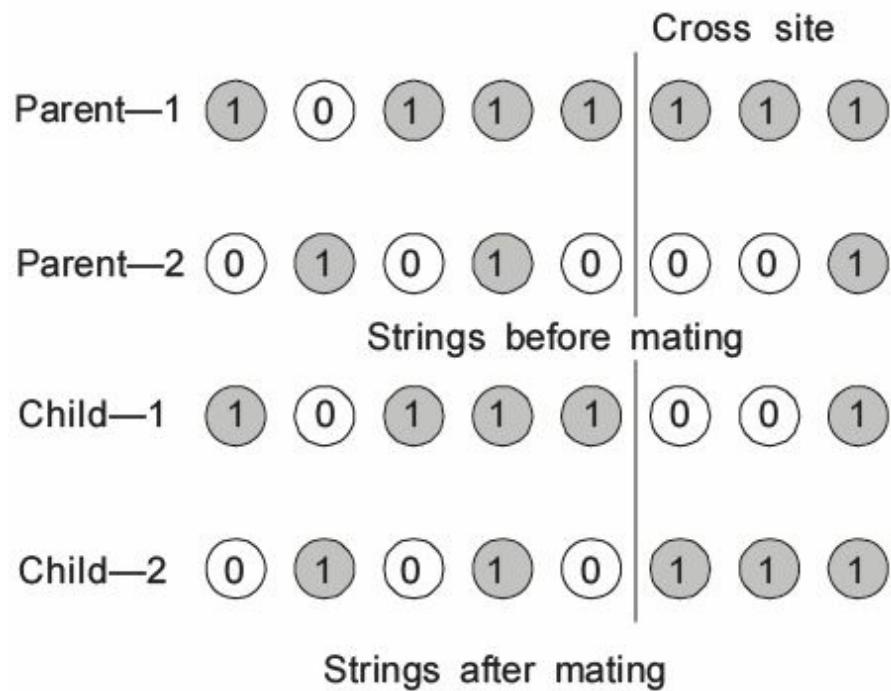


Fig. 9.1 Single-site cross over.

9.2.2 Two-point Cross Over

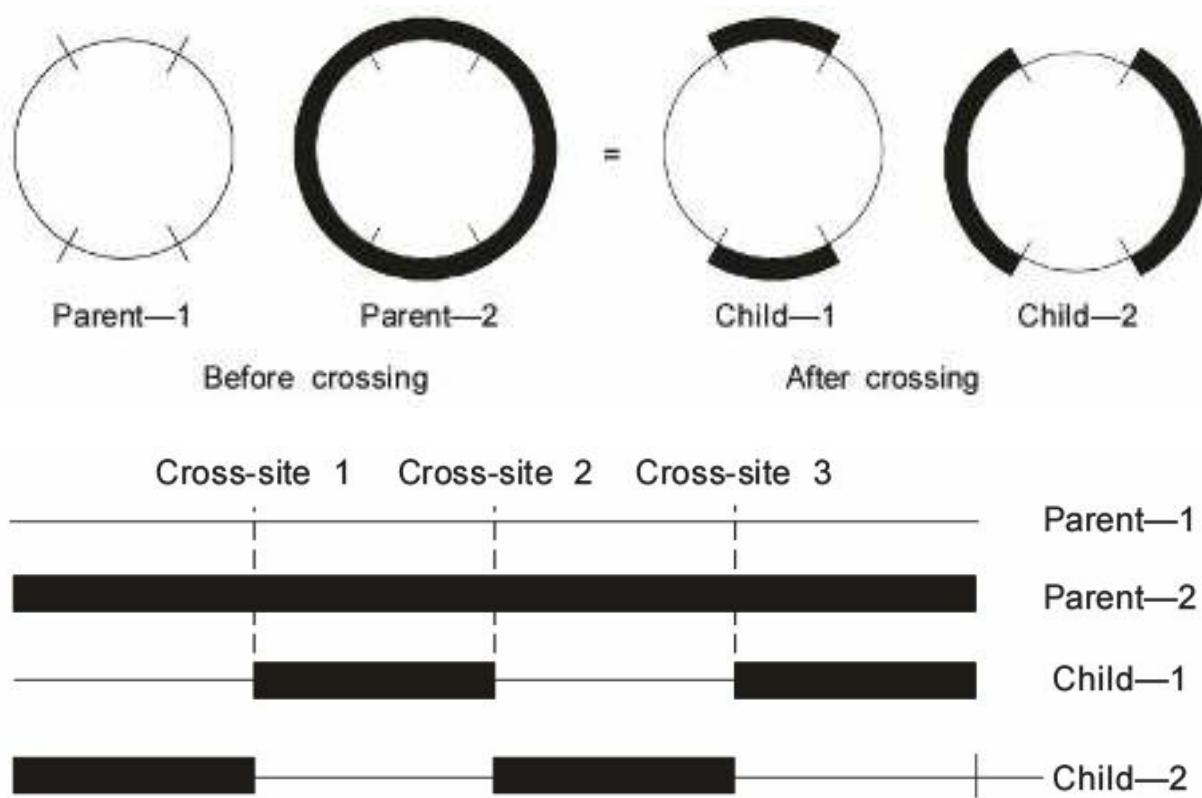
In a two-point cross over operator, two random sites are chosen and the contents bracketted by these sites are exchanged between two mated parents.

If the cross-site 1 is three and cross-site 2 is six, the strings between three and six are exchanged as shown in Fig. 9.2.

Fig. 9.2 Two-point cross over.

9.2.3 Multi-point Cross Over

In a multi-point cross over, again there are two cases. One is even number of cross-sites and second one is the odd number of cross-sites. In case of even



numbered cross-sites, the string is treated as a ring with no beginning or end.

The cross-sites are selected around the circle uniformly at random. Now the information between alternate pairs of sites is interchanged as shown in Fig. 9.3. If the number of cross-sites is odd, then a different cross-point is always

assumed at the string beginning. The information (genes) between alternate pairs is exchanged as shown

in Fig. 9.4.

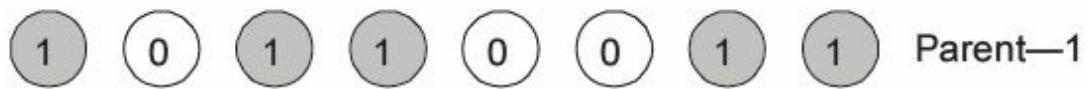
Fig. 9.3 Multi-point cross over with even number of cross-sites.

Fig. 9.4 Multi-point cross over with odd number of cross-sites.

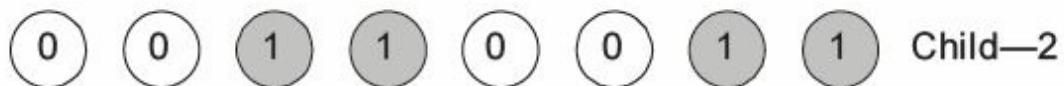
9.2.4 Uniform Cross Over

An extreme of multi-point cross over is the uniform cross over operator. In a uniform cross over operator, each bit from either parent is selected with a probability of 0.5 and then interchanged as shown in Fig. 9.5(a). It is seen that uniform cross over is radically different from one-point cross over.

Sometimes gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a randomly generated cross over mask. When there is 1 in the mask, the gene is copied from the first parent and when there is 0, the gene is copied from second parent as shown in Fig. 9.5(b). The process is repeated with the parents exchanged to

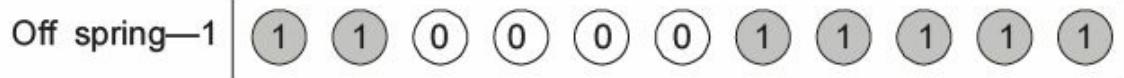


Before crossing



Interchange Interchange Interchange

Cross over mask 1 0 0 1 0 1 1 1 0 0 1



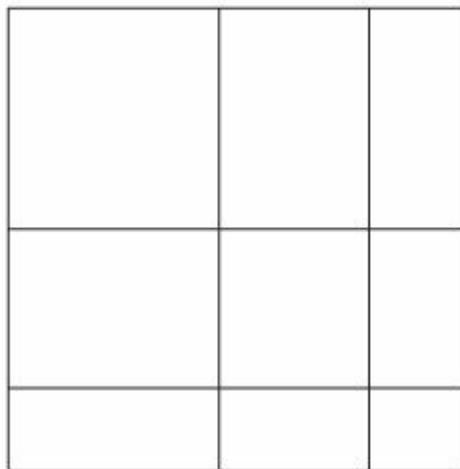
produce the second offspring. A new cross over mask is randomly generated for each pair of parents. Offspring therefore contains a mixture of genes from each parent. The number of effective crossing points is not fixed but averages to $L/2$ (where L is chromosome length).

Fig. 9.5(a) Uniform cross over.

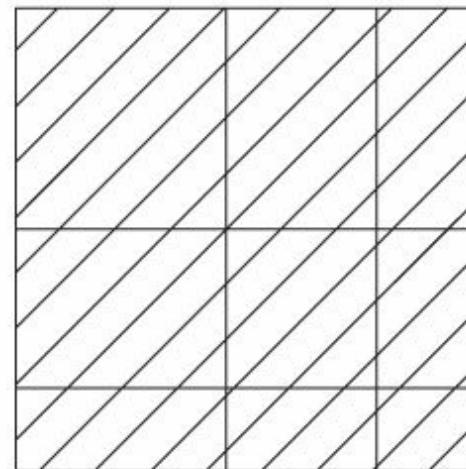
Fig. 9.5(b) Uniform cross over using mask.

9.2.5 Matrix Cross Over (Two-dimensional Cross Over)

String—1	1	0	1	1	1	0	0	1
String—2	0	1	0	1	1	1	1	0
Substring—1					Substring—2			

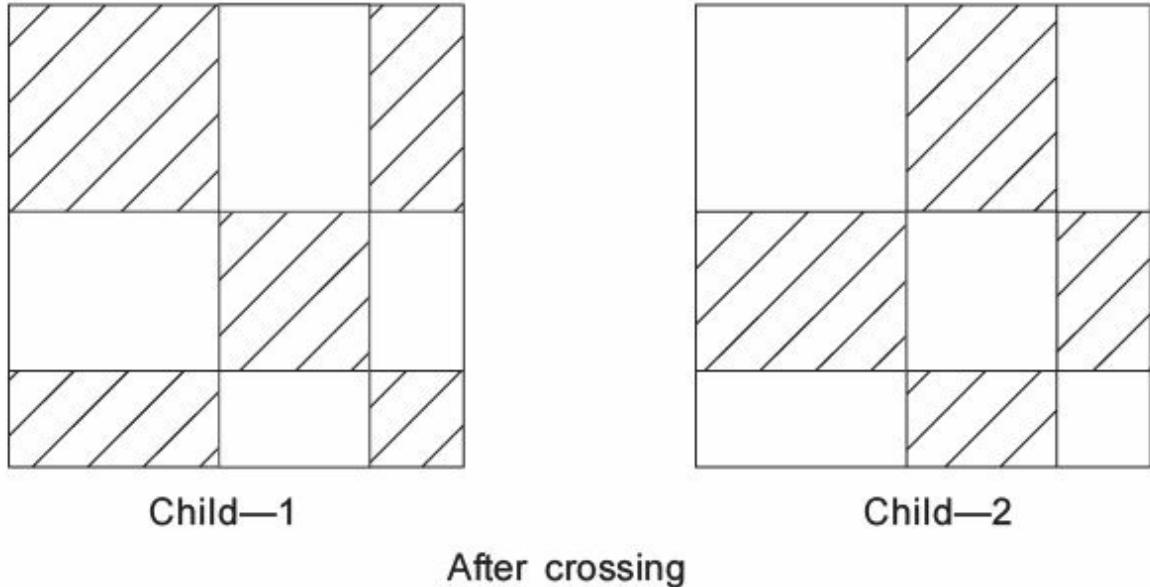


Parent—1



Parent—2

Before crossing



Normally, the strings are represented as a single dimensional array as shown in Fig. 9.6. In the above case, two strings of length 4 are concatenated to form an individual. So, the cross-sites selected for this case are obviously single-dimensional whereas in the case of two-dimensional cross over, each individual is represented as a two-dimensional array of vector to facilitate the process. The process of two-dimensional cross over is depicted in Fig. 9.7.

Fig. 9.6 Single-dimensional strings.

Fig. 9.7 Matrix cross over.

Two random sites along row and column are chosen, then the string is divided into utmost nonoverlapping rectangular regions. Two cross-sites, both row- and column-wise, will decide each individual into utmost nine overlapping rectangular regions. Two cross-sites, both row- and column-wise will decide each individual into three layers horizontally and vertically.

Select any region in each layer, either vertically or horizontally and then exchange the information in that region between the mated populations. The selection of cross over operators is made such that the search in genetic space is proper. In case of a single-point cross over operator, the search is not extensive but maximum information is preserved between parents and

children. Some studies have been made to find an optimal cross over operator. According to Deb (1995), it is difficult to generalize the optimal cross over operator selection. So, it is left to personal interest to select the cross over operator.

9.2.6 Cross Over Rate

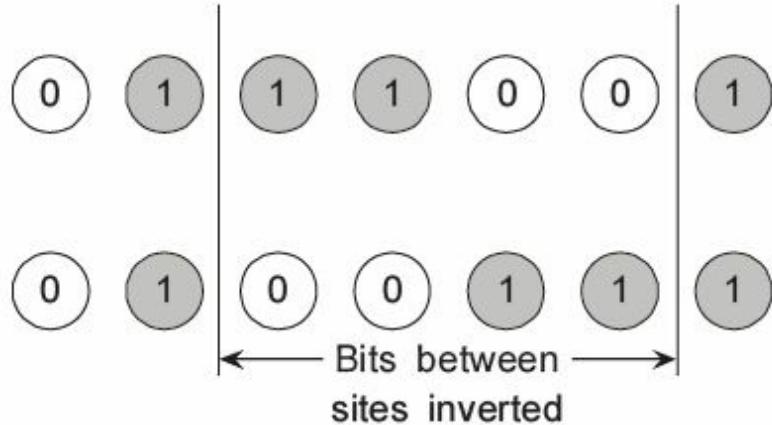
In GA literature, the term cross over rate is usually denoted as PC , the probability of cross over. The probability varies from 0 to 1. This is calculated in GA by finding out the ratio of the number of pairs to be crossed to some fixed population. Typically for a population size of 30 to 200, cross over rates are ranged from 0.5 to 1.

We have seen that with random cross-sites, the children strings produced may not have a combination of good substrings from parent strings depending on whether or not the crossing site falls in the appropriate place.

But we do not worry about this too much because if good strings are created by cross over, there will be more copies of them in the next mating pool generated by the reproduction operator. But if good strings are not created by cross over, they will not survive too long, because reproduction will select against those strings in subsequent generations. It is clear from this discussion that the effect of cross over may either be detrimental or beneficial. Thus, in order to preserve some of good strings that are already present in the mating pool, not all strings in the mating pool are used in cross over.

When a cross over probability of PC is used only $100 PC$ percent strings in the population are used in the cross over operation and $100(1 - PC)$ percentage of the population remains as it is in the current population. Even

though the best $100(1 - PC)\%$ of the current population can be copied deterministically to the new population, this is usually preferred at random. A cross over operation is mainly responsible for the search of new strings.



9.3 INVERSION AND DELETION

9.3.1 Inversion

A string from the population is selected and the bits between two random sites are inverted as shown in Fig. 9.8.

Fig. 9.8 Inversion.

Linear+end-inversion

Linear+end-inversion performs linear inversion with a specified probability of 0.75. If linear inversion was not performed, the end inversion would be performed with equal probability of 0.125 at either the left or right end of the string. Under end inversion, the left or right end of the string was picked as one inversion-point and a second inversion-point was picked uniformly at random from the point no farther away than one half of the string length.

Linear+end-inversion minimizes the tendency of linear inversion to disrupt bits located near the centre of the string disproportionately to those bits located near the ends.

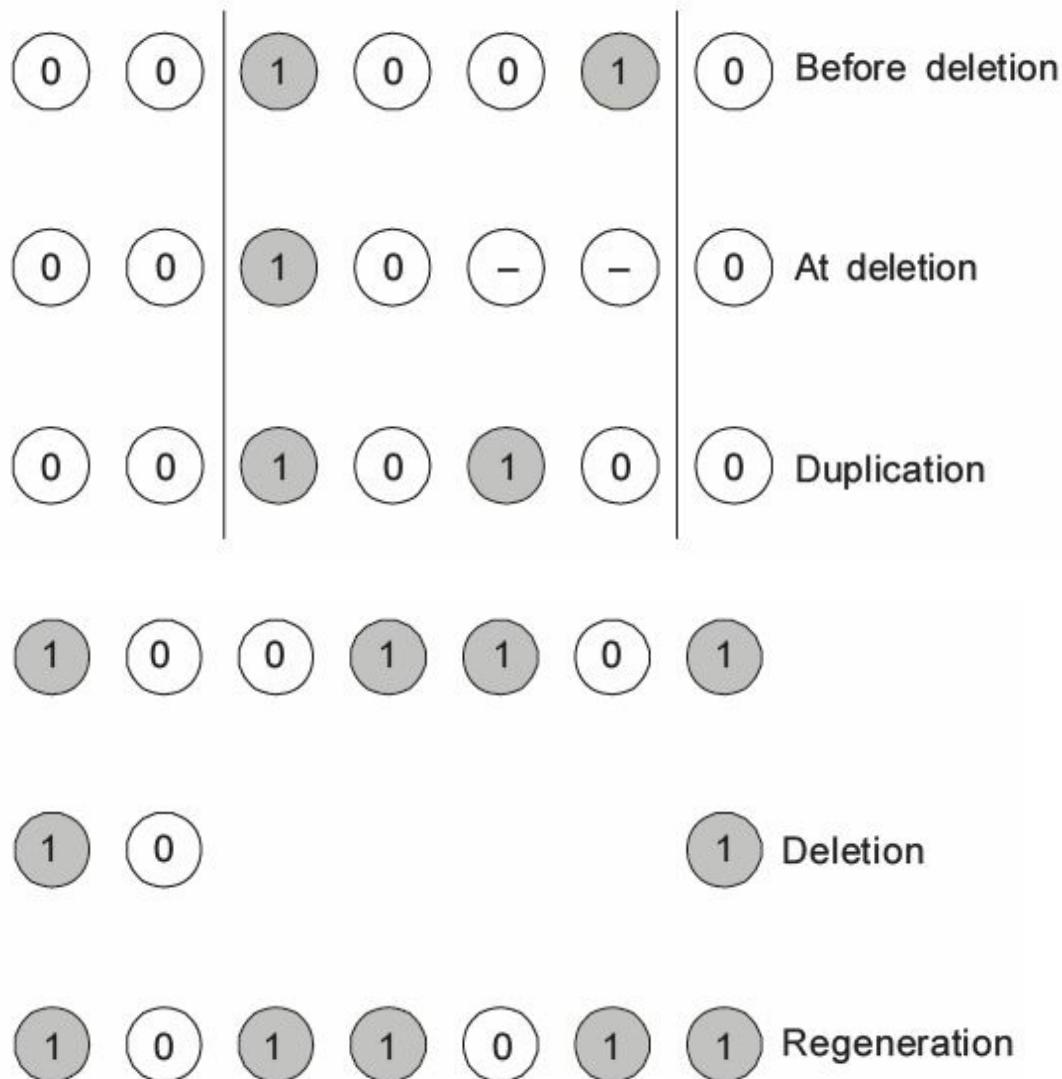
Continuous inversion

In *continuous inversion*, inversion was applied with specified inversion probability Pr to each new individual when it is created.

Mass inversion

No inversion takes place until a new population is created and thereafter, one-half of the population undergoes identical inversion (using the same two inverting points).

9.3.2 Deletion and Duplication



Any two or three bits at random in order are selected and the previous bits are duplicated and it is shown in Fig. 9.9.

Fig. 9.9 Deletion and duplication.

9.3.3 Deletion and Regeneration

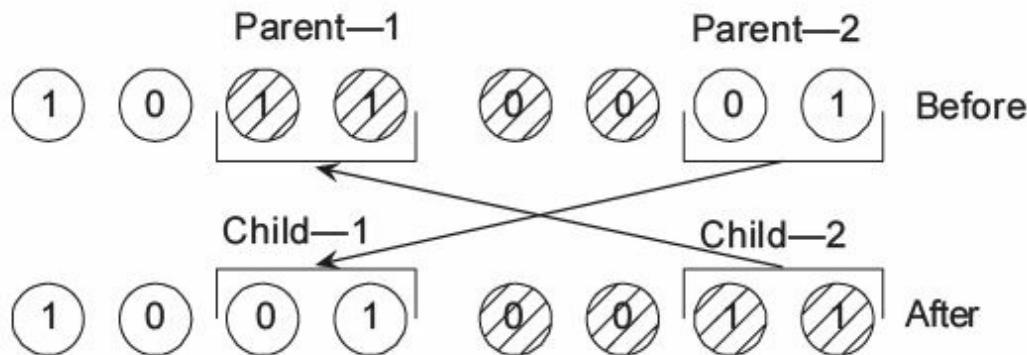
Genes between two cross-sites are deleted and regenerated randomly as shown in Fig. 9.10.

Fig. 9.10 Deletion and regeneration.

9.3.4 Segregation

The bits of the parents are segregated and then crossed over to produce offspring as shown in

Fig. 9.11.



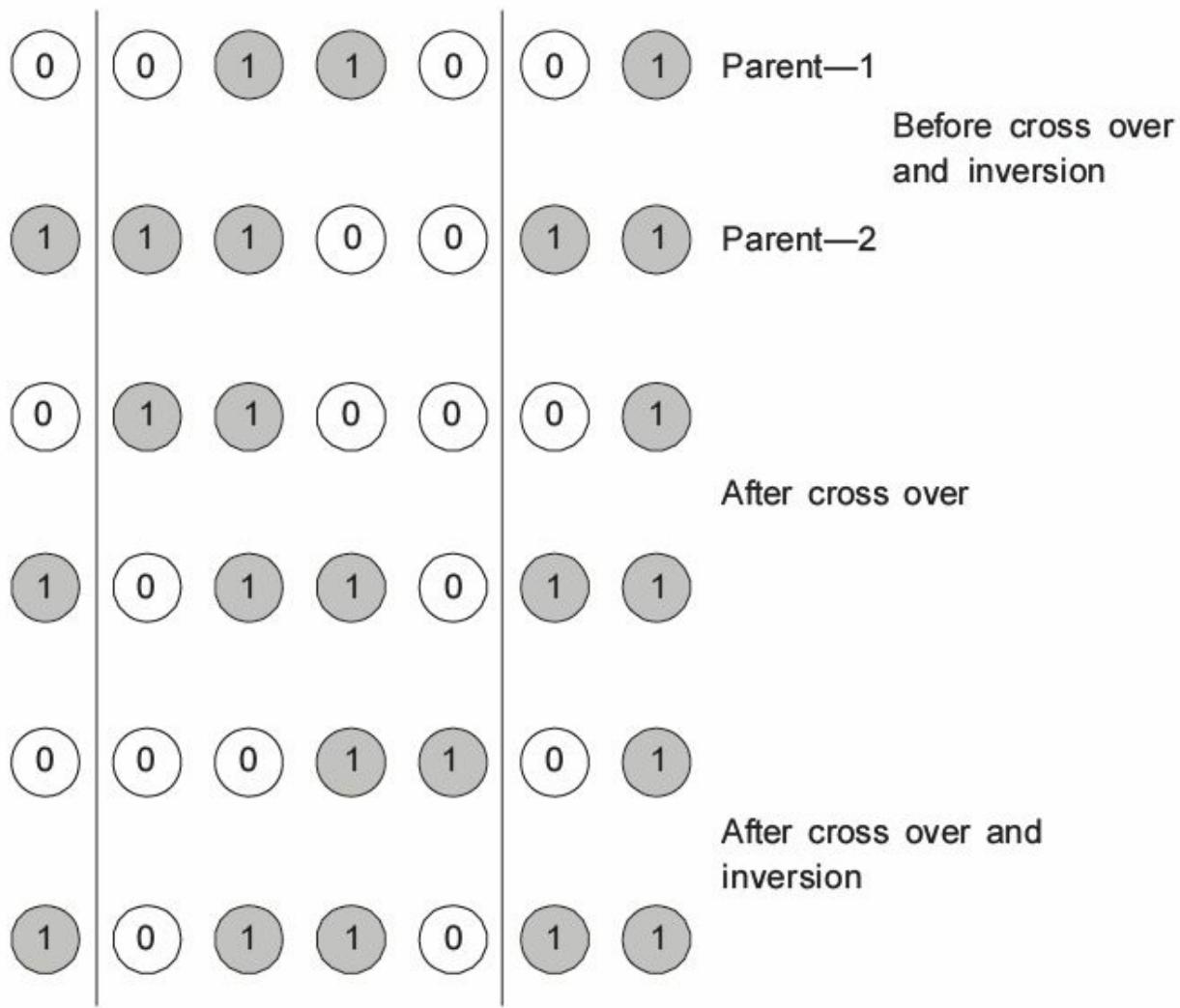


Fig. 9.11 Segregation.

9.3.5 Cross Over and Inversion

Cross over and inversion operator is the combination of both cross over and inversion operators. In this, two random sites are chosen, the contents bracketted by these sites are exchanged between two mated parents and, the end points of these exchanged contents switch place. For example, if the cross-sites in parents shown in Fig. 9.12 are 2 and 7, the cross over and inversion operation is performed in the way shown in Fig. 9.12.

Fig. 9.12 Cross over and inversion.

9.4 MUTATION OPERATOR

9.4.1 Mutation

After cross over, the strings are subjected to mutation. *Mutation* of a bit involves flipping it, changing 0 to 1 and vice versa with a small mutation probability P_m . The bit-wise mutation is performed bit-by-bit by flipping a coin with a probability of P_m . Flipping a coin with a probability of P_m is simulated as follows.

A number between 0 to 1 is chosen at random. If the random number is smaller than P_m then the outcome of coin flipping is true, otherwise the outcome is false. If at any bit, the outcome is true then the bit is altered, otherwise the bit is kept unchanged. The bits of the strings are independently muted, that is, the mutation of a bit does not affect the probability of mutation of other bits. A simple genetic algorithm treats the mutation only as a secondary operator with the role of restoring lost genetic materials.

Suppose, for example, all the strings in a population have conveyed to a zero at a given position and the optimal solution has a one at that position, then cross over cannot regenerate a one at that position while a mutation could. The mutation is simply an insurance policy against the irreversible loss of genetic material.

The mutation operator introduces new genetic structures in the population by randomly modifying some of its building blocks. It helps the search algorithm to escape from local minima's traps since the modification is not related to any previous genetic structure of the population. It creates different structure representing other sections of the search space. The mutation is also used to maintain diversity in the population. For example, consider the following population having four eight-bit strings.

0110 1011

0011 1101

0001 0110

0111 1100

Notice that all four strings have a zero in the leftmost bit position. If the true optimum solution requires a one in that position, then neither

reproduction nor cross over operator described above will be able to create one in that position. The inclusion of mutation introduces some probability (Npm) of turning zero to one as

0110 1011

0011 1101

0001 0110

1111 1100

Mutation for real numbers can be done as

Before (1.38 -69.4 326.44 0.1)

After (1.38 -67.5 326.44 0.1)

Hence, mutation causes movement in the search space (local or global) and restores lost information to the population.

9.4.2 Mutation Rate Pm

Mutation rate is the probability of mutation which is used to calculate number of bits to be muted. The mutation operator preserves the diversity among the population which is also very important for the search. Mutation probabilities are smaller in natural populations leading us to conclude that mutation is appropriately considered a secondary mechanism of genetic algorithm adoption. Typically, the simple genetic algorithm uses the population size of 30 to 200 with the mutation rates varying from 0.001 to 0.5.

	θ_1	θ_2
$a = 0100\ 0001 \rightarrow$	4	1
$24^\circ\ 16^\circ$		
$\sim a = 1011\ 1110 \rightarrow$	11	14
$66^\circ\ 84^\circ$		

9.5 BIT-WISE OPERATORS

Usually, binary coding is used more extensively in the coding mechanism to generate algorithm structure. This involves the coding of real variables to binary strings and genetic operators work on these coded strings. In the present work, genetic algorithm program as written in “C” language with the help of built in operators in “C” (the bit-wise operators), we can directly manipulate the individual bits within a word of memory. These operators can be carried out easily and efficiently. These can operate on integers and characters but not on floats and doubles. Byron S. Gotfried (1990) categorizes bit-wise operators into three, namely, 1. The one’s complement operator,

2. The logical bit-wise operator, and 3. The shift operator.

9.5.1 One’s Complement Operator

The one’s complement operator (\sim) is an unary operator that causes the bits of its operand to be inverted (i.e. reversed), so that 1 becomes zero and zero becomes 1. This operator always precedes its operand.

Consider the example problem 8.1, where two variables θ_1, θ_2 , are four-bit strings each respectively and hence, the total string length is eight.

9.5.2 Logical Bit-wise Operators

There are three logical bit-wise operators, namely, 1. Bit-wise AND, 2. Bit-wise Exclusive-OR, and 3. Bit-wise OR.

Each of these operators require two integer-type operands and hence, can be used instead of cross over. While operating upon two operands, they are

compared on bit by bit basis. The truth table is shown in Table 9.1.

Bit-wise AND (&) operator

A bit-wise AND (&) expression returns 1 if both the bits have a value 1, otherwise it returns a value 0.

Parent 1a = 1010 1010 \rightarrow 10

Parent 2b = 1100 0011 \rightarrow 12

Child a&b = 1000 0010 \rightarrow 8

Table 9.1 Truth table

AND '&'

Exclusive OR

OR '|'

a

b

operator

' \wedge ' operator

operator

a&b

a \wedge b

a | b

0

0

0

0

0

0

1

0

1

1

1

0

0

1

1

1

1

1

0

1

Bit-wise exclusive-OR (\wedge) operator

A bit-wise exclusive-OR (\wedge) expression returns a 1 if one of the bits have a value of 1 and the other has a value of 0 otherwise it returns a value 0.

Parent 1a = 1010 1010 \rightarrow 10 10

Parent 2b = 1100 0011 \rightarrow 12 3

Child a&b = 0110 1001 \rightarrow 6 9

Bit-wise OR (|) operator

A bit-wise OR (|) expression returns a 1 if one or more bits have a value of 1 otherwise it returns a value 0.

Parent 1a = 1010 1010 \rightarrow 10 10

Parent 2b = 1100 0011 \rightarrow 12 3

Child a&b = 1110 1011 \rightarrow 13 11

The three bit-wise operators are summarized in Table 9.1. In this Table, A and B represent the corresponding bits within the first and second operands respectively.

9.5.3 Shift Operators

Two bit-wise shift operators are, *shift left* ($<<$) and *shift right* ($>>$) operators.

Each operator operates on a single variable but requires two operands. The first operand is an integer type operand that represents the bit pattern to be shifted and the second is an unsigned integer that indicates the number of displacements (i.e. whether the bits in the first operand will be shifted by 1 bit position, 2 bit position and so on). This value cannot exceed the number of bits associated with the word size of the first operand.

Shift left operator ($<<$)

The shift left operator causes all the bits in the first operand to be shifted to the left by the number of positions indicated by the second operand. The leftmost bits (i.e. the overflow bits) in the original bit pattern is lost. The rightmost bit positions that become vacant are to be filled with zeroes.

$$a = 1010\ 0110 \rightarrow \quad 10 \quad 6$$

$$a << 2 = 1001\ 1000 \rightarrow \quad 9 \quad 8$$

Shift right operator ($>>$)

The shift right operator causes all the bits in the first operand to be shifted to the right by the number of positions indicated by the second operand. The right most bits (i.e. the underflow bits) in the original bit pattern are lost. The left most bit positions that become vacant are then filled with zeroes.

$$a = 1010\ 0110 \rightarrow \quad 10 \quad 6$$

$$a >> 2 = 0010\ 1001 \rightarrow \quad 2 \quad 9$$

Masking

Masking is a process in which a given bit pattern is transformed into another bit pattern by means of logical bit-wise operation. The original bit pattern is one of the operands in the bit-wise operation. The second operand called *mask*, is a specially selected bit pattern that brings about the desired transformation.

There are several different kinds of masking operations. For example, a portion of a given bit pattern can be copied to a new word, while the remainder of the new word is filled with 0. Thus, part of the original bit pattern will be “masked off” from the final result.

9.6 BIT-WISE OPERATORS USED IN GA

Logical bit-wise operators are used in different combinations. Each operator operates on two individuals and generates one resultant so as to keep the number of individuals in the population constant. Two different operators are used in GA process.

Populations are selected randomly for mating and on each pair bit-wise AND and bit-wise OR operators are performed. Similarly, AND and exclusive-OR or OR and exclusive-OR operations can be performed to produce children or population for the next generation.

9.7 GENERATIONAL CYCLE

Table 9.2 shows a *generational cycle* of the genetic algorithm with a population of four ($P_1 = 4$) strings with 10 bits each. In this example, the objective functions which can assume values in the string 0 to 10, give the number of 1s in the decimal place. The fitness function performs “divide by 10” operation to normalize the objective function in the range of 0 to 1. The four strings thus have fitness values of 0.3, 0.6, 0.6, and 0.9. Ideally, the proportional selection scheme should allocate $0.5(0.3/0.6)$, $1.0(0.6/0.6)$, $1.0(0.6/0.6)$, and $1.5(0.9/0.6)$ values for selection to be offspring (since $f(\text{average}) = (0.3 + 0.6 + 0.6 + 0.9)/4 = 0.6$) to the strings. However, according to Darwinian theory of survival of the fittest, the strongest individual will have two copies, the weakest individual dies, and average individuals will have one copy each. Hence, the string with fitness value of 0.5 will have 0 copy with 1.5 has two copies and others 1 copy. In Table 9.2, the population P_2 represents this selected set of strings. Next, the four strings are paired randomly for cross over. Strings 1 and 4 forms one pair, and 2 and 3 forms the other pair. At a cross over probability rate of 0.5, only the pair 2 and 3 is left intact. The cross over point falls between 1 and 5 and hence, portion of the strings between 1 and 5 are swapped.

The action of mutation on population P_3 can be seen in population P_4 on the sixth bit of string 2 and the first bit of string 4. Only two bits out of 40

have muted representing an effective mutation probability rate of 0.05.

Population P4 represents the next generation. In effect, P1 and P4 are the populations while P2 and P3 represent the intermediate stages in the generational cycle.

The parameters, namely the population size, mutation rate, and cross over rate are together referred to as the *control parameters* of the simple genetic algorithm and must be specified before its execution.

To terminate the execution of simple genetic algorithm, we must specify a stopping criterion. It could be terminated after a fixed number of generations, after a string with a certain high fitness value is located or after all the strings in the populations have attained a degree of homogeneity (a large number of strings have identical bits at most positions).

Table 9.2 A generational cycle of the simple genetic algorithm

Population P1				
String	f = Fitness	$f/f(av)$	Copy	
00000 11100	0.3	0.5	0	
10000 11111	0.6	1.0	1	
01101 01011	0.6	1.0	1	
11111 11011	0.9	1.5	2	

Population P2 after reproduction				
String	f = Fitness	Cross over	CS1	CS2
10000 11111	0.6	4	1	5
01101 01011	0.6	—		
11111 11011	0.9	—		
11111 11011	0.9	1	1	5

Population P3 after cross over	
String	f = Fitness
<u>1</u> 1111 11111	1.0
01101 <u>0</u> 1011	0.6
11111 11011	0.9
10000 11011	0.5

Population P4 after mutation	
String	f = Fitness
01111 11111	0.9
01101 11011	0.7
11111 11011	0.9
10000 11011	0.5

\bar{F}_i

Example Problem

Consider Example 8.1 (Two bar pendulum) using cross over and mutation.

We have seen various selection procedures of obtaining the populations for the mating pool as given in Table 8.13 and as given in Table 9.3). One procedure is described here.

Table 9.3 Generation of population ($F = 3.95$)

Population

Population

Random

Population

Pop.

Mate

F

Actual

Population

CS1

CS2

after cross

bits for

for next

i

Fi/

no.

with

$= \varphi$

count

2

4

5

over

mutation

generation

1

3

6

7

8

9

10

11

1

0010 0001

5

2

6

0010 1001

0010 1001

4.11

1.04

1

0010 1001

2

0001 0101

6

1

5

0110 1101

0110 1101

4.53

1.14

2

0110 1101

3

0010 1000

7

4

8

0010 1100

21

0010 0100

3.15

0.796

0

0110 1101

4

0110 1010

8

4

6

0110 1110

0110 1110

4.40

1.11

1

0110 1110

5

0110 1010

1

2

6

0110 0010

0110 0010

3.00

0.759

0

0111 1100

6

1110 1101

2

1

5

1001 0101

1001 0101

3.49

0.883

1

1001 0101

7

0111 1100

3

4

8

0111 1000

0111 1000

4.44

1.12

1

0111 1000

8

0111 1100

4

4

6

0111 1000

62

0111 1100

4.55

1.15

2

0111 1100

Step 1: Randomly select eight populations of eight-bit strings and decode the population for angles, and substituting in the potential expression find the fitness function.

Step 2: Use any of the selection methods discussed in Chapter 8 to get the population for the mating pool.

(The above two steps have been performed in Chapter 8 and the column 8 of Table 8.13 gives the populations for the mating pool.)

Step 3: Randomly select the parents for the mating pool such as 1 with 5, 2 with 6, 3 with 7, and 4 with 8.

Step 4: Two-point cross over is selected such that bits between strings 2–6 of the population parents 1 and 5, are swapped. The population after cross over is shown in Table 9.3 in the sixth column. We used the cross over probability of 100% in all the parent pairs that are crossed .

Figure 9.13 shows how points cross over and form new points. The points marked with small boxes are the points in the mating pool and the points marked with small circles are children points created after cross over operation. The complete population at the end of cross over operation is shown as last column in Table 9.3. Figure 9.13 shows that some good points and some not-so-good points are created after cross over. In some cases, points far away from the parent points are created and in some cases, points close to the parent points are created.

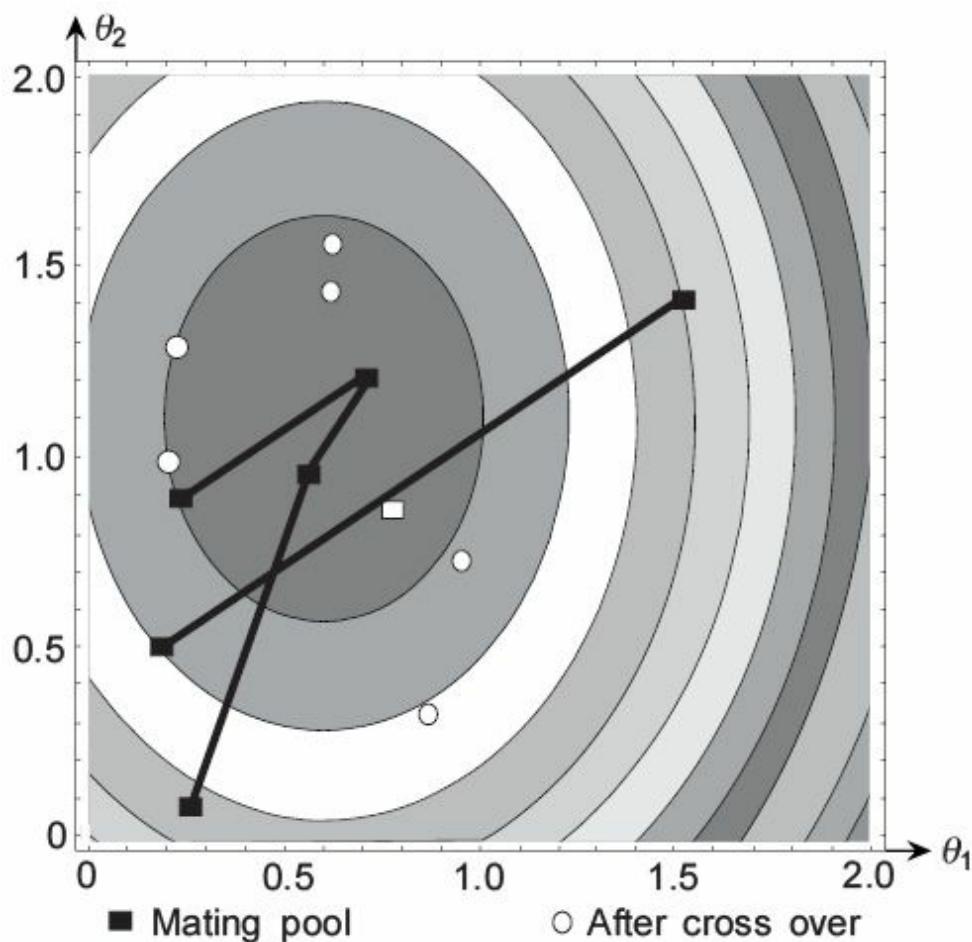
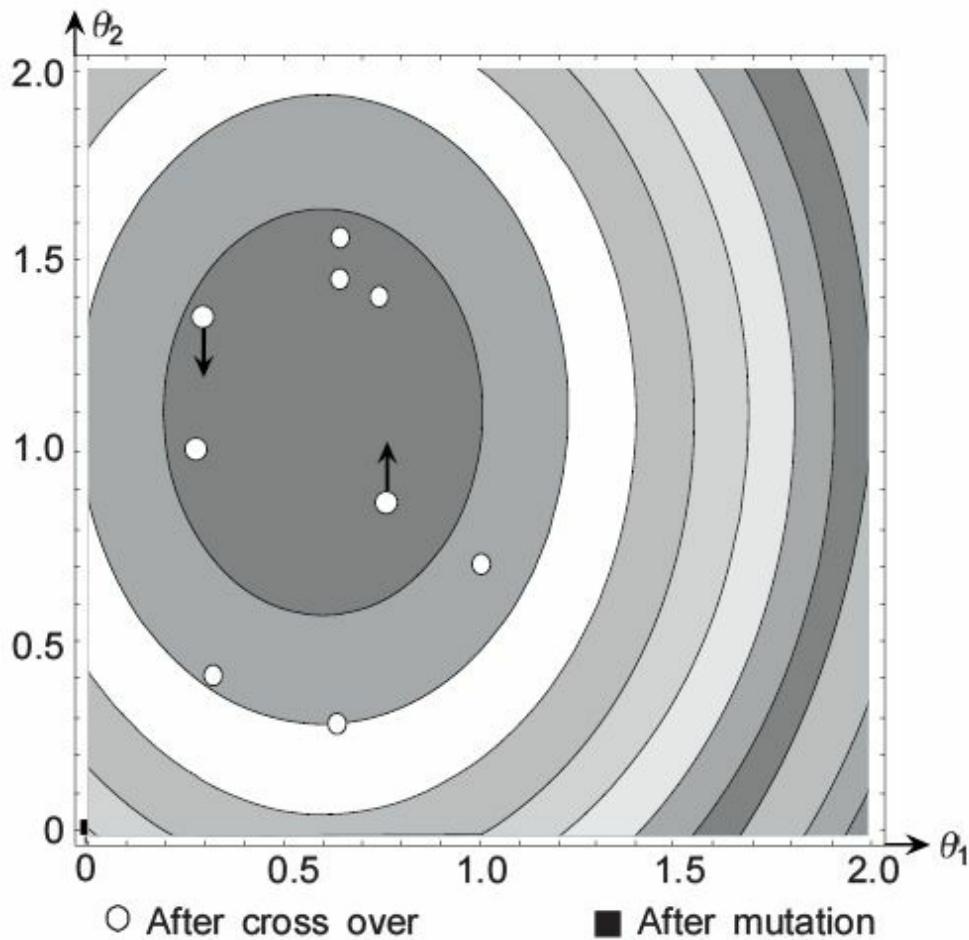


Fig. 9.13 Populations after cross over. (Two-point cross over for all populations) Step 5: The next step is to perform mutation on strings in the intermediate population. For bit-wise mutation, we flip a coin with a probability of $P_m =$

3% for every bit. If the outcome is true, we alter the bit to 1 or 0 depending on the bit value with a probability of $P_m = 0.03$ and for a population size of 8

and a string length of 8, we can expect to alter a total of about $0.03 \times 8 \times 8 = 1.92$ or two bits in the population. These two bits are selected at random as 21 and 62. The 21st bit which is 1, is flipped to 0 and 62nd bit which is zero, is flipped to zero as shown in Table 9.3. Figure 9.14 shows the effect of mutation on the intermediate population. In some cases, the mutation operator changes a point locally and in others it can bring a large change. The points marked with a circle are the points of intermediate population and the points marked with a small box constitute new population (obtained after reproduction, cross over, and mutation). It is interesting to note that if only one bit is muted in a string, the point is moved along a particular variable only. Similar to the cross over operator, the mutation operator has created some points worse than the original points. This flexibility enables GA operators to explore the search space properly before converging to a region prematurely. Although this requires extra computation, this flexibility is essential to solve global optimization



problems.

Step 6: The resulting population becomes a new population as shown in column 8 of Table 9.3. We now evaluate each string as before by first identifying substrings for each variable and mapping the decoded values of the substrings in the chosen intervals. This completes one iteration of genetic algorithm. We increment the generation counter to $t = 1$ and proceed to step 2

for next generation. The new population after one iteration of GA is shown in Fig. 9.14 (marked with empty boxes). The figure shows that in one iteration, some good points have been found. Table 9.3 also shows the fitness values and objective function value of the new population number.

Fig. 9.14 Population after mutation operation. (Average fitness value increased from 3 to 3.95) The average fitness of the new population is calculated to be 3.95

(compared to 3 to start with), a remarkable improvement from that in the initial population. The best point in the population is found to have fitness value equal to 4.67. This process continues until the maximum allowable generation is reached or some other criterion is met. The population after 5

iterations, the best point is found to be $(35^\circ, 66^\circ)$ with a function value of 11.67. In our process the total number of function evaluations required to obtain this solution is $8 \times 5 = 40$ (including the evaluation of the initial

population).

Computer program

A computer program (GAOPT) for optimization of a function subjected to constraints by GA is developed and is given in CD-ROM attached with this book.

Table 9.4 Generation of population ($F = 3.15$)

Applying

Population

Population

Pop.

Mate

right shift

F

Actual

for next

Population

after cross

for next

$i =$

$Fi /$

no.

with

operator to

φ

count

mating

2

over

generation

8

1

3

5

7

9

pool

4

6

5

10

1

0010 0001

5 &

0010 0000

0010 0000

0010 0000

1.7

0.539

0

0110 1000

2

0001 0101

6 &

0000 0101

0000 0101

0000 0101

2.73

0.866

1

0000 0101

3

0010 1000

7 &

0010 1000

0010 1000

0010 1000

4.01

1.27

1

0010 1000

4

0110 1010

8 &

0110 1000

0110 1000

0110 1000

4.51

1.438

2

0110 1000

5

0110 1010

1 \wedge

0100 1011

0010 0101

0010 0101

3.43

1.088

1

0010 0101

6

1110 1101

2 \wedge

1111 1000

1111 1000

1111 1000

1.31

0.415

0

0101 0110

7

0111 1100

3 \wedge

0101 0110

0101 0110

0101 0110

4.16

1.32

2

0101 0110

8

0111 1100

4 \wedge

0001 0110

0001 0110

0001 0110

3.35

1.06

1

0001 0110

Example Problem Using Bit-wise Operators

Let us start with the populations obtained in Chapter 8 for the mating pool.

First two steps are the same as previous example.

Step 3: Randomly select the parents from the mating pool such as 1 with 5, 2 with 6, 3 with 7, and 4 with 8.

Step 4: All the populations are selected for mating since the cross over probability is 100. Since there are eight individuals, there are four pairs.

The operator which is responsible for search in the genetic space bit-wise AND (&) and exclusive-OR (\wedge) is carried out as follows:

1 → 0010 0001	2	1
5 → 0110 1010	6	10
1 & 5 → 0010 0000	2	0
1 \wedge 5 → 0100 1011	4	11

Similar to mutation, we can also apply shift operators. Assuming probability is 10%, we can apply to one string say random 5 as

$a \rightarrow 0100\ 1011$
 $a \gg 1 \rightarrow 0010\ 0101$

Compared to genetic operators, bit-wise operators do not carry genetic information all through generations and hence, it takes longer time to converge.

9.8 CONVERGENCE OF GENETIC ALGORITHM

The situation of good strings in a population set and random information exchange among good strings are simple and straightforward. No mathematical proof is available for convergence of GA. According to Rajeev and Krishnamoorthy (1992), one criterion for convergence may be such that when a fixed percentage of columns and rows in population matrix becomes the same, it can be assumed that convergence is attained. The fixed percentage may be 80% or 85%.

In genetic algorithms as we proceed with more generations, there may not be much improvement in the population fitness and the best individual may not change for subsequent populations. As the generation progresses, the population gets filled with more fit individuals with only slight deviation from the fitness of best individuals so far found, and the average fitness comes very close to the fitness of the best individuals. We can specify some fixed number of generations after getting the optimum point to confirm that there is no change in the optimum in the subsequent generations.

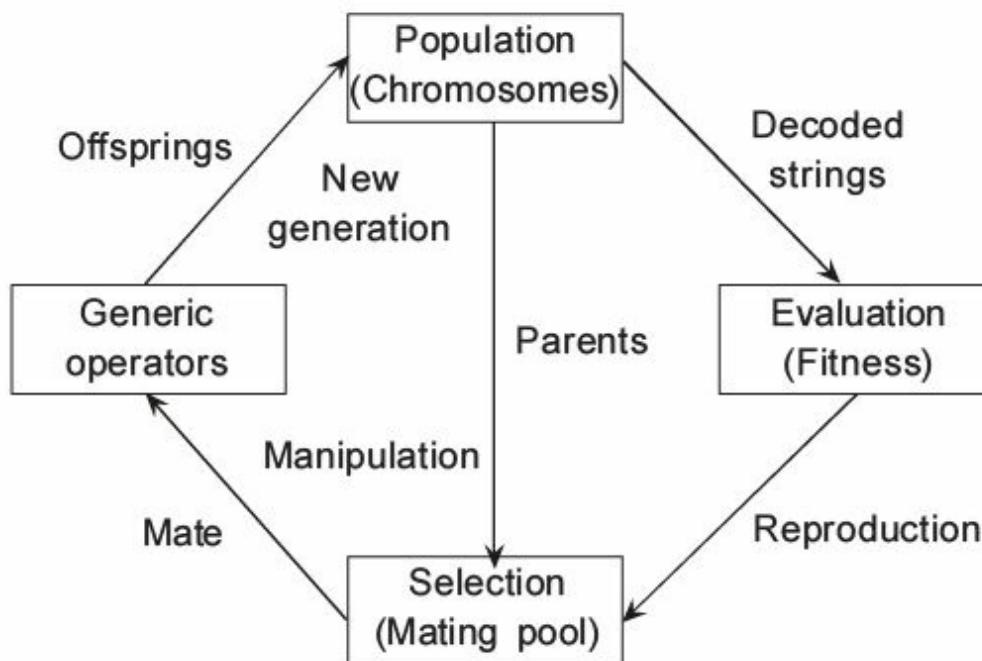
The convergence criteria can be explained from schema point of view in the lines of Goldberg (1989). A *schema* is a similarity template describing a subset of strings with similarities at certain positions. In other words, a schema represents a subset of all possible strings that have the same bits at certain string positions. As an example, consider a string with five bits. A schema ****000** represents the strings 00000, 01000, 10000, and 11000.

Similarly a schema **1*00*** represents the strings 10000, 10001, 11000, and 11001. Each string represented by a schema is called an *instance* of the schema. The symbol * signifies that a 0 or 1 could occur at the string position. Thus, the schema ********* represents all possible strings of five bits.

The fixed positions of a schema are the string positions that have 0 or 1 (In

$**000$, the third, fourth and fifth positions). The number of fixed positions in a schema is its *order* ($**000$ is of order 3). The *defining length* of schema 1^*00* is 3. Any specific string is simultaneously an instance of 2^p schemata (p is the string length).

Since schema represents a robust of strings, we can associate a fitness value with a schema, i.e. the *average fitness* of the schema. Hence, a schema's average fitness value varies with the population's composition from



one generation to another.

One can visualize GA's search for the optimal strings as a simultaneous competition among schemata increases the number of their instances in the population. If we describe the optimal string as just a position of schemata with short defining lengths and high average fitness values, then the winners of individual schema competitions could potentially form the optimal string.

Such schemata with high fitness values and small defining lengths are appropriately called *building blocks*. While genetic operators are applied on a population of strings, a number of such building blocks in various parts along the string get emphasized. Individual processing shows that the worst

schema will die. But there is every danger that the cross over operator may destroy good schemata. So, selection of good appropriate cross over operator plays a vital role here. If the cross-site cuts the well defined position in the schema, this may not be preserved in the next generation unless otherwise. Second parent also will have the same schema in that position. In case of single-point cross over, falling of cross-sites within the defined positions has less probability while in the case of uniform cross over, disturbance of good schema takes place with a higher probability. As far as GA to engineering field is concerned single- and two-point cross over are common. The meaning of search in the genetic space is the development of building blocks.

Building blocks are combined together due to combined action of genetic operators to form bigger and better building blocks and finally converge to the optimal solution. The GA cycle is shown in Fig. 9.15.

Fig. 9.15 The GA cycle.

9.9 APPLICATIONS

9.9.1 Composite Laminates

Composite laminates are used for various applications. They are ideal for structural applications where high strength to weight ratio is required.

Aircraft and other space vehicles are typical weight sensitive structures in which composite materials such as Boron/Epoxy, Carbon/Epoxy, Graphite/Epoxy has resulted in the use of laminated fibre composites and shells. Laminated composites are made by binding together number of layers of at least two different materials. By lamination one can achieve two aspects of constituent layers in order to improve the material quality. Fibre-reinforced composites can be tailored to achieve the required stiffness and strength in a structure by a laminate. This is because the mechanical properties of each ply constituting the laminate can be altered by varying its fibre orientation. A composite offers a weight saving of

24–40% as compared to metallic material if used efficiently.

In a symmetric-angle ply laminate, the fibres are oriented, as shown in Fig.

9.16, symmetrically with respect to the middle layer whereas in an antisymmetric-angle ply laminate, the fibres are oriented as shown in Fig.

9.16. Antisymmetric-angle ply laminate can have only even number of layers whereas symmetric-angle ply laminate can have both odd and even number of layers.

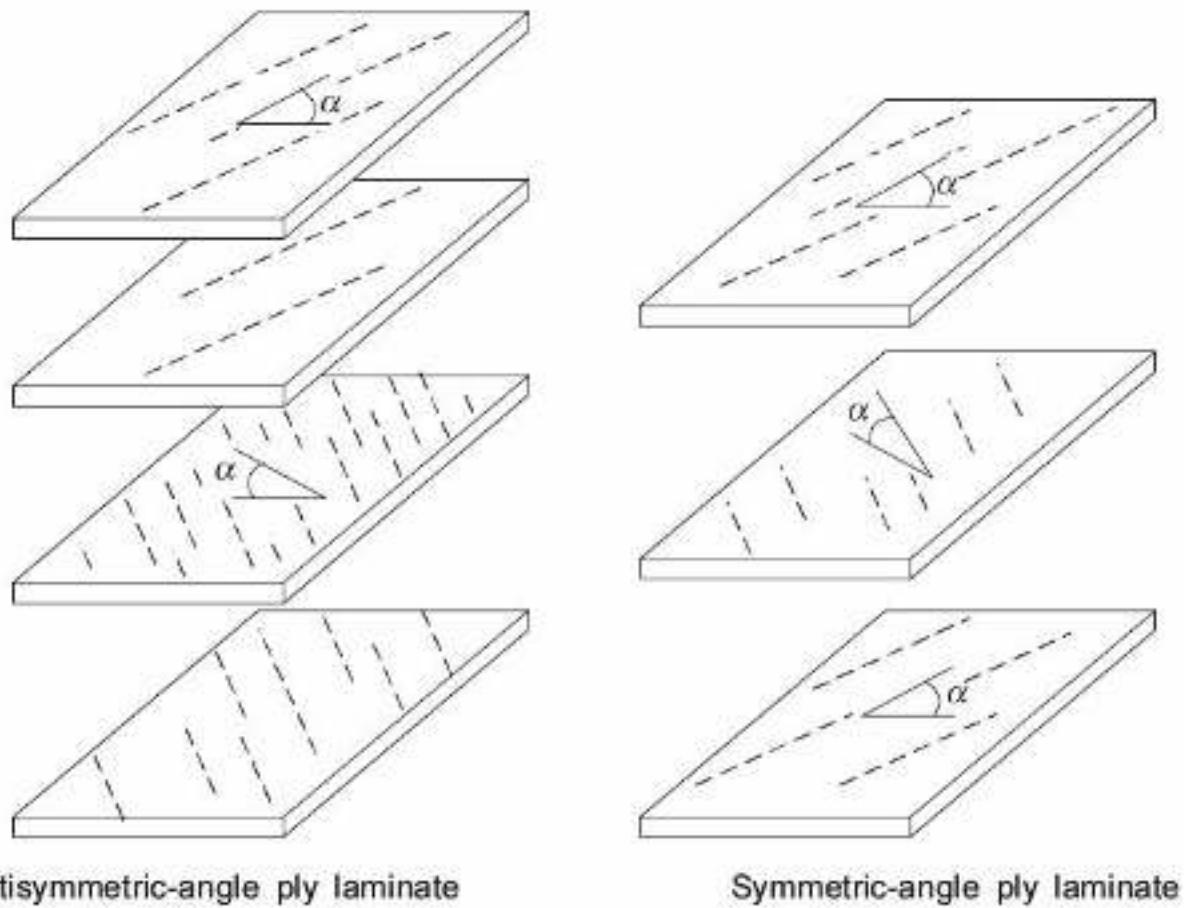


Fig. 9.16 Layered composite plate.

Vibration problem is a predominant problem in aerospace structures where minimum weight design is an important criterion. The frequency of a composite laminate varies with the orientation of fibres in the laminates. The basic design problem of a composite structure is to find the optimum orientation of fibres for maximum frequency. Fibre angles, in case of even

number of layers including the middle surface, are taken as design variable for GA. The bottom half of layers for symmetric orientation (the same number of layers as the top half) and for antisymmetric orientation, the layers in the top half with negative sign are used. Only one half of the layers orientations are used in the coding of GA alongwith the well known operations such as reproductions and mutations.

In this subsection, working of GA is explained with reference to a three-layered symmetric orientation of a thin composite square plate with carbon/epoxy subjected to free vibrations. The assumed data for square plate is—side of square plate = 40 mm, thickness = 0.8 mm.

In this example design variable for the three-layered plate is 2. Since the plate has a symmetric orientation, only half of the layers above the middle

layer including the middle are considered. The orientation of fibres can be varied as discrete values from +90 to -80. A four-bit substring is used to code each variable and in this case, a variable can take 16 discrete values (since the angle varies form 90 to -80, it is divided as -80, -75, -60, -45, -30, -20,

-10, 0, 10, 20, 30, 45, 60, 75, 80, 90, i.e. sixteen angles to select a four-bit binary string) as shown in Table 9.5. Here, eight concatenated strings are adopted to represent two design variables.

The number of populations depends on the importance of the problem and the complexity involved. The number should be even to facilitate mating. In this example, number of populations is limited to eight for the purpose of illustration.

Table 9.5 Binary representation of angles

S.no.

Binary coding

Decoding angle

Fibre angle

1

0000

0

0

2

0001

1

10

3

0010

2

20

4

0011

3

30

5

0100

4

45

6

0101

5

60

7

0110

6

75

8

0111

7

80

9

1000

8

90

10

1001

9

-10

11

1010

10

-20

12

1011

11

-30

13

1100

12

-45

14

1101

13

-60

15

1110

14

-75

16

1111

15

-80

The string representing individuals in the population is generated randomly as shown in column 2 of Table 9.6. In the third column, first value shows the design variable corresponding to layer 1. This is obtained by decoding the first substring of length four of column 2 and getting the corresponding angle from Table 9.5. For example, the substring corresponding to layer 1 from the

first string is 1000. The decimal equivalent of the binary number is 8 and the corresponding angle is 90 degrees. The fibre orientation of second layer is 1001, i.e. binary number is 9 and the angle is -10 degrees. Similarly, other strings are also decoded and the corresponding angles from the list are obtained. After knowing the angles, they are repeated for the second half, which is the mirror image of first half being symmetric. For the above angles in fibres, the frequency is calculated using FEAST-C (1997) for each population. Column 4 shows the frequency for each population.

Table 9.6 Details of computations

Population

Decoded

Mating

Angles

Mating

S.no.

Population

Frequency

Count

Mate

CS1

CS2

after cross

Frequency

Count

angles

pool

decoded

pool

1

2

4

5

7

8

9

over

12

13

3

6

11

14

10

-45,

90, -10,

1000

1100

1

1000 1001

.000535

1

3

1

4

1100 1001

-10,

.000646

1

90

1001

1001

-45

-20,

-20,

1010

1010

2

1010 1100

-45,

.000571

1

6

2

4

1010 1100

-45,

.000571

1

1100

1100

-20

-20

45, -10,

0100

0, -10,

1100

3

0100 1001

.000662

1

1

1

4

0000 1001

.000534

0

45

1001

0

0101

-45, 60,

1100

-45, 60,

1100

4

1100 0101

.000685

2

7

2

4

1100 0101

.000685

1

-45

0101

-45

0101

10, 20,

0001

75, 20,

0110

5

0001 0010

.000542

1

8

1

4

0110 0010

.000553

1

10

0010

75

0010

75, 60,

0110

75, 60,

0110

6

0110 0101

.000555

1

2

2

4

0110 0101

.000555

1

75

0101

75

0101

90, 80,

1100

-45, 60,

1100

7

1000 0111

.000534

0

4

2

4

1100 0101

.000685

2

90

0101

-45

0101

-75,

1110

-10, 80,

1001

8

1110 1111

-80,

.000551

1

5

1

4

1001 1111

.000541

1

1111

-10

1111

-75

Having obtained the fitness values for all the populations, the next step is to generate the population for the next generation which are the offsprings of the current generation. The genetic operators, namely reproduction and cross over are used to generate population for the next generation. The reproduction operator selects the best individuals from the current population and places them in the mating pool. The reproduction process is carried out in

the following manner. The population, which gives the highest frequency, gets two copies and the population which gives the lowest dies off. The other populations get single copy in the mating pool. This process of reproduction confirms the Darwinian principle of survival of the fittest. The mating pool is shown in column 6 of Table 9.6.

The operator responsible for search in genetic space called cross over, is carried now. The cross over rate is selected as 1, that is, all the populations in the mating pool are to be mated. Before selecting the cross-sites, the individuals in the mating pool are matched. Since there are eight individuals there are eight matching pairs. The pairs are selected randomly are shown in column 7 of Table 9.6. Two cross-sites are chosen randomly along the length of the string for each pair as shown in column 8 and 9. Column 10 shows the population after cross over, which is the population set for Generation 2.

Now the same process is repeated for Generation 2 and the details are shown in Table 9.6.

It can be observed from the Table 9.6 that the average frequency is more than the previous generation. It clearly shows the improvement among the set of populations. As one proceeds with more generations, there may not be much improvement in the populations and the best individual may progress.

The population gets filled by more fit individuals with only slight deviation from the fitness of the best individual so far found and the average fitness comes very close to the fitness of the best individual. Number of generations is left to the personal interest. If a satisfactory result is obtained, iterations can be stopped or it can be stopped when there is no significant improvement in the performance from generation to generation for a particular number of generations. In the present study, the convergence of 80% to 85% of the population matrix becomes the same when compared to the previous generation and the iteration has been stopped. Table 9.7 shows the optimum orientation of fibres for maximum frequency for various layers.

Table 9.7 Optimum fibre orientation for max frequency

Symmetric

Antisymmetric

No. of layers

Fibre angle

Max freq

Fibre angle

Max freq

2

45, 45

0.000646

90, -90

0.000533

3

-45, 30, -45

0.000686

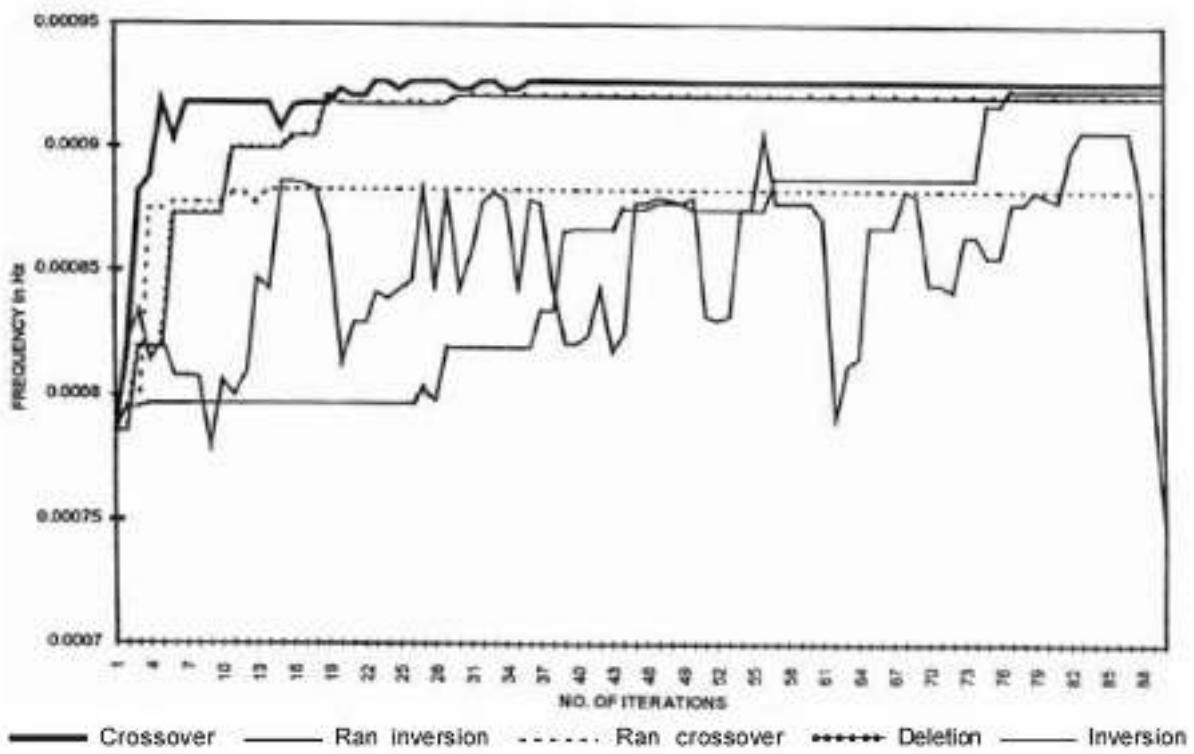
4

-45, 45, 45, -45

0.000782

45, -45, 45, -45

0.000808



5

$-45, 45, 30, 45, -45$

0.000846

6

$-45, 45, 45, 45, 45, -45$

0.000888

$-45, 45, 45, -45, -45, 45$

0.000920

7

$45, -45, -45, 60,$

$-45, -45, 45$

0.000907

8

$-45, 45, 45, 45, 45,$

0.000921

$-45, 45, 45, -45, 45,$

0.000927

$45, 45, -45$

$-45, -45, 45$

9

$-45, 45, 30, 45, -60,$

0.000914

$45, 30, 45, -45$

The convergence history, in number of iterations vs frequency for antisymmetric orientation of fibres for 8 layers is shown in Fig. 9.17.

Fig. 9.17 Convergence history (frequency versus no. of iterations).

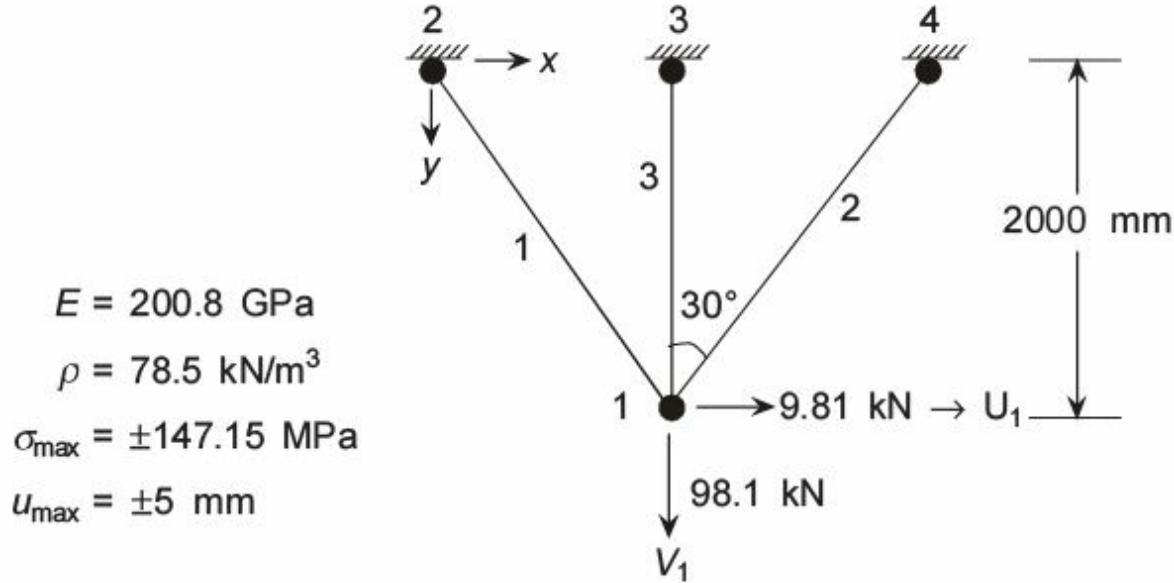
9.9.2 Constrained Optimization

Application from structural engineering

Consider a three bar truss shown in Fig. 9.18. The data assumed is $E = 200.8$

GPa, Density

$\rho = 7850 \text{ kg/cu m}$ (78.5 kN/cu m), maximum stress, $\sigma_{\max} = 147.15 \text{ MPa}$, and



$$f(X) = \sum_{i=1}^3 \rho A_i L_i$$

maximum displacement, $u_{\max} = 5 \text{ mm}$. The truss is symmetric.

Fig. 9.18 Three bar truss.

It is necessary to find out the optimum cross-sectional areas of members 1, 2, and 3 for the given loading conditions. The mathematical programming formulation of this problem can be written as follows.

minimize $f(X)$ subject to

$$g_j(X) \leq 0; j = 1, 2, \dots, m \quad (9.1) \text{ where } m \text{ is the number of constraints.}$$

Since the objective function is to minimise the weight $f(X)$, three bar truss problem can be written as

(9.2)

where A_i is the cross sectional area of i th member, L_i is the length of the i th member, and ρ is the weight density of the material. The constrained equations are written as

$$g_i(X)$$

$$\sigma_j \leq \sigma_a \quad j = 1, 2, 3$$

$$u_1 \leq u_a; v_1 \leq v_a \quad (9.3)$$

where σ_j is the stress in j th member and σ_a is the allowable stress, u_1 and v_1 are the horizontal and vertical displacements of node 1 and u_a is the allowable displacement.

$$\frac{\sigma_j}{\sigma_a} - 1$$

$$\left| \frac{u_1}{u_a} \right| - 1$$

$$\left| \frac{v_1}{v_a} \right| - 1$$

$$\sum_{j=1}^m C_j$$

Here, all the constraints cannot be directly described in terms of design variables hence, they are implicit and their evaluation requires analyzing a truss. Assume, for this purpose, a program is available for analyzing the truss to give the stresses in various members as well as displacements at the nodes.

We have seen that GAs are ideally suited to unconstrained optimization problems. As the present problem is a constrained optimization problem, it is

necessary to transform it to an unconstrained optimization problem to be able to solve it using GA. Transformation methods achieve this either by using exterior or interior penalty functions. Such transformations are ideally suited for sequential searches. GA performs the search in parallel using populations of points in search space. Hence, traditional transformations using penalty or barrier functions are not appropriate for genetic algorithm. A formulation based on the violation of normalized constraints is generally adopted. It is found to work very well for the class of problems. The constraint in normalized form is given by

$$\begin{aligned} &\leq 0 \\ &\leq 0 \\ &\leq 0 \end{aligned} \quad (9.4)$$

A violation coefficient C is computed in the following manner $C_i = g_i(X)$, if $g_i(X) > 0$

$$C_i = 0, \quad \text{if } g_i(X) \leq 0 \quad (9.5)$$

$$C =$$

$$(9.6)$$

where m is the number of constraints.

Now the modified objective function $\phi(X)$ is written as

$\phi(X) = f(X) \{1 + KC\}$ (9.7) where parameter K has to be judiciously selected depending on the required influence of a violation individual in the next generation. A value of 10 was found to be suitable for most of the problems. Now the genetic algorithm is used to carry out unconstrained optimization of $\phi(X)$ as seen for two bar pendulum.

$$\phi(X) = f(X) + (\gamma_0 \times t)^\alpha \sum_{j=1}^m g_j \beta(X)$$

$$\phi(X) = f(X) + (\gamma_0 \times t)^2 \sum_{j=1}^m g_j(K)$$

$$\phi(X) = f(X) + Cp \sum_{j=1}^m g_j(X) K$$

Similar to the approach discussed above for converting constrained optimization to unconstrained optimization, many approaches are studied by Michalewicz (1995). Amongst them, the one proposed by Joines and Houck uses a dynamic penalty with respect to generation

count ‘ t ’ as

(9.8)

In Eq. (9.8), g_j is the j th constraint function, which is zero in case of no violation, and is positive otherwise. The $(\gamma_0 \times t)\alpha$ component of the penalty term is the penalty multiplier, whereas γ_0 stands for the penalty coefficient.

For γ_0 , α , and β the values of 0.5, 2, and 2 are recommended respectively (Erbatur et al., 2000).

Modifying the Eq. (9.8) as

(9.9)

Hasancebi and Erbatur (1999) suggested for $\phi(X)$ as

(9.10)

where ‘ C ’ is the control parameter and ‘ p ’ is the penalty parameter.

Let us use the approach given by Rajeev and Krishnamoorthy (1992).

Going back to truss shown in Fig. 9.18, the objective function is taken as $\phi(X) = f(X) (1 + 10 C)$ (9.11) Because the design variables are

discrete it is necessary to supply a list of values that the design variables can take. The available sections assumed for assigning the value for design variables are given in the list S as $S = \{1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.2, 3.4, 3.6, 3.8, 4.0, 4.4\}$

sq.cm which are given in Table 9.8.

Table 9.8 The available sections for truss members

S.no.

Bit

Decoded value

Area

S.no.

Bit

Decoded value

Area

1

0000

0

1.2

9

1000

8

2.8

2

0001

1

1.4

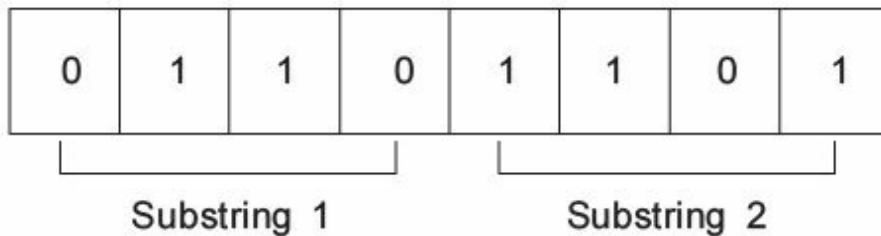
10

1001

9

3.0

$$\frac{(X_u - X_L)}{(2^n - 1)}$$



3

0010

2

1.6

11

1010

10

3.2

4

0011

3

1.8

12

1011

11

3.4

5

0100

4

2.0

13

1100

12

3.6

6

0101

5

2.2

14

1101

13

3.8

7

0110

6

2.4

15

1110

14

4.0

8

0111

7

2.6

16

1111

15

4.4

It is to be noted that the areas are not incremented uniformly. From 1 to 15, increment is 0.2 and from 15 to 16, the increment is 0.4. If the increment is uniform we can write as

$X_{\text{inc}} =$

(9.12)

where ‘ n ’ is the number of digits and

$X_i = XL + (\text{Decoded value}) \times X_{\text{inc}}$ (9.13) A four-bit string can define sixteen different values. The design variables are the areas of two members. Since the truss is symmetric, the area of third member is the same as the first member and hence, there are only two design variables for the optimization problem corresponding to two groups. A binary string of length four is capable of representing 16 different values, and since there are two groups, the total length of the string becomes eight with two substrings of length four each as shown in Fig. 9.19.

Fig. 9.19 Individual string of length 8.

In this problem the fitness function can be calculated as

$F_i = [\phi(X)_{\max} + \phi(X)_{\min}] - \phi_i(X)$ (9.14) The procedure is exactly same as two bar pendulum except that GA will call *analysis program* for the analysis of three bar truss to get the stresses in all the members and the displacements. The details of computations for three generations are given in Tables 9.9, 9.10, and 9.11, respectively.

Table 9.9 Details of computations—Generation 1

S.no.	Population	A_1	A_2	δ_{CD}	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{16}	θ_{17}	θ_{18}	θ_{19}	θ_{20}	CS1	CS2	Pct. gen. 2
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Mate	CS1	CS2	Pct. gen. 2				
1	0111 0111	2.6	2.6	15.50	160.3	85.2	154.1	0.37	-1.62	203	41.6	88.9	1.15	1	0111 0111	3	4	8	01111111								
2	1001 1111	3.0	4.4	17.38	121.4	33.97	116.2	0.75	-1.15	400	47.3	112.7	1.46	2	1001 1111	7	2	4	00001111								
3	0100 0111	2.0	2.8	11.32	170.5	92.48	138.67	1.12	-1.31	273	76.3	33.61	0.69	1	1001 1111	1	4	8	00001111								
4	1111 0011	4.4	1.8	18.78	126.2	75.58	129.48	0.51	-1.31	400	18.1	111.74	1.45	1	0011 1101	8	1	4	00110111								
5	0011 0110	1.8	2.4	10.29	258.3	101.8	208.93	1.25	-2.06	833	96.9	54.48	0.45	0	1111 0011	6	2	6	11110011								
6	0110 0000	2.4	1.2	10.59	211.25	134.5	237.14	0.08	-2.78	979	11.6	16.31	0.51	0	1010 1001	5	3	6	10101001								
7	0000 1000	4.2	3.2	16.31	111.5	72.14	112.01	0.71	-1.31	400	16.1	111.21	1.45	2	1010 1001	2	3	4	10011001								
8	0011 1101	1.8	3.4	17.49	124.0	65.75	159.4	1.25	-1.53	371	46.1	32.16	1.09	0	0011 1101	8	1	4	01001101								

Table 9.10 Details of computations—Generation 2

S.no.	Population	A_1	A_2	δ_{CD}	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{16}	θ_{17}	θ_{18}	θ_{19}	θ_{20}	CS1	CS2	Pct. gen. 2
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Mate	CS1	CS2	Pct. gen. 2				
1	0111 1111	2.6	4.4	16.33	141.5	86.9	126.1	0.37	-1.26	400	16.1	57.6	1.18	1	0111 1111	4	2	8	00011111								
2	1010 1111	3.2	4.4	15.31	118.1	55.22	114.8	0.71	-1.15	400	18.2	85.6	1.12	1	1010 1111	8	4	8	00001111								
3	1100 1011	4.0	2.6	14.96	161	101.58	100.9	0.78	-1.8	165	10.8	35.1	1.15	1	00000111	5	4	8	00010011								
4	0011 1011	1.8	2.6	10.61	211.5	94.48	149.8	1.25	-1.96	373	80.4	15.6	0.21	0	00010000	1	2	8	00110000								
5	1110 1011	4.0	3.4	16.84	110.1	61.06	114.1	0.56	-1.15	400	19.8	34.1	1.12	1	11001011	3	4	8	11100111								
6	1011 1011	3.0	1.8	15.15	117.2	39.48	153.8	0.66	-1.57	400	26.1	77.3	1.48	1	00100011	7	3	7	10010001								
7	1100 1000	3.0	3.0	15.84	134.0	23.97	142.2	0.78	-1.02	400	18.8	35.4	1.19	2	00010000	6	2	7	10100001								
8	0100 1101	2.0	3.9	13.21	164.1	65.98	121	1.12	-1.23	127	24.1	70.0	0.94	1	01001101	2	2	4	01011011								

Table 9.11 Details of computations—Generation 2

S.no.

A 1

A 2

 $f(X)$ σ_1 σ_2

$\sigma 3$

$u\ 1$

$u\ 2$

c

$\varphi(\ x)$

1

2

3

4

5

6

7

8

9

10

11

1

2.2

4.4

14.88

146

56.8

135.1

1.03

-1.35

0.0

14.88

2

2.8

4.4

17.1

126.6

56.5

122.0

0.81

-1.22

0.0

17.06

3

3.0

3.4

16.21

133.6

68.1

134.5

0.75

-1.34

0.0

16.21

4

3.4

3.0

17.03

128.1

70.3

132.2

0.66

-1.32

0.0

17.03

5

4.0

2.6

18.58

118.9

69.8

125.8

0.56

-1.25

0.0

18.58

6

3.0

3.0

15.58

139.4

73.9

142.2

0.75

-1.42

0.0

15.58

7

3.4

1.8

15.15

147.2

89.5

157.8

0.66

-1.57

0.072

26.13

8

2.4

3.8

14.67

147.3

65.5

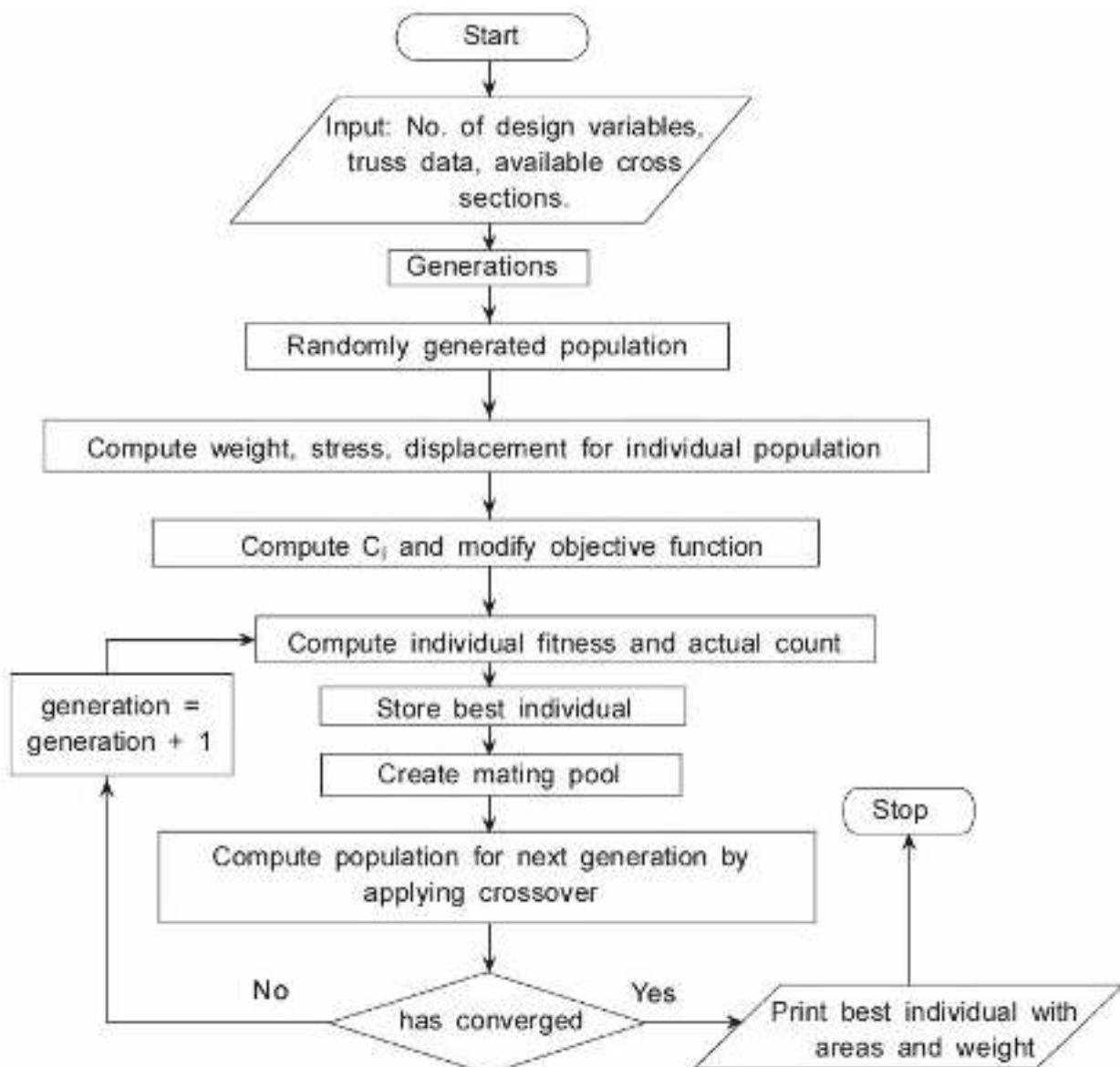
141.8

0.94

-1.41

0.0007

14.78



\bar{F}

The flow chart of the genetic algorithm to solve the truss problem is shown in Fig. 9.20.

Fig. 9.20 Flowchart for truss optimization by GA.

reduces from 53.48 in first generation to 17.53 in the third generation.

From

Table 9.11, we can accept $\phi(X) = 14.778$ with slight violation, $C = 0.00074$ and hence

the minimum weight of the truss is 14.667 kg (0.14 kN) with the area of inclined member as

2.4 sq cm and the vertical member as 3.8 sq cm.

9.10 MULTI-LEVEL OPTIMIZATION

A drawback of GA

Optimization of any system comprising numerous design variables is a challenging problem due to its huge size of search space. In fact, it is not easy for any optimization algorithm to carry out an effective exploration without locating local optimum. The dimensions of the search space grow exponentially either with the addition of extra design variables or with the enlargement of profile lists. The tests performed show that GA is generally quite successful in locating the regions of the search space containing the global optimum, but not the true optimum itself.

Here, multi-level optimization approach as given by Erbatur et al. (2000), is implemented and proved to eliminate the effect of such a major drawback.

The approach rests on reducing the size of the search space for individual design variables in each successive level of the optimization process. In this approach an initial optimization, named the first level optimization, is carried out with an original profile list (discrete set) used by all the design variables.

An original discrete set must be arranged in such a manner that the ready sections are placed in increasing order of the cross-sectional areas. In other words, the first entry of the discrete set is the smallest section and the last entry is the largest section. Following the first level of optimization, the algorithm automatically divides the original discrete set into several subsets (sub-profile lists with smaller sized search space, to be employed in the second level optimization).

The procedure used to create these subsets is as follows.

1. The best design obtained in the first level optimization is taken as the reference design.
2. The original discrete set is equally divided into prescribed number of subsets.
3. Individual design variables are directed to appropriate subsets according to the values obtained in reference design.
4. The enlargement of subset is performed.

Such a treatment of subsets avoids the risk of locating local optimum. In the second level optimization, the design variables use smaller sized subsets.

Hence, the second level optimization is performed on more restricted regions of the search space. The process continues in a similar fashion by dividing the subsets into new subsets and directing the design variables to the most appropriate search space.

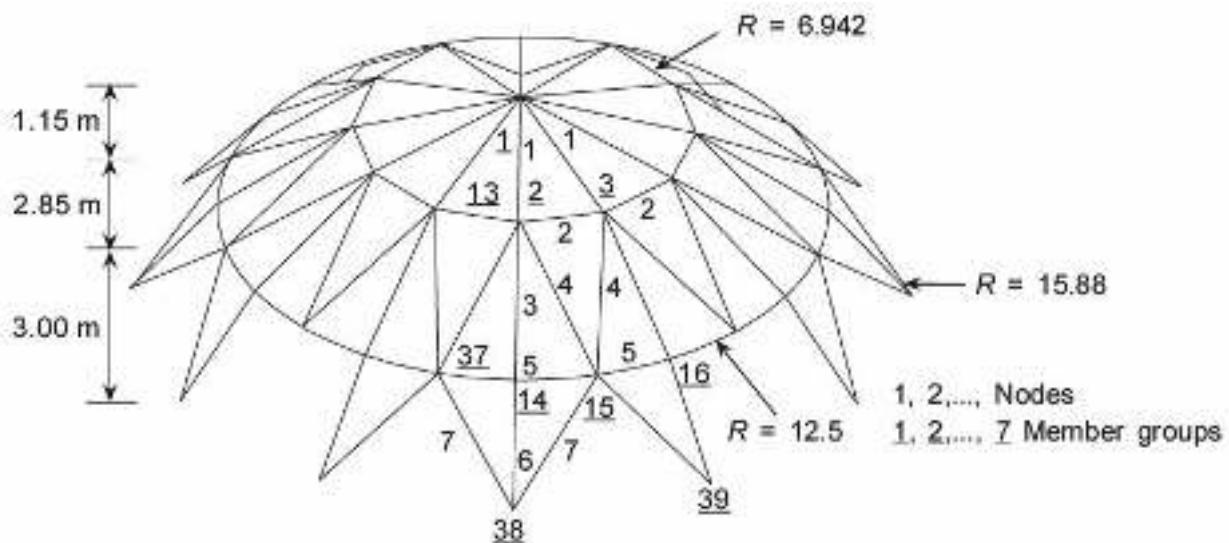
The characteristics of multi-level optimization are:

1. Firstly, it encourages the optimization process to investigate better solutions in more restricted favourable regions of the search space.

Therefore, each optimization level may be interpreted as one step of climbing up a hill towards the summit. Also, it performs well in each search space which is occupied by closely placed optima.

2. Secondly, since the capacity of a discrete set is kept constant, any subset formed by dividing a discrete set must contain fewer ready sections than its capacity. That means, excess slots are produced in the subsets. For continuous optimization problems, these slots can be utilized to obtain better approximation to the continuous solution.

It is observed from experience that two to three levels of optimization are adequate for the convergence to true optimum for discrete and continuous optimizations respectively.



9.11 REAL LIFE PROBLEM

Figure 9.21 shows the 112 bar steel dome that has been previously discussed by Saka (1990) and Erbatur et al. (2000). When applying GA to 112 bar dome, it is not practical to find the areas of 112 members individually.

Consider a 4-bit string in each unknown, each population will consist of $112 \times 4 = 448$ bits. Even during construction, the dome is assembled using 10 or 15 types of members instead of 112 types of members. In the present study, the members are linked into 10 groups and string length of $4 \times 10 = 40$ bits is assumed such that the length of the substring corresponding to each design variable is four. The design variable for the problem is to find the optimal area of these ten types of members so that we achieve minimum weight

design subjected to the constraints. The design data for 112 bar dome is shown in Table 9.12.

Fig. 9.21 Dimensions and member groups of the 112 bar dome.

Table 9.12 Loading and properties data set

Force in

Force in

Force in

Case no.

Joint no.

X dirn

Y dirn

Z dirn

1

0

0

-500 kg (-5 kN)

17, 23, 29, 35

0

0

-40 kg (-0.4 kN)

$$1 \\$$

$$16, 18, 22, 24$$

$$0 \\$$

$$0 \\$$

$$-120\,\mathrm{kg}\,(-1.2\,\mathrm{kN})$$

$$28, 30, 31, 32$$

$$0 \\$$

$$0 \\$$

$$-120\,\mathrm{kg}\,(-1.2\,\mathrm{kN})$$

$$\rho \sum_{i=1}^{i=112} A_i L_i$$

$$\begin{cases} C_{i1}=0 & \text{if } \frac{\sigma_{iT}}{\sigma_{\text{all(ten)}}}-1 \leq 0 \\ C_{i1}=\frac{\sigma_{iT}}{\sigma_{\text{all(ten)}}}-1 & \text{if } > 0 \end{cases}$$

$$\begin{cases} C_{i2}=0 & \text{if } \frac{\sigma_{iC}}{\sigma_{\text{all(com)}}}-1 \leq 0 \\ C_{i2}=\frac{\sigma_{iC}}{\sigma_{\text{all(com)}}}-1 & \text{if } > 0 \end{cases}$$

$$\begin{cases} C_{i3} = 0 & \text{if } \left| \frac{u_i}{u_{\text{all}}} \right| - 1 \leq 0 \\ C_{i3} = \left| \frac{u_i}{u_{\text{all}}} \right| - 1 & \text{if } > 0 \end{cases}$$

$$\sum_{i=1}^{112} \sum_{j=1}^2 C_{ij} + \sum_{i=1}^{49} C_{i3}$$

Rest

0

0

-200 kg (-2 kN)

Modulus of elasticity

210 GPa

Displacement <

0.5 m in any direction

Allowable tensile stress <

1650 kg/sq cm (165 MPa)

Allowable compressive stress <

400 kg/sq cm (40 MPa)

Density of steel

7850 kg/cu m (78.5 kN/cu m)

The objective function is to minimize the weight of the dome and is given by
 $f =$

(9.15)

Constraints for tensile stress are given as

(9.16)

Constraints for compressive stress are given as

(9.17)

Constraints for displacements in X, Y and Z directions are

(9.18)

Hence, the violation coefficient is given as

$C =$

(9.19)

The objective function for constrained problem is

$$\varphi(X) = \varphi(1 + KC) \quad (9.20)$$

and the value of K can be taken as 10.

Along with genetic algorithm program, the analysis program FEAST

(FEAST, 1997) is combined to analyze the 112 bar dome to get the displacements of various nodes and stresses in various members.

Since we do not have any idea of the areas of members, the lowest areas of all groups of members may be assumed to be 400 sq mm and the upper bound for the areas of all groups of members may be assumed to be 1300 sq mm.

After two generations, we get areas for the ten groups and the objective function is given as 3850 (i.e. the weight of the dome

= 0.3850 kg (38.5 kN)). The areas of ten groups are given in Table 9.13. In the second level optimization, we assume the lower bound for areas of first six groups to be 500 sq mm and for the next three groups to be 900 sq mm, and the upper bounds for the above two as 800 sq mm, and 1300 sq. mm respectively. The objective function is given as 3390 kg (33.90 kN) and the areas for the ten groups are shown in Table 9.13. In the third level sub-optimization, the lower bound and the upper bounds for areas of ten groups are shown in Table 9.13. Since we are narrowing down the genetic space, we are able to direct the path towards the minimum. The obtained areas are also shown in the table. Similarly, further level sub-optimization is performed.

The minimum weight of the 112 bar dome is obtained as 3300 kg (33 kN) and the corresponding areas are also shown in Table 9.13. The values obtained by present analysis are compared with earlier investigators.

Table 9.13 Multi-level optimization (112 bar dome)

Weight in kg

Level

Details

A 1

A 2

A 3

A 4

A 5

A 6

A 7

A 8

A 9

A 10

(kN)

1

min

400

400

400

400

400

400

400

400

400

400

max

1300 1300 1300 1300 1300 1300 1300 1300 1300

4881 kg (48.81

First itn

obtained

1010

913

1130 1130 1350

400

1060 1130

913

1060 kN)

3850 kg (38.5

Second itn

obtained

693

1130 1130 1130 1350 1210

840

913

1350

620

kN)

min

500

500

500

500

500

500

900

900

900

900

max

800

800

800

800

800

800

1300 1300 1300 1300

3390 kg (33.90

obtained

640

640

780

600

520

540

1030

980

1100 1060 kN)

min

600

600

700

580

500

500

1000

450

1000 1000

max

700

700

800

650

650

650

1100 1100 1100 1100

3300 kg (33.0

obtained

600

625

720

580

540

580

1030

970

1050 1000 kN)

3468 kg (34.68

Saka (1990)

714

778

—

—

—

—

—

—

—

—

Other

kN)

investigators Erbatur

3390 kg (33.90

707

557

667

707

707

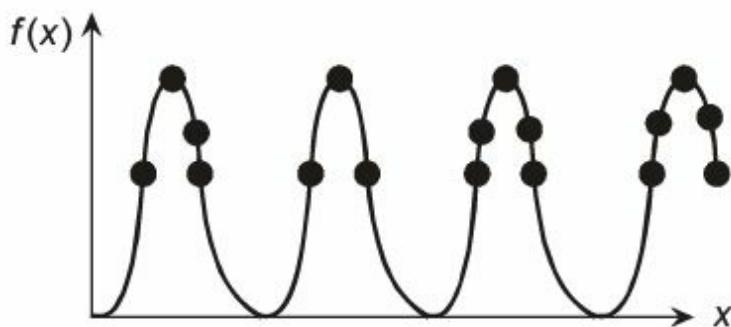
523

—
—
—

(2000)

kN)

It is seen from the real life problem that GA does not need any gradient or either supplementary problem information. This, in fact makes the optimization process easy with respect to more conventional techniques and explains why GA is applied to a wide range of problems. We can also conclude that GA is one of the most robust and promising strategies among discrete optimization techniques and the multi-level optimization approach makes it possible for the GA to compete with continuous optimization techniques.



9.12 DIFFERENCES AND SIMILARITIES BETWEEN GA

AND OTHER TRADITIONAL METHODS

As seen from the working principle of GA, GAs are radically different from most of the traditional optimization methods. Genetic algorithms work with a string coding of variables instead of the variables. The advantages of working with a coding of variable is that coding discretizes the search space

even though the function may be continuous. On the other hand, since GA requires only function values at discrete points, a discrete or discontinuous function can be handled with no extra cost. This allows GA to be applied to a wide variety of problems. Genetic algorithm operators exploit the similarities in string structure to make an effective search. Genetic algorithm works with a population of points instead of a single point. In GA, previously found good information is emphasized using reproduction operator and propagated adaptively through cross over and mutation operators. Genetic algorithm is a population based search algorithm and multiple optimal solutions can be captured as shown in Fig. 9.22, thereby reducing the effort to use the algorithm many times.

Fig. 9.22 Multi-modal functions.

Even though GAs, are different than most traditional search algorithms, there are some similarities. In traditional search methods, where a search direction is used to find a new point, at least two points are either implicitly or explicitly used to define the search direction. In the cross over operator, (which is mainly responsible for GA search) two points are used to create new points. Thus, cross over operator is similar to a directional search method with an exception that the search direction is not fixed for all points in the population and that no effort is made to find the optimal point in any particular direction. Since two points used in cross over operator are chosen

at random, many search directions are possible. Among them, some may lead to global basin and some may not. The reproduction operator has an indirect effect of filtering the good search direction and helps to guide the search. The purpose of mutation operator is to create a point in the vicinity of the current point. The search in the mutation operator is similar to a local search method such as exploratory search used in Hooke-Jeeves method.

$$X_i^{\text{new}} = (1 - \lambda)X_i^{(j)} + \lambda X_i^{(k)}$$

9.13 ADVANCES IN GA

Recently, new and efficient cross over operators have been designed so that search along variables is also possible. Let us consider $X(j)(k)$

i

, X_i values of

design variables X_i in two-parent string j and k . The cross over between these two values may produce the new value as

(9.21)

Here, the parameter λ is a random number between 0 and 1.

The above equation calculates new value bracketting the above two-parent values. This calculation is performed for each variable in the string. This cross over has uniform probability of creating a point inside the region bounded by two parents. An extension of the cross over to create points outside the range is bounded by the parents.

GA with memory

A local improvement procedure for GA is described in this section. The binary tree data structure given by Kernighan and Ritchie (1988) provides a way to store previous designs which permit efficient search of duplicate designs. The binary tree structure is used to store all the data pertinent to the design of the encoded design string. Figure 9.23 shows the pseudo code for the calculation of the objective function using binary tree. After a new generation of design strings is created, the binary tree is searched for the new design. If the design is found, the objective function value is obtained from the tree without analysis, otherwise the tree is searched for design with identical parameters. New designs and their relevant data are then inserted into the binary tree. This algorithm is given in Fig. 9.23.

Algorithm

```
Evaluation of objective function using binary tree.  
begin  
    Search for the given design in binary tree  
    if found;  
        Get objective function value from tree;  
    else  
        Search for the design having identical parameters  
        if found  
            Get normalised values from the tree  
        else  
            Perform analysis  
        end if  
        Add design to binary tree  
    end if  
end
```

$$Sh(d_{ij}) = \begin{cases} 1 - d_{ij}/\sigma & \text{if } d_{ij} < \sigma \\ 0 & \text{otherwise} \end{cases}$$

Fig. 9.23 Finding objective function using binary tree.

Multi-modal Optimization

Many real world problems contain multiple solutions that are optimal or near optimal. The knowledge of multiple optimal solutions in a problem provides flexibility in choosing alternate yet good solutions as and when required. The idea of a number of optimal solutions coexisting in a population requires some change in the simple genetic algorithm described in the previous section. In nature, for example, we recognize that multiple niches (humans and animals) exist by sharing available resources. A similar sharing concept is introduced artificially in GA population in sharing functions as given by Deb (1989) which calculate the extent of sharing that needs to be done between two strings. If d_{ij} is the distance between i th and j th string. Then (9.22)

Here, the parameter σ is the maximum distance between two strings for them to be shared and is fixed beforehand. *Shared fitness* (Sh) is calculated and this is used for reproduction. Using with this sharing strategy, GAs have solved a number of multi-modal optimization problems having more than five million local optima, of which 32 are global optima as given by

Goldberg, et al. (1992).

Searching for optimal schedule

Job shop scheduling, time tabling, and travelling salesman problems are solved using GA. A solution in these problems is a permutation of N objects (names of machines or cities). Although reproduction operator is based on fitness function which is nothing but the distance travelled by salesman, the cross over and mutation operators are different. The operators are designed to produce offsprings which are valid and yet have certain properties of both parents as suggested by Goldberg (1989).

Non-stationary function optimization

Diploid and dominance concepts can be implemented in GA to solve non-stationary optimization problem. Information about earlier good solutions can be stored in recessive alleles and when needed, can be expressed by suitable genetic operators.

Multi-objective optimization

There are many objective functions in multi-objective optimization. The usual practice is to convert multiple objectives into one objective function as $\phi = W_1 f_1 + W_2 f_2 + \dots + W_n f_n$ (9.23) where W_1, W_2 are the weights and f_1, f_2, \dots, f_n are multi-objective functions. Equation (9.23) is one way of converting multi-objective function into single-objective optimization problem. The solution of multi-objective optimization problem can be considered as a collection of optimal solutions obtained by solving different single-objective functions formed using different weight vectors. These solutions are known as *Paretooptimal solutions*. This can be solved

using the concept of non-dominated sorting of population members. For details one may refer to Srinivas and Deb (1995).

Issues for GA practitioners

The following issues are important while applying GA to practical problems, namely

1. Choosing basic implementation issues such as
 - (a) Representation,
 - (b) Population size and mutation rate,
 - (c) Selection, deletion policies, and
 - (d) Cross over and mutation operators.
2. Termination criterion
3. Performance and scalability
4. Solution is only as good as the evaluation functions (often a difficult task).

Benefits of GA

The concept of genetic algorithms is

1. easy to understand,
2. modular, separate from application,
3. supports multi-objective optimization,
4. good for noisy environment,
5. we always get an answer and the answer gets better with time,
6. inherently parallel and easily distributed,

7. there are many ways to speed up and improve a GA's basic applications as knowledge about the problem domain is general,
8. easy to exploit for previous or alternate solutions,
9. flexible in forming building blocks for hybrid applications, and
10. has substantial history and range of use.

When to use GA

Genetic algorithm should be used in case,

1. alternate solutions are too slow or overly complicated,
2. need an exploratory tool to examine new approaches,
3. problem is similar to one that has already been successfully solved by using GA,
4. we want to hybridize with an existing solution, or
5. benefits of GA technology meet key problem requirements.

Table 9.14 gives the fields where GA is successfully applied.

Table 9.14 GA applications

Domains

Application types

Control

Gas pipe line, pole balancing, missile evasion, pursuit

Semi conductor layout, aircraft design, keyboard configuration, communication Design

networks

Scheduling

Manufacturing, facility scheduling, resource allocation

Robotics

Trajectory planning

Machine learning

Designing neural networks, classification algorithms

Signal processing

Filter design

Game playing

Poker, checker, prisoner's dilemma

Combinatorial

Set covering, travelling salesman, routing, bin packing, graph colouring and optimization

partitioning

Research directions

During the course of a run, the optima value for each operator probability may vary. When the population converges, the cross over rate is reduced to give more opportunity to mutation to find new variations. In the past, much research has been empirical, but gradually theoretical insights are being gained. In many cases, it is still too early to say which techniques are robust or general purpose and which are special purpose. The most promising ideas are steady-state replacement, fitness ranking, two-point cross over and are often good methods to use. Knowledge-based operators and dynamic

probabilities are help to solve real world problems. If the fitness function has many local maxima, no search technique is ever going to perform well on it.

Better methods for designing fitness functions are needed which can avoid such pitfalls. There is no doubt that research into GA will be a very active area for some time to come. Genetic algorithms can perform multi-modal optimization technique which is beyond the scope of this book. Besides, a plethora of other applications including non-stationary function, optimization job scheduling problems, routing problem, travelling salesman problems have been tried. The growing list of successful applications of GA to different engineering problems confirms robustness of GAs and shows promise of GA in solving engineering design problems. According to Goldberg “Genetic Algorithms are rich-rich in applications across a large and growing number of disciplines”.

SUMMARY

In this chapter,

The cross over and mutation operators are discussed.

Bit-wise operators are illustrated.

Examples of unconstrained optimization are solved.

GA to constrained optimization is described.

GA is applied to real life problems and the results are compared.

Advances in GA are also illustrated.

PROGRAMMING ASSIGNMENT

P9.1 Solve the nonlinear optimization problem

Minimize

$$(X_1 - 1.5)^2 + (X_2 - 4)^2$$

subject to

$$4.5 X$$

$$2$$

$$1 + X_2 - 18 \leq 0$$

$$2X_1 - X_2 - 1 \geq 0$$

$$0 \leq X_1, X_2 \leq 4$$

Show calculations for three generations. Use cross over probability as 80% and a mutation probability of 3%.

P9.2 Use the Program “GAOPT” given in CD-ROM to solve the Problem 9.1.

(a) Use 4 bits for each variable.

(b) Use 5 bits for each variable. Use multilevel optimization technique.

P9.3 Minimize the function

$$f(X)$$

$$2$$

$1, X_2 = (X_1 + X_2 - 1)2 + (X_1 + X_2 - 7)$ in the interval $0 \leq X_1, X_2 \leq 6$. Use bit-wise operators and show atleast two generations.

P9.4 Use the program “GAOPT” given in CD-ROM to solve the Problem P9.3.

P9.5 Modify the program by including a program to solve a truss and hence find the optimal design for the truss shown in Fig. 9.18.

SUGGESTED FURTHER READING

Deb, K. (1999), *An Introduction to Genetic Algorithms*, Sadhana, Vol. 24, Parts 4 and 5, Aug. and Oct., pp. 293–315.

Gen, M. and R. Cheng (1997), *Genetic Algorithm and Engineering Design*, John Wiley, New York.

Goldberg, D.E. and J. Richardson (1987), Genetic Algorithms with Sharing for Multi-modal Function Optimization, *Proc. of the Second Intl. Conf. On Genetic Algorithms*, (Ed.)

J.J. Grefenstette, pp. 41–49.

Starkweather, T., S. McDaniel, K. Mathias, D. Whitley, and C. Whitley (1991), A Comparison of Genetic Scheduling Operators, *Proc. of Fourth Intl. Conf. on Genetic Algorithms*, (Eds.)

R. Belew, L.B. Booker, (San Mateo, C.A., Morgan Kaufmann), pp. 69–76.

SOME USEFUL WEBSITES

1. <http://cs.felk.cvut.cz/~xobitko/ga>
2. http://www.elec.gla.ac.uk/~yuunli/ga_demo
3.
<http://www.eng.buffalo.edu/Research/MODEL/wcsmo3/proceedings/numlist.html>
4. <http://ls11-www.informatik.uni-dortmund.de/evenet/coordinator/resources/softwareanddemos.html>
5. <http://lancet.mit.edu/~mbwall/presentations/Introto GA>

REFERENCES

Anonymous (1997), *FEAST-C User Manual*, SEG, SDS group, V.S.S.C, ISRO, Trivandrum.

Deb, K. (1989), Genetic Algorithms in Multi-modal Function Optimization, *Masters thesis* (TCGA report No. 89002), Tuscaloosa, University of Alabama.

Deb, K. (1995), *Optimization for Engineering Design—Algorithms and Examples*, Prentice-Hall of India, New Delhi.

Erbatur, F., O. Hasancebi, I. Tutuncu, and H. Kilic (2000), Optimal Design of Planar and Space Structures with Genetic Algorithms, *Computers and Structures*, Vol. 75, pp. 209–224.

Goldberg, D.E. (1989), *Genetic Algorithm in Search Optimization and Machine Learning*, Addison Wesley, Reading, Mass., USA.

Goldberg, D.E., K. Deb, and J. Horn (1992), Massive Multi-modality, Deception, and Genetic Algorithms in Parallel Problem Solving from Nature, (Eds.) R. Manner, B. Manderich, Berlin Springer Verlag, pp.

37–46.

Gottfried, B.S. (1990), *Programming with C*, Schaum's outline series, Tata McGraw-Hill,

New Delhi, 1990.

Hasancebi, O. and F. Erbatur (1999), Constraint Handling in Genetic Algorithm Integrated Structural Optimization”, *Acta Mechanica*.

Kernighan, B.W. and D.M. Ritchie (1988), *The C Programming Language*, 2nd ed., Prentice Hall, Englewood Cliffs, N.J., USA., pp. 139–143.

Michalewicz, Z. (1995), Genetic Algorithms, Numerical Optimization and Constraints, *Technical Report*, University of North Carolina, USA.

Rajeev, S. and C.S. Krishnamoorthy (1992), Discrete Optimization of Structures using Genetic Algorithms, *Joul. of Structural Engg*, ASCE, Vol. 118, No. 5, pp. 1223–1250.

Saka, M.P. (1990), Optimum Design of Pin-jointed Steel Structures with Practical Applications, *Joul. of Structural Engineering*, ASCE, Vol. 116, No. 10, pp. 2599–2620.

Srinivas, N. and K. Deb (1995), Multi-objective Function Optimization using Non-dominated Sorting Genetic Algorithm, *Evolutionary Computation*, Vol. 2, pp. 221–225.

PART 4

HYBRID SYSTEMS

- Integration of Neural Networks, Fuzzy Logic, and Genetic Algorithms
- Genetic Algorithm based Backpropagation Networks
- Fuzzy Backpropagation Networks
- Simplified Fuzzy ARTMAP
- Fuzzy Associative Memories
- Fuzzy Logic Controlled Genetic Algorithms

Chapter 10

Integration of Neural Networks,

Fuzzy Logic, and Genetic Algorithms

Neural networks, fuzzy logic, and genetic algorithms are soft computing methods which have been inspired by biological computational processes and nature's problem solving strategies.

Neural networks (NNs) are highly simplified models of the human nervous system which mimic our ability to adapt to circumstances and learn from past experience. Part I of the book discusses three NN systems each representing the three major classes of NN architectures, namely single layer feedforward, multilayer feedforward, and recurrent network architectures. The NN systems presented are the backpropagation network, associative

memories, and adaptive resonance theory. Backpropagation network is a multilayer feedforward network architecture with gradient descent learning. Associative memories are single layer feedforward or recurrent network architectures adopting Hebbian learning. ART networks are recurrent network architectures with a kind of competitive learning termed adaptive resonance theory.

Fuzzy logic systems address the imprecision or vagueness in input output descriptions of systems using fuzzy sets. Fuzzy sets have no crisp boundaries and provide a gradual transition between membership and non-membership of elements in a set. Part II of the book discusses the fundamental concepts of fuzzy logic and its applications.

Genetic algorithms (GAs) inspired by the process of biological evolution, are adaptive search and optimization algorithms. Part III of the book discusses the basic concepts, namely the genetic inheritance operators and applications of GA.

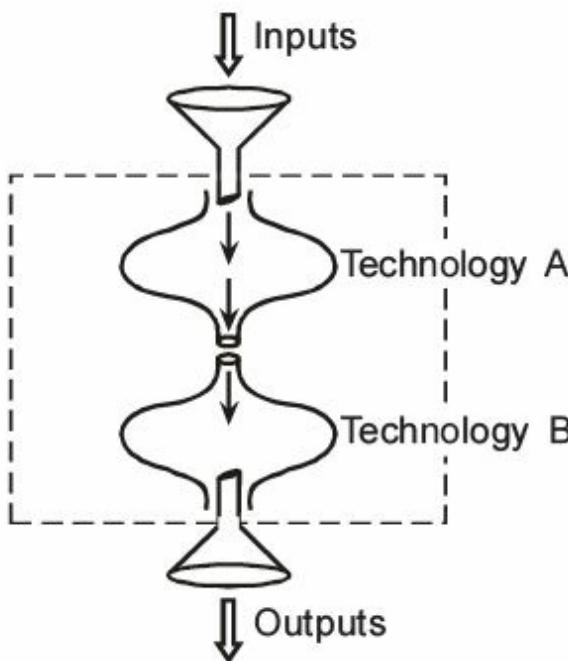
Each of the technologies, in their own right and merit, have provided efficient solutions to a wide range of problems belonging to different domains. At the same time, as mentioned in Chapter 1, various attempts have been successfully made to synergize the three different technologies in whole or in part, to solve problems for which these technologies could not find solutions individually. The objective of the synergy or hybridization has been

to overcome the weaknesses in one technology during its application, with the strengths of the other by appropriately integrating them. More often, the complexity surrounding a problem has called for a judicious combination of the technologies, when a technology individually applied has failed to obtain an efficient solution.

However, hybridization of technologies can have its pitfalls and therefore needs to be done with care. If a technology can solve a problem then a hybrid technology ought to be used only if its application results in a better solution or provides a better method to arrive at the solution or at worst, finds an alternative method to arrive at the solution. Hybridization should only be

performed for the purpose of investigating better methods of problem solving.

Hence, though hybrid systems have a tremendous potential to solve problems, an inappropriate use of the technology can backfire. For, it is improper to expect that if the individual technologies are good then hybridization of technologies should turn out to be even better. In fact, it is not unlikely for a hybrid technology to exhibit most of the weaknesses of the participating technologies and less of their strengths.



10.1 HYBRID SYSTEMS

Hybrid systems are those for which more than one technology is employed to solve the problem. Hybrid systems have been classified as (Refer Gray and Kilgour, 1997) 1. Sequential Hybrids,

2. Auxiliary Hybrids, and 3. Embedded Hybrids.

10.1.1 Sequential Hybrid Systems

As the name indicates, *sequential hybrid systems* make use of technologies in a pipeline-like fashion. Thus, one technology's output becomes another's

input and so on. Figure 10.1 illustrates the schema for a sequential hybrid.

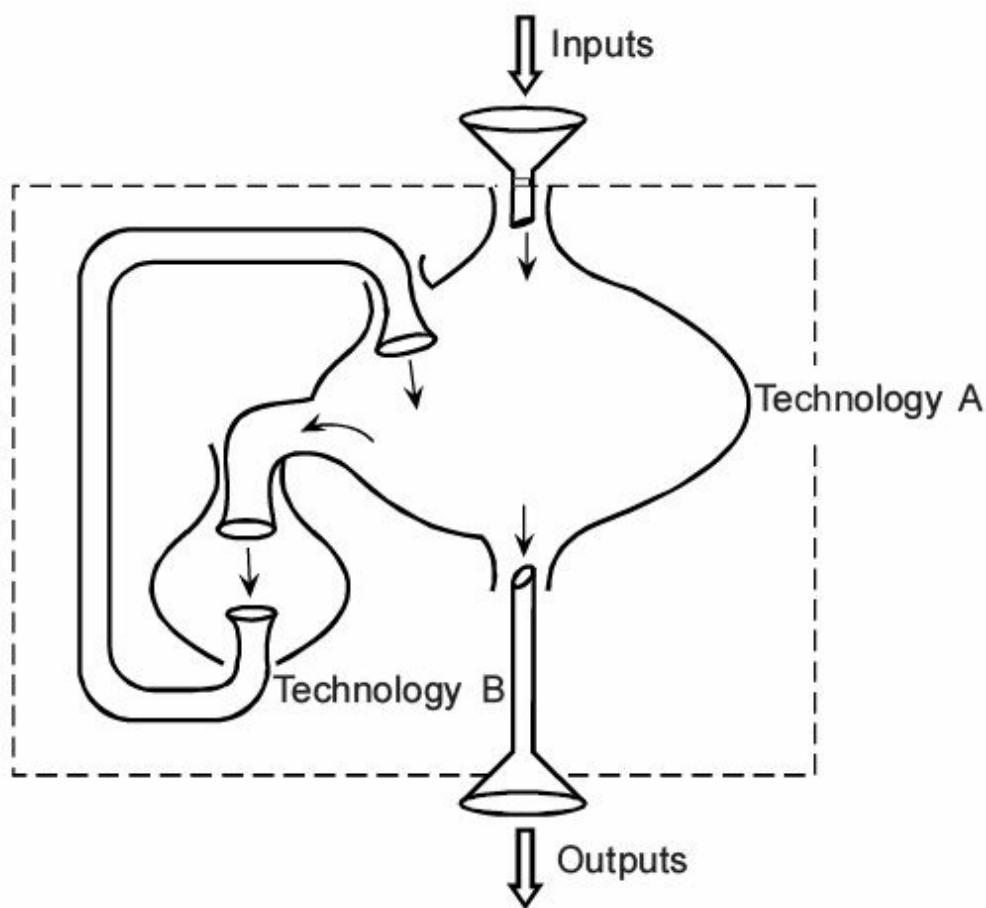
Fig. 10.1 A sequential hybrid system.

This is one of the weakest forms of hybridization since an integrated combination of the technologies is not present.

An example is a GA preprocessor which obtains the optimal parameters for different instances of a problem and hands over the ‘preprocessed’ data set to an NN for further processing.

10.1.2 Auxiliary Hybrid Systems

In this, one technology calls the other as a “subroutine” to process or manipulate information needed by it. The second technology processes the information provided by the first and hands it over for further use. Figure



10.2 illustrates an auxiliary hybrid system. This type of hybridization though better than sequential hybrids, is considered to be of intermediary level only.

An example is a neuro-genetic system in which an NN employs a GA to optimize its structural parameters, i.e. parameters which defines its architecture.

Fig. 10.2 An auxiliary hybrid system.

10.1.3 Embedded Hybrid Systems

In *Embedded hybrid systems*, the technologies participating are integrated in such a manner that they appear intertwined. The fusion is so complete that it would appear that no technology can be used without the others for solving the problem. Figure 10.3 illustrates the schema for an embedded hybrid system. Here, the hybridization is absolute.

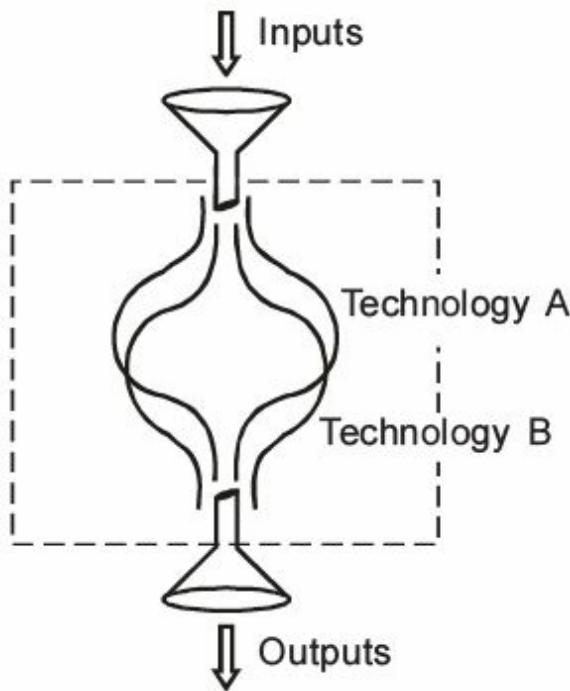


Fig. 10.3 An embedded system.

For example, an NN-FL hybrid system may have an NN which receives fuzzy inputs, processes it and extracts fuzzy outputs as well.

10.2

NEURAL

NETWORKS,

FUZZY

LOGIC,

AND

GENETIC

ALGORITHMS HYBRIDS

In this book, we confine ourselves to hybridization of the three technologies, namely neural networks, fuzzy logic, and genetic algorithms. Neural networks, fuzzy logic, and genetic algorithms are three distinct methodologies each with its own advantages and disadvantages. It is therefore appropriate that a hybridization of the technologies is attempted to overcome the weaknesses of one with the strengths of the other.

10.2.1 Neuro-Fuzzy Hybrids

This is one of the most researched forms of hybrid systems and has resulted in a stupendous quantity of publications and research results.

Neural networks and fuzzy logic represent two distinct methodologies to deal with uncertainty. Each of them has its own merits and demerits. Neural networks can model complex nonlinear relationships and are appropriately suited for classification phenomenon into predetermined classes. On the other hand, the precision of outputs is quite often limited and does not admit zero error but only minimization of least squares errors. Besides, the training time required for an NN can be substantially large. Also, the training data has to be chosen carefully to cover the entire range over which the different variables are expected to change.

Fuzzy logic systems address the imprecision of inputs and outputs directly by defining them using fuzzy sets and allow for a greater flexibility in formulating system descriptions at the appropriate level of detail.

Neural networks and fuzzy logic though different technologies, can be used to accomplish the specification of mathematical relationships among numerous variables in a complex dynamic process, perform mappings with some degree of imprecision, in different ways, and can be used to control nonlinear systems to an extent not possible with conventional linear control systems.

Neuro-Fuzzy systems which are an integration of NN and FL have demonstrated the potential to extend the capabilities of systems beyond either of these technologies when applied individually (Haykin, 1994; Kartalopoulos, 1996).

There are two ways of looking at this hybridization. One is to endow NNs with fuzzy capabilities, thereby increasing the network's expressiveness and flexibility to adapt to uncertain environments. The second aspect is to apply neuronal learning capabilities to fuzzy systems to make the fuzzy systems more adaptive to changing environments. This approach is also known, in the literature, as *NN driven fuzzy reasoning* (Takagi and Hayashi, 1991).

10.2.2 Neuro-Genetic Hybrids

Neural networks can learn various tasks from training examples, classify phenomena, and model nonlinear relationships. However, the primary features that are of concern in the design of the network are problem specific.

Despite the availability of some guidelines, it would be helpful to have a computational procedure in this aspect, especially for the optimum design of an NN. Genetic algorithms have offered themselves as potential candidates for the optimization of parameters of NN (Harp et al., 1989, 1990). The integration of GAs with NNs has turned out to be useful (Schaffer et al., 1992). Genetically evolved nets have reported comparable results against their conventional counterparts (Kitano, 1990; Whitley and Hanson, 1989).

While gradient descent learning algorithms have reported difficulties in learning the topology of the networks whose weights they optimize, GA based algorithms have provided encouraging results especially with regard to face recognition, animation control, and other problems.

Genetic algorithms encode the parameters of NNs as a string of the properties of the network, that is, chromosomes. A large population of chromosomes representing the many possible parameter sets for the given NN is generated. Combined GA-NN technology also known as ‘GANN’ have the ability to locate the neighbourhood of an optimal solution quicker than other conventional search strategies. But once in the neighbourhood of the optimal solution GANN strategies tend to converge slower than the conventional ones. Drawbacks of GANN algorithms are:

The large amount of memory required to handle and manipulate chromosomes for a given network, and

The question whether this problem scales as the size of the networks becomes large.

Parallel versions of the GA computational paradigm have also been applied on NN (Maniezzo, 1994).

10.2.3 Fuzzy-Genetic Hybrids

Fuzzy systems have been integrated with GAs. Kosko (1992) has shown that fuzzy systems like NNs (feedforward) are universal approximators in the fact that they exhibit the capability to approximate general nonlinear functions to any desired degree of accuracy. The adjustment of system parameters that is called for in the process, so that the system output matches the training data, has been tackled using GAs. Several parameters which a fuzzy system is involved with, namely input/output variables and the membership functions that define the fuzzy systems, have been optimized using GAs. Nomura et al.

(1994) proposed a genetic approach to the problem of fuzzy system adaptation.

10.3 PREVIEW OF THE HYBRID SYSTEMS TO BE DISCUSSED

In this section, we present a preview of the hybrid systems to be discussed in Part IV of the book.

The systems discussed are:

1. Genetic algorithm based backpropagation network (*Neuro Genetic Hybrid*).
2. Fuzzy backpropagation network (*Neuro-Fuzzy Hybrid with Multilayer Feedforward Network as the host architecture*).
3. Simplified fuzzy ARTMAP (*Neuro-Fuzzy Hybrid with Recurrent Network as the host architecture*).
4. Fuzzy associative memory (*Neuro-Fuzzy Hybrid with Single layer Feedforward or Recurrent Network as its host architecture*).
5. Fuzzy logic controlled genetic algorithms (*Fuzzy-Genetic Hybrid*).

We now briefly review each of these hybrid systems.

10.3.1 Genetic Algorithm based Backpropagation Network

This is a Neuro-Genetic hybrid which makes use of GAs to determine the weights of a multilayer feedforward network with backpropagation learning.

Conventional backpropagation networks make use of gradient descent learning to obtain their weights. However, there remains the problem of the network getting stuck in local minimum. On the other hand, the GA based backpropagation though not guaranteed to obtain global optimum solution, has been found to obtain ‘acceptably good’ solutions ‘acceptably quickly’.

Though several successful propositions regarding the GA-backpropagation integration have been implemented, the network discussed is a

backpropagation network architecture in which the GA makes use of real-coded chromosomes, instead of the conventional binary chromosomes to determine its weights.

The hybrid system has been demonstrated on two problems, namely k-factor design and electrical load forecasting.

10.3.2 Fuzzy-Backpropagation Network

This is a Neuro-Fuzzy hybrid system in which the host is a multilayer feedforward network. Proposed by Lee and Liu (1994), the network maps fuzzy input vectors to crisp outputs making use of backpropagation like learning.

The architecture has been applied to the problems of knowledge-based evaluation and earthquake damage evaluation.

10.3.3 Simplified Fuzzy ARTMAP

Fuzzy ARTMAP (Carpenter et al., 1992) is a neuro-fuzzy hybrid in which the host is a recurrent network with a kind of competitive learning termed adaptive resonance theory by its authors (Carpenter and Grossberg, 1988).

Proposed by Kasuba (1992), simplified fuzzy ARTMAP is a vast simplification of fuzzy ARTMAP. It classifies inputs by its fuzzy set of features and unlike its predecessor has reduced computational overhead and architectural redundancy.

The application of simplified fuzzy ARTMAP to image recognition under noisy and noise-free conditions is elaborated.

10.3.4 Fuzzy Associative Memories

Fuzzy Associative Memory (FAM) is a neuro-fuzzy system with the host architecture as a recurrent or a single layer network. FAMs map fuzzy sets and can encode fuzzy rules. It is on account of this mapping capability that they behave like associative memories.

Fuzzy associative memories make use of graphical method of inference and correlation matrix encoding schemes for inference.

The application of FAM has been demonstrated on two problems, namely balancing an inverted pendulum and fuzzy truck backer-upper system.

10.3.5 Fuzzy Logic Controlled Genetic Algorithms

This is a Fuzzy-Genetic hybrid system applicable on fuzzy optimization problems. The system obtains optimal solution to problems with fuzzy constraints and fuzzy variables.

The hybrid system has been demonstrated on the problems of optimization of structures (civil/machine tool) and obtaining the optimal mix for high performance concrete.

SUMMARY

Hybrid systems are those which employ integrated technologies to effectively solve problems. Hybrid systems are classified as sequential, auxiliary and embedded hybrids.

The soft computing techniques of neural networks, fuzzy logic, and genetic algorithms have offered themselves as candidates for a healthy integration or hybridization of technologies for effective problem solving.

The synergy of the three technologies has led to neuro-fuzzy, neuro-genetic, and fuzzy-genetic hybrids. In this book, five hybrid systems, namely genetic algorithm based back-propagation network (*Neuro-Genetic hybrid*), fuzzy backpropagation network, simplified fuzzy ARTMAP and fuzzy associative memories (*Neuro-Fuzzy hybrid s*) and fuzzy logic based genetic algorithms (*Fuzzy-Genetic hybrid*), are presented.

REFERENCES

Carpenter, G.A. and S. Grossberg (1988), The Art of Adaptive Pattern Recognition by a Self-Organizing Neural Network, *IEEE Computer*

Magazine, Vol. 21, No. 3, pp. 77–88.

Carpenter, G.A., S. Grossberg, Markuzon Natalya, J.H. Reynolds and D.B.

Rosen (1992), Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps, *IEEE Trans on Neural Networks*, Vol. 3, No. 5, pp. 698–713.

Gray, Andrew and Richard Kilgour (1997), Hybrid Systems FAQ, Version Draft 1.00.

Harp, S., T. Samad, and A. Guha (1989), Towards the Genetic Synthesis of Neural Networks, *Proc of the Third International Conf on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.

Harp, S., T. Samad, and A. Guha (1990), Designing Application Specific Neural Networks using the Genetic Algorithm, *Advances in Neural Information Processing Systems 2*, D.S.Touretzky, Ed., Morgan Kaufmann, San Mateo, CA.

Haykin, S. (1994), Neural Networks: A Comprehensive Foundation, IEEE Computer Society Press, Macmillan NY.

Kartalopoulos, S. (1996), Understanding Neural Networks and Fuzzy Logic, IEEE Press, NY.

Kasuba, Tom (1992), Simplified Fuzzy ARTMAP, *AI Expert*, pp. 18–25.

Kitano, H. (1990), Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms, *Proc of the Eighth National Conf on AI (AAAI-90)*, pp. 789–795.

Kosko, B. (1992), Neural Networks and Fuzzy Systems, Prentice Hall, Englewood Cliffs, NJ.

Lee Hahn Ming and Liu Bing Hui (1994), Fuzzy BP: A Neural Network Model with Fuzzy Inference; *Proc ICANN 94*, pp. 1583–1588.

Maniezzo, V. (1994), Genetic Evolution of the Topology of Weight Distribution of Neural Networks, *IEEE Trans on Neural Networks*, Vol. 5, No. 1, pp. 39–53.

Nomura, H., I. Hayashi, and N. Wakami (1994), A Self Tuning Method of Fuzzy Reasoning by Genetic Algorithms, *In. Fuzzy Control Systems*, Kandel, A. and Langholz, G., Eds, CRC Press, Boca Raton, FL, pp.

338–354.

Schaffer, J.D., D. Whitley, and L.J. Eshelman (1992), Combination of Genetic Algorithms and Neural Networks: A Survey of the State of the Art, *Proc of Intl Wks on Combinations of Genetic Algorithms and Neural Networks (COGANN 92)*, pp. 1–37.

Takagi, H. and I. Hayashi (1991), NN Driven Fuzzy Reasoning, *Intl Journal of Approximate Reasoning*, Vol. 5, No. 3, pp. 191–212.

Whitley, D. and T. Hanson (1989), Optimizing Neural Networks using Faster More Accurate Genetic Search, *Proc of the Third Intl Conf on Genetic Algorithms*, Morgan Kaufmann,

San Mateo, CA.

Chapter 11

$$\frac{1}{2} \sum (T_j - O_j)^2$$

Genetic Algorithm Based

Backpropagation Networks

Neural networks solve problems by self-learning and self-organization. The backpropagation network (BPN) is probably the most well known and widely used among the currently available neural network systems. Backpropagation network has been applied to classification problems, speech synthesis from

text, adaptive robotics control, system modelling, and various problems in engineering.

The learning algorithm behind BPN (Rumelhart et al., 1986) is a kind of gradient descent technique with backward error (gradient) propagation.

However, ‘while the network can recognize patterns similar to those they have learnt, they do not have the ability to recognize new patterns’ (Fu, 1994). Also, while the network must be sufficiently trained to extract a sufficient set of general features applicable to both seen and unseen instances, overtraining the network may lead to undesired effects.

Backpropagation (BP) searches on the error surface by means of the gradient descent technique in order to minimize the error criterion $E =$

(11.1)

where T_j is the target output and O_j is the output calculated by the network. It is therefore likely to get stuck in a local minimum.

On the other hand, there exist genetic algorithms (GAs) which are adaptive search and optimization algorithms that mimic the principles of natural genetics. Genetic algorithms are quite different from traditional search and optimization techniques used in engineering design problems but at the same time exhibit simplicity, ease of operation, minimal requirements, and global perspective.

Conventionally, a BPN determines its weights based on a gradient search technique and therefore runs the risk of encountering the local minimum problem. GAs on the other hand, though not guaranteed to find global optimum solution to problems, have been found to be good at finding

“acceptably good” solutions to problems “acceptably quickly” (less number of iterations).

The idea to hybridize the two approaches, namely GA and BPN follows naturally. Whitley and co-workers (Whitley and Bogart, 1989, 1990; Whitley

and Hanson, 1989; Whitley and Starkwerther, 1990) used GAs to guide backpropagation network in finding the necessary connections instead of full connections in order to enhance the speed of training. Though Kitano (Kitano, 1990) proposed some empirical evidence to show that GA/BP

mating does not provide any advantage over a randomly initialized multiple application of Quickprop (a fast variant of BP) alone, atleast for shallow networks and easy fitness functions, successful reports have been reported with a hybrid approach. Harp, Samad and Guha (1989) proposed a combined genetic/backpropagation learning algorithm and encoded BP parameters in the individuals together with the network structure.

In this chapter, a GA based technique for the determination of weights in a BPN (GA/BPN) (Rajasekaran and Pai, 1996) is discussed. However, the network unlike its predecessors employs real-coded chromosomes to determine its weights while employing a two-point cross over operator (TCO).

11.1 GA BASED WEIGHT DETERMINATION

Genetic algorithms which use a direct analogy of natural behaviour, work with a population of individual strings, each representing a possible solution to the problem considered. Each individual string is assigned a fitness value which is an assessment of how good a solution is, to a problem. The high-fit individuals participate in “reproduction” by cross-breeding with other individuals in the population. This yields new individual strings as offspring which share some features with each parent. The least-fit individuals are kept out from reproduction and so “die out”. A whole new population of possible solutions to the problem is generated by selecting the best (high fit) individuals from the current generation. This new generation contains characteristics which are better than their ancestors.

Progressing in this way, after many generations, owing to mixing and exchange of good characteristics, the entire population inherits the best characteristics and therefore turns out to be fit solutions to the problem. If the GA has been designed well, then most promising areas of search space are

explored, resulting in the population converging to an optimal solution to the problem.

Before a GA is executed, a suitable coding for the problem needs to be devised. The fitness function, which assigns merit to each of the individuals in the population, has to be formulated. During the run, parents must be selected for reproduction and crossed over to generate offspring. These aspects of the GA for the weight determination of the BPN are described in the following sections:

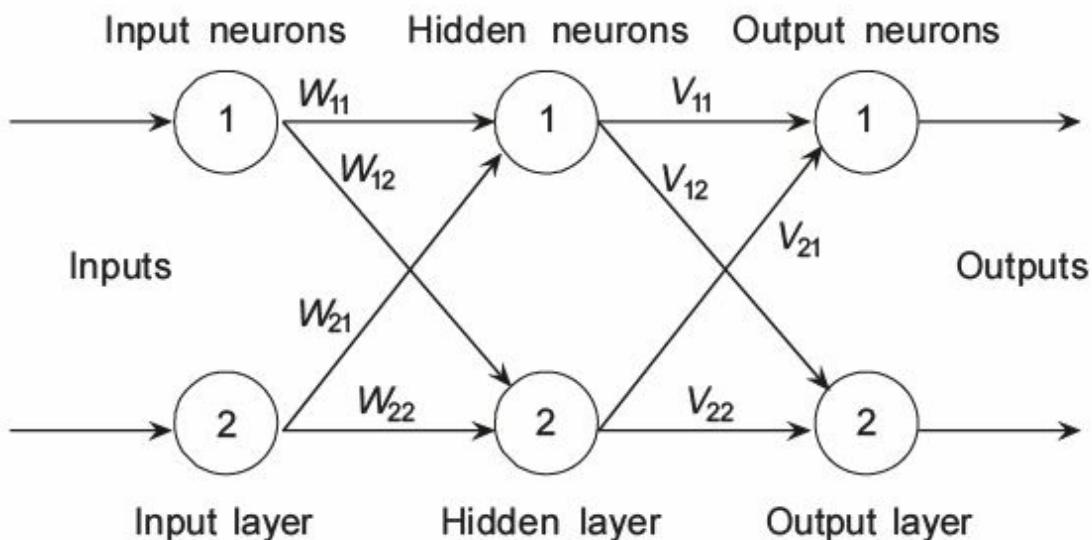
11.1.1 Coding

The parameters which represent a potential solution to the problem, *genes*, are joined together to form a string of values referred to as a *chromosome*.

Most conventional GAs code these chromosomes into *binary alphabet*.

However, in this work, binary coding has been dispensed with and a *real (decimal) coding* system adopted.

Assume a BPN whose network configuration is $l\text{-}m\text{-}n$ (l input neurons, m hidden neurons, and n output neurons). The number of weights that are to be determined are $(l + n)m$. With each weight (gene) being a real number and



assuming the number of digits (*gene length*) in the weight to be d , a string S of decimal values representing the $(l + n)m$ weights and therefore having a string length $L = (l + n)md$ is randomly generated. The string S represents the weight matrices of the input-hidden and hidden-output layers, in a linear form, arranged according to row-major or column-major order as selected by the designer. An initial population of p chromosomes is randomly generated where p is referred to as the *population size*.

Example

Consider a BPN network with a configuration 2-2-2 ($l = 2$ input neurons, $m =$

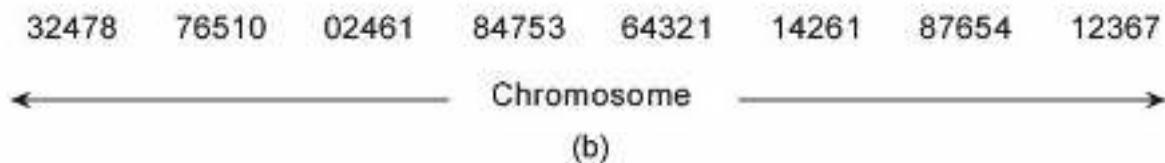
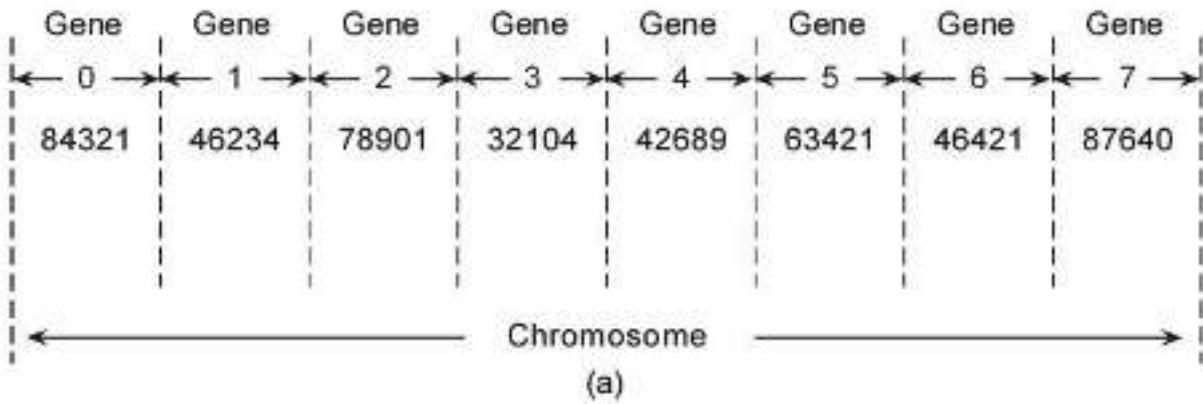
2 hidden neurons, and $n = 2$ output neurons) as shown in Fig. 11.1. The number of weights that are to be determined are $(2 \times 2) + (2 \times 2) = 8$.

With each weight being a real number and assuming the number of digits d to be randomly generated for representing a weight value as 5, the string S representing the chromosome of weights is $8 \times 5 = 40$, in length.

Fig. 11.1 A BPN with a 2-2-2 configuration.

Some sample chromosomes, randomly generated, are shown in Fig. 11.2.

Observe that a chromosome is made up of 8 genes representing 8 weights.



$$w_k = \begin{cases} + \frac{x_{kd+2}10^{d-2} + x_{kd+3}10^{d-3} + \dots + x_{(k+1)d}}{10^{d-2}}, & \text{if } 5 \leq x_{kd+1} \leq 9 \\ - \frac{x_{kd+2}10^{d-2} + x_{kd+3}10^{d-3} + \dots + x_{(k+1)d}}{10^{d-2}}, & \text{if } 0 \leq x_{kd+1} < 5 \end{cases}$$

Fig. 11.2 Sample chromosomes randomly generated for the BPN weights.

Choosing a population size of $p = 40$, an initial population of 40 chromosomes is randomly generated.

11.1.2 Weight Extraction

To determine the fitness values for each of the chromosomes, we extract weights from each of chromosomes.

Let $x_1, x_2, \dots, x_d, \dots, x_L$ represent a chromosome and $x_{kd+1}, x_{kd+2}, \dots, x_{(k+1)d}$ represent the k th gene ($k \geq 0$) in the chromosome. The actual weight w_k is given by

w_k

=

(11.2)

Example

For the chromosome given in Fig. 11.2, we have the weights extracted from the 8 genes as

Gene 0: 84321 We have $k = 0$, $d = 5$, and $xkd + 1$ which is $x 1$, is such that $5 \leq x 1 = 8 \leq 9$

Hence, the weight extracted is

$$+ \frac{4 \times 10^3 + 3 \times 10^2 + 2 \times 10 + 1}{10^3}$$

$$+ \frac{4 \times 10^3 + 3 \times 10^2 + 2 \times 10 + 1}{10^3}$$

$$\begin{cases} (I_{11}, I_{21}) & (T_{11}, T_{21}) \\ (I_{12}, I_{22}) & (T_{12}, T_{22}) \\ (I_{13}, I_{23}) & (T_{13}, T_{23}) \end{cases}$$

$$\bar{W}_1^0, \bar{W}_2^0, \dots, \bar{W}_{40}^0$$

$$w 0 =$$

$$= 4.321$$

Gene 1: 46234 We have $k = 1$, $d = 5$, and $xkd + 1$ which is $x 6$, is such that $0 \leq x 6 = 4 < 5$

Hence, the weight extracted is

$w_1 =$
 $= -6.234.$

Similarly,

Gene 2: 78901 yields $w_2 = +8.901$

Gene 3: 32104 yields $w_3 = -2.104$

Gene 4: 42689 yields $w_4 = -2.689$

Gene 5: 63421 yields $w_5 = +3.421$

Gene 6: 46421 yields $w_6 = -6.421$ and,

Gene 7: 87640 yields $w_7 = +7.640$

11.1.3 Fitness Function

The fitness function must be devised for each problem to be solved.

Algorithm FITGEN () illustrates the procedure.

Example

Let

represent a set of input-output pairs for a problem P to be solved by the BPN illustrated in Fig. 11.1.

We first randomly generate the initial population P_0 of size $p = 40$. Let C_0

0

0

$1, C_2, \dots, C_{40}$ represent the 40 chromosomes of P_0 . Also, let be the weight sets extracted from each of C_0

$i, i = 1, 2, \dots, 40$

using Eq. (11.2).

\bar{W}_i^0

$\bar{O}_1, \bar{O}_2, \bar{O}_3$

$$E = \sqrt{\frac{(E_1 + E_2 + E_3)}{3}}$$

$$F_1 = \frac{1}{E}$$

Algorithm FITGEN ()

{ Let (\bar{I}_i, \bar{T}_i) , $i = 1, 2, \dots, N$ where $\bar{I}_i = (I_{1i}, I_{2i}, \dots, I_{ni})$ and $\bar{T}_i = (T_{1i}, T_{2i}, \dots, T_{ni})$ represent the input-output pairs of the problem to be solved by BPN with a configuration l-m-n.
For each chromosome C_i , $i = 1, 2, \dots, p$ belonging to the current population P_i whose size is p
{ Extract weights \bar{W}_i from C_i with the help of equation (11.2);

For a fixed weight set

, the BPN is trained for all the input instances

given, i.e. the three input-output pairs given.

Let

be the calculated outputs of the BPN.

Compute

$$E 1 = (T_{11} - O_{11})^2 + (T_{21} - O_{21})^2$$

$$E_2 = (T_{12} - O_{12})^2 + (T_{22} - O_{22})^2$$

$$E_3 = (T_{13} - O_{13})^2 + (T_{23} - O_{23})^2$$

The root mean square of the error is

The fitness F

0

1 for the chromosome C_1 is given by

Keeping \bar{w}_i as a fixed weight setting, train the BPN for the N input instances;

Calculate error E_i for each of the input instances using the formula,

$$E_i = \sum_j (T_{ji} - O_{ji})^2 \quad (11.3)$$

where \bar{O}_i is the output vector calculated by BPN;

Find the root mean square E of the errors E_i , $i = 1, 2, \dots, N$

$$E = \sqrt{\frac{\sum_i E_i}{N}} \quad (11.4)$$

Calculate the fitness value F_i for each of the individual string of the population as

$$F_i = \frac{1}{E} \quad (11.5)$$

}

Output F_i for each C_i , $i = 1, 2, \dots, p$;

)

END FITGEN

$$\bar{W}_2^0$$

Similarly, the next weight set

is extracted from the next chromosome

C_0

2 . The BPN as before is trained using the extracted weights, for all the given input instances. The root mean square of the errors E is computed and the fitness value of the chromosome C_0

2 is given by $F_2 = 1/E$. Proceeding in

this way, the fitness values for all other chromosomes in the initial population are computed. Since the population size is $p = 40$, for the initial population $F_i, i = 1, 2, \dots, 40$ are computed. Figure 11.3 illustrates the computation of fitness values for the (initial) population.

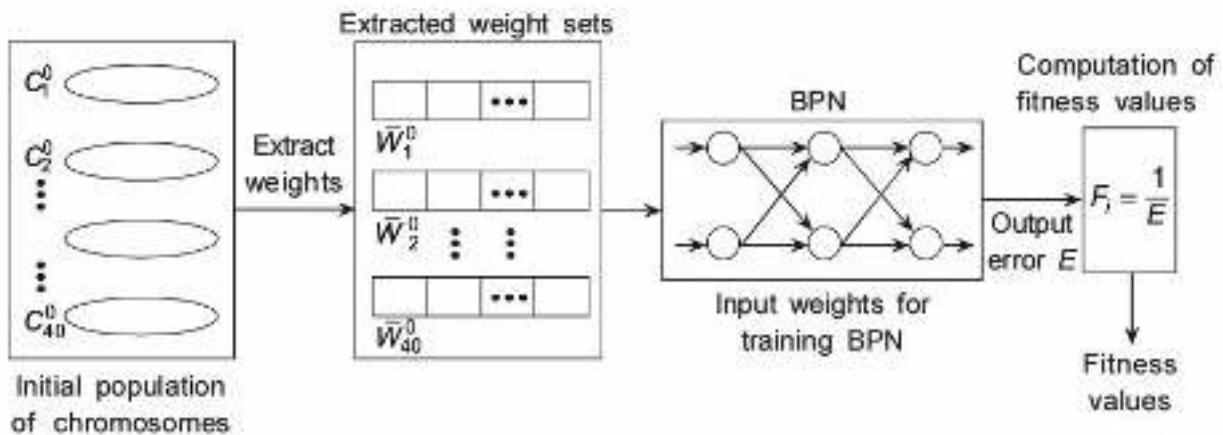


Fig. 11.3 Fitness function computation for the (initial) population.

11.1.4 Reproduction

In this phase, the mating pool is first formed, before the parent chromosomes reproduce to deliver offspring with better fitness. For the given problem, the mating pool is first formed by excluding that chromosome C_1 with the least fitness F_{\min} and replacing it with a duplicate copy of the chromosome C_k reporting the highest fitness F_{\max} . That is, the best fit individuals have multiple copies while worst fit individuals die off.

Having formed the mating pool, the parents are selected in pairs at random.

The chromosomes of the respective pairs are recombined using the two-point cross over operator of a standard GA. Recollect that in two-point cross over, the exchange of gene segments by the parent pair requires selection of cross-sites (cut-points). If P_a and P_b are two parent chromosomes, the off-springs O_a and O_b are produced as a result of executing the two-point cross over operator. This is illustrated in Fig. 11.4.

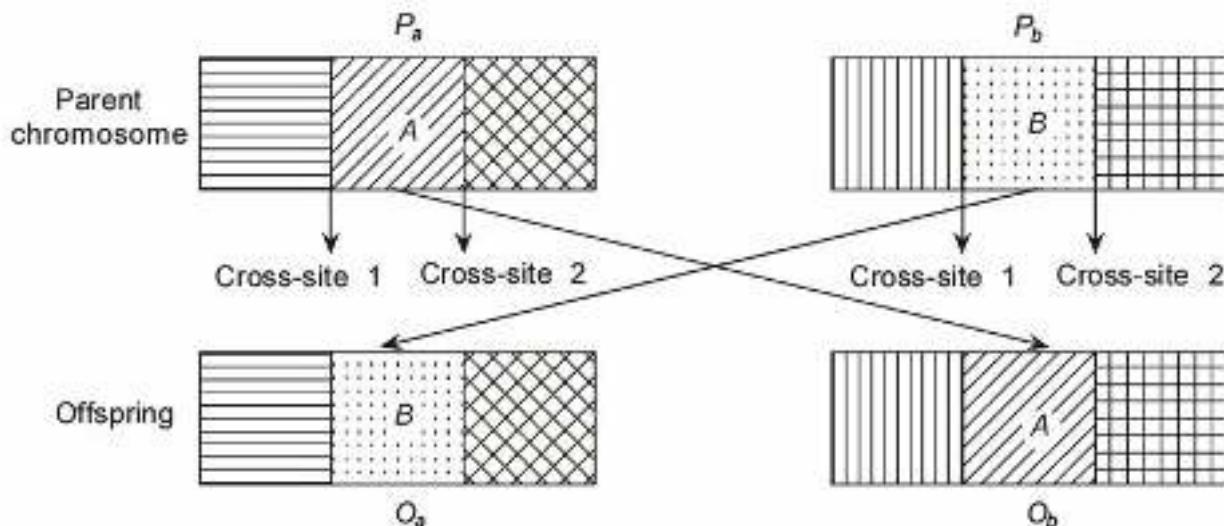


Fig. 11.4 Two-point cross over operator.

The offsprings which now form the current population again have their fitness calculated as illustrated by algorithm FITGEN().

Example

Consider the initial population of chromosomes P_0 generated earlier, with F_i , where $i = 1, 2, \dots, 40$, as their fitness values.

Let $F_{\max} = F_k$ and $F_{\min} = F_l$, (for $1 \leq l, k \leq 40$ where $l \neq k$) be the maximum and minimum fitness values. We remove all chromosomes with a fitness value F_{\min} and replace it with copies of chromosomes whose fitness value is F_{\max} .

Figure 11.5 illustrates the formation of the mating pool.

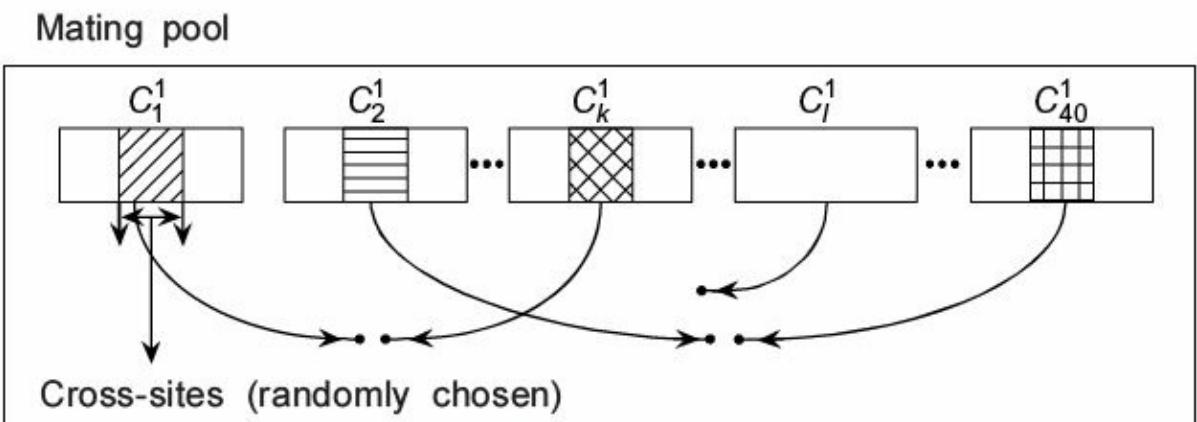
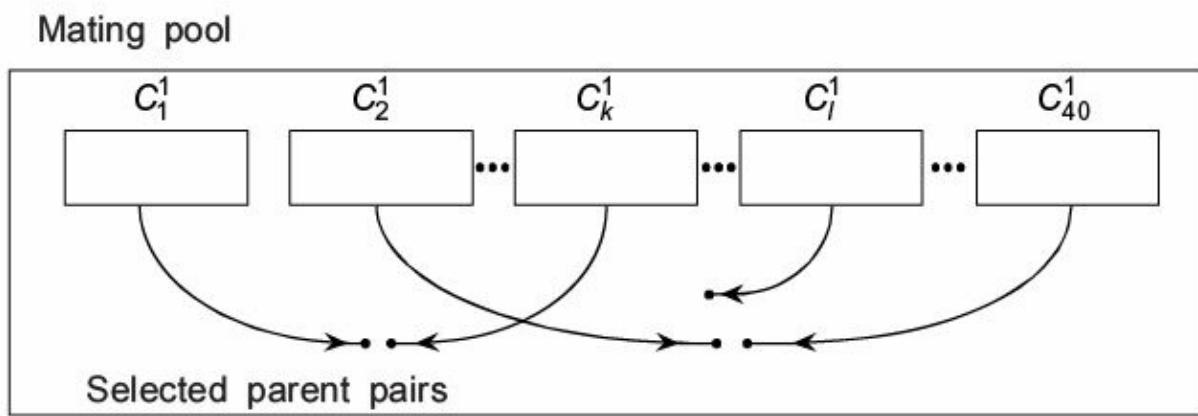
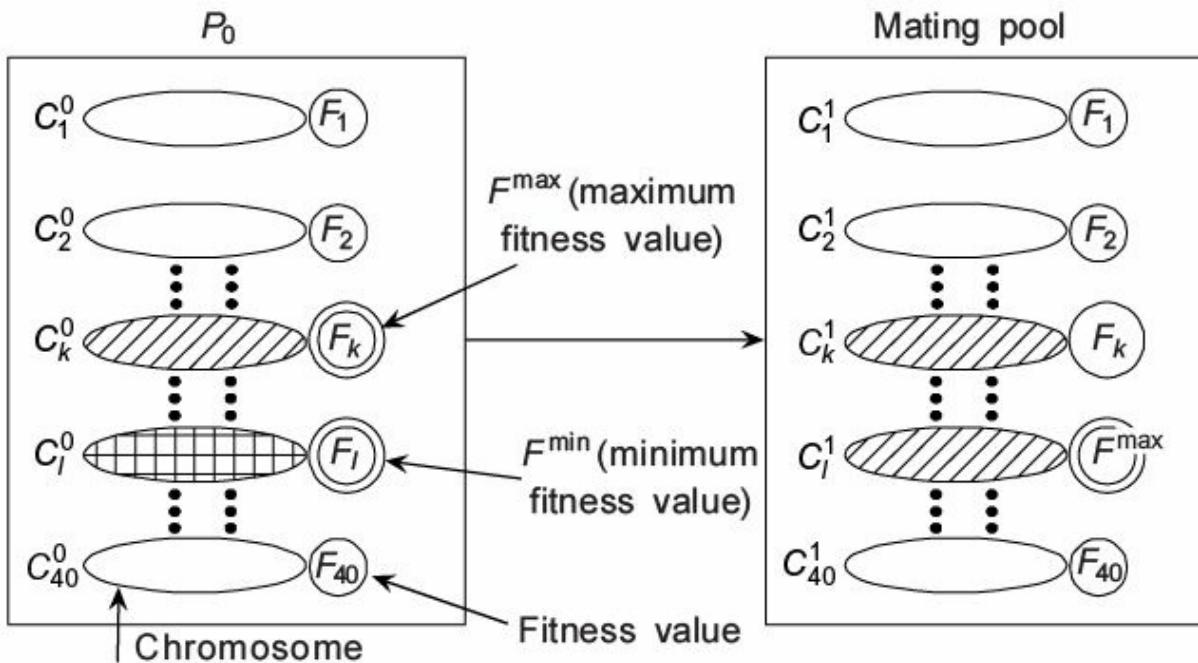


Fig. 11.5 Formation of the mating pool.

Now the reproduction of the offspring is done. Figure 11.6 illustrates a sample selection of parents for the application of the two-point cross over operator to produce offspring chromosomes. Here, the parents are selected in pairs at random. The cross-sites of the chromosome parent pairs are randomly determined for each pair as shown as Fig. 11.7. The genes are exchanged as shown in Fig. 11.8.

Fig. 11.6 Random selection of parent chromosomes.

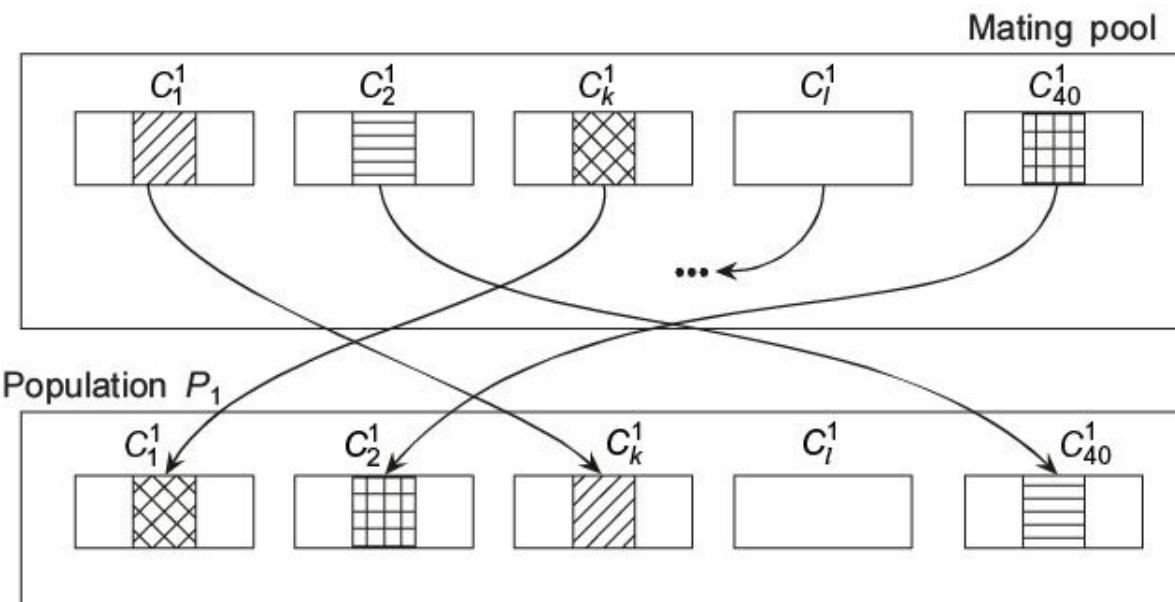


Fig. 11.7 Selection of cross-sites for the parent chromosome pairs.

Fig. 11.8 The new population P_1 after application of two-point cross over operator.

We call the new population P_1 . P_1 comprises 40 chromosomes which are the offsprings of the earlier population generation P_0 .

11.1.5 Convergence

For any problem, if the GA is correctly implemented, the population evolves over successive generations with the fitness value increasing towards the

global optimum. Convergence is the progression towards increasing uniformity. A population is said to have converged when 95% of the individuals constituting the population share the same fitness value.

The population P_1 now undergoes the process of selection, reproduction, and cross over. The fitness values for the chromosomes in P_1 are computed, the best individuals replicated and reproduction carried out using two-point cross over operator to form the next generation P_2 of chromosomes. The process of generation proceeds until at one stage 95% of the chromosomes in the population P_i converge to the same fitness value. At that stage, the weights extracted from the population P_i are the final weights to be used by the BPN.

The algorithm for the GA based weight determination can be summarized as illustrated in **Algorithm GA-NN-WT()**.

```
Algorithm GA-NN-WT()

{ i ← 0;

  Generate the initial population  $P_i$  of real-coded chromosomes
   $C_j^i$  each representing a weight set for the BPN;

  While the current population  $P_i$  has not converged
    { Generate fitness values  $F_j^i$  for each  $C_j^i \in P_i$  using the
      Algorithm FITGEN();
```

```

    Set the mating pool ready by terminating worst fit individuals
    and duplicating high fit individuals;

    Using the cross over mechanism, reproduce offspring from
    the parent chromosomes;

        i ← i + 1;

    Call the current population  $P_i$ 

    Calculate fitness values  $F_j^i$  for each  $C_j^i \in P^i$ ;

    }

    Extract weights from  $P_i$  to be used by the BPN ;

}

END GA-NN-WT.

```

Illustration

The problem is to determine the weights for a BPN with a configuration 2-2-1 using GA, for the following input-output set of data.

S. no.

Input

Output

1

(0.33, 0.66)

0.5

2

(0.33, 1)

0.67

3

(0.33, 0.33)

0.33

4

(0.66, 0.66)

0.67

5

(0.66, 1)

0.83

6

(1, 0.66)

0.83

7

(0.66, 0.33)

0.5

8

(1, 0.33)

0.67

9

(1, 1)

1

The number of weights to be determined are 6 and choosing $d = 5$, the chromosome length is 30. We first generate the initial population P_0 of 30 randomly generated chromosomes.

Figure 11.9 illustrates a sample P_0 . The input-hidden layer weights and the hidden-output layer weights extracted from P_i are shown in Fig. 11.10.

1 722553040782560513957266122277
2 036817284432815831745141984492
3 802190277603921708897318962768
4 419032336621777646940174170808
5 928601102550642618319386744630
6 822995001720253710305088110377'
7 744342402274624732342263362035
8 923253154182050036478123221475
9 808083108221026924189121104292
10 901882326354891102647450733990
11 449771252353867247298119584884
12 803606259242332545892494141890
13 239851184513755314301353152083
14 111227180961268310077018920559
15 110947016044073816348397261537
16 627117214754344342000321300941
17 53522540438403421111376683589
18 126857480741973911201297063328
19 128306000442705649147276924568
20 012648254831460941511253543686
21 322692304682483746427048761994
22 038009160042511025959365591162
23 244943497743605506497282660685
24 438672027332440137878221471817
25 733023069761922522591112954836
26 803411185660750208009361230397
27 241137400442701634004396751076
28 816863387873670225026268573075
29 438672278833711634004396751076
30 139846259021330406593051393147

Fig. 11.9 P_0 —the initial population of chromosomes.

Input-hidden layer weights (extracted from P_0)				Hidden-output layer weights (extracted from P_0)	
2.255000	-0.407000	2.560000	1.395000	2.661000	-2.277000
-3.681000	2.844000	-2.815000	3.174000	1.419000	4.492000
0.219000	-2.776000	3.921000	0.889000	3.189000	2.768000
-1.903000	-3.366000	-1.777000	4.694000	-1.741000	0.808000
2.860000	-1.025000	0.642000	1.831000	3.867000	-4.630000
2.299000	0.017000	0.253000	1.030000	0.881000	-0.377000
4.434000	-4.022000	4.624000	3.234000	-2.633000	2.035000
2.325000	-1.541000	2.050000	-3.647000	1.232000	-1.475000
0.808000	-1.082000	-1.026000	2.418000	1.211000	-4.292000
0.188000	-3.263000	4.891000	-0.264000	4.507000	-3.990000
-4.977000	-2.523000	3.867000	-4.729000	1.195000	4.884000
0.360000	2.592000	-2.332000	4.589000	-4.941000	-1.890000
-3.985000	-1.845000	-3.755000	-1.430000	-3.531000	2.083000
-1.122000	1.809000	1.268000	1.007000	0.189000	-0.559000
-1.094000	0.160000	-4.073000	1.634000	3.972000	1.537000
2.711000	2.147000	4.344000	4.200000	-3.213000	-0.941000
3.522000	4.043000	4.034000	-1.111000	-3.766000	3.589000
-2.685000	4.807000	-1.973000	1.120000	-2.970000	3.328000
-2.830000	0.004000	-2.705000	4.914000	2.769000	-4.568000
-1.264000	2.548000	-1.460000	4.151000	-2.535000	-3.686000
-2.269000	-3.046000	2.483000	4.642000	0.487000	1.994000
-3.800000	1.600000	-2.511000	-2.595000	3.655000	1.162000
-4.494000	-4.977000	-3.605000	0.649000	2.826000	0.685000
-3.867000	-0.273000	-2.440000	-3.787000	2.214000	1.817000
3.302000	-0.697000	1.922000	2.259000	-1.129000	4.836000
0.341000	-1.856000	0.750000	0.800000	3.612000	-0.397000
-4.113000	4.004000	-2.701000	3.400000	-3.967000	1.076000
1.686000	-3.878000	3.670000	-2.502000	2.685000	3.075000
-3.867000	-2.788000	-3.711000	3.400000	-3.967000	1.076000
-3.984000	2.590000	-1.330000	-0.659000	-0.513000	3.147000
4.924431	2.676982	3.675898	5.396611	4.209667	5.162605
1.806555	5.360716	1.647623	2.795751	4.145098	1.562493
4.156507	3.307374	4.140387	1.509401	2.628218	2.965435
1.472241	1.479448	4.008221	4.259435	3.915377	3.882587
3.331693	3.299846	5.608877	3.042581	5.500490	3.335596
Minimum fitness value		Maximum fitness value			

Fig. 11.10 Weights extracted from P_0 .

The fitness values computed for the chromosomes in population P_0 are as shown in

Fig. 11.11. The mating pool after duplicating worst-fit chromosomes in P_0

with best-fit chromosomes is shown in Fig. 11.12.

722553040782560513957266122277	
036817284432815831745141984492	
802190277603921708897318962768	
419032336621777646940174170808	
928601102550642618319386744630	
822995001720253710305088110377	
744342402274624732342263362035	
923253154182050036478123221475	
808083108221026924189121104292	
901882326354891102647450733990	
449771252353867247298119584884	
803606259242332545892494141890	
239851184513755314301353152083	
111227180961268310077018920559	
110947016044073816348397261537	
627117214754344342000321300941	
53522540438403421111376683589	
126857480741973911201297063328	
241137400442701634004396751076	← Duplication of best-fit chromosome after removal of worst-fit chromosome
012648254831460941511253543686	
322692304682483746427048761994	
038009160042511025959365591162	
244943497743605506497282660685	
438672027332440137878221471817	
733023069761922522591112954836	
803411185660750208009361230397	
241137400442701634004396751076	← Best-fit chromosome
816863387873670225026268573075	
438672278833711634004396751076	
139846259021330406593051393147	

Fig. 11.11 Fitness values of chromosomes in P_0 .

Fig. 11.12 Mating pool after duplicating worst-fit chromosomes with best-fit chromosomes.

The random selection of parent pairs and their cross-sites is shown in Fig. 11.13.

Parent pairs represented by chromosome number	Cross-sites	
	Start position	End position
(1, 16)	29	30
(2, 26)	3	19
(3, 11)	26	29
(4, 23)	4	13
(5, 6)	18	19
(7, 19)	6	25
(8, 21)	12	16
(9, 27)	22	25
(10, 15)	10	11
(12, 22)	4	26
(13, 20)	29	30
(14, 24)	1	12
(17, 28)	22	30
(18, 25)	22	24
(29, 30)	1	6

Fig. 11.13 Selection of parent pairs and their cross-sites.

After reproduction using a two-point cross over operator, the new population P_1 is as shown in Fig. 11.14.

722553040782560513957266122241
033411185660750208045141984492
802190277603921708897318984888
419943497743677646940174170808
928601102550642610319386744630
822995001720253718305088110377
744347400442701634004396762035
923253154182483736478123221475
808083108221026924189396704292
901882326044891102647450733990
449771252353867247298119562764
803009160042511025959365591890
239851184513755314301353152086
438672027332268310077018920559
110947016354073816348397261537
627117214754344342000321300977
53522540438403421111268573075
126857480741973911201112063328
241132402274624732342263351076
012648254831460941511253543683
322692304682050046427048761994
038606259242332545892494141162
244032336621705506497282660685
111227180961440137878221471817
733023069761922522591297954836
806817284432815831709361230397
241137400442701634004121151076
816863387873670225026376683589
139846278833711634004396751076
438672259021330406593051393147

Fig. 11.14 The new population P_1 after two-point cross over.

The next set of weights extracted from P_1 to determine the fitness values of chromosomes in P_1 is shown in Fig. 11.15.

← Input-hidden layer weights →				← Hidden-output layer weights →	
2.255000	-0.407000	2.560000	1.395000	2.661000	-2.241000
-3.341000	-1.856000	0.750000	-0.804000	1.419000	4.492000
0.219000	-2.776000	-3.921000	0.889000	3.189000	4.888000
-1.994000	-4.977000	-3.677000	4.694000	-1.741000	0.808000
2.860000	-1.025000	0.642000	1.031000	3.867000	-4.630000
2.299000	0.017000	-0.253000	1.830000	0.881000	-0.377000
4.434000	4.004000	-2.701000	3.400000	-3.967000	2.035000
2.325000	-1.541000	2.483000	3.647000	1.232000	-1.475000
0.808000	-1.062000	-1.026000	2.418000	3.967000	-4.292000
0.188000	-3.260000	-4.891000	-0.264000	4.507000	-3.990000
-4.977000	-2.523000	3.867000	-4.729000	1.195000	2.764000
0.300000	1.600000	-2.511000	-2.595000	3.655000	1.890000
-3.985000	-1.845000	-3.755000	-1.430000	-3.531000	2.086000
-3.867000	-0.273000	-2.268000	-1.007000	0.189000	-0.559000
-1.094000	0.163000	4.073000	1.634000	3.972000	1.537000
2.711000	2.147000	4.344000	-4.200000	-3.213000	-0.977000
3.522000	4.043000	4.034000	-1.111000	-2.685000	3.075000
-2.685000	4.807000	-1.973000	1.120000	-1.120000	3.328000
-4.113000	-4.022000	4.624000	3.234000	-2.633000	1.076000
-1.264000	2.548000	-1.460000	4.151000	-2.535000	-3.683000
-2.269000	-3.046000	2.050000	-4.642000	0.487000	1.994000
-3.860000	2.592000	-2.332000	4.589000	-4.941000	-1.162000
-4.403000	-3.366000	-1.705000	0.649000	2.826000	0.685000
-1.122000	1.809000	1.440000	-3.787000	2.214000	1.817000
3.302000	-0.697000	1.922000	2.259000	-2.979000	4.836000
0.681000	2.844000	-2.815000	3.170000	3.612000	-0.397000
-4.113000	4.004000	-2.701000	3.400000	-1.211000	1.076000
1.686000	-3.878000	3.670000	-2.502000	3.766000	3.589000
-3.984000	2.788000	-3.711000	3.400000	-3.967000	1.076000
-3.867000	-2.590000	-1.330000	-0.659000	-0.513000	3.147000

Fig. 11.15 Extraction of weights from $P 1$

We now proceed to demonstrate the computation of fitness values on the chromosomes of population $P 1$ illustrated in Fig. 11.14.

Computation of fitness values

$$\begin{bmatrix} 2.255 & -0.407 \\ 2.56 & -2.241 \end{bmatrix}$$

$$\begin{bmatrix} 2.661 \\ -2.241 \end{bmatrix}$$

$$[H]^{\text{in}} = [IH]^T I_1^T = \begin{bmatrix} 2.255 & 2.56 \\ -0.407 & 1.395 \end{bmatrix} \begin{bmatrix} 0.33 \\ 0.66 \end{bmatrix} = \begin{bmatrix} 2.43375 \\ 0.786390 \end{bmatrix}$$

$$\begin{bmatrix} 0.919365 \\ 0.687056 \end{bmatrix}$$

$$[O^{\text{in}}] = [HO]^T [H^{\text{out}}] = [2.661 - 2.241] \begin{bmatrix} 0.919365 \\ 0.687056 \end{bmatrix} = 0.906739$$

Consider the first chromosome of population $P 1$. The aim is to compute its fitness value.

The weights extracted from the first chromosome are (Refer Fig. 11.14 and Fig. 11.15)

Input-hidden layer weight matrix

$$[IH] =$$

Hidden-output layer weight matrix

[*HO*] =

(11.6)

We now begin a series of computations to obtain the cumulative error E of training the BPN for the weight set given in Eq. (11.6), for all the input instances of the problem.

For the first input instance, $I_1 = (0.33, 0.66)$ with a target output $T_1 = 0.5$.

The input to the hidden layer neurons are

(11.7)

Applying sigmoidal function, i.e., $f(x) = 1/(1 + e^{-x})$ to [H in], the output of the hidden layer neurons is

[*H* out] =

(11.8)

The input to the output layer neurons are

(11.9)

Applying sigmoidal function to [O in] the calculated output of the BPN

$$[O \text{ out}] = 0.712332 \quad (11.10)$$

The error is

$$(T1 - [O_{out}])2 = (0.5 - 0.712332)2 = 0.045085 \quad (11.11) A$$

similar set of computations are repeated for the same weight set but for the other input instances. Table 11.1 illustrates the computations.

Table 11.1 Computation of fitness value for the first chromosome in P_1

Weights		Input	Output	Hidden layer output	Calculated output	Cumulative error
Input-hidden layer	Hidden-output layer					
$\begin{bmatrix} 2.255 & 0.407 \\ 2.56 & 1.395 \end{bmatrix}$	$\begin{bmatrix} 2.661 \\ -2.241 \end{bmatrix}$	(0.33, 0.66)	0.5	(0.919365, 0.687056)	0.712332	0.045085
		(0.33, 1)	0.67	(0.964571, 0.779145)	0.694376	0.045679
		(0.33, 0.33)	0.33	(0.830468, 0.580796)	0.712655	0.192104
		(0.66, 0.66)	0.67	(0.959994, 0.657479)	0.746708	0.197988
		(0.66, 1)	0.83	(0.982847, 0.755170)	0.715657	0.211602
		(1, 0.66)	0.83	(0.981009, 0.625673)	0.770003	0.214662
		(0.66, 0.33)	0.5	(0.911582, 0.547786)	0.768192	0.286589
		(1, 0.33)	0.67	(0.956885, 0.513334)	0.801530	0.303889

$$F_1 = \frac{1}{E} = \frac{1}{0.375524} = 2.6629$$

The fitness value for the first chromosome is given by

Table 11.2 illustrates the computations using the second weight set extracted from the second chromosome of population P_1 to arrive at its fitness value. The fitness value is given by 1.40564. Thus, the computations of the fitness values for all other chromosomes in the population are done.

The generations progress and for this problem 95% convergence was attained in less than 150 generations.

Table 11.2 Computation of fitness value for the second chromosome of population P_1

Weights		Input	Output	Hidden layer output	Calculated output	Cumulative error
Input-hidden layer	Hidden-output layer					
$\begin{bmatrix} -3.341 & -1.856 \\ 0.75 & -0.804 \end{bmatrix}$	$\begin{bmatrix} 1.419 \\ 4.492 \end{bmatrix}$	(0.33, 0.66)	0.5	(0.352623, 0.241748)	0.830095	0.108963
		(0.33, 1)	0.67	(0.412769, 0.195214)	0.811933	0.129108
		(0.33, 0.33)	0.33	(0.298379, 0.293634)	0.850992	0.400541
		(0.66, 0.66)	0.67	(0.153156, 0.147342)	0.706654	0.401884
		(0.66, 1)	0.83	(0.189224, 0.116196)	0.687930	0.422068
		(1, 0.66)	0.83	(0.054888, 0.084197)	0.612089	0.469553
		(0.66, 0.33)	0.5	(0.123731, 0.183880)	0.731366	0.523083
		(1, 0.33)	0.67	(0.043376, 0.107042)	0.632367	0.524500
		(1, 1)	1	(0.069720, 0.065375)	0.596908	0.686983

11.2 APPLICATIONS

11.2.1 K-factor Determination in Columns

This problem chosen from structural engineering is to determine the K-factor for columns given their relative stiffness ratios ψA and ψB at the two ends A and B of a column.

Conventional methods call for the solving of equations, thereby demanding heavy computer analysis. Making use of a BPN for the problem requires training the BPN for a set of input instances before the network can determine the K-factor for a given $(\psi A, \psi B)$ pair of values.

Here, the BPN has a 2-3-1 configuration. The two neurons in the input layer represent the intake of ψA and ψB as the network inputs and one neuron in the output layer represents the output of K-factor.

Table 11.3 illustrates a sample set of $(\psi A, \psi B)$ values and their corresponding K-factor values all of which have been normalized to lie between 0 and 1.

Table 11.3 Sample $(\psi A, \psi B)$ and K-factor values A

B

K

0

0

0.5

0.08

0.08

0.841615

0.8

0.8

0.986750

0.16

0.04

0.832753

1.0

0.2
0.968245
0.2
0.04
0.841905
0.8
0.2
0.965504
0.12
0.4
0.933234
0.2
0.4
0.952807
0.04
0
0.628571

Table 11.4 illustrates the weight determined by the GA based BPN after about 140 generations. Table 11.5 shows the calculated K-factor values by the GA based BPN and the expected values for the data set also shown in

Table 11.4 Weights determined by the GA based BPN

Input-hidden layer Weight-matrix	Hidden-output layer Weight-matrix
$\begin{bmatrix} 2.86 & -3.85 \\ 4.891 & -2.595 \end{bmatrix}$	$\begin{bmatrix} 3.817 \\ -2.967 \end{bmatrix}$

Table 11.3.

.....

Table 11.5 Expected value and calculated K-factor values by the GA based BPN

K-factor value

Expected

Calculated

0.5

0.604679

0.841615

0.797861

0.986750

0.977954

0.832753

0.823423

0.968245

0.975865

0.841905

0.849938

0.965504

0.973354

0.933224

0.949219

0.952807

0.957605

0.628571

0.656611

11.2.2 Electrical Load Forecasting

This application discussed by Montgomery and Askin, (Montgomery and Askin, 1981) investigates the factors which influence the load or demand for electricity by residential customers. The factors identified were: x_1 —the size of the customers' house, x_2 — the annual family income, x_3 —tons of the air conditioning capacity, x_4 —the appliance index for the house, and x_5 —the number of residents in the house on a weekday.

However, in this problem we have only considered the influence of x_1 , x_3 and x_4 on the load y . The GA based BPN has a configuration 3-2-1 since the three inputs are x_1 , x_3 and x_4 and the output is y .

Table 11.6 illustrates a sample set of x_1 , x_3 , x_4 and the corresponding y values which have been normalized to lie between 0 and 1. The weight-matrix determined by the BPN after about 500 generations is given in Table 11.7. The testing data set, the expected, and computed values by the GA based BPN are illustrated in Table 11.8.

Table 11.6 Sample training set for the electrical load forecasting problem x_1

x_3

x_4

y

0.7791

1

0.8273

1

0.4750

0.2143

0.5577

0.4761

0.6434

0.9286

0.6718

0.7861

0.5755

0.5714

0.6089

0.6371

0.6789

0.5714

0.6602

0.6647

0.3443

0.1423

0.3306

0.2982

0.8289

0.7143

1

0.9879

0.5856

0.4286

0.6524

0.6030

0.7094

0.5

0.8178

0.7968

0.5166

0.3571

0.5547

0.5455

0.2901

0

0.2735

0.2241

0.5811

0.5714

0.5880

0.6065

0.5506

0.3571

0.6594

0.6195

0.4361

0.1429

0.5537

0.4191

0.4561

0.1429

0.4949

0.3959

0.4489

0.2143

0.4730

0.3814

0.3886

0.0714

0.4225

0.3541

0.5213

0.3571

0.6624

0.5803

0.5053

0.2143

0.5196

0.3979

0.4223

0.2143

0.4206

0.3679

0.5745

0.7143

0.5351

0.7080

0.7109

0.7857

0.8301

0.8716

0.4856

0.2857

0.4707

0.4969

0.7107

0.8571

0.7476

0.9040

0.4615

0.4283

0.5581

0.5980

0.3467

0.2143

0.3151

0.3766

Table 11.7 Weights determined by the GA based BPN for the training set

Input-hidden layer weights	Hidden-output layer weights
$\begin{bmatrix} -0.755 & -4.082 \\ -2.51 & 4.062 \\ -0.529 & 2.789 \end{bmatrix}$	$\begin{bmatrix} -4.977 \\ 1.659 \end{bmatrix}$

0.6220

0.6423

0.6884

0.7309

0.6947

0.7857

0.7737

0.8853

0.3270

0.0714

0.3879

0.3125

0.7033

0.8571

0.7633

0.9077

0.5474

0.7143

0.6336

0.7122

0.6129

0.8571

0.7288

0.7718

0.6045

0.4286

0.6380

0.6371

0.6698

0.7857

0.7132

0.8357

0.5420

0.1429

0.5975

0.4522

0.6634

0.7143

0.7381

0.8091

0.6048

0.5

0.6525

0.7635

0.4363

0.2857

0.5166

0.4787

0.5548

0.7143

0.5284

0.6664

0.5905

0.3571

0.7379

0.6152

....

Table 11.8 Sample testing set, the expected, and calculated y values by the GA based BPN

y

x_1

x_3

x_4

Computed by

Expected

GA based BPN

0.4750

0.2143

0.5577

0.4761

0.46705

0.5014

0.3571

0.5928

0.5265

0.59559

0.6434

0.9286

0.6718

0.7861

0.79923

0.4149

0.1429

0.3905

0.3472

0.34498

0.5856

0.4286

0.6524

0.6030

0.64507

0.5629

0.4286

0.5555

0.6325

0.62272

0.5166

0.3571

0.5547

0.5455

0.58065

0.3723

0.1429

0.4061

0.3519

0.35984

0.5506

0.3571

0.6594

0.6195

0.60631

0.5531

0.2857

0.6320

0.5692

0.54105

0.4361

0.1429

0.5537

0.4191

0.40459

0.5713

0.4286

0.5926

0.6261

0.63159

0.4561

0.1429

0.4949

0.3959

0.37690

0.5605

0.4286

0.6493

0.6040

0.64859

0.4910

0.2857

0.5646

0.5176

0.53111

0.4856

0.2857

0.4707

0.4969

0.49761

0.6220

0.6423

0.6884

0.7309

0.74319

0.3270

0.0714

0.3879

0.3125

0.29251

0.4767

0.1429

0.4786

0.3891

0.36598

0.6129

0.8571

0.7288

0.7718

0.79333

0.6045

0.4286

0.6380

0.6371

0.63761

0.5905

0.3571

0.7379

0.6152

0.62157

SUMMARY

GA based backpropagation network is a hybrid architecture in which a BPN employs genetic algorithms for the determination of its weights.

For a BPN with a configuration $l \cdot m \cdot n$, the number of weights that are to be determined are $(l + n) m$. Selecting a gene length of d , the length of the chromosome string S comprising $(l + n) m$ genes is $(l + n) md$.

Initially a population P_0 of Chromosomes of size N is randomly generated. The weight sets for the BPN are extracted from P_0 . For each weight set, the BPN is trained for all the input instances of the given problem. The error in

the training is utilized to compute the fitness value for each of the chromosomes.

In the next phase, the worst-fit chromosomes are terminated and replaced by best-fit chromosomes. The parent chromosomes are randomly selected in pairs and a two-point cross over operator is applied to produce offspring. The new population P_1 again has its fitness values computed after extraction of weights and computation of error. The generations progress until the population converges to the same fitness values.

The weights extracted from the ‘converged’ population are the final weights of the BPN.

The application of GA based BPN has been demonstrated on two problems, namely

1. K-factor design and
2. Electrical load forecasting.

PROGRAMMING ASSIGNMENT

P11.1 Compare the performance of a conventional BPN with that of the GA based BPN for the Electrical load forecasting problem discussed in Sec. 11.2.2.

- (a) Make use of MATLAB’s neural network tool box to implement the conventional BPN algorithm.
- (b) Make use of the data set in Table 11.6 to train the BPN.
- (c) Obtain the results for the testing set in Table 11.8 and compare it with the computed output of the GA based BPN.

P11.2 Implement a GA based BPN using C/C++. Compare its performance with that of a conventional BPN with regard to the number of training iterations/generations.

P11.3 Modify the GA based BPN algorithm to handle binary chromosomes.

- (a) Design a procedure to extract weights from the binary chromosomes
- (b) Make use of algorithm FITGEN() and the procedure designed in P11.3(a) to compute the fitness values of the chromosomes.
- (c) Employ multi-point cross over technique for the reproduction.

SUGGESTED FURTHER READING

Montana and Davis (Montana & Davis, 1989) discuss the use of some domain specific GA operators to train the BPN instead of the conventional learning rule. Maniezzo (Maniezzo, 1994) discusses a hybrid system involving the application of GA for neural network design. Schiffman *et al.* (Schiffman *et al.*, 1992, 1993) propose approaches for automatic topology optimization of a BPN.

REFERENCES

Fu Limin (1994), *Neural Networks in Computer Intelligence*, McGraw Hill Inc.

Harp, S., T. Samad, and A. Guha (1989), Towards the Genetic Synthesis of Neural Networks , *Proc of the Third Intl Conf on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.

Kitano, H. (1990), Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms, *Proc of the Eighth Natl Conf on AI (AAAI-90)*, pp. 789–795.

Montgomery, D.C. and R.G. Askin (1981), Problems of Non-normality and Multi-collinearity for Forecasting Methods based on Least Squares, *AIEE Trans*, Vol. 13, pp. 102–115.

Rajasekaran, S. and G.A. Vijayalakshmi Pai (1996), Genetic Algorithm based Weight Determination for Backpropagation Networks, *Proc of the*

Fourth Intl Conf on Advanced Computing, pp. 73–79.

Rumelhart, D.E., G.R. Hinton and R.J. Williams (1986), Learning Internal Representations by Error Propagation, in *Parallel Distributed Processing*, Vol. 1, Rumelhart, D.E. and

McClelland, J.L., Eds., MIT Press, Cambridge MA.

Whitley, D. and C. Bogart (1989), Optimizing Neural Networks using Faster More Accurate Genetic Search, *Proc of the Third Intl Conf on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.

Whitley, D. and C. Bogart (1990), The Evolution of Connectivity Pruning Neural Networks using Genetic Algorithms, *Proc of IJCNN-90-Wash DC*, Jan.

Whitley, D. and T. Hanson (1989), Optimizing Neural Networks using Faster More Accurate Genetic Search, *Proc of the Third Intl Conf on Genetic Algorithms and their Applications*, J D Schaffer, Ed., Morgan Kaufmann, San Mateo, CA, pp. 391–396.

Whitley, D. and Starkwerther (1990), Optimizing Small Neural Networks using a Distributed Genetic Algorithm, *Proc of IJCNN-90-Wash DC*, Jan.

Chapter 12

Fuzzy Backpropagation Networks

In Chapter 3, we discussed in detail, the backpropagation network (BPN) architecture and its applications. Various attempts have been initiated to hybridize BPN by incorporating fuzzy logic (Tsoukalas and Uhrig, 1997; Adeli and Hung, 1995). In this chapter, we elaborate on such an NN-FL hybrid architecture, namely Lee and Lu's Fuzzy BP network (Lee and Lu, 1994). Fuzzy BP is a hybrid architecture which maps fuzzy inputs to crisp outputs. The fuzzy neurons in the model make use of LR-type fuzzy numbers. Besides, triangular type of LR-type fuzzy numbers have been used for simplification of architecture and reduction of computational load.

The LR-type fuzzy numbers and their operations required by the fuzzy BP

architecture are first presented. The structure of a fuzzy neuron, the architecture of fuzzy BP, its learning mechanism, and algorithms are elaborated next. Finally, the application of Fuzzy BP to the problems of knowledge base evaluation and Earthquake damage evaluation are discussed.

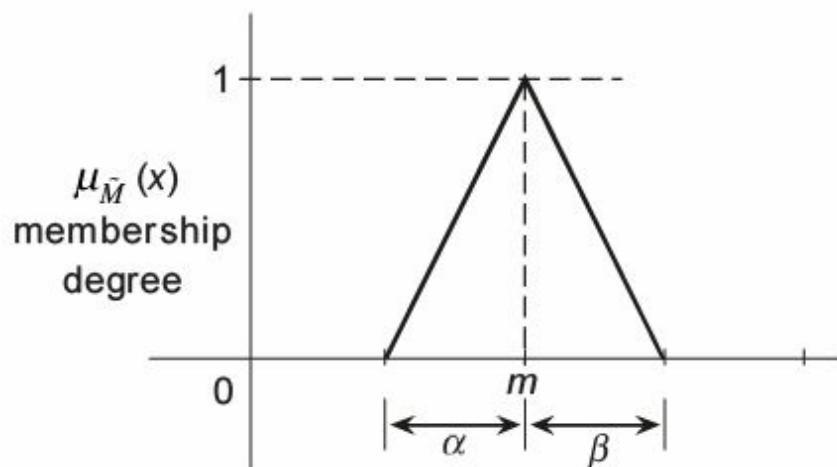
\tilde{M}

$$\mu_{\tilde{M}}(x) = \begin{cases} L\left(\frac{m-x}{\alpha}\right) & \text{for } x \leq m \\ R\left(\frac{x-m}{\beta}\right) & \text{for } x \geq m \end{cases}$$

\tilde{M}

$\mu_{\tilde{M}}$

\tilde{M}



12.1 LR-TYPE FUZZY NUMBERS

The LR-type fuzzy numbers are special type of representations for fuzzy numbers, proposed by Dubois and Prade (1979). They introduced functions called L (and R) which map $R^+ \rightarrow [0, 1]$ and are decreasing shape functions if

$$L(0) = 1,$$

$$\begin{aligned} L(x) < 0, \forall x < 1, \\ x) > 0 \forall x, \text{ and } L(\infty) = 0] \end{aligned} \quad (12.1) \quad L(1) = 0 \text{ or } [L($$

Definition

A fuzzy number

is of LR-type if there exist reference functions L (for left), R (for right) and scalars, $\alpha > 0$, $\beta > 0$ with

(12.2)

Here, m , called the mean value of

, is a real number and α and β are

called the left and right spreads, respectively.

Here,

is the membership function of fuzzy number

. An LR-type

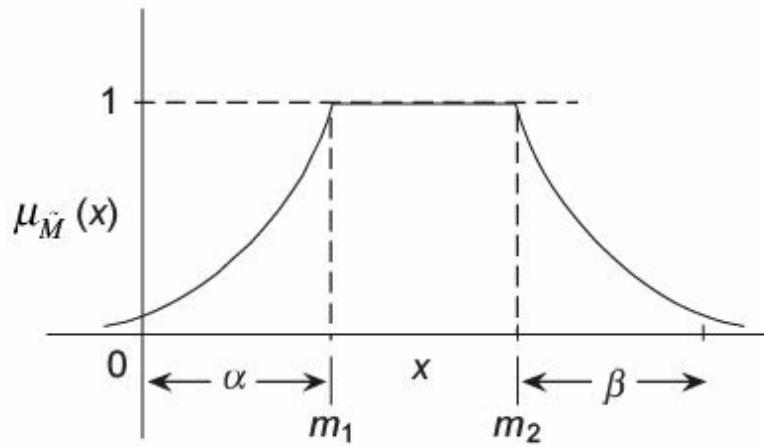
fuzzy number M can be expressed as $(m, \alpha, \beta) LR$ (Refer Fig. 12.1). If α and β

are both zero, the LR-type function indicates a crisp value.

Fig. 12.1 Symmetric triangular LR-type fuzzy number.

For $L(z)$, different functions may be chosen. Dubois and Prade (Dubois and Prade, 1988) mention $L(x) = \max(0, 1 - xp)$ $p > 0$, $L(x) = \exp(-x)$, $L(x) =$

$$\mu_{\tilde{M}}(x) = \begin{cases} L\left(\frac{m_1 - x}{\alpha}\right) & \text{for } x \leq m_1, \alpha > 0 \\ R\left(\frac{x - m_2}{\beta}\right) & \text{for } x \geq m_2, \beta > 0 \\ 1 & \text{otherwise} \end{cases}$$



\tilde{M}

\tilde{N}

\tilde{M}

\tilde{N}

$$(m, \alpha, \beta)_{LR} \oplus (n, \gamma, \delta)_{LR} = (m+n, \alpha+\gamma, \beta+\delta)_{LR}$$

$$(m, \alpha, \beta)_{LR} \oplus (n, \gamma, \delta)_{RL} = (m-n, \alpha+\delta, \beta+\gamma)_{LR}$$

$\exp(-x^2)$ to list a few, thus suggestive of a wide scope of $L(z)$. However, the choice of the L and R functions is specific to the problem in hand.

In the case of trapezoidal fuzzy numbers (Refer Fig. 12.2) the LR-type flat fuzzy numbers defined below are made use of

(12.3)

Fig. 12.2 Trapezoidal LR-type flat fuzzy number.

Briefly, the above equation is represented by the quadruple $(m_1, m_2, \alpha, \beta)$ LR . A triangular

LR-type fuzzy number can also be represented by the quadruple (m, m, α, β) .

12.1.1 Operations on LR-type Fuzzy Numbers

Let

and be two LR-type fuzzy numbers given by

$= (m, \alpha, \beta)$ and $=$

(n, γ, δ) .

The basic operations are

Addition

(12.4)

Subtraction

(12.5)

Multiplication

$$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} = (mn, m\gamma + n\alpha, m\delta + n\beta)_{LR} \text{ for } m \geq 0, n \geq 0$$

$$(m, \alpha, \beta)_{RL} \otimes (n, \gamma, \delta)_{LR} = (mn, n\alpha - m\delta, n\beta - m\gamma)_{RL} \text{ for } n \geq 0, m < 0$$

$$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} = (mn, -n\beta - m\delta, -n\alpha - m\gamma)_{RL} \text{ for } m < 0, n < 0$$

$$\lambda \bullet (m, \alpha, \beta)_{LR} = (\lambda m, \lambda \alpha, \lambda \beta)_{LR}, \quad \forall \lambda \geq 0, \lambda \in R$$

$$\lambda \bullet (m, \alpha, \beta)_{LR} = (\lambda m, -\lambda \alpha, -\lambda \beta)_{RL}, \quad \forall \lambda < 0, \lambda \in R$$

(12.6)

Scalar Multiplication

(12.7)

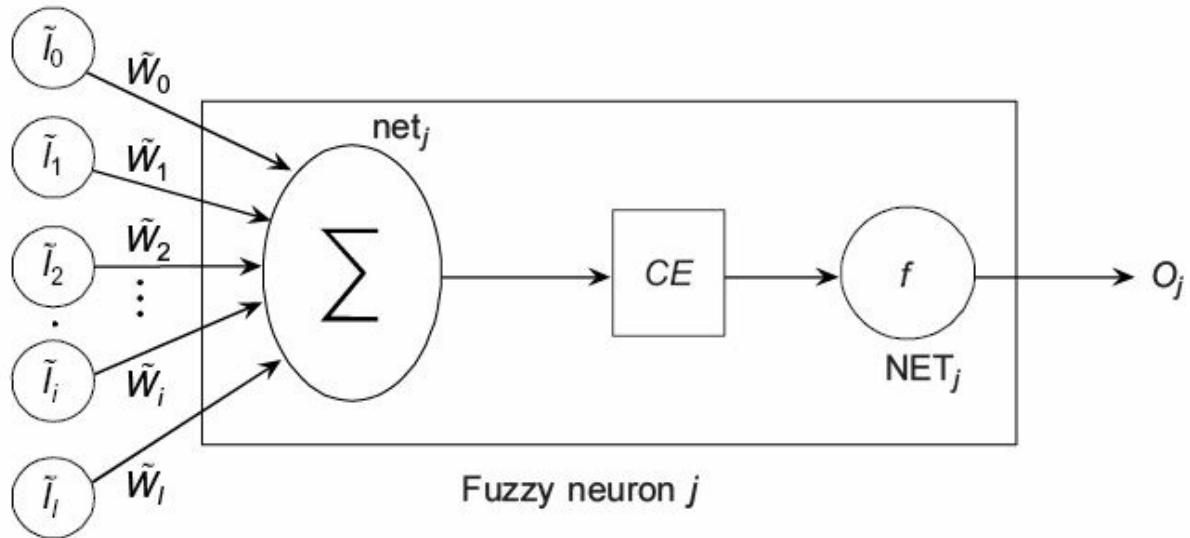
$$\tilde{I} = (\tilde{I}_0, \tilde{I}_1, \dots, \tilde{I}_l)$$

$$\tilde{W} = (\tilde{W}_0, \tilde{W}_1, \dots, \tilde{W}_l)$$

$$O = f(NET) = f\left(CE\left(\sum_{i=0}^l \tilde{W}_i \cdot \tilde{I}_i\right)\right)$$

$$\tilde{I}_0$$

$$net = \sum_{i=0}^l \tilde{W}_i \cdot \tilde{I}_i$$



$$NET = CE(\tilde{net})$$

$$\tilde{net} = (net_m, net_\alpha, net_\beta)$$

$$CE(\tilde{net}) = net_m + \frac{1}{3}(net_\beta - net_\alpha) = NET$$

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

12.2 FUZZY NEURON

The fuzzy neuron is the basic element of the fuzzy BP model. Figure 12.3 illustrates the architecture of the fuzzy neuron. Given the input vector and weight vector

, the fuzzy neuron computes

the crisp output O given by

(12.8)

where, $= (1, 0, 0)$ is the bias. Here, the fuzzy weighted summation **Fig. 12.3** Fuzzy neuron architecture of fuzzy BP model.

is first computed and

is computed next. The function CE is

the centroid of the triangular fuzzy number and can be treated as a defuzzification operation which maps fuzzy weighted summation value to a crisp value. Thus, if

is the fuzzy weighted summation

then the function CE is given by

(12.9)

The function f is the sigmoidal function which performs nonlinear mapping between the input and output. f is defined as

(12.10)

\tilde{I}

\tilde{W}

$$\tilde{I} = (\tilde{I}_0, \tilde{I}_1, \dots, \tilde{I}_l)$$

\tilde{I}_i

$$\tilde{W} = (\tilde{W}_0, \tilde{W}_1, \dots, \tilde{W}_l)$$

\tilde{W}_i

$$\tilde{W}_i = (W_{\alpha i}, W_{\beta i})$$

This is the final computation to obtain the crisp output value O .

In the fuzzy neuron, both input vector

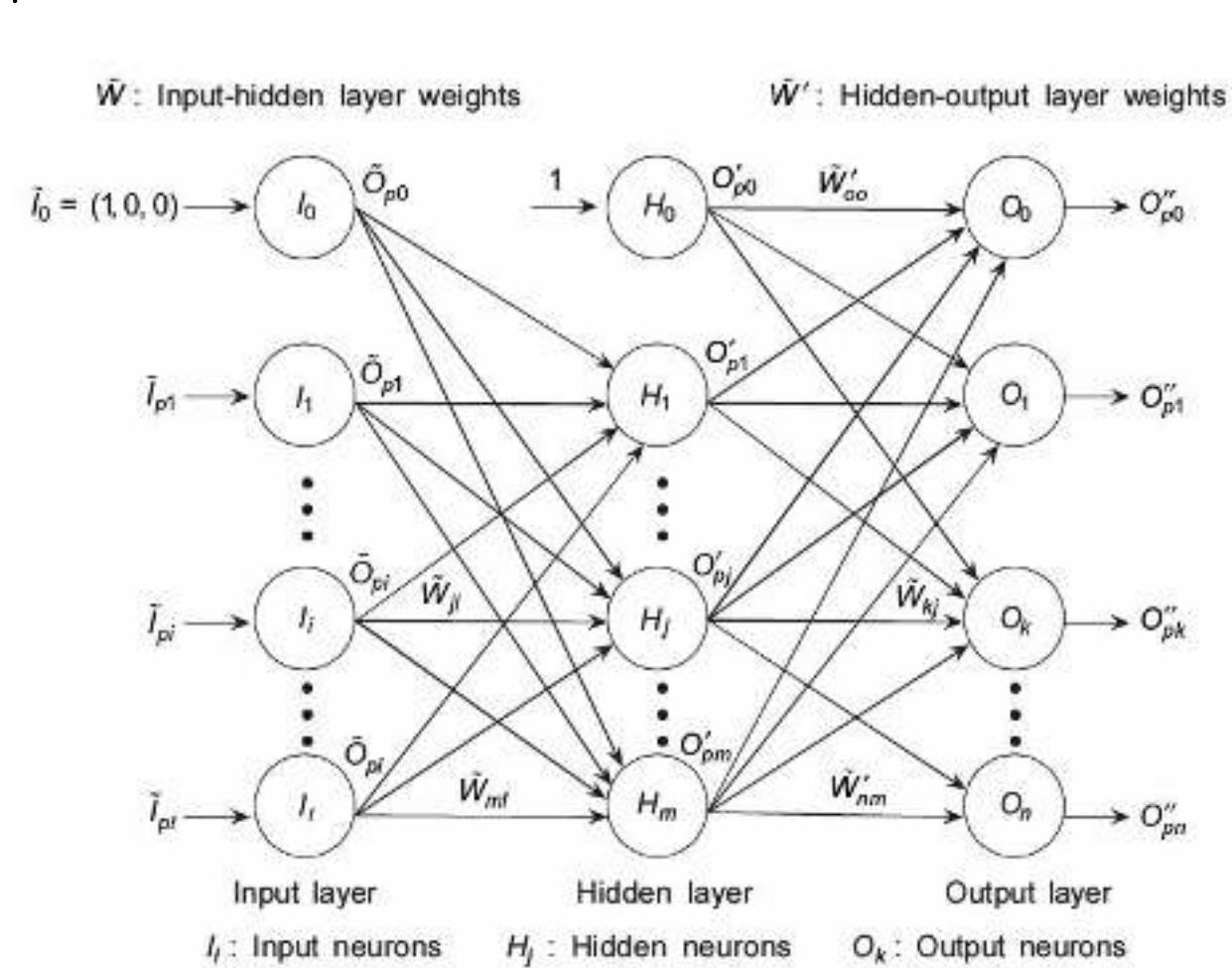
and weight vector

are

represented by triangular LR-type fuzzy numbers. Thus, for the input component vector is represented by the LR-type fuzzy number (I_{mi} , $I \alpha i$, $I \beta i$). Similarly, for

, the weight vector component

is represented as



$$\tilde{I}_p = (\tilde{I}_{p1}, \tilde{I}_{p2}, \dots, \tilde{I}_{pl}) \quad p = 1, 2, \dots, N$$

$$\tilde{I}_o$$

12.3 FUZZY BP ARCHITECTURE

Fuzzy BP is a three layered feedforward architecture. The three layers are \tilde{N} input layer, hidden layer, and output layer. As in BPN, the functioning of fuzzy BP proceeds in two stages, namely

1. Learning or Training, and
2. Inference.

Learning is detailed in Sec.12.4 and inference in Sec. 12.5.

Consider a configuration of $l\text{-}m\text{-}n$ (l input neurons, m hidden neurons, and n output neurons) for the fuzzy BP model. Figure 12.4 illustrates the architecture of fuzzy BP.

Fig. 12.4 Architecture of fuzzy BP.

Let

be the p th pattern among N input patterns

that fuzzy BP needs to be trained, with $= (1, 0, 0)$ as the bias.

$$\tilde{I}_{pi}$$

$$\tilde{I}_{pi} = (I_{pmi}, I_{p\alpha i}, I_{p\beta i})$$

$$\tilde{O}_{pi}$$

$$\tilde{W}_{ji}$$

$$\tilde{W}_{kj}$$

$$\tilde{O}_{pi} = \tilde{I}_{pi} = 1, 2, \dots, l; \quad \tilde{O}_{po} = (1, 0, 0)$$

$$O'_{pj} = f(NET_{pj}), \quad j = 1, 2, \dots, m; \quad O'_{po} = 1$$

$$NET_{pj} = CE \left(\sum_{i=0}^l \tilde{W}_{ji} \tilde{O}_{pi} \right)$$

$$O''_{pk} = f(NET'_{pk}), \quad k = 0, 1, 2, \dots, n-1$$

$$NET'_{pk} = CE \left(\sum_{j=0}^m \tilde{W}'_{kj} O'_{pj} \right)$$

Here,

indicates the i th input component of the input pattern p and is an LR-type triangular fuzzy number, i.e.

. Let

be the output

value of the i th input neuron, O'_{pj} and O''_{pk} are the j th and k th crisp defuzzification outputs of the hidden and output layer neurons respectively.

and

are the LR-type fuzzy connection weights between the i th input neuron and the j th hidden neuron, and the j th hidden neuron and k th output neuron respectively. In addition, CE and f are the sigmoidal and centroid functions as explained in

Eqs. (12.9) and (12.10).

The computations carried out by each layer are:

Input neurons

(12.11)

Hidden neurons

(12.12)

Output neurons

(12.13)

$$E_p = \sum_{i=1}^n \frac{1}{2} (D_{pi} - O''_{pi})^2$$

$$E = \sum_p E_p$$

$$\Delta \tilde{W}(t) = -\eta E_p(t) + \alpha \Delta \tilde{W}(t-1)$$

$$\alpha \Delta \tilde{W}(t)$$

$$\nabla E_p(t) = \frac{\partial E_p}{\partial \tilde{W}(t)} = \left(\frac{\partial E_p}{\partial W_m(t)}, \frac{\partial E_p}{\partial W_\alpha(t)}, \frac{\partial E_p}{\partial W_\beta(t)} \right)$$

$$\tilde{W}$$

$$\tilde{W}'_{ji} = (W'_{mji}, W'_{\alpha ji}, W'_{\beta ji})$$

$$\tilde{W}'_{kj} = (W'_{mkj}, W'_{\alpha kj}, W'_{\beta kj})$$

$$\begin{aligned}
\frac{\partial E_p}{\partial W'_{mkj}} &= \frac{\partial E_p}{\partial net'_{pmk}} \frac{\partial net'_{pmk}}{\partial W'_{mkj}} \\
&= \left(\frac{\partial E_p}{\partial O''_{pk}} \frac{\partial O''_{pk}}{\partial net'_{pmk}} \right) \frac{\partial net'_{pmk}}{\partial W'_{mkj}} \\
&= -\left(D_{pk} - O''_{pk} \right) \left(\frac{\partial O''_{pk}}{\partial NET'_{pk}} \frac{\partial NET'_{pk}}{\partial net'_{pmk}} \right) \frac{\partial net'_{pmk}}{\partial W'_{mkj}}
\end{aligned}$$

12.4 LEARNING IN FUZZY BP

The learning procedure of fuzzy BP follows the gradient descent method of minimizing error due to the learning. Here, the mean square error function for pattern p is defined as

(12.14)

where D_{pi} is the desired (target) output value of the i th output neuron and O''_{pi} is the computed value of the i th output neuron.

The overall error of the training pattern

During the learning phase, the weights are adjusted so as to minimize E .

The weight change at time t is given by

(12.15)

where η is the learning rate and α is a constant value.

is the

momentum term to be added for speeding up convergence.

The term $\nabla E_p(t)$ is given by

(12.16)

where

$$(t) = (Wm(t), W\alpha(t), W\beta(t)).$$

Also, recollect that

and

are the fuzzy

connection weights between the input-hidden and hidden-output layer neurons.

We now begin the derivations to obtain $\partial E_p / \partial W'm$, $\partial E_p / \partial W'\alpha$, and $\partial E_p / \partial W'\beta$. Consider the hidden-output layer. Applying chain rule,

(12.17)

$$= -(D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) 1.O'_{pj}$$

$$\frac{\partial E_p}{\partial W'_{mkj}} = -(D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) O'_{pj}$$

$$\frac{\partial E_p}{\partial W'_{akj}} = \frac{\partial E_p}{\partial net'_{pak}} \frac{\partial net'_{pak}}{\partial W'_{akj}}$$

$$= \left(\frac{\partial E_p}{\partial O''_{pk}} \frac{\partial O''_{pk}}{\partial net'_{pak}} \right) \frac{\partial net'_{pak}}{\partial W'_{akj}}$$

$$= -(D_{pk} - O''_{pk}) \left(\frac{\partial O''_{pk}}{\partial NET'_{pk}} \frac{\partial NET'_{pk}}{\partial net'_{pak}} \right) \frac{\partial net'_{pak}}{\partial W'_{akj}}$$

$$= - (D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{-1}{3} \right) O'_{pj}$$

$$\frac{\partial E_p}{\partial W'_{\alpha kj}} = - (D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{-1}{3} \right) O'_{pj}$$

$$\frac{\partial E_p}{\partial W'_{\beta kj}} = - (D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{1}{3} \right) O'_{pj}$$

$$\frac{\partial E_p}{\partial \tilde{W}(t)}$$

$$\delta_{pmk} = \frac{\partial E_p}{\partial net'_{pmk}} = - (D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk})$$

$$\delta_{pak} = \frac{\partial E_p}{\partial net'_{pak}} = - (D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{-1}{3} \right)$$

$$\delta_{p\beta k} = \frac{\partial E_p}{\partial net'_{p\beta k}} = - (D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{1}{3} \right)$$

$$\nabla E_p(T) = \frac{\partial E_p}{\partial \tilde{W}(t)}$$

Hence,

(12.18)

Again

(12.19)

Hence,

(12.20)

Similarly,

(12.21)

Thus, Eqs. (12.18), (12.20), and (12.21) give the

terms for the hidden-

output layer.

Now consider the input-hidden layer. Let us define the error values δp_{mk} , $\delta p \alpha k$, and $\delta p \beta k$ as,

(12.22)

(12.23)

(12.24)

To obtain,

$$\begin{aligned} \frac{\partial E_p}{\partial W_{mji}} &= \frac{\partial E_p}{\partial O'_{pj}} \frac{\partial O'_{pj}}{\partial W_{mji}} \\ &= \frac{\partial E_p}{\partial O'_{pj}} \left(\frac{\partial O'_{pj}}{\partial NET_{pj}} \frac{\partial NET_{pj}}{\partial net_{pmj}} \frac{\partial net_{pmj}}{\partial W_{mji}} \right) \\ &= \frac{\partial E_p}{\partial O'_{pj}} O'_{pj} (1 - O'_{pj}) \cdot 1 \cdot \tilde{O}_{pi} \end{aligned}$$

$$= \left(\sum_{k=0}^n \frac{\partial E_p}{\partial net'_{pmk}} \frac{\partial net'_{pmk}}{\partial O'_{pj}} \right) O'_{pj} (1 - O'_{pj}) \cdot 1 \cdot \tilde{O}_{pi}$$

$$\frac{\partial E_p}{\partial W_{mji}} = \left(\sum_{k=0}^n \delta_{pmk} W'_{mkj} \right) O'_{pj} (1 - O'_{pj}) \cdot 1 \cdot \tilde{O}_{pi}$$

$$\frac{\partial E_p}{\partial W_{\alpha ji}} = \frac{\partial E_p}{\partial O'_{pj}} \frac{\partial O'_{pj}}{\partial W_{\alpha ji}}$$

$$= \frac{\partial E_p}{\partial O'_{pj}} \left(\frac{\partial O'_{pj}}{\partial NET_{pj}} \frac{\partial NET_{pj}}{\partial net_{p\alpha j}} \frac{\partial net_{p\alpha j}}{\partial W_{\alpha ji}} \right)$$

$$= \frac{\partial E_p}{\partial O'_{pj}} \left(O'_{pj} (1 - O'_{pj}) \left(\frac{-1}{3} \right) \tilde{O}_{pi} \right)$$

$$= \left(\sum_{k=0}^n \frac{\partial E_p}{\partial net'_{p\alpha k}} \frac{\partial net'_{p\alpha k}}{\partial O'_{pj}} \right) O'_{pj} (1 - O'_{pj}) \left(\frac{-1}{3} \right) \tilde{O}_{pi}$$

$$\frac{\partial E_p}{\partial W_{\alpha ji}} = \left(\sum_{k=0}^n \delta_{\alpha k} W'_{\alpha kj} \right) O'_{pj} (1 - O'_{pj}) \left(\frac{-1}{3} \right) \tilde{O}_{pi}$$

$$\frac{\partial E_p}{\partial W_{\beta ji}} = \left(\sum_{k=0}^n \delta_{\beta k} W'_{\beta kj} \right) O'_{pj} (1 - O'_{pj}) \left(\frac{1}{3} \right) \tilde{O}_{pi}$$

$$\Delta \tilde{W}(t)$$

$$\tilde{W}'(t)$$

$$\tilde{W}'$$

$$\tilde{W}'$$

(12.25)

(12.26a)

Therefore,

(12.26b)

Again,

(12.27)

(12.28a)

Therefore,

(12.28b)

Similarly,

(12.29)

Thus, Eqs. (12.26), (12.28), and (12.29) give the $\nabla Ep(t)$ term for the input-hidden layer weights.

Now, the change in weights

for the input-hidden and hidden-output

layer weights can be obtained using Eq. (12.15).

The updated weights at time t are given by

=

(t

-

1)

+

$\Delta(t),$

for

the

hidden-output

$\tilde{W}(t)$

\tilde{W}

\tilde{W}

Algorithm 12.1

```
Procedure Fuzzy_BP_TRAINING ()  
/* Let the configuration of fuzzy BP be l-m-n */  
  
Step 1: Randomly generate the initial weight sets  $\tilde{W}$  for the  
input-hidden layer where each  $\tilde{W}_{ji} = (W_{mji}, W_{ujj}, W_{gji})$  is an  
LR-type fuzzy number. Also generate the weight set  $\tilde{W}'$   
for the hidden-output layer where  $\tilde{W}'_{kj} = (W'_{mkj}, W'_{ukj}, W'_{gkj})$ .  
  
Step 2: Let  $(\tilde{I}_p, D_p), p = 1, 2, \dots, N$  be the  $N$  input-output pattern  
set, that fuzzy BP needs to be trained with. Here,  $\tilde{I}_p$   
=  $(\tilde{I}_{p0}, \tilde{I}_{p1}, \dots, \tilde{I}_{pl})$  where each  $\tilde{I}_{pi}$  is an LR-type fuzzy  
number, i.e.  $\tilde{I}_{pi} = (\tilde{I}_{p\alpha i}, \tilde{I}_{p\beta i}, \tilde{I}_{p\gamma i})$ .  $D_p$  is a crisp output.  
  
Step 3: Let ITRNS denote the number of iterations. Set the  
counters for the number of iterations and number of  
pattern sets to be trained to zero.  
i.e. COUNT_OF_ITRNS = 0; p = 1;
```

layer (12.30)

= $(t - 1) + \Delta(t)$, for the input-hidden layer (12.31) Algorithm
12.1 illustrates the training of fuzzy BP.

Step 4: Assign values for η and α .

Initialise: $\Delta W(t-1) = 0;$

$\Delta W'(t-1) = 0;$

Also, $\tilde{W}(t-1) = 0$ and $\tilde{W}'(t-1) = 0$

Step 5: Get next pattern set $(\tilde{x}_p, \tilde{d}_p)$. Assign $\tilde{o}_{pi} = \tilde{x}_{pi}$,
 $i = 1, 2, \dots, l$; $\tilde{o}_o = (1, 0, 0)$ for the input neurons.

Step 6: Compute

$$O'_{pj} = f(NET_{pj}), \quad j = 1, 2, \dots, m; \quad O'_{po} = 1$$

$$\text{where } NET_{pj} = CE \left(\sum_{i=0}^l \tilde{w}_{ji} \tilde{o}_{pi} \right)$$

for the hidden neurons.

Step 7: Compute

$$O''_{pk} = f(NET'_{pk}), \quad k = 0, 1, \dots, n - 1$$

$$\text{where } NET'_{pk} = CE \left(\sum_{j=0}^m \tilde{w}_{kj} O'_{pj} \right)$$

for the output neurons.

Step 8: Compute change of weights $\Delta W(t)$ for the hidden-output layer as follows:

8.1 Compute

$$\nabla E_p(t) = \left(\frac{\partial E_p}{\partial W'_{mkj}}, \frac{\partial E_p}{\partial W'_{\alpha kj}}, \frac{\partial E_p}{\partial W'_{\beta kj}} \right)$$

where

$$\frac{\partial E_p}{\partial W'_{mkj}} = -(D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \cdot 1 \cdot O'_{pj}$$

$$\frac{\partial E_p}{\partial W'_{\alpha kj}} = -(D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{-1}{3} \right) O'_{pj}$$

$$\frac{\partial E_p}{\partial W'_{\beta kj}} = -(D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{1}{3} \right) O'_{pj}$$

8.2 Compute

$$\tilde{\Delta}W'(t) = -\eta \nabla E_p(t) + \alpha \tilde{\Delta}W'(t-1).$$

Step 9: Compute change of weights $\Delta \hat{W}(t)$ for the input-hidden layer as follows:

9.1

$$\text{let } \delta_{puk} = -(D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \cdot 1$$

$$\delta_{\nu uk} = -(D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{-1}{3} \right)$$

$$\delta_{\beta uk} = -(D_{pk} - O''_{pk}) O''_{pk} (1 - O''_{pk}) \left(\frac{1}{3} \right)$$

Compute

$$\nabla E_p(t) = \left(\frac{\partial E_p}{\partial W_{\alpha ji}}, \frac{\partial E_p}{\partial W_{\beta ji}}, \frac{\partial E_p}{\partial W_{\gamma ji}} \right) \text{ where}$$

$$\frac{\partial E_p}{\partial W_{\alpha ji}} = \left(\sum_k \delta_{puk} W'_{\alpha kj} \right) O'_{pj} (1 - O'_{pj}) \cdot 1 \cdot \tilde{o}_{pi}$$

$$\frac{\partial E_p}{\partial W_{\beta ji}} = \left(\sum_k \delta_{\nu uk} W'_{\beta kj} \right) O'_{pj} (1 - O'_{pj}) \left(\frac{-1}{3} \right) \tilde{o}_{pi}$$

$$\frac{\partial E_p}{\partial W_{\gamma ji}} = \left(\sum_k \delta_{\beta uk} W'_{\gamma kj} \right) O'_{pj} (1 - O'_{pj}) \left(\frac{1}{3} \right) \tilde{o}_{pi}$$

9.2 Compute

$$\Delta \tilde{W}'(t) = -\eta \nabla E_p(t) + \alpha \tilde{\Delta} W'(t)$$

Step 10: Update weights for the input-hidden and hidden-output layers as

$$\tilde{W}(t) = \tilde{W}(t-1) + \Delta \tilde{W}(t)$$

$$\tilde{W}'(t) = \tilde{W}'(t-1) + \Delta \tilde{W}'(t)$$

Step 11: $p = p + 1$;

If ($p \leq N$) goto Step 5;

Step 12: COUNT_OF_ITRNS = COUNT_OF_ITRNS + 1;

If COUNT_OF_ITRNS < ITRNS

{ Reset pointer to first pattern in the
training set;
 $p=1$;
goto Step 5;
}

Step 13: Output \tilde{W} and \tilde{W}' the final weight sets.

}

end Fuzzy_BP_TRAINING.

\tilde{F}_p

$\tilde{F}_p = (\tilde{F}_{p1}, \tilde{F}_{p2}, \dots, \tilde{F}_{pl})$

\tilde{F}_{pi}

$\tilde{F}_{pi} = (\tilde{F}_{p\alpha i}, \tilde{F}_{p\beta i}, \dots, \tilde{F}_{p\gamma i})$

\tilde{F}_p

\tilde{F}_p

$$O''_k$$

$$\tilde{F}_p$$

$$\tilde{F}_p$$

$$\bar{W}$$

$$\bar{W}'$$

$$\tilde{F}_p$$

$$\tilde{O}_{pi}$$

$$\tilde{F}_{pi}, \quad i = 1, 2, \dots, l$$

$$\tilde{O}_o$$

$$O'_{pj} = f(NET_{pj}) j = 1, 2, \dots, m;$$

$$O'_{po} = 1$$

$$NET_{pj} = CE \left(\sum_{i=0}^l \tilde{W}_{ji} \tilde{O}_{pi} \right)$$

12.5 INFERENCE BY FUZZY BP

Once the fuzzy BP model has been trained for a given set of input-output patterns a definite number of times, it is ready for inference.

Given a set of patterns

to be inferred, where

and

is

an

LR-type fuzzy number given by

. The aim is to obtain

O_p , the output corresponding to .

O_p is computed in one pass by allowing to pass through the series of computations illustrated in Eqs. (12.11)–(12.13). The

, computed by the

output neurons, is the output corresponding to

.

Algorithm 12.2 illustrates the inference of fuzzy BP.

Algorithm 12.2

Procedure Fuzzy_BP_INFERENCE()

{

/* Let

, $p = 1, 2, \dots, N'$ be the patterns whose output values are to be inferred.

Let and be the weight sets obtained after training fuzzy BP */

Step 1 : $p = 1$;

Step 2 : Get next pattern ;

Step 3 : Compute

=

$= (1,0,0);$

for the input neurons.

Step 4 : Compute

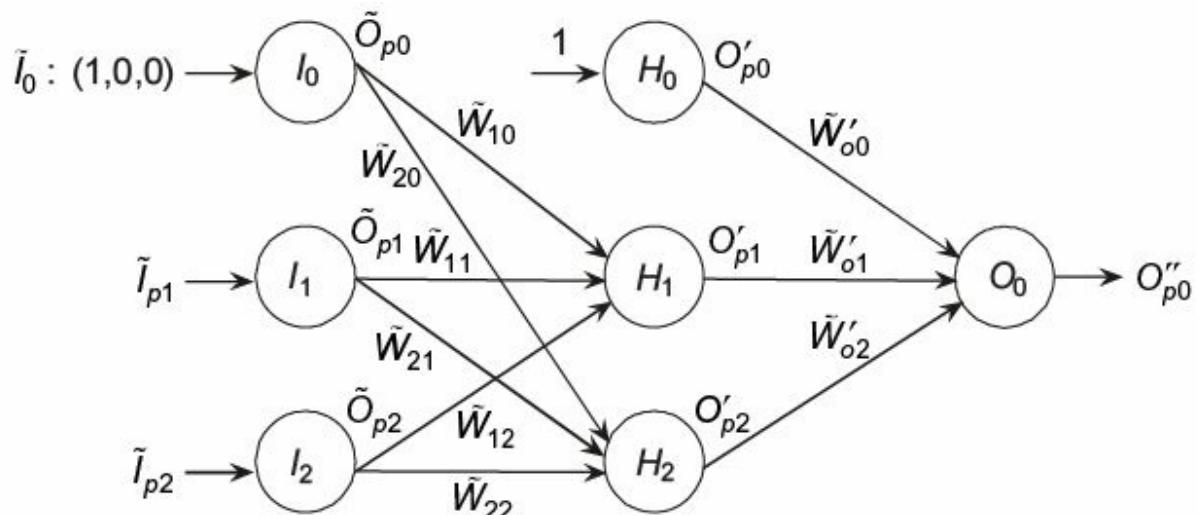
Where...

for the hidden neurons.

$$O''_{pk} = f(NET'_{pk}), \quad k = 0, 1, 2, \dots, n - 1$$

$$NET'_{pk} = CE \left(\sum_{j=0}^m \tilde{W}_{kj} O'_{pj} \right)$$

$$O''_{pk},$$



Step 5 : Compute

where

for the output neurons.

Step 6 : Output the associated output

$$k = 0, 1, 2, \dots, n - 1$$

Step 7 : $p = p + 1;$

If ($P \leq N'$) goto Step 2;

}

end FUZZY_BP_INFERENCE.

Illustration

In this section, we demonstrate the learning method of fuzzy BP on an illustrative toy problem.

Consider a fuzzy BP model with a 2-2-1 configuration (Refer Fig. 12.5).

Fig. 12.5 A 2-2-1 fuzzy BP configuration.

Table 12.1 gives the input patterns that fuzzy BP needs to be trained with.

The input patterns have been normalized to lie between 0 and 1. Table 12.2(a) and (b) show the initial set of weight vectors that have been randomly generated.

Table 12.1 Input patterns for training

Pattern p	Input			Output
	I_0	\tilde{I}_{p1}	\tilde{I}_{p2}	
1	(1, 0, 0)	(1, 0.2, 0.3)	(0, 0.1, 0.4)	0.8
2	(1, 0, 0)	(0.5, 0.2, 0.1)	(0.4, 0.5, 0.4)	0.1

Table 12.2 Initial weight sets

(a) Input-hidden layer		(b) Hidden-output layer	
\hat{W}_{00}	\hat{W}_{01}	\hat{W}_{02}	\hat{W}'_{00}
—	—	—	(0.154, 0.033, 0.498)
\hat{W}_{10} (0.62, 0.505, 0.405)	\hat{W}_{11} (0.894, 0.634, 0.101)	\hat{W}_{12} (0.66, 0.567, 0.64)	\hat{W}'_{01} (0.092, 0.277, 0.223)
\hat{W}_{20} (-0.235, 0.329, 0.498)	\hat{W}_{21} (-0.723, 0.71, 0.855)	\hat{W}_{22} (0.134, 0.719, 0.153)	\hat{W}'_{02} (0.972, 0.137, 0.72)

\tilde{I}_1

\tilde{I}_1

\tilde{O}_1

\tilde{I}_1

The computations performed by fuzzy BP in the first iteration are: **Iteration 1**
1 Input pattern 1 (, $D 1$)

$$= ((1, 0, 0), (1, 0.2, 0.3), (0, 0.1, 0.4))$$

$$D 1 = 0.8$$

The output of the input neurons

$$= = ((1, 0, 0), (1, 0.2, 0.3), (0, 0.1, 0.4))$$

The input to the hidden neurons are

$$net\ 10 = \text{nil}$$

$$\begin{aligned} net\ 11 &= (1, 0, 0) (0.62, 0.505, 0.405) + (1, 0.2, 0.3) (0.894, 0.634, 0.101) + \\ &\quad (0, 0.1, 0.4) (0.66, 0.567, 0.64) \\ &= (1.5140, 1.3838, 1.0382) \end{aligned}$$

$$\begin{aligned} net\ 12 &= (1, 0, 0) (-0.235, 0.329, 0.498) + (1, 0.2, 0.3) (-0.723, 0.71, 0.855) \\ &\quad + \\ &\quad (0, 0.1, 0.4) (0.134, 0.719, 0.153) \end{aligned}$$

$$O'_{10} = 1$$

$$O'_{11} = f(1.3988) = \frac{1}{1 + e^{-1.3988}} = 0.8020$$

$$O'_{12} = f(-1.0254) = \frac{1}{1 + e^{-1.0254}} = 0.2640$$

$$\begin{aligned} net'_{10} &= \sum_{j=0}^2 \tilde{W}'_{kj} O'_{1j} \\ &= 1 \times (0.154, 0.033, 0.498) + 0.8020 \times (0.092, 0.277, 0.223) \\ &\quad + 0.264 \times (0.972, 0.137, 0.72) \\ &= (0.4844, 0.2913, 0.8669) \end{aligned}$$

$$NET'_{10} = CE(net'_{10}) = 0.6762$$

$$O''_{10} = f(NET'_{10}) = \frac{1}{1 + e^{-0.6762}} = 0.6629$$

$$\Delta \tilde{W}(t)$$

$$\Delta \tilde{W}$$

$$\nabla E_1(t) = \left(\frac{\partial E_1}{\partial W'_{m00}}, \frac{\partial E_1}{\partial W'_{\alpha00}}, \frac{\partial E_1}{\partial W'_{\beta00}} \right) \text{ gives}$$

$$\begin{aligned} \frac{\partial E_1}{\partial W'_{m00}} &= -(0.8 - 0.6629)(0.6629)(1 - 0.6629)1.1 \\ &= -0.0306 \end{aligned}$$

$$= (-0.9580, 1.5108, 1.3087)$$

From the calculations above,

NET 10: Nil

NET 11: CE (1.514, 1.3838, 1.0382)

$$= 1.514 + 1/3(1.0382 + 1.3838)$$

$$= 1.3988$$

NET 12: CE(-0.9580, 1.5108, 1.3087)

$$= -1.0254$$

The outputs of the hidden neurons are

The output of the output neuron is

We now proceed to compute the change of weights

given by Eq.

(12.15) for the input-hidden and hidden-output layers.

Initially set

($t - 1$) = 0. Choose $\eta = 0.9$ and $\alpha = 0.1$.

Now,

$$\frac{\partial E_1}{\partial W'_{\alpha 00}} = 0.0102$$

$$\frac{\partial E_1}{\partial W'_{\beta 00}} = -0.0102$$

$$\left(\frac{\partial E_1}{\partial W'_{m01}}, \frac{\partial E_1}{\partial W'_{\alpha 01}}, \frac{\partial E_1}{\partial W'_{\beta 01}} \right) = (-0.0246, 0.0082, -0.0082)$$

$$\left(\frac{\partial E_1}{\partial W'_{m02}}, \frac{\partial E_1}{\partial W'_{\alpha 02}}, \frac{\partial E_1}{\partial W'_{\beta 02}} \right) = (-0.0081, 0.0027, -0.0027)$$

$$\begin{aligned} \Delta \tilde{W}'_{00}(t) &= -0.9(-0.0306, 0.0102, -0.0102) + (0.1 \times 0) \\ &= (0.0276, -0.0092, 0.0092) \end{aligned}$$

$$\Delta \tilde{W}'_{01}(t) = (0.0221, -0.0074, 0.0074)$$

$$\Delta \tilde{W}'_{02}(t) = (0.0073, -0.0024, 0.0024)$$

$$\begin{aligned}
W'_{00}(t) &= W'_{00}(t-1) + \Delta \tilde{W}'(t) \\
&= (0.154, 0.033, 0.498) + (0.0276, -0.0092, 0.0092) \\
&= (0.1816, 0.0238, 0.5072)
\end{aligned}$$

$$W'_{01}(t) = (0.1141, 0.2696, 0.2304)$$

$$W'_{02}(t) = (0.9793, 0.1346, 0.7224)$$

$$\begin{aligned}
\delta_{1m0} &= -(0.8 - 0.6629)(0.6629)(1 - 0.6629) \\
&= -0.0306
\end{aligned}$$

$$\delta_{1\alpha 0} = 0.102$$

$$\delta_{1\beta 0} = -0.0102$$

$$\begin{aligned}
\frac{\partial E_1}{\partial W_{m10}} &= (-0.0306 \times 0.092)(0.8020)(1 - 0.8020) \cdot 1 \\
&= -0.000448
\end{aligned}$$

$$\frac{\partial E_1}{\partial W_{\alpha 10}} = 0$$

Similarly,

Also,

The updated weights for the hidden-output layer are

Now the change in weights for the input-hidden layer are as follows:
Therefore,

$$\frac{\partial E_1}{\partial W_{\beta 10}} = 0$$

$$\left(\frac{\partial E_1}{\partial W_{m11}}, \frac{\partial E_1}{\partial W_{\alpha 11}}, \frac{\partial E_1}{\partial W_{\beta 11}} \right) = (-0.000448, -0.00003, -0.000036)$$

$$\left(\frac{\partial E_1}{\partial W_{m12}}, \frac{\partial E_1}{\partial W_{\alpha 12}}, \frac{\partial E_1}{\partial W_{\beta 12}} \right) = (0, -0.000015, -0.000048)$$

$$\left(\frac{\partial E_1}{\partial W_{m20}}, \frac{\partial E_1}{\partial W_{\alpha 20}}, \frac{\partial E_1}{\partial W_{\beta 20}} \right) = (-0.005786, 0, 0)$$

$$\left(\frac{\partial E_1}{\partial W_{m21}}, \frac{\partial E_1}{\partial W_{\alpha 21}}, \frac{\partial E_1}{\partial W_{\beta 21}} \right) = (-0.005786, -0.000018, -0.000143)$$

$$\left(\frac{\partial E_1}{\partial W_{m22}}, \frac{\partial E_1}{\partial W_{\alpha 22}}, \frac{\partial E_1}{\partial W_{\beta 22}} \right) = (0, -0.000009, -0.000190)$$

$$\begin{aligned} \Delta \tilde{W}_{10}(t) &= -0.9 \times (-0.000448, 0, 0) + 0.1 \times 0 \\ &= (0.000403, 0, 0) \end{aligned}$$

$$\Delta \tilde{W}_{11}(t) = (0.000403, 0.000027, 0.000033)$$

$$\Delta \tilde{W}_{12}(t) = (0, 0.000013, 0.000043)$$

$$\Delta \tilde{W}_{20}(t) = (0.005207, 0, 0)$$

$$\Delta \tilde{W}_{21}(t) = (0.005207, 0.000016, 0.000129)$$

$$\Delta \tilde{W}_{22}(t) = (0, 0.000008, 0.000171)$$

$$\begin{aligned}
\tilde{W}_{10}(t) &= \tilde{W}_{10}(t-1) + \Delta \tilde{W}(t) \\
&= (0.62, 0.505, 0.405) + (0.000403, 0, 0) \\
&= (0.6204, 0.505, 0.405)
\end{aligned}$$

$$\tilde{W}_{11}(t) = (0.8944, 0.634, 0.101)$$

Similarly,

The change in weights are given by

The updated weights for the input-hidden layer are

$$\tilde{W}_{12}(t) = (0.6600, 0.567, 0.64)$$

$$\tilde{W}_{20}(t) = (-0.2298, 0.329, 0.498)$$

$$\tilde{W}_{21}(t) = (-0.7178, 0.71, 0.8551)$$

$$\tilde{W}_{22}(t) = (0.1320, 0.719, 0.1532)$$

$$\tilde{I}_2$$

$$\tilde{I}_2 = ((1, 0, 0) (0.5, 0.2, 0.1) (0.4, 0.5, 0.4))$$

$$D_2 = 0.1$$

$$O'_{20} = 1; \quad O'_{21} = 0.7627; \quad O'_{22} = 0.3274$$

$$\left(\frac{\partial E_2}{\partial W'_{m00}}, \frac{\partial E_2}{\partial W'_{\alpha 00}}, \frac{\partial E_2}{\partial W'_{\beta 00}} \right) = (0.12619, -0.042064, 0.042064)$$

$$\left(\frac{\partial E_2}{\partial W'_{m01}}, \frac{\partial E_2}{\partial W'_{\alpha 01}}, \frac{\partial E_2}{\partial W'_{\beta 01}} \right) = (0.09624, -0.03208, 0.03208)$$

$$\left(\frac{\partial E_2}{\partial W'_{m02}}, \frac{\partial E_2}{\partial W'_{\alpha 02}}, \frac{\partial E_2}{\partial W'_{\beta 02}} \right) = (0.0413, -0.01377, 0.01377)$$

$$\Delta W'_{00}(t) = (-0.1108, 0.0369, -0.0369)$$

$$\Delta W'_{01}(t) = (-0.0844, 0.02814, -0.02814)$$

$$\Delta W'_{02}(t) = (-0.0365, 0.0122, -0.0122)$$

Thus, the first set of updated weights have been obtained at the end of training the network for the first pattern set. Now proceeding on similar lines for the second input pattern set, the computations are as follows: **Iteration 1**
Input Pattern 2 (, D 2) The output of hidden layer neurons are

The output of the output layer neurons

$$O''20 = 0.6909$$

Also,

The change in weights $\Delta W'(t)$ for the hidden-output layer neurons is given as The updated weights for the hidden-output layer are

$$W'_{00}(t) = (0.0708, \, 0.0607, \, 0.4703)$$

$$W'_{01}(t) = (0.0297, \, 0.2978, \, 0.2022)$$

$$W'_{02}(t) = (0.9428, \, 0.1467, \, 0.7103)$$

$$\left(\frac{\partial E_2}{\partial W_{m10}}, \frac{\partial E_2}{\partial W_{\alpha10}}, \frac{\partial E_2}{\partial W_{\beta10}}\right) = (0.002606, \, 0, \, 0)$$

$$\left(\frac{\partial E_2}{\partial W_{m11}}, \frac{\partial E_2}{\partial W_{\alpha11}}, \frac{\partial E_2}{\partial W_{\beta11}}\right) = (0.001303, \, 0.000137, \, 0.000058)$$

$$\left(\frac{\partial E_2}{\partial W_{m12}}, \frac{\partial E_2}{\partial W_{\alpha12}}, \frac{\partial E_2}{\partial W_{\beta12}}\right) = (0.001043, \, 0.000342, \, 0.000234)$$

$$\left(\frac{\partial E_2}{\partial W_{m20}}, \frac{\partial E_2}{\partial W_{\alpha20}}, \frac{\partial E_2}{\partial W_{\beta20}}\right) = (0.027211, \, 0, \, 0)$$

$$\left(\frac{\partial E_2}{\partial W_{m21}}, \frac{\partial E_2}{\partial W_{\alpha21}}, \frac{\partial E_2}{\partial W_{\beta21}}\right) = (0.013605, \, 0.000083, \, 0.000223)$$

$$\left(\frac{\partial E_2}{\partial W_{m22}}, \frac{\partial E_2}{\partial W_{\alpha22}}, \frac{\partial E_2}{\partial W_{\beta22}}\right) = (0.010884, \, 0.000208, \, 0.000892)$$

$$\Delta \tilde{W}(t)$$

$$\Delta \tilde{W}_{10}(t) = (-0.00231, 0, 0)$$

$$\Delta \tilde{W}_{11}(t) = (-0.00113, -0.00012, -0.000049)$$

$$\Delta \tilde{W}_{12}(t) = (-0.000938, -0.000307, -0.000206)$$

$$\Delta \tilde{W}_{20}(t) = (-0.024, 0, 0)$$

$$\Delta \tilde{W}_{21}(t) = (-0.011724, -0.000073, -0.000188)$$

$$\Delta \tilde{W}_{22}(t) = (-0.009796, -0.000186, -0.000786)$$

For the input-hidden layer neurons,

Change in weights

for the input-hidden layer is given by

The updated weights of the Input-Hidden layer are:

$$\tilde{W}_{10}(t) = (0.6181, 0.5050, 0.4050)$$

$$\tilde{W}_{11}(t) = (0.8933, 0.6339, 0.1010)$$

$$\tilde{W}_{12}(t) = (0.6591, 0.5667, 0.6398)$$

$$\tilde{W}_{20}(t) = (-0.2538, 0.329, 0.498)$$

$$\tilde{W}_{21}(t) = (-0.7295, 0.7099, 0.8549)$$

$$\tilde{W}_{22}(t) = (0.1222, 0.7188, 0.1524)$$

Table 12.3 Weight sets update for the toy problem

Iteration	Input-hidden layer	Hidden-output Layer
Initialisation	\hat{W}	\hat{W}'
Iteration 1	$\begin{bmatrix} (0.62, 0.505, 0.405), (0.894, 0.634, 0.101), (0.66, 0.567, 0.64), \\ (-0.235, 0.329, 0.498), (-0.723, 0.71, 0.855), (0.134, 0.719, 0.153) \end{bmatrix}$	$\begin{bmatrix} (0.154, 0.033, 0.498) \\ (0.092, 0.277, 0.223) \\ (0.972, 0.137, 0.72) \end{bmatrix}$
	$\begin{bmatrix} (0.6204, 0.505, 0.405), (0.8944, 0.634, 0.101), (0.66, 0.567, 0.64) \\ (-0.2298, 0.329, 0.498), (-0.7178, 0.71, 0.8551), (0.1320, 0.719, 0.1532) \end{bmatrix}$	$\begin{bmatrix} (0.1816, 0.0238, 0.5072) \\ (0.1141, 0.2696, 0.2304) \\ (0.9793, 0.1346, 0.7224) \end{bmatrix}$
Input pattern 1	$\begin{bmatrix} (0.6181, 0.505, 0.405), (0.8933, 0.6339, 0.1010), (0.6591, 0.5667, 0.6398), \\ (-0.2538, 0.329, 0.498), (-0.7295, 0.7099, 0.8549), (0.1222, 0.7188, 0.1524) \end{bmatrix}$	$\begin{bmatrix} (0.0708, 0.0607, 0.4703) \\ (0.0297, 0.2978, 0.2022) \\ (0.9428, 0.1467, 0.7103) \end{bmatrix}$

Summing up, at the end of the first iteration after training the fuzzy BP model with the two input patterns, the updated weight sets are as shown in Table 12.3. The iterations are repeated for a definite number of times before the training is deemed to have come to an end.

12.6 APPLICATIONS

In this section, we discuss two applications of the fuzzy BP model, namely Knowledge base evaluation and

Earthquake damage evaluation.

12.6.1 Knowledge Base Evaluation

In their paper, Lee and Lu have illustrated the working of fuzzy BP on a *Knowledge Base Evaluation* (KBE) system. KBE is an expert system to evaluate how suitable an expert system is with regard to its application in a specific domain. KBE is governed by 18 instances as shown in Table 12.4. It comprises six input attributes/features, namely worth, employee acceptance, solution available, easier solution, teachability, and risk. Figure 12.6

illustrates the fuzzy linguistic terms associated with the attributes. The output is the suitability of the expert system and is given by one of the two crisp output values, namely Good (1) or Poor (0). For the ‘don’t care’ terms marked by ‘*’ in Table 12.4, each possible fuzzy linguistic term associated with that feature is generated. Thus, the 18 instances give rise to 340 instances.

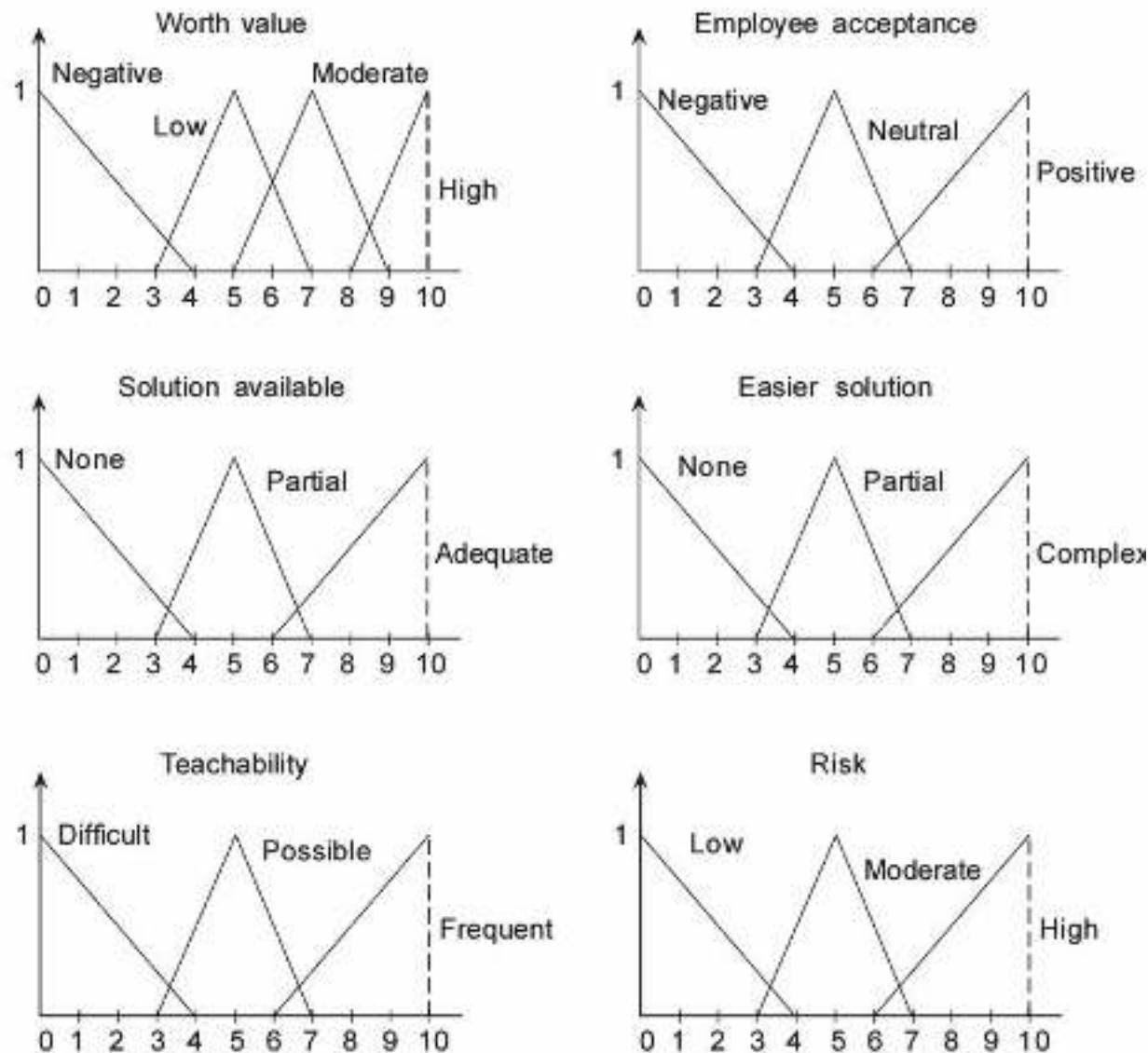


Fig. 12.6 Fuzzy linguistic terms associated with the input attributes of KBE.

Table 12.4 The instances of knowledge base evaluation

Feature

Employee

Solution

Easier

Output

Worth

Teachability

Risk

acceptance

available

solution

(suitability)

Instances

1

High

Positive

None

None

Frequent

Low

Good

2

Negative

*

*

*

*

*

Poor

3

Low

*

*

*

*

High

Poor

4

Moderate

Neutral

Adequate

Complete

Difficult

High

Poor

5

Low

Negative

None

Partial

Frequent

Low

Poor

Feature Instances	Worth	Employee acceptance	Solution availability	Easier solution	Teachability	Risk	Output
1	High	Positive	None	None	Frequent	Low	Good

6

High

Negative

Partial

None

Difficult

Moderate

Good

7

High

Positive

Partial

Complete

Frequent

High

Poor

8

High

Positive

Partial

Partial

Possible

Low

Poor

9

Low

Positive

Adequate

None

Frequent

Low

Good

10

High

Negative

Partial

None

Frequent

High

Good

11

Low

Positive

None

Complete

Difficult

Moderate

Poor

12

Low

Neutral

Adequate

Complete

Frequent

Low

Good

13

Low

Neutral

None

None

Difficult

Low

Good

14

Moderate

Positive

Adequate

None

Difficult

High

Poor

15

High

Negative

Adequate

Partial

Frequent

High

Poor

16

High

Negative

Partial

Complete

Possible

Low

Good

17

Moderate

Negative

None

Partial

Difficult

High

Good

18

Moderate

Neutral

Adequate

Partial

Difficult

Low

Poor

Of these, Lee and Lu have used 290 randomly selected instances as training instances and the remaining as testing instances. A three layered fuzzy BP

with a configuration 6-6-1 has been employed and the values of η and α are chosen as $\alpha = 0.1$ and $\eta = 0.9$. The interpretation of the output values computed by the output neuron is as follows:

If computed output value ≥ 0.5 suitability is good.

If computed output value < 0.5 suitability is poor.

We now detail a method of presenting the input attributes listed in Table 12.4 for computation by the fuzzy BP model. Consider an instance (instance no. 1) of Table 12.4, which reads

We need to convert each of the fuzzy linguistic terms associated with the attribute into their equivalent LR-type fuzzy numbers. For the attribute

“Worth”, the LR-type fuzzy number equivalent for its associated attribute values as gathered from Fig. 12.6 is

Fuzzy linguistic term

LR-type fuzzy number equivalents

Negative

(0, 0.0001, 4)

Low

(5, 2, 2)

Moderate

(7, 2, 2)

High

(10, 2, 0.0001)

Note here that since α, β the left and right spreads need to be such that $\alpha, \beta > 0$, we have chosen a small quantity 0.0001 to indicate the zero spreads.

Also, the LR-type fuzzy number equivalents could be normalized to lie between 0 and 1, thereby giving rise to the following:

Negative : (0, 0.0001, 0.4)

Low : (0.5, 0.2, 0.2)

Moderate : (0.7, 0.2, 0.2)

High : (1, 0.2, 0.0001)

Thus, the normalized LR-type fuzzy number equivalent for the instance 1 of Table 12.4 becomes

Feature

Employee

Solution

Easier

Worth

Teachability

Risk

Output

Instances

acceptance

availability

solution

(1, 0.2,

(1, 0.4,

(0, 0.0001,

(0, 0.0001,

(1, 0.4,

(0, 0.0001,

1

1

0.0001)

0.0001)

0.4)

0.4)

0.0001)

0.4)

Similarly, the LR-type fuzzy number equivalents of the other instances are obtained.

Table 12.5 illustrates a sample set of input patterns that the fuzzy BP model was trained with. The weights obtained after a training session of 450

iterations is shown in Table 12.6. The output values inferred by fuzzy BP for a set of instances is shown in Table 12.7.

Table 12.5 Sample training set for the KBE system

Employee

Solution

Easier

S. no.

Worth

Teachability

Risk

Output

acceptance

available

solution

(1, 0.2,

(1, 0.4,

(0, 0.0001,

(0, 0.0001,

(1, 0.4,

(0, 0.0001,

1

1

0.0001)

0.0001)

0.4)

0.4)

0.0001)

0.4)

2

(0, 0.0001,

(0, 0.0001,

(0, 0.0001,

(0, 0.0001,

(0, 0.0001,

(0, 0.0001,

0

0.4)

0.4)

0.4)

0.4)

0.4)

0.4)

(0, 0.0001,

(0.5, 0.2,

(0.5, 0.2,

(0.5, 0.2,

(0.5, 0.2,

(0.5, 0.2,

3

0

0.4)

0.2)

0.2)

0.2)

0.2)

(0, 0.0001,

(1, 0.4,

(1, 0.4,

(1, 0.4,

(1, 0.4,

(1, 0.4,

4

0

0.4)

0.0001)

0.0001)

0.0001)

0.0001)

0.0001)

(0.5, 0.2,

(0, 0.0001,

(0, 0.0001,

(0, 0.0001,

(0, 0.0001,

5

0

0.2)

0.4)

0.4)

0.4)

0.4)

0.4)

(1, 0.2,

(0, 0.0001,

(0.5, 0.2,

(0, 0.0001,

(0, 0.0001,

(0.5, 0.2,

6

1

0.0001)

0.4)

0.2)

0.4)

0.4)

0.2)

(1, 0.2,

(1, 0.4,

(0.5, 0.2,

(1, 0.4,

(1, 0.4,

(1, 0.4,

7

0

0.0001)

0.0001)

0.2)

0.0001)

0.0001)

0.0001)

(0.5, 0.2,

(1, 0.4,

(0, 0.0001,

(1, 0.4,

(0, 0.0001,

(0.5, 0.2,

8

1

0.2)

0.0001)

0.4)

0.0001)

0.4)

0.2)

(1, 0.2,

(0, 0.0001,

(0.5, 0.2,

(0, 0.0001,

(1, 0.4,

(1, 0.4,

9

1

0.0001)

0.4)

0.2)

0.4)

0.0001)

0.0001)

(0.5, 0.2,

(1, 0.4,

(0, 0.0001,

(1, 0.4,

(0, 0.0001,

(0.5, 0.2,

10

0

0.2)

0.0001)

0.4)

0.0001)

0.4)

0.2)

....

Table 12.6 Weight sets obtained after training, for the KBE problem Input-hidden layer weights

W_{10}

-1.253724

0.781000

0.579000

$W\,20$

0.130494

0.988000

0.869000

$W\,30$

-0.624522

0.799000

0.750000

$W\,40$

-0.979520

0.645000

0.070000

$W\,50$

0.121162

0.752000

0.168000

$W\,60$

-0.125892

0.481000

0.409000

W 11

0.088709

0.575159

0.542470

W 21

6.221473

0.625117

0.233425

W 31

-5.355558

0.511356

0.319630

W 41

-1.196219

0.188272

0.947940

W 51

0.743903

0.943636

0.116286

W 61

1.205110

0.516413

0.110785

1.283483

0.757850

0.793306

W 12

W 22

-3.239704

0.286386

0.747749

W 32

1.565447

0.055406

0.138882

W 42

1.400189

0.412950

0.554903

W 52

4.461302

0.154162

0.607884

W 62

-3.491205

0.551971

0.071648

W 13

-0.017605

0.025093

0.562666

W 23

-4.533030

0.765363

0.196448

W 33

3.892954

0.700272

0.182422

W 43

-1.764890

0.746978

0.404761

W 53

-2.888355

0.329722

0.434618

W 63

-0.680463

0.320811

0.583909

W 14

-5.898080

0.333336

0.661306

W 24

-1.739380

0.505145

0.091717

W 34

-2.283170

0.669047

1.025956

W 44

1.141670

0.198082

0.078835

W 54

0.032954

0.063104

0.550654

W 64

-2.851645

0.751487

0.645628

W 15

1.379645

0.340031

0.872655

W 25

-3.077025

0.287930

0.553491

W 35

-4.642153

0.721427

0.555672

W 45

0.639301

0.113205

0.131933

W 55

0.410148

0.413361

0.927472

W 65

1.663710

0.465324

0.221375

W 16

-2.016176

0.732965

0.411935

W 26

-0.400687

0.884901

0.717908

W 36

3.493105

0.521547

0.435450

W 46

-1.445656

0.044423

0.681075

W 56

-1.904217

0.016407

0.922307

W_{66}

1.385423

0.279829

0.218603

Hidden-output layer weights

W_{00}

1.871553

-0.793852

1.153852

W_{10}

6.002691

-1.598233

2.992223

W_{20}

6.858886

-1.872296

2.578299

W_{30}

-7.659173

3.063065

-2.219060

$W'40$

-1.587710

0.773236

-0.301237

$W'50$

-4.652789

2.412276

-1.444268

$W'60$

-4.662981

2.096997

-0.763998

....

Table 12.7 Sample inference results by fuzzy BP for the KBE problem
Output

S.

Solution

Easier

Teachability

Expected

Worth

Employee

Risk

computed

No.

acceptance

available

solution

output

by fuzzy BP

(0.5,

(0,

(0,

(0.5, 0.2,

(0, 0.0001,

(0, 0.0001,

1

0.2,

0.0001,

0.0001,

0.956

1

0.2)

0.4)

0.4)

0.2)

0.4)

0.4)

(0.5,

(0.5,

(1, 0.4,

(0, 0.0001,

(1, 0.4,

(0, 0.0001,

2

0.2,

0.2,

0.011

0

0.0001)

0.4)

0.0001)

0.4)

0.2)

0.2)

(0,

(0,

(1, 0.2,

(0.5, 0.2,

(1, 0.4,

(1, 0.4,

3

0.0001,

0.0001,

0.961

1

0.0001)

0.2)

0.0001)

0.0001)

0.4)

0.4)

(0.5,

(0,

(1, 0.4,

(1, 0.4,

(1, 0.4,

4

0.2,

0.0001,

0.0001,

0.951

1

0.0001)

0.0001)

0.0001)

0.2)

0.4)

0.4)

(0,

(1, 0.2,

(1, 0.4,

(0.5, 0.2,

(0.5, 0.2,

(0.5, 0.2,

5

0.0001,

0.073

0

0.0001)

0.0001)

0.2)

0.2)

0.2)

0.4)

(1, 0.2,

(1, 0.4,

(0.5, 0.2,

(1, 0.4,

(1, 0.4,

(1, 0.4,

6

0.093

0

0.0001)

0.0001)

0.2)

0.0001)

0.0001)

0.0001)

(0,

(0,

(0.5,

(1, 0.2,

(0.5, 0.2,

(0, 0.0001,

7

0.0001,

0.0001,

0.2,

0.993

1

0.0001)

0.2)

0.4)

0.4)

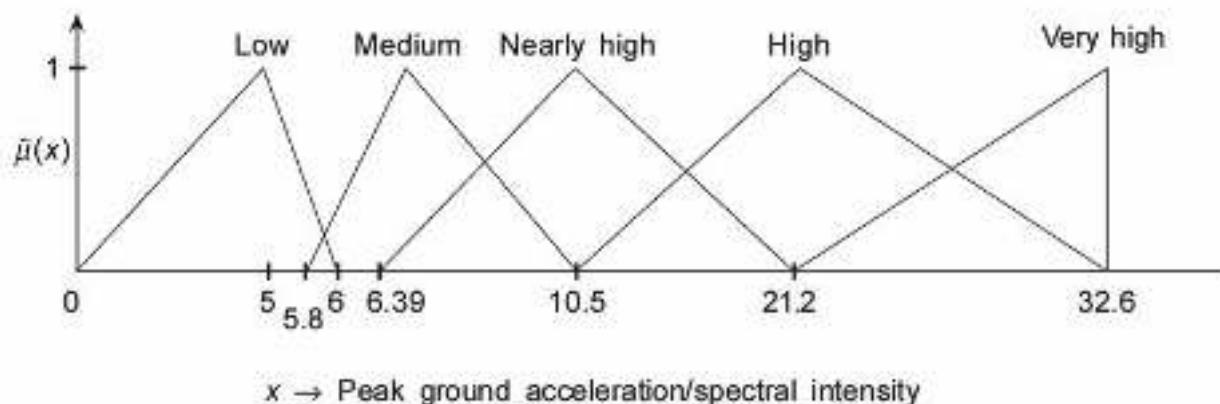
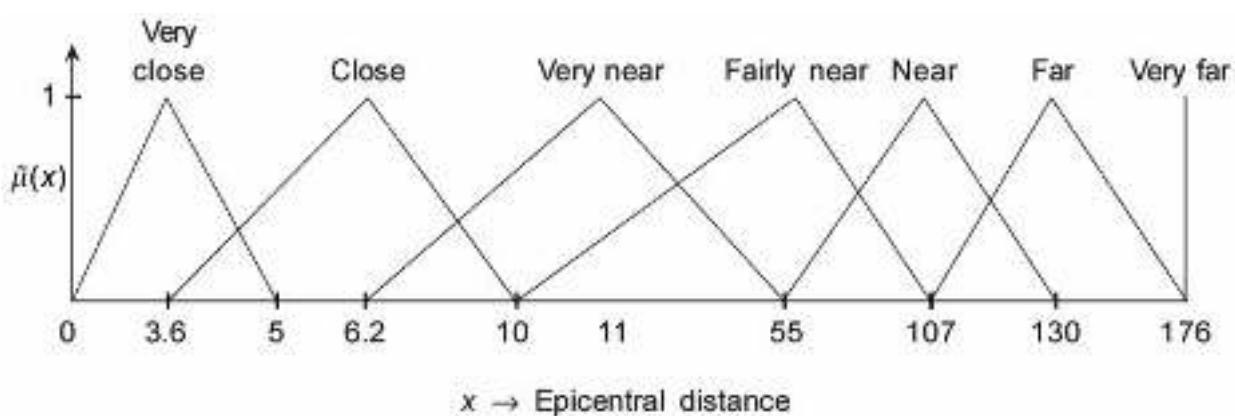
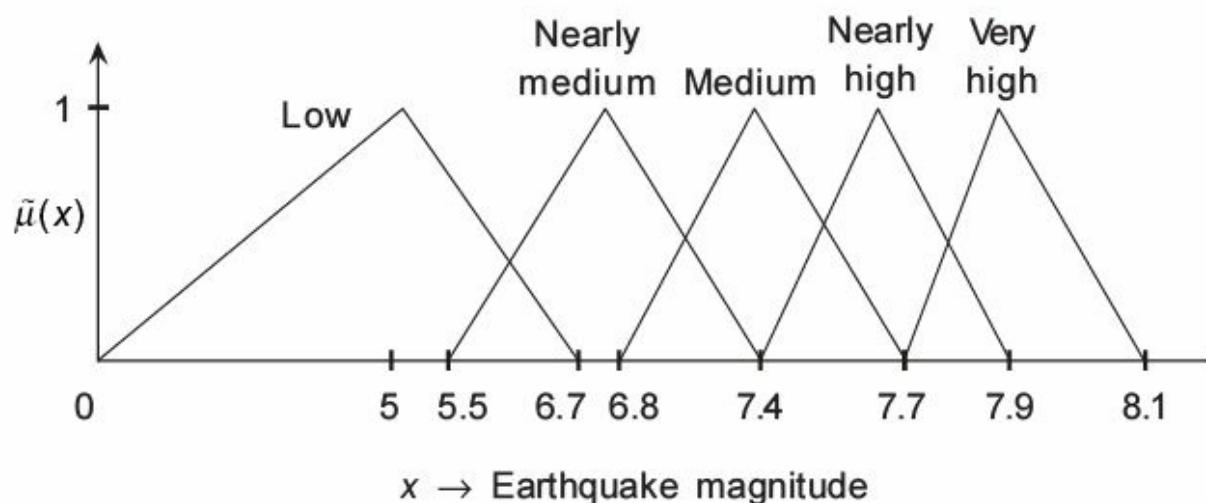
0.4)

0.2)

12.6.2 Earthquake Damage Evaluation

Evaluation of *earthquake damage* involves manipulation of vague concepts.

Song et al. (1996) have evaluated the damage characteristics of a few kinds of



structures situated at different distances from the damage centres for a few earthquakes that occurred in Japan. We make use of the inputs presented in the paper to demonstrate the application of fuzzy BP model for earthquake damage evaluation.

The inputs to fuzzy BP are earthquake magnitude, epicentral distance, and the ratio of peak ground acceleration and spectrum intensity. The fuzzy linguistic values associated with the above three inputs have been given in Fig. 12.7. The output to be determined is the damage membership value.

Fig. 12.7 Fuzzy sets associated with the input attributes for the earthquake damage evaluation

problem.

Table 12.8 illustrates the LR-type fuzzy number equivalents of the fuzzy terms. Table 12.9 presents a sample training set. The damage membership value obtained by fuzzy BP for a testing set and the same obtained by Song et al. has been shown in Table 12.10.

Table 12.8 LR-type fuzzy number equivalents for the fuzzy sets associated with earthquake damage evaluation

Earthquake magnitude

Epicentral distance

Peak ground acceleration/spectrum intensity

LR-type

LR-type

LR-type

Fuzzy set

Fuzzy set

Fuzzy set

fuzzy no.

fuzzy no.

fuzzy no.

Low

(5, 5, 1.7)

Very close

(3.6, 3.6, 1.4)

Low

(5.58, 5.58, 0.2)

Nearly medium

(6.7, 1.2, 0.7)

Close

(6.2, 2.6, 3.8)

Medium

(6.39, 0.5, 4.11)

Very near

(11, 4.8, 44)

Medium

(7.4, 0.6, 0.3)

Fairly near

(55, 45, 52)

Nearly high

(10.5, 4.1, 10.7)

Near

(107, 52, 23)

Nearly High

(7.7, 0.3, 0.2)

Far

(130, 23, 46)

High

(21.2, 10.7, 11.4)

Very high

(7.9, 0.2, 0.2)

Very far

(176, 46, 0.0001)

Very high

(32.6, 11.4, 0.0001)

....

Table 12.9 A sample training set for fuzzy BP for the earthquake damage evaluation problem (normalised data) Peak ground

Damage

Earthquake

Epicentral

S.no.

acceleration/spectrum

membership

magnitude

distance

intensity

value

1

(0.633, 0.633, 0.215)

(0.021, 0.021, 0.008)

(1, 0.35, 0.00003)

0.035

2

(1, 0.025, 0.025)

(1, 0.261, 0.000005)

(0.196, 0.015, 0.126)

0.24

3

(0.937, 0.076, 0.038)

(0.739, 0.131, 0.261)

(0.171, 0.171, 0.006)

0.6

4

(0.975, 0.038, 0.025)

(0.608, 0.295, 0.131)

(0.196, 0.015, 0.126)

0.181

5

(0.848, 0.152, 0.089)

(0.313, 0.256, 0.296)

(0.65, 0.33, 0.35)

0.1

6

(0.633, 0.633, 0.215)

(0.035, 0.015, 0.022)

(0.171, 0.171, 0.006)

0.41

7

(0.937, 0.076, 0.038)

(0.035, 0.015, 0.022)

(0.32, 0.13, 0.33)

1

...

Table 12.10 Sample results inferred by fuzzy BP for the earthquake damage evaluation problem Peak ground

Expected

Damage

Earthquake

Epicentral

acceleration/

damage

membership

Earthquake

magnitude

distance

spectral

membership

value computed

intensity

value

by fuzzy BP

Tokiachi-oki (1968)

79

176

5.96

0.23

0.227

Chiba-ken-toho-oki (1987)

6.7

55

21.2

0.1

0.111

$$\tilde{O}_{pi} = \tilde{I}_{pi}, i = 1, 2, \dots, l$$

$$\tilde{O}_o = (1, 0, 0)$$

$$O'_{pj} = f(\text{NET}_{pj}) \quad j = 1, 2, \dots, m$$

$$O'_{po} = 1$$

$$NET_{pj} = CE \left(\sum \tilde{W}_{ji} \tilde{O}_{pi} \right)$$

$$O''_{pk} = f(NET'_{pk}) \quad k = 0, 1, \dots, n - 1$$

$$NET'_{pk} = CE \left(\sum \tilde{W}'_{kj} O'_{pj} \right)$$

$$\Delta \tilde{W}(t) = -\eta \nabla E_p(t) + \alpha \Delta \tilde{W}(t-1)$$

$$\tilde{W}(t) = \tilde{W}(t-1) + \Delta \tilde{W}(t)$$

SUMMARY

In this Chapter, a neuro-fuzzy hybrid system fuzzy BP proposed by Lee and Lu has been discussed. The network maps fuzzy input vectors to crisp outputs. A backpropagation like learning algorithm is used by fuzzy BP for its training.

The network is three layered and of feedforward type. The fuzzy inputs presented to fuzzy BP need to be LR-type fuzzy numbers.

The input neurons compute their output as

The hidden neurons compute their output as

where

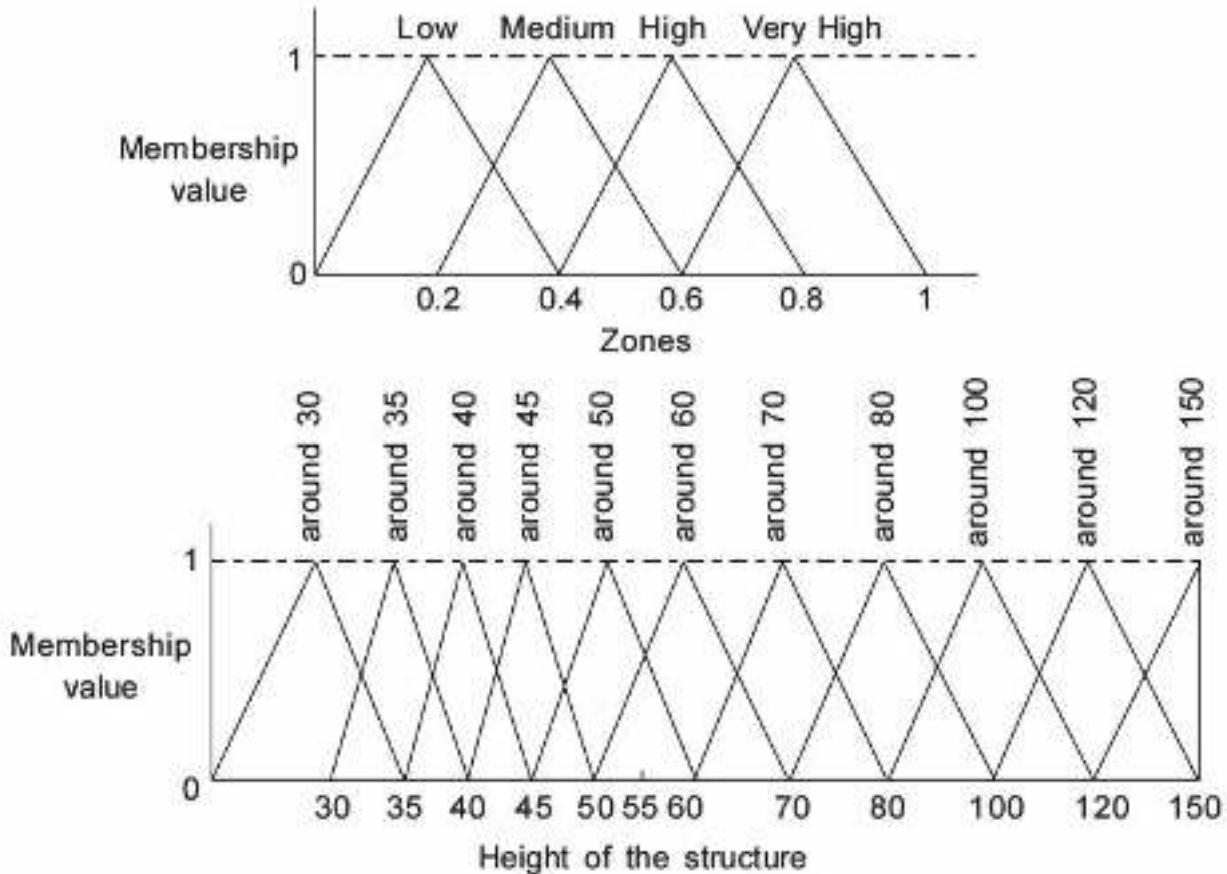
The output neurons compute their output as

where

During the learning session the weight changes in the input-hidden and hidden-output layers are computed as

The updated weights are given by

The model has been applied to two problems, namely knowledge base evaluation and earthquake damage evaluation.



PROGRAMMING ASSIGNMENT

P12.1 Implement the fuzzy BP algorithms shown in Algorithms 12.1 and 12.2.

P12.2 Wind pressure prediction Wind pressure load is of major concern in the design of industrial as well as tall structures. The load exerted by wind on the structure is proportional to the wind pressure. Wind pressure varies with height of the structure and the zone in which the structure is loaded. The fuzzy representation of height of the structure and zones are shown in Fig. P12.1. The corresponding wind pressures (kgf/m²) are given in Table P12.1.

Fig. P12.1 Fuzzy representation of inputs for wind pressure prediction.

(a) Obtain the LR-type fuzzy number equivalents for the input attributes shown in

Fig. P12.1.

(b) Normalize the inputs obtained in (a).

(c) From among the 44 instances that can be obtained using Table

Table P12.1 Wind pressure variation with height and zone

Zone \ Height	30	35	40	45	50	60	70	80	100	120	150
Low	60	62	63	65	67	68	71	73	76	79	83
Medium	100	104	105	108	111	115	118	122	127	132	138
High	150	156	158	163	167	172	177	183	191	198	207
Very high	200	208	210	217	222	230	236	244	254	264	267

P12.1, train the fuzzy BP model for 25 randomly selected instances.

(d) Infer the remaining 9 instances using the weight sets obtained from (c).

REFERENCES

Adeli, H. and S.L. Hung (1995), Machine Learning—Neural Networks, Genetic Algorithms, and Fuzzy Systems, John Wiley & Sons, New York.

Dubois, D. and H. Prade (1979), Fuzzy Real Algebra: Some Results, FSS 2, pp. 327–348.

Dubois, D. and H. Prade (1988), Fuzzy Sets and Systems: Theory and Applications, Academic Press, Boston.

Lee Hahn Ming and Lu Bing Hui (1994), Fuzzy BP: A Neural Network Model with Fuzzy Inference, Proc. ICANN 94, pp. 1583–1588.

Song Bo, S. Hao, Murakami Suminao and Sadohara Satoru (1996), Comprehensive Evaluation Method on Earthquake Damage using Fuzzy Theory, Joul of Urban Planning and Development, March, pp. 1–17.

Tsoukalas, Lefteri H. and E. Robert Uhrig (1997), Fuzzy and Neural Approaches in Engineering, John Wiley & Sons, Inc.

Chapter 13

Simplified Fuzzy ARTMAP

Adaptive resonance architectures are neural networks which in response to arbitrary sequences of input patterns, self-organize stable recognition codes in real time. Grossberg (1976) introduced the basic principles of Adaptive Resonance Theory (ART).

A class of ART architectures specified as systems of differential equations have been evolved by Carpenter and Grossberg (1987a, 1987b). Termed ART1 and ART2, ART1 architecture self-organizes recognition categories for arbitrary sequence of binary input patterns and ART2 does the same for either binary or analog inputs. The third class of architecture termed ART3

(Carpenter and Grossberg, 1990) based on ART2, includes a model of the chemical synapse that solves the memory search problem of ART systems embedded in network hierarchies, where there can be in general, either fast or slow learning and distributed or compressed code representations. ART2A (Carpenter et al., 1991) models the essential dynamics of ART2 architectures and runs two to three orders of magnitude faster than ART2.

ARTMAP is a class of neural network architectures that performs incremental supervised learning of recognition categories and multidimensional maps in response to input vectors presented in arbitrary order.

The first ARTMAP system (Carpenter et al., 1991) was used to classify inputs by the set of features they possess, that is, by an ordered n -tuple of binary values representing the presence or absence of each possible feature.

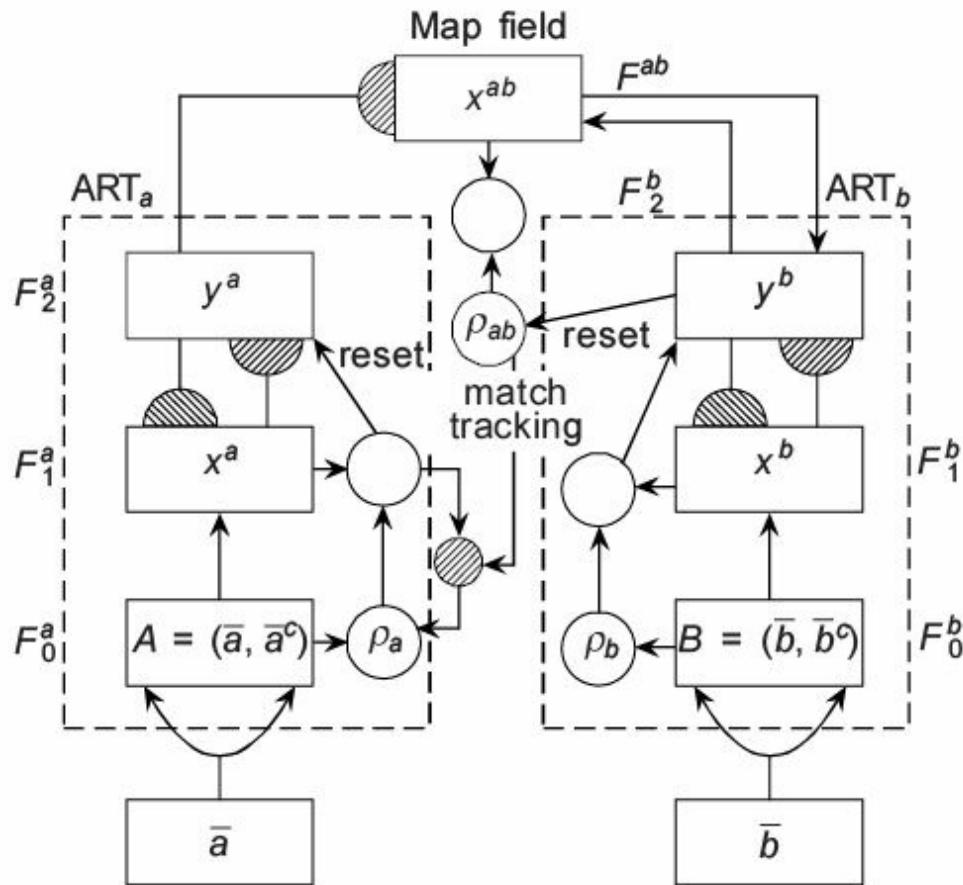
A more general ARTMAP system termed fuzzy ARTMAP (Carpenter et al., 1992) learns to classify inputs by a fuzzy set of features or a pattern of fuzzy membership values between 0 and 1, indicating the extent to which each feature is presented.

The architecture of fuzzy ARTMAP is briefly reviewed in the following section.

$$\{\bar{a}^p, \bar{b}^p\},$$

$$\{\bar{a}^p\}$$

$$\{\bar{b}^p\}$$



$$\bar{a}$$

\bar{b}

$\{\bar{a}, \bar{a}^c\}$

$\{(\bar{b}, \bar{b}^c)\}$

13.1 FUZZY ARTMAP: A BRIEF INTRODUCTION

Fuzzy ARTMAP is an architecture which synthesizes fuzzy logic with adaptive resonance theory neural networks. Figure 13.1 illustrates the architecture of fuzzy ARTMAP. The architecture comprises two ART

modules, ART a and ART b that create stable recognition categories in response to arbitrary sequence of input patterns. During supervised learning, given a set of input patterns

ART a receives a stream

of input

patterns and ART b a stream of

. These modules are linked by an

associative learning network and an internal controller that ensures autonomous system operation in real time. Fab which is the inter-art module that links together ART a and ART b modules and known as the *map field* gets triggered whenever one of the ART a or ART b categories is active.

Fig. 13.1 Architecture of fuzzy ARTMAP.

The *complement coding processor* in ART a and ART b transforms the input patterns and into complement coded input pairs $A =$

and $B =$

respectively. $F a$

b

1 and F 1 receive A and B as inputs.

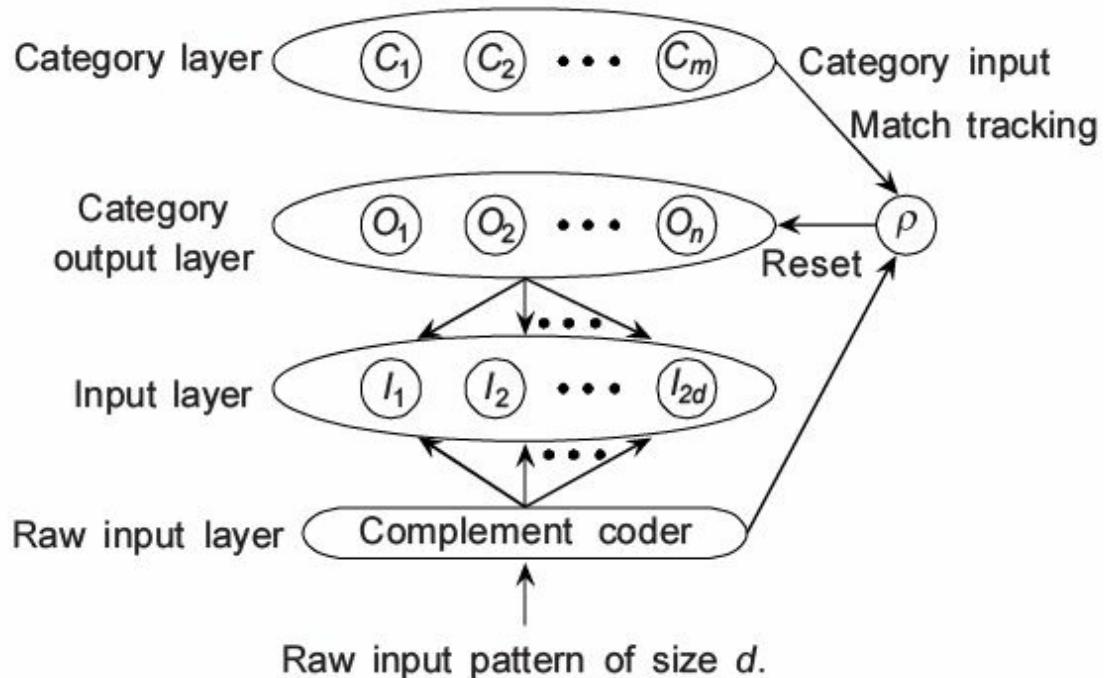
When a prediction made by ART *a* module is disconfirmed at ART *b*, inhibition of the map field activation induces the match tracking process.

Match tracking raises the ART

a

a

a vigilance ρ *a* to just above F 1 to F 0 match ratio. This triggers an ART *a* search which results in the activation of either an ART *a* category that predicts *b* correctly or to a previously uncommitted ART *a* category node.



13.2 SIMPLIFIED FUZZY ARTMAP

Kasuba's *Simplified Fuzzy ARTMAP* (Kasuba, 1993) which is a vast simplification of Carpenter and Grossberg's fuzzy ARTMAP has reduced computational overhead and architectural redundancy when compared to its

predecessor. Also, the model employs simple learning equations with a single user selectable parameter and can learn every single training pattern within a small number of training iterations.

Simplified fuzzy ARTMAP is essentially a two-layer net containing an input and an output layer. Figure 13.2 illustrates the architecture of simplified fuzzy ARTMAP.

Fig. 13.2 Simplified fuzzy ARTMAP.

The input to the network flows through the *complement coder* where the input string is stretched to double the size by adding its complement also. The complement coded input then flows into the input layer and remains there.

Weights (W) from each of the *output category nodes* flow down to the input layer. The *category layer* merely holds the names of the M number of *categories* that the network has to learn. *Vigilance parameter* and *match tracking* are mechanisms of the network architecture which are primarily employed for network training.

ρ which is the vigilance parameter can range from 0 to 1. It controls the granularity of the output node encoding. Thus, while high vigilance values makes the output node much fussier during pattern encoding, low vigilance

\bar{a}

\bar{a}^c

\bar{a}^c

\bar{a}^c

\bar{a}

\bar{a}

\bar{a}

\bar{a}

$$|p| = \sum_{i=1}^d p_i, \text{ for } p = (p_1, p_2, \dots, p_d)$$

$$|I| = |(\bar{a}, \bar{a}^c)| = \sum_{i=1}^d a_i + \left(d - \sum_{i=1}^d a_i \right)$$

renders the output node to be liberal during the encoding of patterns.

The match tracking mechanism of the network is responsible for the adjustment of vigilance values. Thus, when an error occurs in the training phase during the classification of patterns, i.e. when the selected output node does not represent the same output category corresponding to the input pattern presented, match tracking is evoked. Depending on the situation, match tracking may result in the network adjusting its learning parameters and the network opening new output nodes.

13.2.1 Input Normalization

Complement coding is used for input normalization and it represents the presence of a particular feature in the input pattern and its absence. For example, if a is the given input pattern vector of d features, i.e. = ($a_1, a_2,$

\dots, a_d) the complement coded vector

represents the absence of each

feature, where

is defined as

$$= (1 - a_1, 1 - a_2, \dots, 1 - a_d) \quad (13.1) \text{ The normalization process is essential since simplified fuzzy ARTMAP}$$

needs all its input values to lie between 0 to 1. Therefore, the complement coded input vector I obtained by concatenating c with c is given by the vector

$$I = (c, c) = (a_1, a_2, \dots, a_d, 1 - a_1, 1 - a_2, \dots, 1 - a_d) \quad (13.2) \text{ The learning equations of the architecture call for the computation of } |I|.$$

Here, ' $| |$ ' is the norm of a vector defined as

$$(13.3)$$

Observe that for a complement coded vector I , $|I|$ results in the automatic normalization of input vectors, i.e.

$$(13.4)$$

13.2.2 Output Node Activation

When the simplified fuzzy ARTMAP is presented the complement coded

$$T_j(I) = \frac{|I \wedge W_j|}{\alpha + |W_j|}$$

$$\frac{|I \wedge W_j|}{|I|}$$

forms of input patterns, all output nodes become active to varying degrees.

This output activation, denoted by T_j and referred to as the activation function for the j th output node, where W_j is the corresponding top-down weight, is given by

$$(13.5)$$

Here, α is kept as a small value close to 0 usually about 0.0000001. That node which registers the highest activation function is deemed winner, i.e.

$$\text{Winner} = \max(T_j) \quad (13.6)$$

In the event of more than one node emerging as the winner, owing to the same activation function value, some mechanism such as choosing a node with the smallest index may be devised to break the tie. The category associated with the winner is the one to which the given input pattern belongs to, as classified by the network.

The match function which helps to determine whether the network must adjust its learning parameters is given by

$$(13.7)$$

As mentioned earlier, the match function in association with the vigilance parameter decides on whether a particular output node is good enough to encode a given input pattern or whether a new output node should be opened to encode the same. The network is said to be in a state of *resonance* if the match function value exceeds vigilance parameter. However, for a node to exhibit resonance, it is essential that it not only encodes the given input pattern but should also represent the same category as that of the input pattern.

On the other hand, the network is said to be in a state of *mismatch reset* if the vigilance parameter exceeds match function. Such a state only means that the particular output node is not fit enough to learn the given input pattern and thereby cannot update its weights even though the category of the output node may be the same as that of the input pattern. This is so, since the output node has fallen short of the expected encoding granularity indicated by the vigilance parameter.

The weight updating equation of an output node j when it proceeds to learn

$$W_j^{\text{new}} = \beta(I \wedge W_j^{\text{old}}) + (1 - \beta)W_j^{\text{old}}$$

the given input pattern I is given by

$$\\\\\\\\(13.8)$$

where, $0 < \beta \leq 1$

Once the network has been trained, the inference of patterns, known or unknown, i.e. the categories to which the patterns belong, may be easily computed. This is accomplished by passing the input pattern into the complement coder and then to the input layer. All the output nodes compute the activation functions with respect to the input. The winner, which is the node with the highest activation function, is chosen. The category to which the winning output node belongs is the one to which the given input pattern is classified by the network.

Algorithms 13.1 and 13.2 illustrate the training and inference phases of simplified fuzzy ARTMAP.

Algorithm 13.1 Simplified Fuzzy ARTMAP: Training

ALGORITHM SFARTMAP-TRAIN (ρ, α, I)

Step 1: Choose an appropriate value for the vigilance parameter ($0 < \rho < 1$) and a small value for α . Set NO_OF_TRAINING_EPOCHS to the desired number of training epochs and COUNT_OF_TRAINING_EPOCHS to 0.

Step 2: $i \leftarrow 1$;

COUNT_OF_TRAINING_EPOCHS = COUNT_OF_TRAINING_EPOCHS + 1;
while (COUNT_OF_TRAINING_EPOCHS ≤ NO_OF_TRAINING_EPOCHS)
Repeat Steps 3-12;

Step 3: Input the pattern vector $I_i = (a_{i1}, a_{i2}, \dots, a_{id})$ of dimension d and its category C_i .

Step 4: Compute the augmented input vector

$$AI_i = (a_{i1}, a_{i2}, \dots, a_{id}, 1 - a_{i1}, 1 - a_{i2}, \dots, 1 - a_{id})$$

Step 5: If AI_i is the first input in the given category C_i set the top down weight vector W_i as AI_i
i.e. $W_i = AI_i$;
Link W_i to the category C_i ;
Go to Step 12;

Step 6: If AI_i is an input pattern vector whose category already exists then compute the activation function $T_j(AI_i)$ for each of the existing top-down weight nodes W_j

$$T_j(AI_i) = \frac{|AI_i \wedge W_j|}{\alpha + |W_j|};$$

Step 7 : Choose that top-down weight node k which records the highest activation function

$$T_k(AI_i) = \max_j T_j(AI_i)$$

Step 8 : Compute the match function $MF_k(AI_i)$ of the winning node k ;

If $MF_k(AI_i) > \rho$ and C_i is same as that category C_k linked to W_k

then update weight vector W_k as $W_k^{\text{new}} = W_k^{\text{old}} + (I \wedge W_k^{\text{old}})$

(Here $\beta = 1$ has been chosen in Eq.(13.8))

goto step 12;

Step 9 : If $MF_k(AI_i) > \rho$ and C_i is not the category C_k linked to W_k then

Undertake match tracking by setting ρ to $MF_k(AI_i)$ and incrementing by a small value ϵ .

$$\rho = MF_k(AI_i) + \epsilon;$$

If some more top-down weight nodes exist

then consider the next highest winner W_k among the top-down weight nodes;

goto Step 8;

else goto Step 11;

```
Step 10: If  $MF_k(AI_i) < \rho$ 
then
    If some more top-down weight nodes exist
    then
        Consider the next highest winner  $W_k$  among the top-
        down weight nodes.
        goto Step 8;
    else goto Step 11;

Step 11: Create a new top-down weight node  $W_i$  such that  $W_i = AI_i$ 
and link the node to the category  $C_j$ ;

Step 12: If no more input patterns then goto step 13;
else
     $i \leftarrow i + 1$ 
    goto Step 3;

Step 13: goto Step 2;

END SFARTMAP-TRAIN.
```

Algorithm 13.2 Simplified Fuzzy ARTMAP-Inference

ALGORITHM SFARTMAP-INF(W, I)

Step 1 : Let W_j , $j = 1, 2, \dots, s$ indicate s top-down weight vectors obtained after training the network with a given set of training patterns;

Let I_i be the inference pattern set each of whose category is to be inferred by the network;

$i \leftarrow 1;$

Step 2 : Read input I_i ;

Step 3 : Compute the augmented input AI_i ;

Step 4 : for $j \leftarrow 1$ to s
Compute the activation functions

$$T_j(AI_i) = \frac{|AI_i \wedge W_j|}{\alpha + |W_j|}$$

end

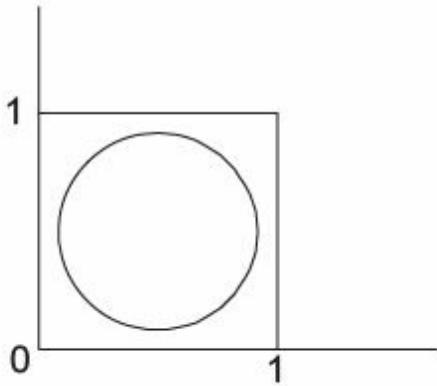
Step 5 : Choose the winner k among the S activation functions

$$T_k(AI_i) = \max_j T_j(AI_i)$$

Step 6 : Output the category C_k linked to $T_k(AI_i)$ as the one to which I_i belongs to.

Step 7 : If no more inference pattern vectors
then exit
else $i \leftarrow i + 1$;
goto Step 2;

END SFARTMAP-INF.



13.3 WORKING OF SIMPLIFIED FUZZY ARTMAP

Illustration— *Simulation of Circle in Square*

The circle in the square problem requires a system to identify which pockets of a square lie inside and which lie outside a circle whose area equals half that of the square (refer Fig. 13.3).

Fig. 13.3 Circle in the square.

The problem, specified as a benchmark problem for system performance evaluation in the DARPA *Artificial Neural Network Technology* (ANNT) program, has been used to illustrate the working of simplified fuzzy ARTMAP (Kasuba, 1993). In this, the network is trained with a set of points for a definite number of training epochs. The inputs during training are the points (x, y) and the category to which they belong, namely inside the circle (IN) or outside the circle (OUT). The training of the network has been illustrated in the following examples.

Training

Example 13.1 (*Learning the input (0.7, 0.7) (IN)*) Consider the point (0.7, 0.7) as input I and whose category is IN.

Complement of $I = (0.3, 0.3)$

Augmented input $I = (0.7, 0.7, 0.3, 0.3)$

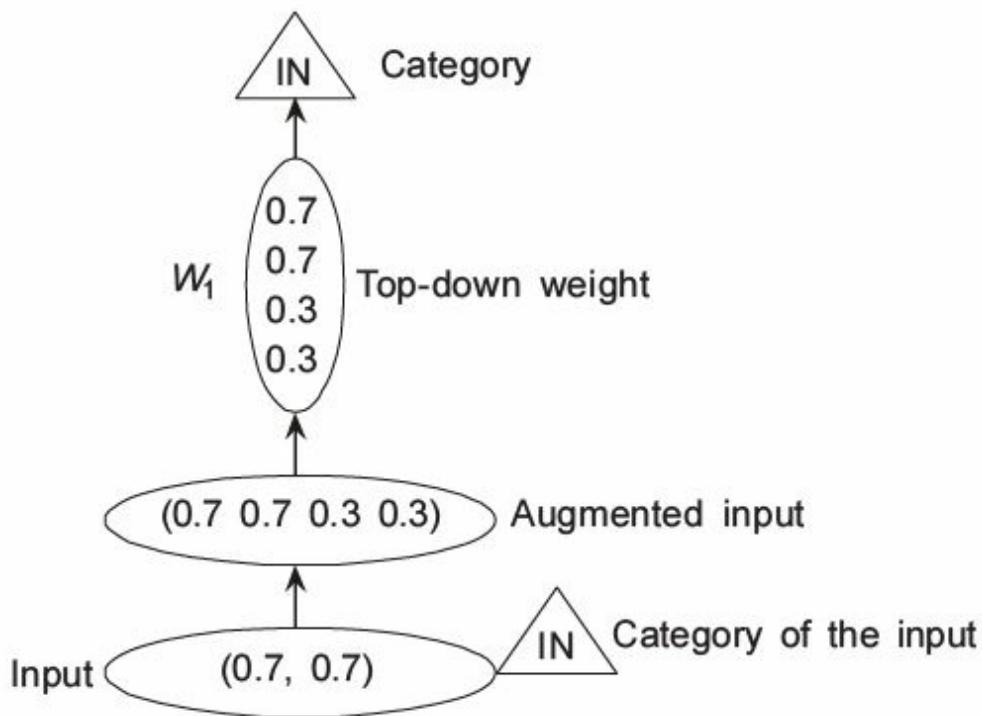
Since I is the first input seen by the network in the IN category, the top-down weights

W

c

c

$i = (W_x, W_y, W_x, W_y)$ are set to the augmented input values and is therefore given by $W_1 = (0.7, 0.7, 0.3, 0.3)$. The activation function value



Consider

$$I = (0.3, 0.8) \text{ with category as IN.}$$

$$\text{Complement of } I = (0.7, 0.2)$$

$$\text{Augmented input } I = (0.3, 0.8, 0.7, 0.2)$$

$$\text{Activation function } T_1(I) = \frac{|I \wedge W_1|}{\alpha + |W_1|} = 0.7499$$

(Choosing $\alpha = 0.0000001$)

$$\begin{aligned}\text{Match function value} &= MF(I) = \frac{|I \wedge W_1|}{|I|} \\ &= 0.75\end{aligned}$$

$T_1(I)$ of the top-down weight node W_1 for the input I seen, is set to null, and is set to point the category IN in the category layer. Figure 13.4 illustrates the sequence for handling the input $(0.7, 0.7)$.

Fig. 13.4 Training simplified fuzzy ARTMAP—learning the input $(0.7, 0.7)$.

Example 13.2 (*Learning the input $(0.3, 0.8)$ (IN)*) Here, we choose the vigilance parameter ρ to be 0.5. Now $MF(I)$ is greater than ρ and since the category of I is the same as that pointed by W_1 , W_1 is fit enough to learn the current input I . This is accomplished by updating the weights of W_1 as illustrated in Eq. (13.8).

Choosing $\beta = 1$,

W_{new}

1

$$= (0.3, 0.7, 0.3, 0.2)$$

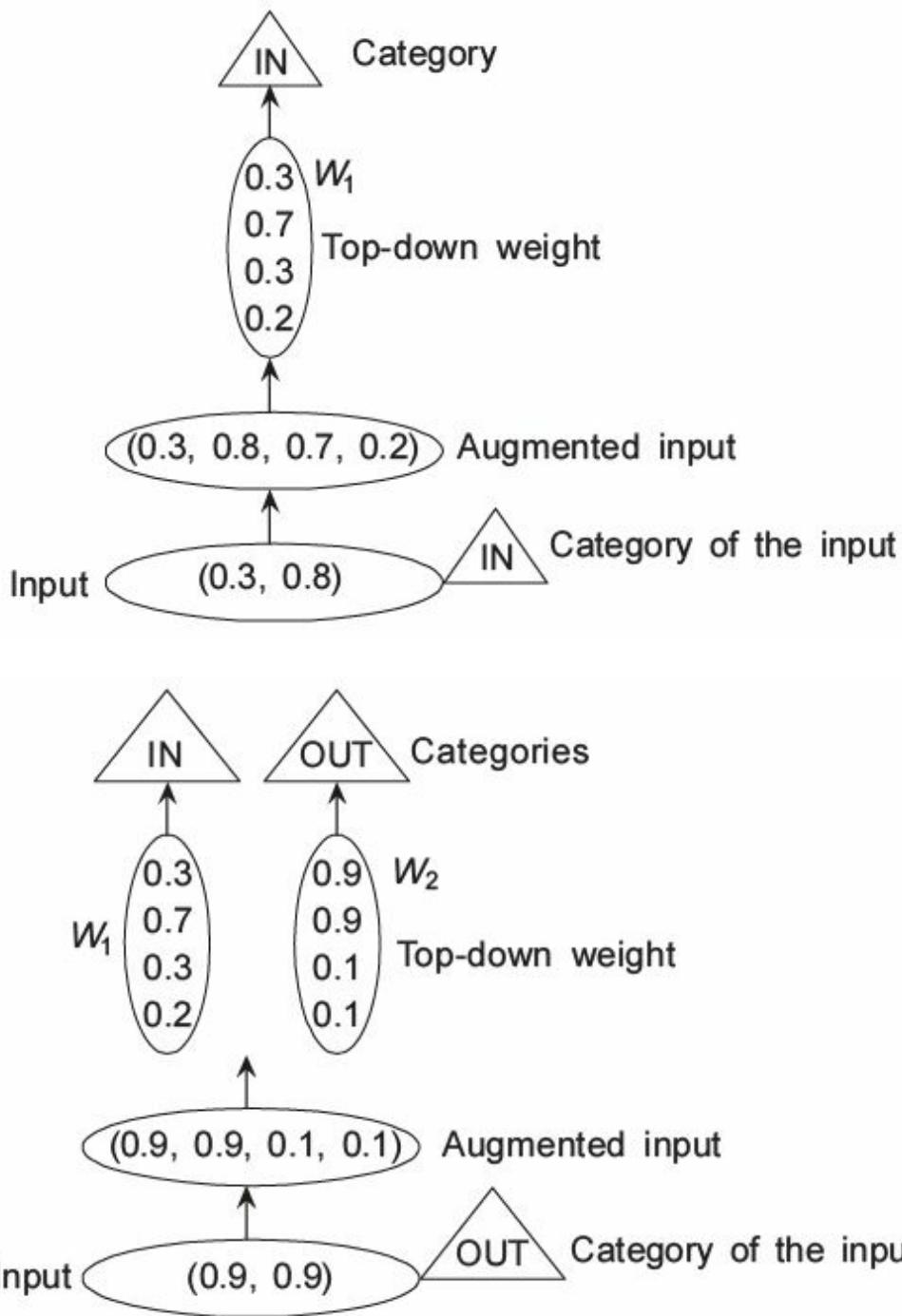


Figure 13.5 illustrates the sequence for processing the input $(0.3, 0.8)$.

Fig. 13.5 Training of simplified fuzzy ARTMAP—learning the input $(0.3, 0.8)$.

Example 13.3 (*Learning the input (0.9, 0.9) (OUT)*) Consider $I = (0.9, 0.9)$ and category = OUT.

Since this is a new category, repeating the computations illustrated in Example 13.1, the new top-down weight node W_2 , which points to the category OUT, is given by

$$W_2 = (0.9, 0.9, 0.1, 0.1) \text{ and}$$

$$T_2(I) = \text{Null}$$

Figure 13.6 illustrates the sequence for handling the input (0.9, 0.9).

Fig. 13.6 Training of simplified fuzzy ARTMAP—learning the input (0.9, 0.9).

Example 13.4 (*Learning the input (0.7, 0.9) (OUT)*) Consider $I = (0.7, 0.9)$ and category = OUT.

The augmented input $I = (0.7, 0.9, 0.3, 0.1)$

Since there are two weight nodes, to decide which node is fit enough to learn the new input I , the activation function values of the two nodes are computed, i.e.

$$T_1(I) = 0.9333$$

$$T_2(I) = 0.8999$$

The node with the highest activation function value W_1 in this case is chosen to learn the new input. Also, the match function of I with W_1 is greater than the vigilance parameter, i.e.

$$(MF(I) = 0.7) > (\rho = 0.5)$$

However, there is a category mismatch since the category pointed to by W_1 (IN) and that represented by I (OUT) are different. In such a case, W_1 is not fit to learn the input and hence, the next node is to be considered. Before

proceeding to the next node, match tracking is done by updating the vigilance parameter to the match function value $MF(I)$ and incrementing it by a small quantity, i.e.

$$\rho = 0.701$$

The next node W_2 gives

$$MF(I) = 0.9$$

Since $MF(I) > \rho$ and the categories are also the same, learning occurs in W_2 , given by the updating of W_2 as

W_{new}

2

$$= (0.7, 0.9, 0.1, 0.1)$$

Figure 13.7 illustrates the sequence for handling the input $(0.7, 0.9)$.

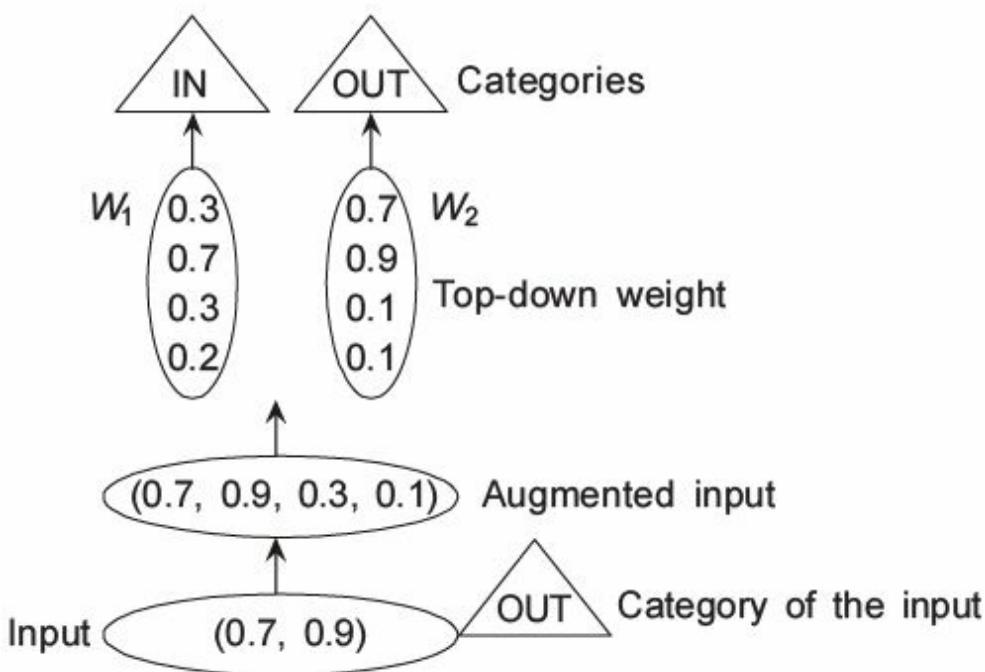


Fig. 13.7 Training of simplified fuzzy ARTMAP—learning the input (0.7, 0.9).

Example 13.5 (Learning the input (0.1, 0.3) (IN)) Consider the input $I = (0.1, 0.3)$ and category = IN.

The augmented input $I = (0.1, 0.3, 0.9, 0.7)$

The activation function values of W_1 and W_2 are $T_1(I) = 0.5999$

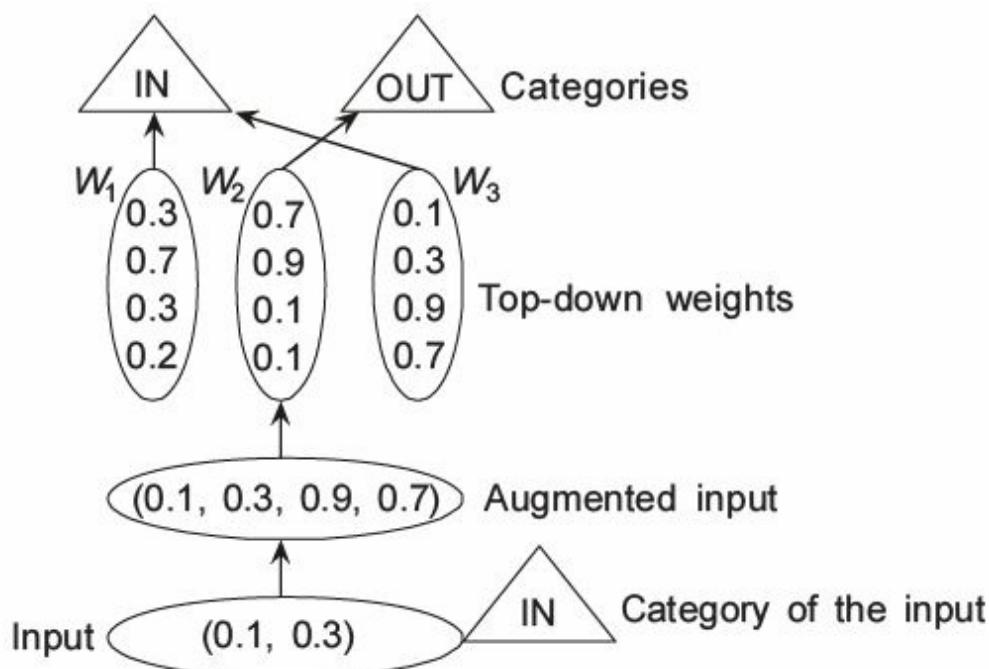
$$T_2(I) = 0.33$$

Choosing the highest, namely W_1 , the match function yields $MF(I) = 0.45$

which is less than ρ , rendering the node to be misfit to learn the pattern I . The choice of W_2 also results in a similar case with ($MF(I) = 0.3 < \rho = 0.701$).

In such a case a new top-down weight node W_3 pointing to IN is created with $W_3 = (0.1, 0.3, 0.9, 0.7)$.

Figure 13.8 illustrates the handling of input (0.1, 0.3) by the network.



$$T_1(I') = \frac{|I' \wedge W_1|}{\alpha + |W_1|} = 0.7999$$

$$T_2(I') = \frac{|I' \wedge W_2|}{\alpha + |W_2|} = 0.4999$$

$$T_3(I') = \frac{|I' \wedge W_3|}{\alpha + |W_3|} = 0.8499$$

Fig. 13.8 Training of simplified fuzzy ARTMAP—learning the input (0.1, 0.3).

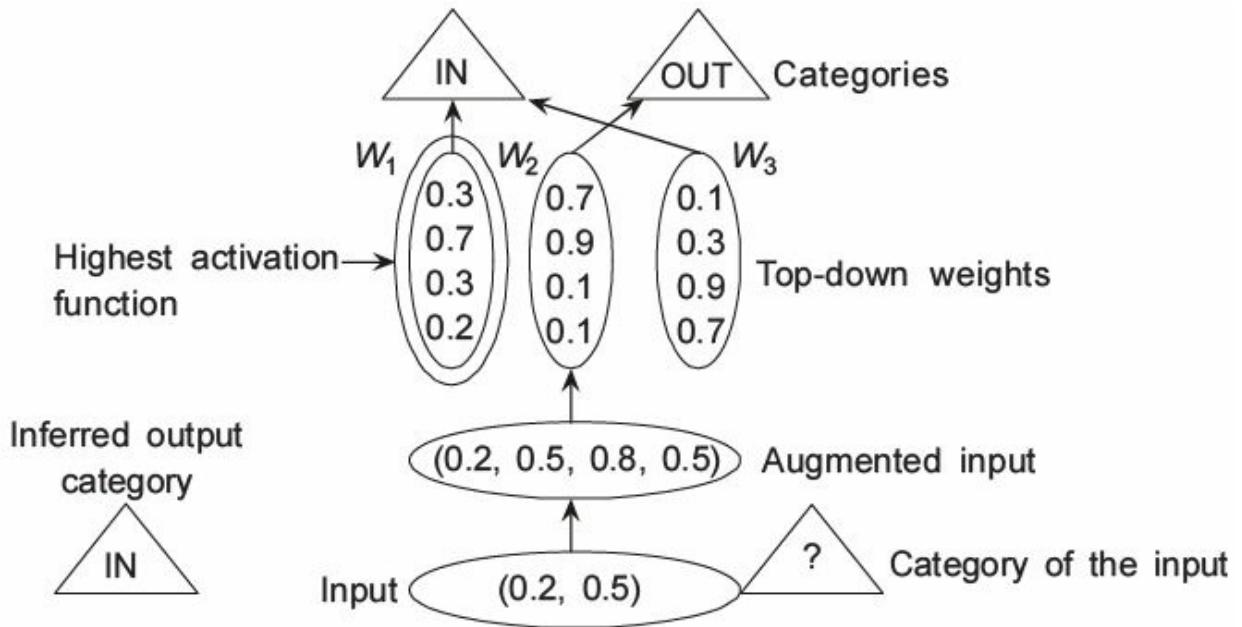
Inference

During inference, the architecture is presented points (x, y) alone, to determine the category. In this case, that top-down weight node which reports the highest activation function value for the given (x, y) is the winner and the category pointed to by the node is the category to which (x, y) belongs.

Example 13.6 (*Inferring the category of (0.2, 0.5)*) Consider the network evolved out of training as illustrated in Examples (13.1)–(13.5). The weight nodes are $W_1 = (0.3, 0.7, 0.3, 0.2)$, $W_2 = (0.7, 0.9, 0.1, 0.1)$, and $W_3 = (0.1, 0.3, 0.9, 0.7)$.

Consider $I' = (0.2, 0.5)$ whose category is IN. The objective now, is to test whether the given input is correctly categorized by the network belonging to IN. The activation functions for the input I' corresponding to the three weight nodes are

Choosing a weight node with the highest activation function, we select W_3



which is attached to the category IN. Thus, the input given is inferred to belong to the category IN.

Figure 13.9 illustrates the inference of $(0.2, 0.5)$.

Fig. 13.9 Inference by simplified fuzzy ARTMAP—Inferring $(0.2, 0.5)$.

Example 13.7 (*Inferring the category of $(0.9, 0.75)$*) For the input $I'' = (0.9, 0.75)$ which belongs to the category OUT, inference by the architecture assumed in Example 13.6 yields the following activation function computations.

$$T 1(I'') = 0.8667$$

$$T 2(I'') = 0.9167$$

$$T 3(I'') = 0.3749$$

Choosing $W 2$ which yields the highest activation function value, yields the category OUT which is indeed the correct result. Figure 13.10 illustrates the inference of $(0.9, 0.75)$.

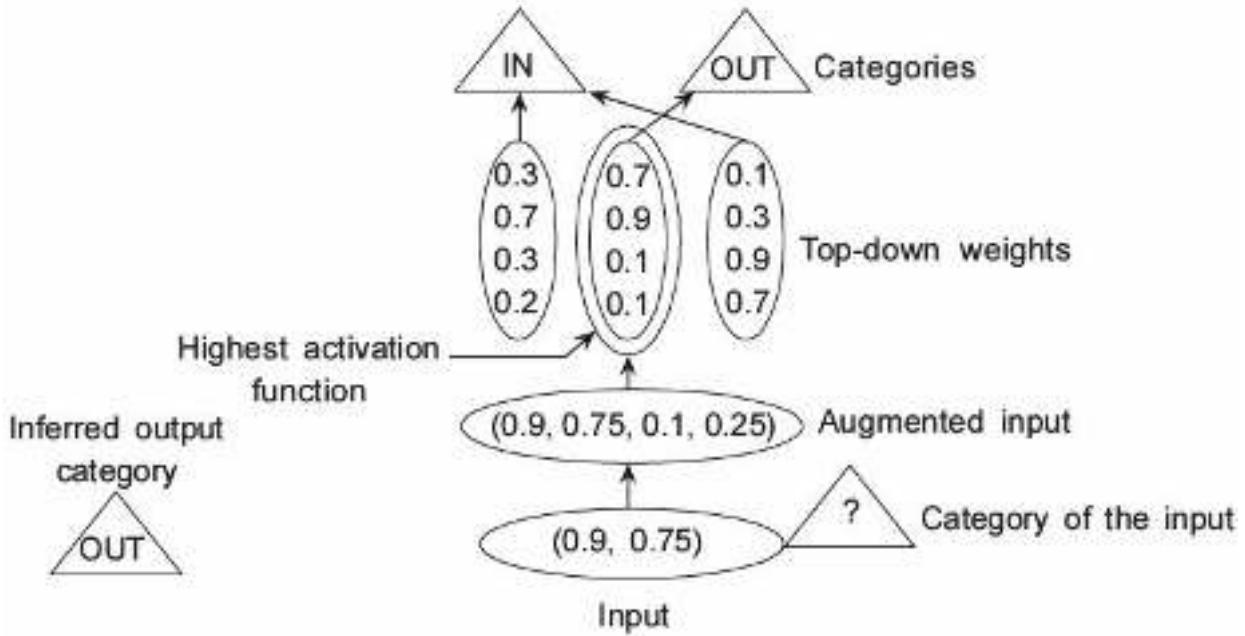


Fig. 13.10 Inference by simplified fuzzy ARTMAP—inferring (0.9, 0.75).

13.4 APPLICATION: IMAGE RECOGNITION

The simplified fuzzy ARTMAP can be applied to the solution of pattern classification/recognition problems. The input vectors to be presented to the network for training as well for inference should comprise components which lie between 0 and 1. The outputs to be associated are the categories/classes to which the inputs belong.

In this section, we discuss the application of the network for the recognition of patterns (images). Rajasekaran and Pai (1999) have experimented with the potential of the simplified fuzzy ARTMAP network to recognize graphical images, both coloured and monochrome.

Table 13.1 illustrates a set of images and the classes to which they belong.

Here, the images have been engraved on a (40×40) grid to facilitate the representation of the image as a

gray level matrix. Thus, for monochrome image as shown in Fig. 13.11(a), the matrix representation would be as in Fig. 13.11(b). Here, a black square

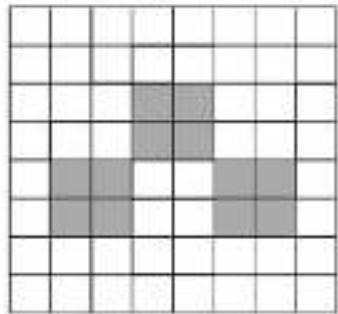
is represented as 1 while a white square is represented as 0, in the image matrix.

In this application, the possibility of a partially shaded square is ruled out for simplicity. In the case of coloured images, a weight value between 0 and 1 is assigned to different colours. Thus, for a coloured pattern (colours represented by different shades) as shown in Fig. 13.2(a), the equivalent image matrix is as shown in

Fig. 13.12(b).

Table 13.1 Sample input patterns

Pattern	Category
	A
	B
	C

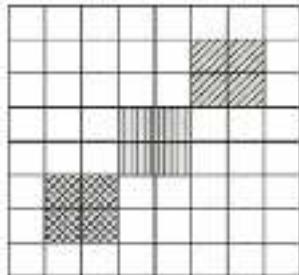


(a) Monochrome image engraved on an (8×8) grid

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b) Equivalent matrix representation

Fig. 13.11 Monochrome image and its matrix representation.



(a) Coloured image engraved on an (8×8) grid

0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	0
0	0	0	.75	.75	0	0	0
0	0	0	.75	.75	0	0	0
0	.5	.5	0	0	0	0	0
0	.5	.5	0	0	0	0	0
0	0	0	0	0	0	0	0

Weight values for the colours

0.5
0.75
1

(b) Equivalent matrix representation

$$|A| = |[a_{ij}]| = \sum_i \sum_j |a_{ij}|$$

Fig. 13.12 Coloured image and its matrix representation.

Having transformed the images into their equivalent matrix representations, the simplified fuzzy ARTMAP architecture could now be extended to work on input matrices whose elements lie between 0 and 1, using the same governing equations of the architecture [Eqs. (13.1)–(13.8)]. The ‘fuzzy \wedge ’

operating on input vectors is now interpreted as the ‘fuzzy \wedge ’ on matrices and the norm ($|\cdot|$) of the vector is treated as the norm of a matrix, i.e.

(13.9)

where $A = [a_{ij}]$

For a direct application of the network to the image recognition problem, the architecture is trained using the matrix equivalents of the training images set. Once the training is over and the top-down weight matrices (in this case) have been obtained, the inference could be carried out by presenting the matrix equivalents of the inference image set. The expected output as before, would be the category to which the image belonged.

However, observations (Rajasekaran et al., 1997) have shown that though the predictions made by the simplified fuzzy ARTMAP architecture are acceptably good for the recognition of patterns which are perfect or exact reproductions of the training set, the architecture is unable to make correct predictions

in

the

case

of

patterns

that

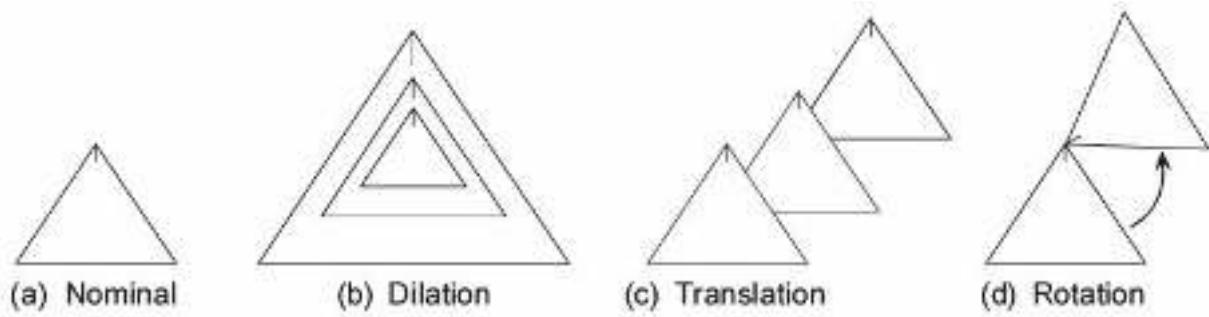
are

perturbed

(e.g.

rotated/scaled/translated or their combinations) or noisy. A solution to this problem is to augment the architecture with a feature extractor to enable the

model exhibit maximum pattern recognition capability. The *feature extractor* extracts feature vectors from the patterns before making their presentation to



$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y) dx dy \quad p,q = 0, 1, 2, \dots, \infty$$

$$m_{pq} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} i^p j^q f(i,j)$$

simplified fuzzy ARTMAP as preprocessed inputs. Digital approximations of moment invariants which have been used to accomplish the extraction of invariant features, has been elaborated in the next section.

13.4.1 Feature Extraction—Moment Based Invariants

The classification of two-dimensional objects from visual image data is an important pattern recognition (PR) task. This task exemplifies many aspects of a typical PR problem, including feature selection, dimensionality reduction, and the use of qualitative descriptors.

Moments are the extracted features derived from raw measurements. In practical imagery, various geometric distortions or pattern perturbations may be observed in the pattern to be classified. Figure 13.13 illustrates some example pattern perturbations. It is therefore essential that features that are invariant to orientations, be used for the classification purpose. For two-dimensional images, moments have been used to achieve Rotation (R), Scaling (S), and Translation (T) invariants.

Fig. 13.13 Pattern perturbations.

Properties of invariance to R, S, T transformations may be derived using function of moments.

The moment transformation of an image function $f(x, y)$ is given by (13.10)

However, in the case of a spatially discretized MXN image denoted by $f(i, j)$, Eq. (13.10) is approximated as

(13.11)

Here, the image function $f(i, j)$ is either 0 or 1 depending on whether the (i ,

$$\sum_{i=0}^M \sum_{j=0}^N (i - \hat{i})^p (j - \hat{j})^q f(i, j)$$

$$\hat{i} = \frac{m_{10}}{m_{00}}, \quad \hat{j} = \frac{m_{01}}{m_{00}}$$

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\left(\frac{p+q}{2} + 1\right)}}, \quad p + q = 2, 3, \dots$$

j)th pixel or its representation is bright or dark for monochrome images.

On the other hand, the intensity is represented by various shades, i.e. $0 \leq f(i, j) \leq 1$ indicating that the intensity lies anywhere between the ends of a spectrum for colour images. However, $f(i, j)$ is constant over any pixel region.

The so called *central moments* are given by

$$\mu_{pq} =$$

(13.12)

where

(13.13)

The central moments are still sensitive to R and S transformation. The scaling invariant may be obtained by further normalizing μ_{pq} as (13.14)

From Eq. (13.14) constraining p, q for $p, q \leq 3$, and using the tools of invariant algebra, a set of seven RST invariant features (as shown in Table 13.2) may be derived (Schalkoff, 1992).

However, though the set of invariant moments shown in Table 13.2 are invariant to Translation, in spite of them being computed discretely, the moments cannot be expected to be strictly invariant under rotation and scaling changes (Schalkoff, 1989).

Table 13.2 Moment based RST invariant features

$$\varphi_1 = \eta_{20} + \eta_{02}$$

$$2$$

$$2$$

$$\varphi_2 = (\eta_{20} + \eta_{02}) + 4\eta_{11}$$

$$2$$

$$2$$

$$\varphi_3 = (\eta_{30} - 3\eta_{12}) + (3\eta_{21} - \eta_{03})$$

$$2$$

$$2$$

$$\varphi_4 = (\eta_{30} + \eta_{12}) + (\eta_{21} + \eta_{03})$$

2

2

2

$$\varphi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12}) - 3(\eta_{21} + \eta_{03})) + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12}) -$$

2

$$(\eta_{21} + \eta_{03}))$$

2

2

$$\varphi_6 = (\eta_{20} - \eta_{02})(\eta_{30} + \eta_{12}) - (\eta_{21} + \eta_{03}) + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})^2$$

2

2

$$\varphi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})(\eta_{30} + \eta_{12}) - 3(\eta_{21} + \eta_{03}) - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12}) - (\eta_{21}$$

2

$$+ \eta_{03}))$$

$$x_i = i - \hat{i}, y_j = j - \hat{j}$$

\hat{i}

\hat{j}

$$\hat{i} = \frac{m_{10}}{m_{00}}, \quad \hat{j} = \frac{m_{01}}{m_{00}}$$

$$\sum_{j=1}^n \sum_{i=1}^n f(i, j) \cdot 1/12$$

Investigations (Rajasekaran and Pai, 1999) reveal that in the definition of μ_{pq} , the contribution made by a pixel has been overlooked. The modified central moments are presented in Table 13.3. The moments listed in the table have been derived for an image engraved on an

$(N \times N)$ grid, without loss of generality. Also, (xi, yj) defined in the table is given by

where , are the centres of mass given by

The moments are still sensitive to R and S transformations. Note that the scaling invariance may be obtained by further normalizing μ_{pq} as given in Eq. (13.14). From Table 13.3, it may be observed that μ_{20} and μ_{02} are different from their conventional definition of central moments. In the conventional definition of μ_{20} , for example, the term

has

been neglected. This omission has resulted in a cascading effect rendering η_{20} and η_{02} and the functions φ_1 , φ_2 , and φ_6 incorrectly defined, leading to the misclassification of images with RST orientations. Using the normalized central moments and tools of invariant algebra, a set of seven RST invariant features same as that shown in Table 13.3 may be derived.

Table 13.3 Revised μ_{pq} definitions

$\mu_{00} = M$ (Mass)	$\mu_{21} = \sum_{j=1}^n \sum_{i=1}^n f(i,j) (x_i^2 y_j)$
$\mu_{10} = 0$	$\mu_{12} = \sum_{j=1}^n \sum_{i=1}^n f(i,j) (x_i y_j^2)$
$\mu_{01} = 0$	$\mu_{11} = \sum_{j=1}^n \sum_{i=1}^n f(i,j) (x_i y_j)$
$\mu_{20} = \sum_{j=1}^n \sum_{i=1}^n f(i,j) \left(x_i^2 + \frac{1}{12} \right)$	$\mu_{30} = \sum_{j=1}^n \sum_{i=1}^n f(i,j) (x_i^3)$
$\mu_{02} = \sum_{j=1}^n \sum_{i=1}^n f(i,j) \left(y_j^2 + \frac{1}{12} \right)$	$\mu_{03} = \sum_{j=1}^n \sum_{i=1}^n f(i,j) (y_j^3)$

13.4.2 Computation of Invariants

In this section, we demonstrate the computation of invariant functions on the three patterns shown in Figs. 13.14(a), (b), and (c). Figure 13.14(a) illustrates a simple nominal pattern engraved on an (8×8) grid. The shaded regions are representative of different colours whose weight values have been provided.

For our computations these weight values are treated as $f(i, j)$.

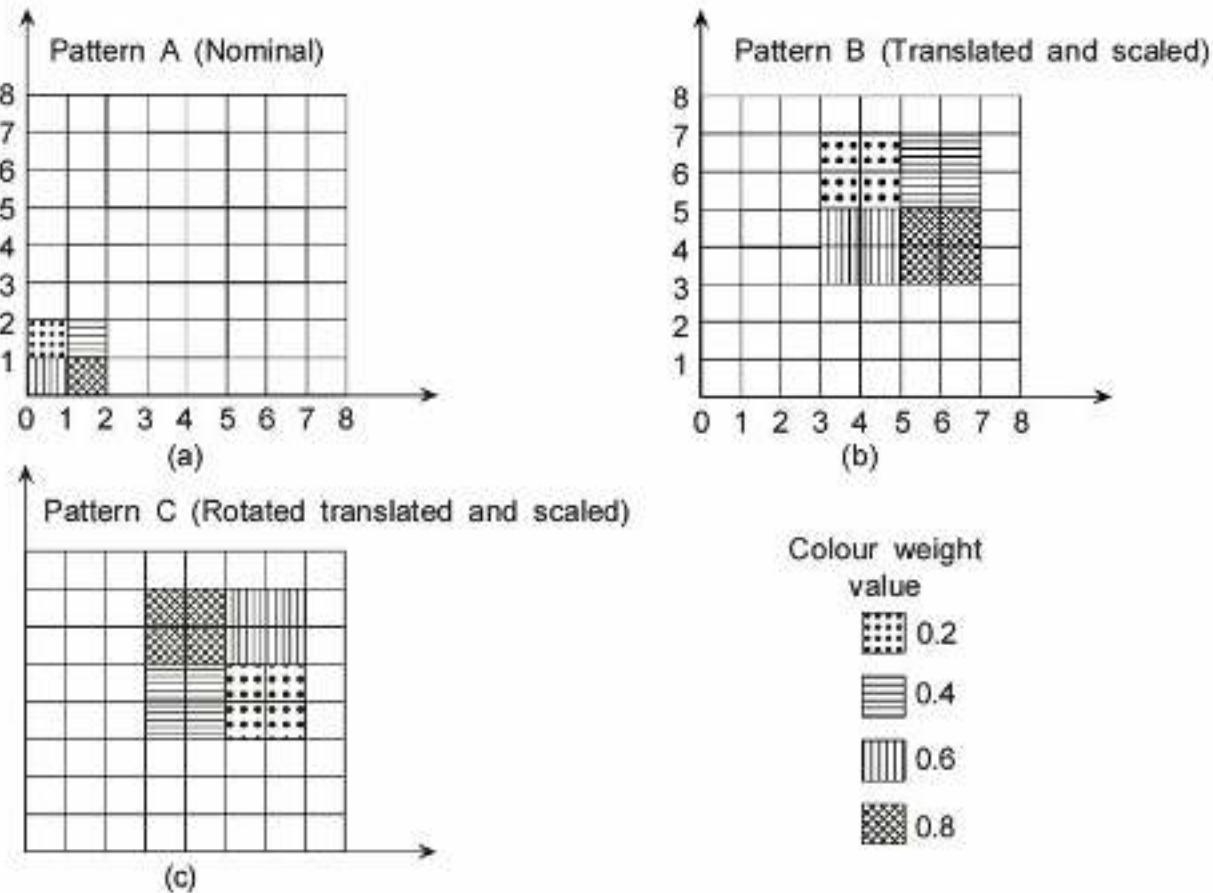
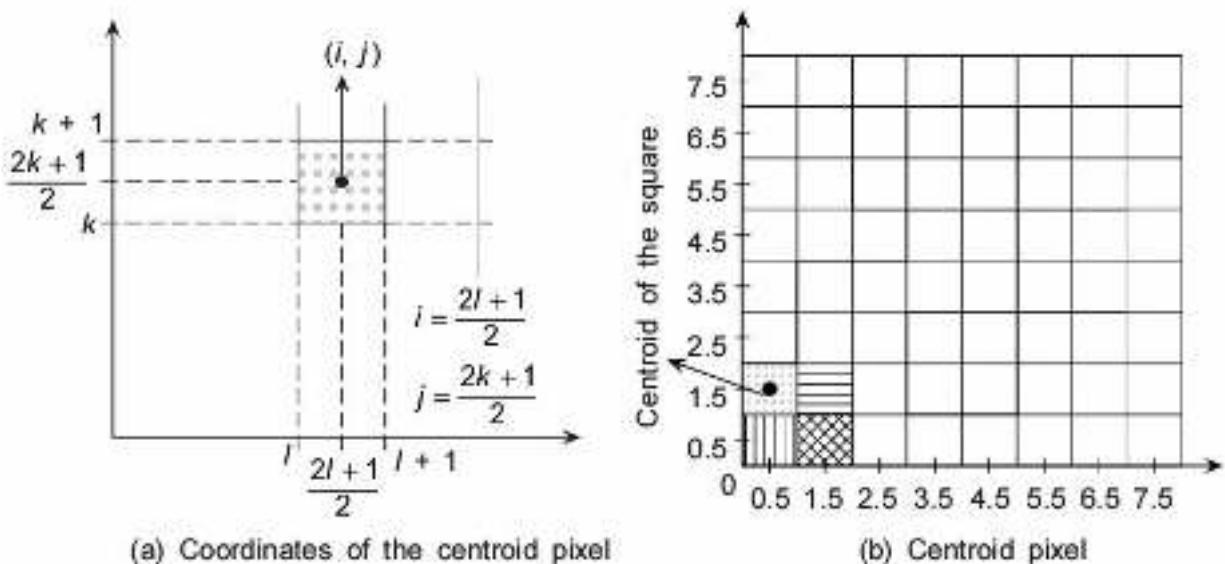


Fig. 13.14 Nominal and perturbed patterns.

It needs to be observed that the pattern engraved in the grid is made up of squared regions of different colours. Each squared region may be made up of a group of pixels as dictated by the grid dimensions. We therefore, for simplicity, discretize the image in such a way that each squared region is represented by the pixel (i, j) which is its centroid (refer Fig. 13.15(a)).

The pixel (i, j), which is the centroid, represents the square region and we assume $f(i, j)$ as a constant over the region. For the (8×8) pattern considered, as shown in Fig. 13.15(b) the index i and j denoting the pixels run from 0.5 to 7.5 in steps of 1.



$$\begin{aligned}
 m_{00} &= \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} i^0 j^0 f(i, j) \\
 &= \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} f(i, j) = f(0.5, 0.5) + f(0.5, 1.5) + f(1.5, 0.5) + f(1.5, 1.5) \\
 &= 0.6 + 0.2 + 0.8 + 0.4 \quad (\because \text{all other } f(i, j) = 0) \\
 &= 2
 \end{aligned}$$

Fig. 13.15 Shift in the coordinate system of a pattern.

Figure 13.14(b) shows a translated and scaled version of the above mentioned nominal pattern of Fig. 13.14(a) and Fig. 13.14(c) represents the rotated, scaled, and translated version of the same. The objective is not only to illustrate the computation of invariant functions of perturbed patterns, but also to show that the invariant functions of the perturbed patterns are the same as that of their nominal versions.

Example 13.8 (*Computation of invariants for pattern A (Fig. 13.14(a))*) We first compute m_{00} , m_{01} and m_{10} as illustrated by Eq. (13.11).

$$\begin{aligned}
m_{10} &= \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} i^1 j^0 f(i, j) \\
&= \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} i f(i, j) \\
&= (0.5 \times 0.6) + (0.5 \times 0.2) + (1.5 \times 0.8) + (1.5 \times 0.4) \\
&= 2.2
\end{aligned}$$

$$m_{01} = \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} j f(i, j)$$

\hat{i}

\hat{j}

$$\hat{i} = \frac{m_{10}}{m_{00}} = 1.1$$

$$\hat{j} = \frac{m_{01}}{m_{00}} = 0.8$$

\hat{i}

\hat{j}

Similarly,

$$\begin{aligned}
&= (0.5 \times 0.6) + (0.5 \times 0.8) + (1.5 \times 0.2) + (1.5 \times 0.4) \\
&= 1.6
\end{aligned}$$

The computation of and , the centroids, yields

Now we proceed to compute μ_{pq} which are the central moments. Here, $xi = i$

– and $yj = j$ – .

Thus,

$$\begin{aligned}
 \mu_{00} &= \sum_{i=-0.5}^{7.5} \sum_{j=-0.5}^{7.5} f(i,j) = 2 \\
 \mu_{01} &= \mu_{10} = 0 \\
 \mu_{11} &= \sum_{i=-0.5}^{7.5} \sum_{j=-0.5}^{7.5} f(i,j) (x_i, y_j) \\
 &= 0.6 \times (-0.6 \times -0.3) + 0.2 \times (-0.6 \times 0.7) + 0.8 \times (0.4 \times -0.3) + 0.4 \times (0.4 \times 0.7) \\
 &\quad - 0.04 \\
 \mu_{20} &= \sum_{i=-0.5}^{7.5} \sum_{j=-0.5}^{7.5} f(i,j) \left(x_i^2 + \frac{1}{12} \right) \\
 &= 0.6 \times \left((-0.6)^2 + \frac{1}{12} \right) + 0.2 \times \left((-0.6)^2 + \frac{1}{12} \right) \\
 &\quad + 0.8 \times \left((0.4)^2 + \frac{1}{12} \right) + 0.4 \times \left((0.4)^2 + \frac{1}{12} \right)
 \end{aligned}$$

$$=0.6467$$

$$\begin{aligned}\mu_{02} = & \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} f(i,j) \left(y_j^2 + \frac{1}{12} \right) \\ & - 0.6 \times \left((-0.3)^2 + \frac{1}{12} \right) + 0.2 \times \left((0.7)^2 + \frac{1}{12} \right) \\ & + 0.8 \times \left((-0.3)^2 + \frac{1}{12} \right) + 0.4 \times \left((0.7)^2 + \frac{1}{12} \right) \\ = & 0.5867\end{aligned}$$

$$\begin{aligned}\mu_{30} = & \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} f(i,j) x_i^3 \\ = & -0.096\end{aligned}$$

$$\begin{aligned}\mu_{03} = & \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} f(i,j) y_j^3 \\ = & 0.168\end{aligned}$$

$$\begin{aligned}\mu_{12} = & \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} f(i,j) (x_i \cdot y_j^2) \\ = & 0.016\end{aligned}$$

$$\begin{aligned}\mu_{21} = & \sum_{i=0.5}^{7.5} \sum_{j=0.5}^{7.5} f(i,j) (x_i^2 \cdot y_j) \\ = & -0.008\end{aligned}$$

$$\eta_{11}=\frac{\mu_{11}}{\mu_{00}^2}=0.01$$

$$\eta_{02}=\frac{\mu_{02}}{\mu_{00}^2}=0.1467$$

$$\eta_{20}=0.1617$$

The computation of η_{pq} using Eq. (13.14) yields

$$\eta_{21} = -0.00141$$

\hat{i}

\hat{j}

\hat{i}

\hat{j}

$$\eta_{12} = 0.0282$$

$$\eta_{30} = 0.0169$$

$$\eta_{03} = 0.0296$$

The invariant function φ_1 – φ_7 as illustrated in Table 13.2 yield $\varphi_1 = 0.3084$

$$\varphi_2 = 0.0006$$

$$\varphi_3 = 0.0114$$

$$\varphi_4 = 0.0009$$

$$\varphi_5 = 0$$

$$\varphi_6 = 0$$

$$\varphi_7 = 0$$

Example 13.9 (*Computation of invariants for pattern B (Fig. 13.14(b))*)

Here, we compute the invariant functions for the pattern illustrated in Fig.

13.14(b).

For the pattern $m_{00} = 8$, $m_{01} = 36.8$ and $m_{10} = 41.6$

The centroids , are = 5.2, = 4.6

The central moments are given by

$$\mu_{00} = 8, \mu_{01} = 0, \mu_{02} = 9.387, \mu_{03} = 5.376$$

$$\mu_{10} = 0, \mu_{11} = 0.64, \mu_{12} = 0.512, \mu_{20} = 10.34$$

$$\mu_{21} = -0.256 \text{ and } \mu_{30} = -3.072$$

The normalized moments η_{pq} yield

$$\eta_{11} = 0.01, \eta_{02} = 0.1467, \eta_{03} = 0.0296$$

$$\eta_{20} = 0.1617, \eta_{12} = 0.00282$$

$$\eta_{30} = -0.01697, \eta_{21} = -0.00141$$

The invariant functions are given by

$$\varphi_1 = 0.3084$$

$$\varphi_2 = 0.0006$$

$$\varphi_3 = 0.0114$$

$$\varphi_4 = 0.0009$$

$$\varphi_5 = 0$$

$$\varphi_6 = 0$$

$$\varphi_7 = 0$$

Example 13.10 (*Computation of invariants for pattern C* (Fig. 13.14(c)) On similar lines as illustrated in Examples 13.8 and 13.9, the invariant functions for the patterns in Fig. 13.14(c) yield

$$\varphi_1 = 0.3084$$

$$\varphi_2 = 0.0006$$

$$\varphi_3 = 0.0114$$

$$\varphi_4 = 0.0009$$

$$\varphi_5 = 0$$

$$\varphi_6 = 0$$

$$\varphi_7 = 0$$

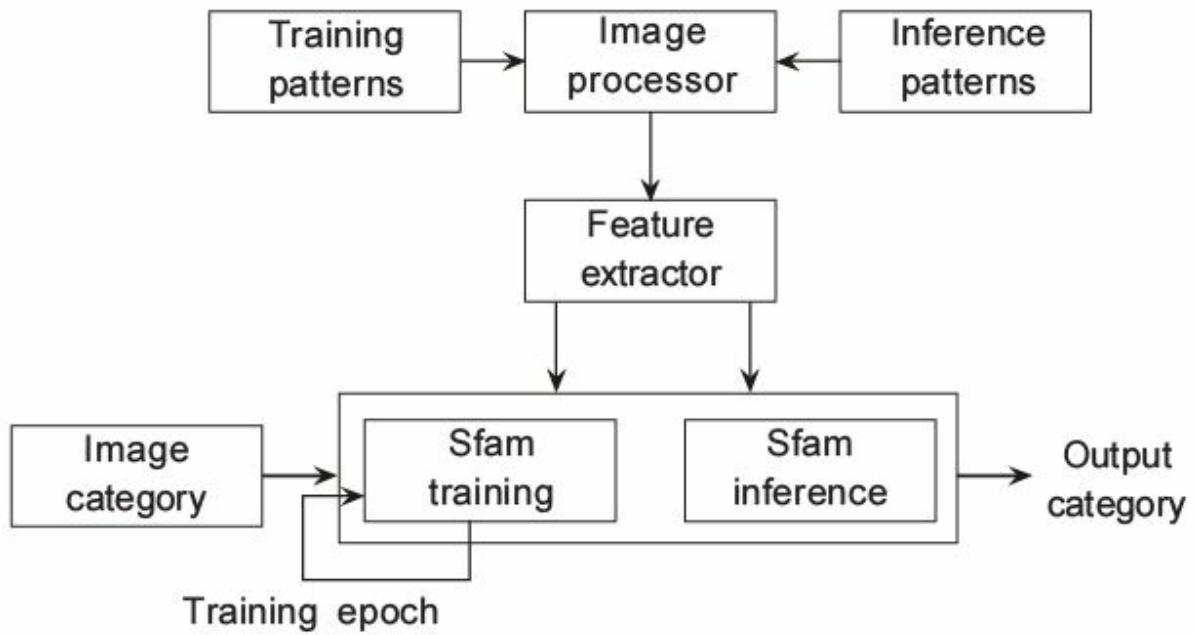
The above three examples suggest that for a pattern and its perturbed versions (rotated, scaled, translated, and their combinations), the invariant functions are the same. This property is what is exploited by a *pattern recognizer* in general and simplified fuzzy ARTMAP in particular in this application. Though the network receives graphical patterns as input, it actually processes only the invariant functions vector ($\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6, \varphi_7$) which is normalized to lie between

0 and 1. The structure of the simplified fuzzy ARTMAP based pattern recognizer is discussed in the following section.

13.4.3 Structure of the Simplified Fuzzy ARTMAP based

Pattern Recognizer

The overall structure of the pattern recognizer is illustrated in Fig. 13.16. The



images (patterns), whether monochrome or coloured, are input through the image processor. In this application, as mentioned earlier, the images are engraved on a (40×40) grid. Also, images can have their colours selected from a fixed palette.

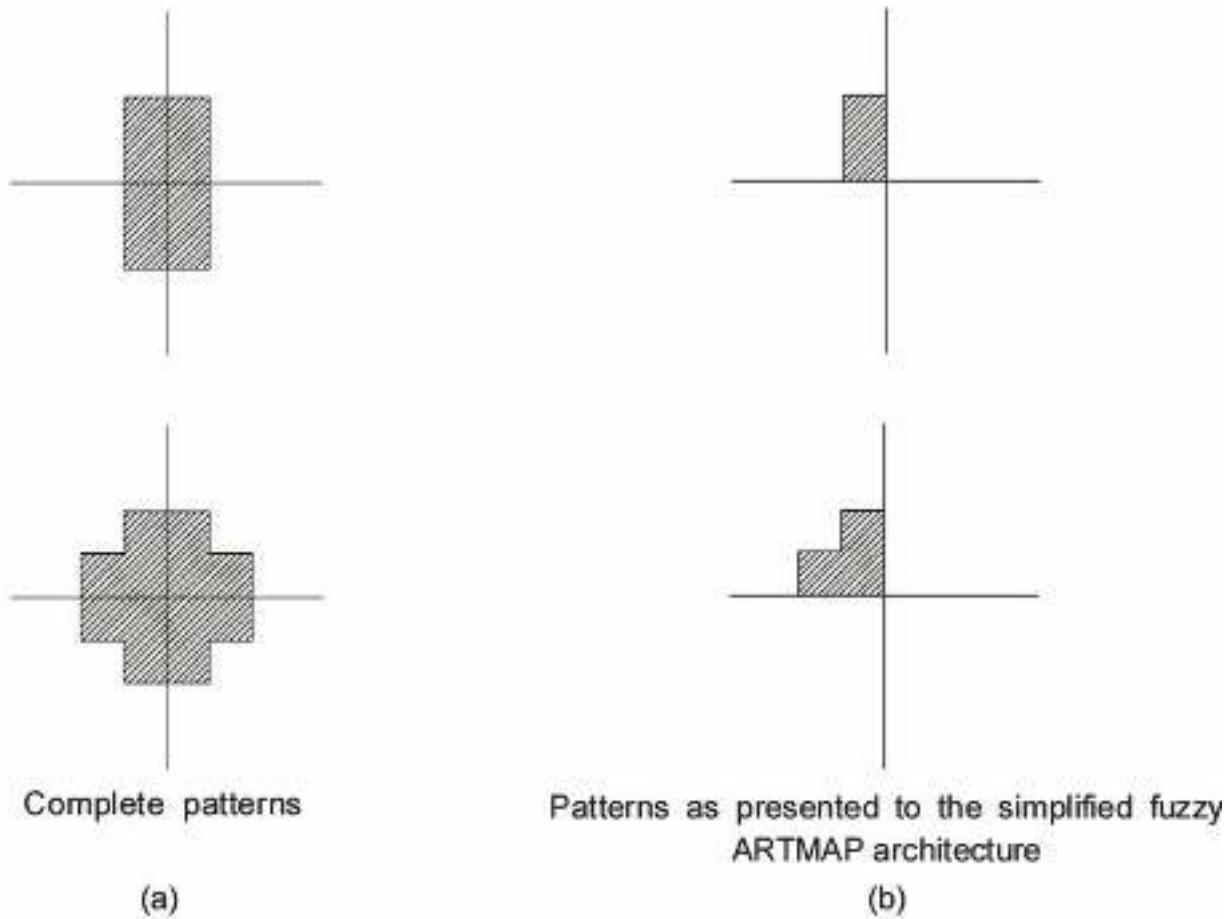
Fig. 13.16 Structure of the simplified fuzzy ARTMAP based pattern recognizer.

The feature extractor obtains the RST invariant features for each image, be it for training or inference. The SFAM (simplified fuzzy ARTMAP) activator functions as two modules, namely the *training module* and the *inference module*. The feature vectors of the training patterns and the categories to which they belong are presented to the SFAM's training module. The only user selectable parameter for the training session is the vigilance parameter ρ , where $0 < \rho < 1$. Once the training is complete, the top-down weight vectors represent the patterns learnt. Next, the feature vectors of the images that are to be recognized/classified are presented to the inference module.

The SFAM

now begins its classification of images by associating the feature vectors with the top-down weight vectors.

The system can handle both symmetric and asymmetric patterns. However, in the case of symmetric patterns, it is essential that only distinct portions of the images be trained. Figure 13.17 illustrates a sample set of doubly symmetric images and their presentation to simplified fuzzy ARTMAP. This is so, since in the case of doubly symmetric or in general, multisymmetric patterns, their RST invariant feature vectors ϕ_4 – ϕ_7 acquire values very close to 0 and ϕ_1 tends to 1. This consequently results in feature vectors, which are almost similar, leading to misclassification of patterns. Hence, in the case of multisymmetric patterns, it is sufficient to consider $(1/2 n)$ th portion of the image.



to 1. This consequently results in feature vectors, which are almost similar, leading to misclassification of patterns. Hence, in the case of multisymmetric patterns, it is sufficient to consider $(1/2 n)$ th portion of the image.

Fig. 13.17 Symmetric patterns and their presentation to simplified fuzzy ARTMAP.

13.4.4 Experimental Study

The simplified fuzzy ARTMAP was trained with nominal patterns of the kind illustrated in

Table 13.1. The performance of the network was observed for varying vigilance parameter values,

$0.5 \leq \rho < 1$. The number of training epochs was kept fixed to a paltry 3. The experiments performed are categorized as

Image

Training set

Testing set

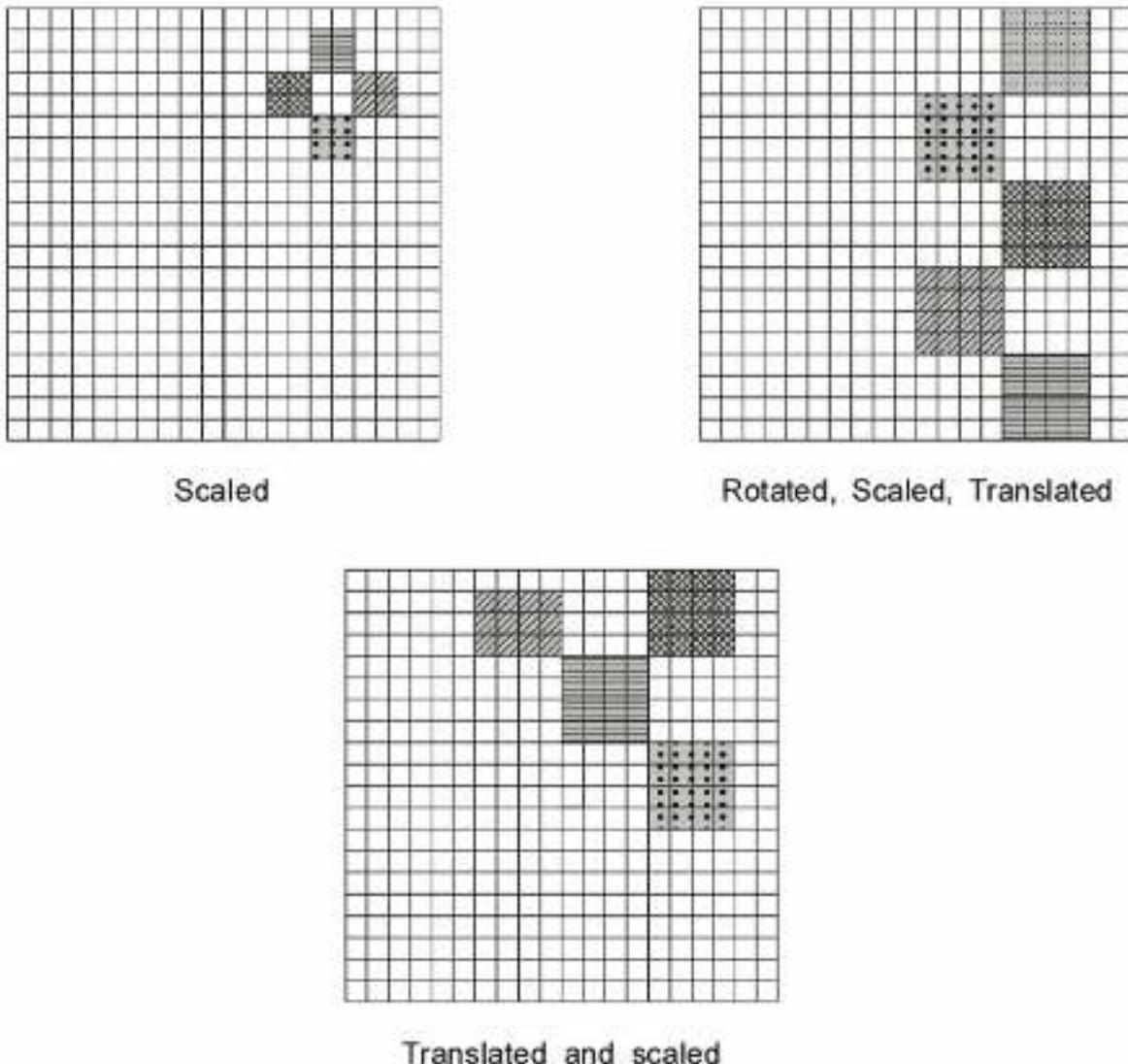
Nominal (noisy) patterns

Coloured

Nominal patterns

Rotated/Scaled/Translated (noise-free)

patterns and their combinations.



Rotated/Scaled/Translated (noisy)

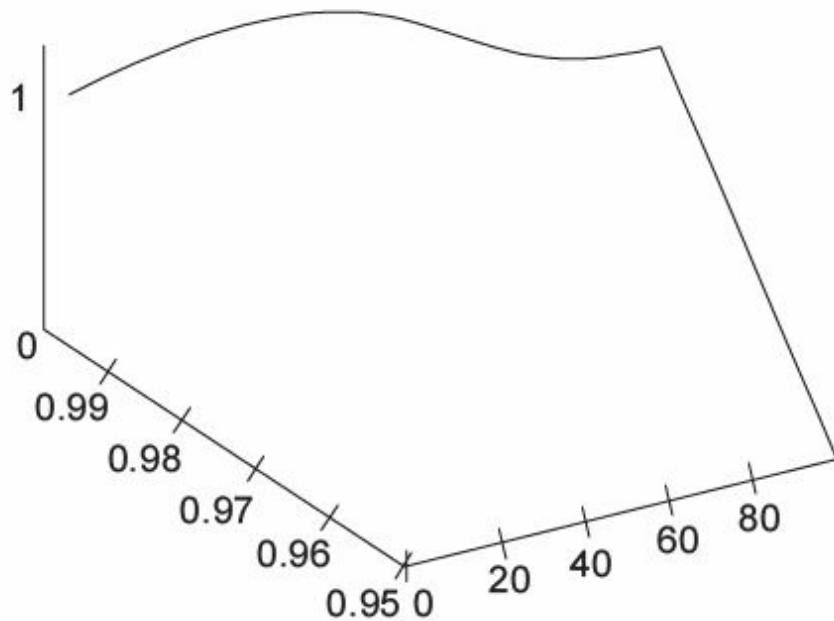
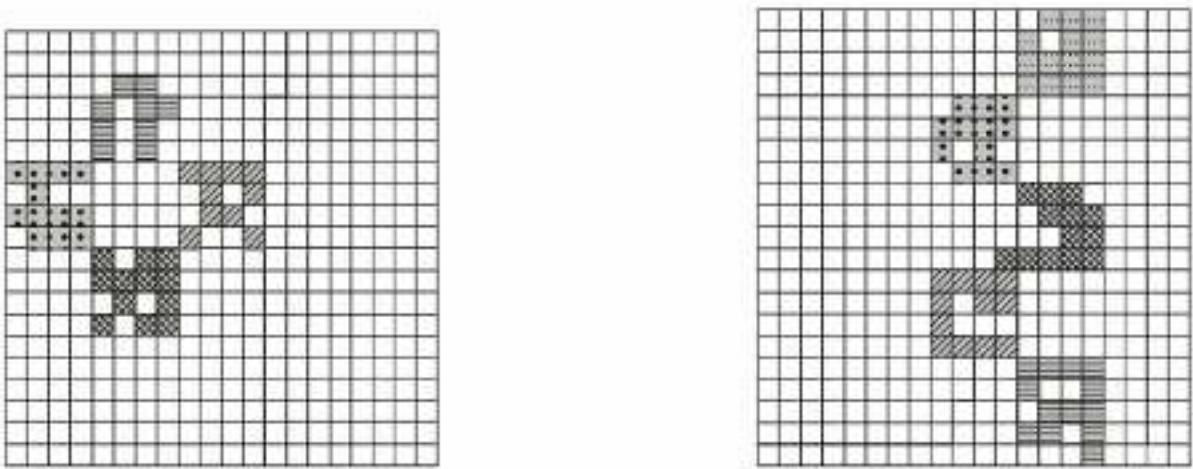
patterns and their combinations.

Figure 13.18 illustrates a set of noise-free patterns but subjected to perturbations—rotation, scaling, translation, and their combinations. Table 13.4 illustrates the results of the experiment.

Fig. 13.18 Noise-free perturbed patterns.

Table 13.4 Recognition of noise free colour images

No. of
Vigilance
training
Training set
Testing set
Nature of the testing set
Recognition rate
parameter
epochs
4 Exemplars
(one exemplar
25
Rotated/Scaled/
3
 $0.5 \leq \rho < 1$
100%
from four
patterns
Translated/Combination
different categories)



In the case of noisy patterns, a sample of which is illustrated in Fig. 13.19, the performance of the model for varying noise levels was observed. The activation value of the top-down vectors during the inference of the noisy image and the recognition capability (0—image unrecognized, 1—image recognized) of the model for the varying noise levels of a given pattern was kept track of. Figure 13.20 illustrates the performance when a sample coloured pattern which is perturbed (Rotated, Scaled and Translated) and subjected to noise, was presented for inference.

Fig. 13.19 Noisy patterns.

Fig. 13.20 Performance of simplified fuzzy ARTMAP during the recognition of noisy and a perturbed pattern.

13.5 RECENT TRENDS

A variation of simplified fuzzy ARTMAP termed *probabilistic simplified fuzzy ARTMAP* has been proposed (Jervis et al., 1999). Simplified fuzzy ARTMAP has also been used for the classification of lithofacies using wireline log data (Wong et al., 1995).

SUMMARY

Adaptive Resonance Theory (ART) architectures are neural networks, which, in response to arbitrary input sequences, self-organize stable recognition codes in real time.

A class of ART architectures such as ART1, ART2, ART3, ART2a have been evolved.

ARTMAP is a class of neural network architectures that performs incremental supervised learning of recognition categories and multidimensional maps in response to input vectors presented in arbitrary order. A general ARTMAP architecture termed fuzzy ARTMAP

classifies inputs by a fuzzy set of features combining fuzzy logic with ART. However, the architecture is complex.

A simplified fuzzy ARTMAP architecture with reduced computational overhead and architectural redundancy has been proposed by Tom Kasuba.

Simplified fuzzy ARTMAP is a two-layer network containing an input and output layer. The input flows through a complement coder. The category layer remembers the various categories that the network has to learn. The vigilance parameter, which is user selectable and match tracking function which adjusts the vigilance parameter, is responsible for network learning. The activation function records the degree of activation of each output node.

When the network learns a category, the corresponding node updates its weights.

Inference is just a feedforward pass where the input which flows through the complement coder associates with the top-down weight nodes to decide the winning node with the highest activation function.

The category associated with that node is the one to which the output belongs.

The working of the simplified fuzzy ARTMAP architecture has been illustrated on the circle-in-the-square problem.

The application of the simplified fuzzy ARTMAP network has been demonstrated on an image recognition problem. The direct application of the network for the problem does not produce acceptable results,

especially when the patterns tested for inference are noisy or perturbed versions of the training set. This handicap could be rectified by augmenting the network with a moment based feature extractor. The conventional invariant functions overlook the contribution made by a pixel and therefore result in incorrect predictions. Rajasekaran and Pai have proposed a modified set of central moments which result in accurate predictions for perturbed patterns.

The structure of the simplified fuzzy ARTMAP based pattern recognizer has been presented. The experimental study carried out on the classification of a set of coloured patterns using the augmented architecture has been discussed.

PROGRAMMING ASSIGNMENT

P13.1 Classification of tiles. A collection of geometrical patterns engraved in a grid of unit length and breadth are to be classified into patterns labelled *A, B, C, D, and E* depending on the number of square tiles covered wholly by the pattern, as illustrated in Table P13.1. Thus, pattern (a) of Fig. P13.1 which covers four whole square tiles is to be classified as Class B and so on.

Table P13.1 Pattern classificationNumber of square tiles covered (X)

Classification label

 $X \leq 2$ A $3 \leq X \leq 5$ B $6 \leq X \leq 8$ C $9 \leq X \leq 11$ D $X \geq 12$ E

.

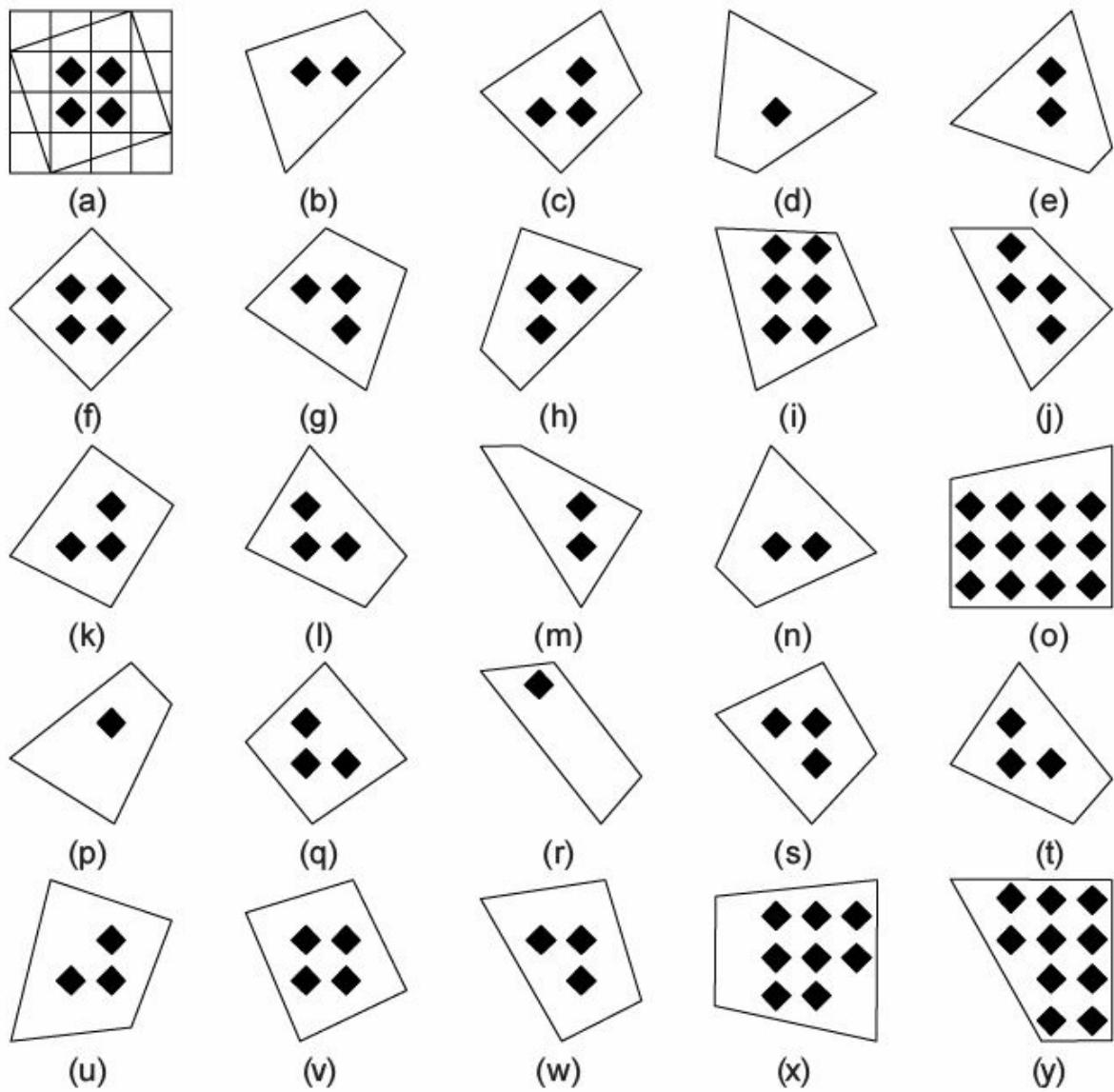


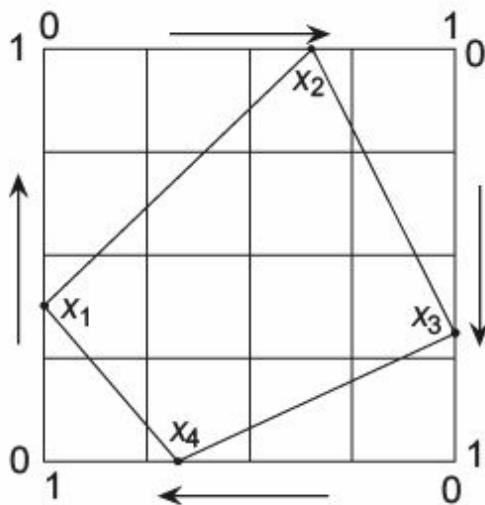
Fig. P13.1 Sample patterns for pattern classification.

A set of pattern PI of various shapes engraved in the grid of the stated size and the corresponding number of tiles subsumed are presented as input to the simplified fuzzy ARTMAP model.

Each of the geometrical pattern is numerically represented as a vector (x_1, x_2, x_3, x_4),

$0 \leq x_i \leq 1$ where x_i represents the corners of the quadrilateral, as the distance from the origin of the appropriate side of the grid. (Refer Fig. P13.2). Table P13.2 illustrates a sample set of inputs.

Another set of patterns, PU , unknown to the model and some of which are distorted (noisy) versions of the patterns belonging to PI are presented for retrieval.



Vector representation: (x_1, x_2, x_3, x_4)

Fig. P13.2 Vector presentation of a geometrical pattern.

Table P13.2 Sample inputs for presentation to simplified fuzzy ARTMAP

Pattern

Actual tiles present in the pattern

Classification

$(0.5, 0.8, 0.2, 0.3)$

2.000

A

(0.35, 0.45, 0.7, 0.8)

3.000

B

(1.0, 0.2, 0.6, 0.25)

0

A

(0.67, 0.67, 0.45, 0.45)

3.000

B

(0.35, 0.5, 0.58, 0.3)

2.000

A

(0.2, 0.45, 0.45, 0.5)

2.000

A

(0.68, 0.8, 0.5, 0.75)

3.000

B

(0.9, 0.9, 0.9, 0.75)

4.000

B

(0.25, 0.2, 0, 0.6)

4.000

B

(1.0, 1.0, 1.0, 0.4)

9.000

D

(0.1, 0, 0, 0.9)

6.000

C

(1.0, 1.0, 0.9, 0.3)

7.000

C

(0.8, 0.95, 0.25, 0.65)

3.000

B

(1.0, 0.9, 0.75, 0.9)

7.000

C

(0.8, 1.0, 0.9, 0.9)

6.000

C

(1.0, 1.0, 0.85, 0.1)

12.000

E

(a) Implement simplified fuzzy ARTMAP architecture.

(b) Prepare a set PI of training pairs comprising pattern vectors and their classifications, and a set PU as a combination of pattern vectors which are both known and unknown to the network.

(c) Observe the behaviour of the network while inferring known patterns

and unknown or noisy versions of the pattern vectors. Comment on the inference capability of simplified fuzzy ARTMAP while processing noisy pattern vectors.

REFERENCES

Carpenter, G.A. and S. Grossberg (1987a), A Massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine, *Computer Vision, Graphics and Image Processing*, 37, pp. 54–115.

Carpenter, G.A. and S. Grossberg (1987b), ART2: Self-organization of Stable Category Recognition Codes for Analog Input Patterns, *Applied Optics*, 26, pp. 4919–4930.

Carpenter, G.A. and S. Grossberg (1990), ART3: Hierarchical Search using Chemical

Transmitters

in

Self-organizing

Pattern

Recognition

Architectures, *Neural Networks*, 3, pp. 129–152.

Carpenter, G.A., S. Grossberg, and J.H. Reynolds (1991), ARTMAP: Supervised Real Time Learning and Classification of Non-stationary Data by a Self-organizing Neural Network, *Neural Networks*, Vol. 4, pp.

565–588.

Carpenter, G.A., S. Grossberg, and David B. Rosen (1991), ART2A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition, *Neural Networks*, Vol. 4,

pp. 493–504.

Carpenter, G.A., S. Grossberg, Natalya Markuzon, J.H. Reynolds, and D.B.

Rosen (1992), Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multi-dimensional Maps, *IEEE Trans on Neural Networks*, Vol. 3, No. 5, 1992,

pp. 698–713.

Grossberg, S. (1976), Adaptive Pattern Classification and Universal Recoding II: Feedback, Expectation, Olfaction and Illusions, *Biological Cybernetics*, 23, pp. 187–202.

Jervis, B.W., T. Garcia, E.P. Giahnakis (1999), Probabilistic Simplified Fuzzy ARTMAP, *IEE Proc.—Science, Measurement and Technology*, Vol. 146, 4, pp. 165–169.

Kasuba, Tom (1993), Simplified Fuzzy ARTMAP, *AI Expert*, November, pp.

18–25.

Rajasekaran, S., G.A. Vijayalakshmi Pai and Jesmon P. George (1997),

Simplified Fuzzy ARTMAP for Determination of Deflection in Slabs of Different Geometry, *Natl Conf on NN and Fuzzy Systems*, pp. 107–116.

Rajasekaran, S. and G.A. Vijayalakshmi Pai (1999), Image Recognition using Simplified Fuzzy ARTMAP Augmented with a Moment based Feature Extractor, *Intl Joul. of Pattern Recognition and Artificial Intelligence*, Vol. 14, No. 8, pp. 1081–1095, 2000.

Rajasekaran, S. and G.A. Vijayalakshmi Pai (2000), Simplified Fuzzy ARTMAP as a Pattern Recognizer, *ASCE Joul. of Computing in Civil Engineering*, Vol. 14, No. 2, pp. 92–99.

Schalkoff, Robert (1989), *Digital Image Processing and Computer Vision*, John Wiley & Sons, Inc.

Schalkoff, Robert (1992), Pattern Recognition—Statistical, Structural and Neural Approaches, John Wiley & Sons.

Wong, P.M., I.J. Taggart, and T.D. Gedeon (1995), The Use of Fuzzy ARTMAP for Lithofacies Classification: a Comparison Study, *Proc. of SPWLA, Thirtysixth Annual Logging Symposium*, Paris.

Chapter 14

Fuzzy Associative Memories

In Chapter 7, we introduced fuzzy rule based systems as a means to encode knowledge of a system in statements of the form,

If {a set of conditions} then {a set of consequents}

In other words, to infer a set of consequents, a set of conditions known as antecedents should be satisfied. Also in Chapter 4, we elaborated on associative memories which are a storehouse of associated patterns. When

the storehouse is incited by an input pattern, the associated pattern pair is recalled.

In this chapter, we discuss fuzzy associative memories (FAMs) which roughly refers to a storehouse of fuzzy associations each of which encodes a fuzzy rule. A FAM, in its elementary form, maps a fuzzy set A to a fuzzy set B and the association is encoded by a fuzzy rule of the form

If X is A then Y is B

where X and Y are fuzzy variables.

A FAM can also map compound associations between fuzzy sets. We first discuss introductory concepts regarding a FAM. Next, the single association FAM, its graphical method of inference, fuzzy Hebb FAMs, and the FAM

system architecture for a rule base are presented. FAM systems for rules with multiple antecedents/consequents and its graphical method of inference are elaborated. Finally, the inference capability of FAMs is demonstrated on two classical problems, namely

1. Balancing an inverted pendulum and
2. Truck backer-upper system.

14.1 FAM—AN INTRODUCTION

A *fuzzy system* maps fuzzy sets to fuzzy sets. Thus, a fuzzy system S is a transformation

$S: In \rightarrow Ip$, where In is the domain space of all fuzzy subsets defined over a universe of discourse $X = \{x_1, x_2, \dots, x_n\}$ and Ip is the range space of fuzzy subsets defined over a universe of discourse $Y = \{y_1, y_2, \dots, y_p\}$.

In general, a fuzzy system S maps families of fuzzy sets to families of fuzzy sets, i.e.

$S : In_1 \times In_2 \times \dots \times In_r \rightarrow Ip_1 \times Ip_2 \times \dots \times Ip_k$

It is on account of this mapping between fuzzy sets to fuzzy sets that fuzzy systems behave like associative memories. Hence, fuzzy systems are referred to as *Fuzzy Associative Memories* (FAMs).

A FAM unlike conventional neural networks, which acquire knowledge through training, directly encodes structured knowledge of the form: *If X is A then Y is B*

where A and B are n -dimensional and p -dimensional fuzzy sets respectively and X, Y are fuzzy variables.

Thus in its simplest form, a FAM encodes a FAM rule/fuzzy rule which associates A with B . Such a FAM is termed a *Single Association FAM* and we represent the association as (A, B) .

A FAM can also represent compound associations encoded by rules of the form

If X is A and Y is B then Z is C

or

If X is A or Y is B then Z is C

or

If X is A then Y is B and Z is C

or

If X is A then Y is B or Z is C

Thus, FAM can represent rules with multiple antecedents/consequents.

A FAM association is also represented using a single linguistic entry matrix termed *FAM bank linguistic matrix*. However, in the case of a single

antecedent and consequent, the association is represented using a one-dimensional table.

Example 14.1

Consider the following FAM rules:

R 1 : If the room temperature is moderately low then turn on the room heater to fairly high.

R 2 : If the pressure is fairly high and temperature very low then adjust the throttle to fairly wide.

Here, *R 1* is a single association FAM rule associating the fuzzy sets,

‘moderately low’ and ‘fairly high’. On the other hand, *R 2* is a compound association FAM rule associating ‘fairly high’ and ‘very low’ with ‘fairly wide’.

14.2 SINGLE ASSOCIATION FAM

A single association FAM associates a single antecedent A_i with a single consequent B_j encoded in a fuzzy rule. In general, a single association FAM

system encodes and processes in parallel, a *FAM bank of m rules* (A_1, B_1), (A_2, B_2), ..., (A_m, B_m).

Example 14.2

Consider the FAM bank comprising the rules

R 1 : If the room temperature is moderately low then turn on the room heater to fairly high

R 2 : If the room temperature is high then turn off the room heater to low.

Here, *R 1* represents the association (moderately low, fairly high) and *R 2*

represents the association (high, low).

Figure 14.1 represents the fuzzy sets associated with R_1 and R_2 . Here we choose the two discrete sets $X = \{10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$

and $Y = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$ as the universes of discourse for the fuzzy sets defined over temperature and heater regulator scale respectively. For simplicity, X is restricted to lie between 10°C and 20°C and Y represents the scale over which the heater regulator can be moved, namely

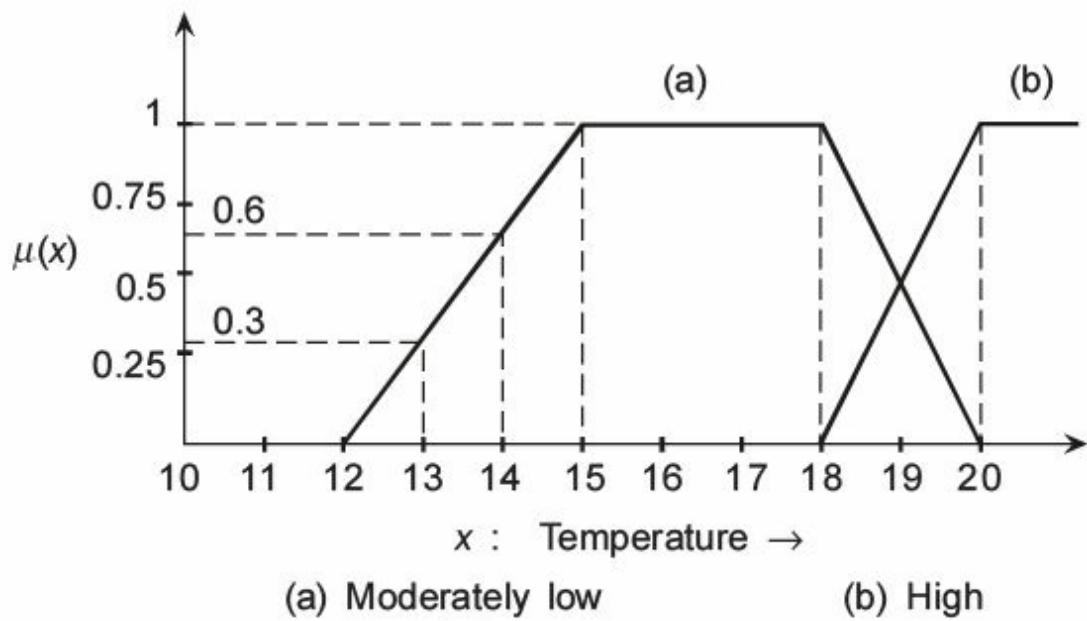
-5 to 5. We assume that when the regulator is positioned over the positive scale, the room temperature rises and when it is positioned over the negative scale, the room temperature falls.

The fuzzy sets, ‘moderately low’ (A_1), ‘fairly high’ (B_1) of R_1 and ‘high’

(A_2), ‘low’ (B_2) of R_2 can be written in their enumerated form as $A_1 = \{(10, 0), (11, 0), (12, 0), (13, 0.3), (14, 0.6), (15, 1), (16, 1), (17, 1), (18, 1), (19, 0.5), (20, 0)\}$

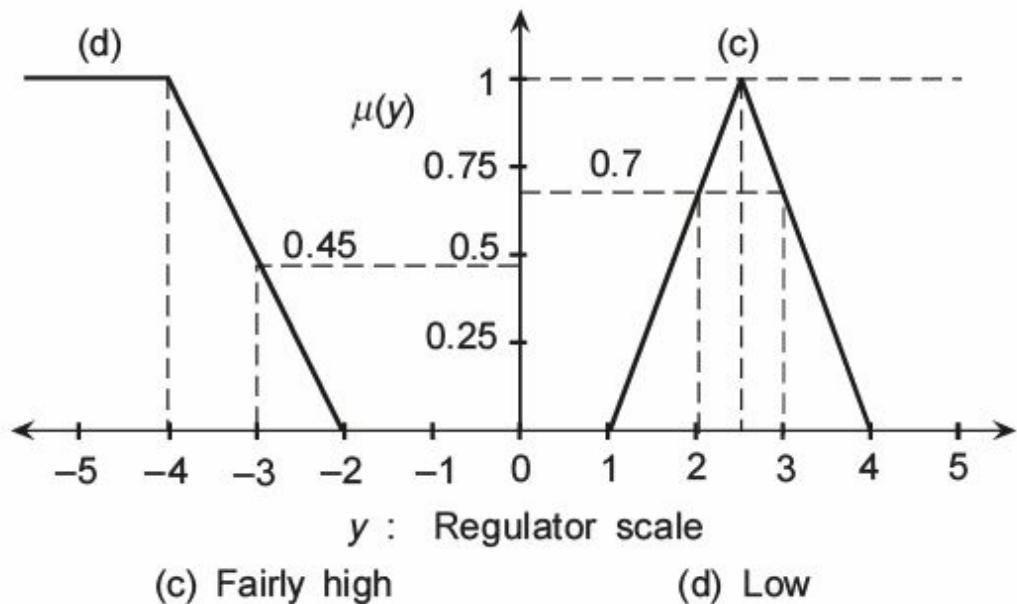
$B_1 = \{(-5, 0), (-4, 0), (-3, 0), (-2, 0), (-1, 0), (0, 0), (1, 0), (2, 0.7), (3, 0.7), (4, 0)\}$

$A_2 = \{(10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 0), (18, 0), (19, 0.5), (20, 1)\}$



(a) Moderately low

(b) High



(c) Fairly high

(d) Low

$$B_2 = \{(-5, 1), (-4, 1), (-3, 0.45), (-2, 0), (-1, 0), (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0)\}$$

Recollect that a fuzzy set A could be defined as $\{(x, \mu A(x)), x \in X\}$ where X

is the universe of discourse and $\mu A(x)$ is the membership value of x in the fuzzy set A . A one-dimensional table representing the FAM bank is shown in

Table 14.1.

Table 14.1 A single association FAM bank

A_i

Moderately low

High

B_i

Fairly high

Low

.

Fig. 14.1 Fuzzy sets defined over temperature and room heater regulator scale.

14.2.1 Graphical Method of Inference

In a FAM system, inference means that given an input A , the system should output the corresponding consequent B as determined by the FAM rule base.

In this, an input A presented to the FAM system activates each of the FAM

rules to varying degrees. Let B_i' be the partially activated versions of B_i for each of the FAM rule. Then the final output B could be taken to be the weighted average of the partially activated set B_i' .

$$\text{i.e. } B = W_1 B_1' + W_2 B_2' + \dots + W_m B_m$$

'

(14.1)

Where W_i indicates the strength of the fuzzy association (A_i, B_i). The more A resembles A_i , the more B_i' resembles B_i . In practice, we prefer to

defuzzify the output B by computing the fuzzy centroid of B with respect to Y , the universe of discourse of the fuzzy set B_i .

In practical applications, the strength of the association W_i is determined using the membership value of the antecedent(s) in the fuzzy rule. The summation in Eq. (14.1) is implemented as ‘fuzzy OR’.

We now illustrate the inference procedure of single association FAM using the FAM system presented in Example 14.2.

Example 14.3

For the FAM system discussed in Example 14.2, let us suppose the input is the current room temperature 19°C . The aim is to infer the turn of the regulator knob over the scale (-5 to 5).

The FAM system has been shown in Table 14.1 and the fuzzy sets involved have been illustrated in Fig. 14.1. Figure 14.2(a) shows the application of the graphical inference procedure to determine the output action. For the input 19°C , rules R_1 and R_2 are both fired with membership values of 0.5 each.

Figure 14.2(b) shows the aggregate of the outputs of the two rules fired.

Now, defuzzification using centroid method yields the output value of -1.4 .

This implies that the output action, to be performed when the room temperature is 19°C , is to turn the regulator knob over the negative scale positioning it at -1.4 .

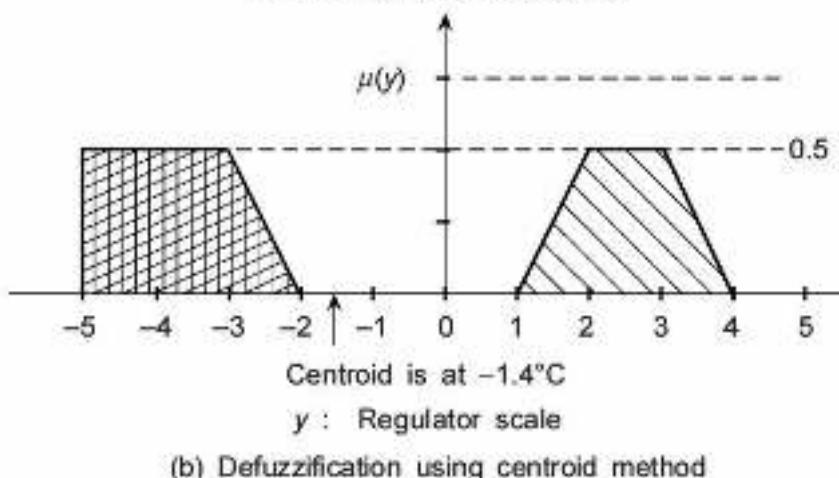
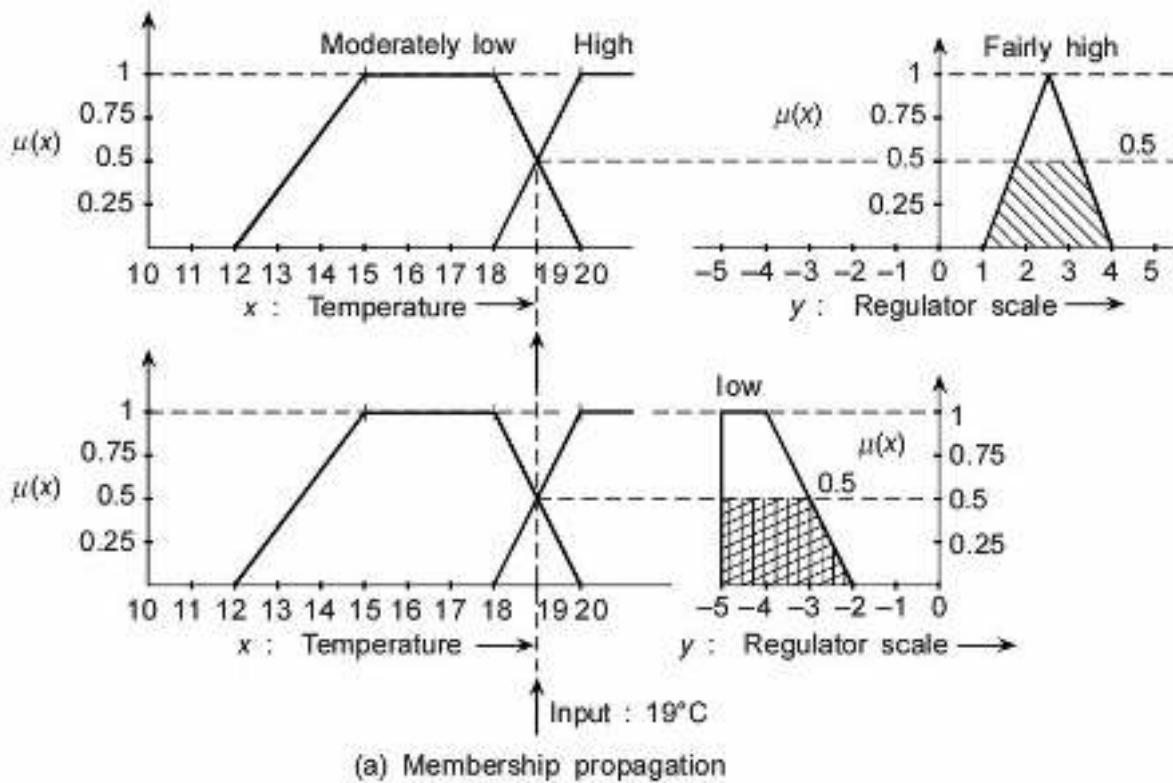


Fig. 14.2 Graphical inference method for single association FAM.

14.2.2 Correlation Matrix Encoding

Let (A, B) be the fuzzy set pair associated by the FAM. Let $X = \{x_1, x_2, \dots, x_n\}$ and

$Y = \{y_1, y_2, \dots, y_p\}$ be the universes of discourse of A and B respectively. We could represent A and B by the numerical fit vectors $A = (a_1, a_2, \dots, a_n)$

) and $B = (b_1, b_2, \dots, b_p)$ where $a_i = \mu_A(x_i)$ and $b_j = \mu_B(y_j)$. Here, μ_A and μ_B are

$$b_j = \max_{1 \leq i \leq n} \min(a_i, m_{ij}) \text{ where } M = [m_{ij}]$$

$$M = \begin{pmatrix} 0.1 & 0.7 & 0.6 \\ 0.6 & 0.5 & 0.5 \\ 0.7 & 0.1 & 0.4 \\ 0 & 0.2 & 0.3 \end{pmatrix}$$

the membership values of x_i and y_j with reference to the fuzzy sets A and B .

Example

The fuzzy sets ‘Moderately low’, ‘High’, ‘Fairly High’ and ‘Low’ in Example 14.2 have been represented using their fit vectors.

‘Moderately low’ : $\{(10, 0), (11, 0), (12, 0), (13, 0.3), (14, 0.6), (15, 1), (16, 1), (17, 1), (18, 1), (19, 0.5), (20, 0)\}$

‘High’ : $\{(10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 0), (18, 0),$

$(19, 0.5), (20, 1)\}$

‘Fairly high’ : $\{(-5, 0), (-4, 0), (-3, 0), (-2, 0), (-1, 0), (0, 0), (1, 0), (2, 0.7), (3, 0.7),$

$(4, 0)\}$

‘Low’ : $\{(-5, 1), (-4, 1), (-3, 0.45), (-2, 0), (-1, 0), (0, 0), (1, 0), (2, 0), (3, 0),$

$(4, 0), (5, 0)\}$

If we are able to frame the *correlation association matrix* [M] $n \times p$ between the fuzzy sets in a suitable manner, then given A the fit vector of the input pattern, it is possible to recall B the associated output vector using $A \circ M = B$

(14.2)

Here, ‘ ’ is the max-min composition relation given by

(14.3)

Example 14.4

If the fit vector for A and the correlation matrix M were given by $A = (0.2, 0.3, 0.7, 1)$

and

Then the recalled fit vector B given component-wise is

$$B = A \circ M = (0.2 \ 0.3 \ 0.7 \ 1) \circ \begin{pmatrix} 0.1 & 0.7 & 0.6 \\ 0.6 & 0.5 & 0.5 \\ 0.7 & 0.1 & 0.4 \\ 0 & 0.2 & 0.3 \end{pmatrix}$$

$$\begin{aligned} b_1 &= \max (\min (0.2, 0.1), \min (0.3, 0.6), \min (0.7, 0.7), \min (1, 0)) \\ &= \max (0.1, 0.3, 0.7, 0) \\ &= 0.7 \end{aligned}$$

$$\begin{aligned} b_2 &= \max (0.2, 0.3, 0.1, 0.2) \\ &= 0.3 \end{aligned}$$

$$\begin{aligned} b_3 &= \max (0.2, 0.3, 0.4, 0.3) \\ &= 0.4 \end{aligned}$$

Hence, the recalled B vector is $(0.7, 0.3, 0.4)$.

If it were possible somehow to encode the association (A, B) in M then the FAM system could exhibit perfect recall in the forward direction.

◦

◦

$$M = A^T \circ B = \begin{pmatrix} 0.2 \\ 0.3 \\ 0.7 \\ 1 \end{pmatrix} \circ (0.7, 0.3, 0.4)$$

$$= \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 \\ 0.7 & 0.3 & 0.4 \\ 0.7 & 0.3 & 0.4 \end{pmatrix}$$

14.3 FUZZY HEBB FAMS

Most fuzzy systems adopted by applications make use of *fuzzy Hebb FAMs*.

Recall that (Chapter 4) for a given pair of bipolar row vectors (X, Y), the associative memory neural networks employ the correlation matrix as the outer product of X and Y , i.e.

$$M = XT Y \quad (14.4)$$

In the case of fuzzy Hebbian matrix, we employ *correlation minimum encoding*. In matrix form, the fuzzy outer product is given by $M = AT B$

where

$$mij = \min (ai, bj) \quad (14.5)$$

An *autoassociative fuzzy Hebb FAM matrix* would encode the pair (A, A) with the fuzzy auto-correlation matrix

$$M = AT A \quad (14.6)$$

Example 14.5

In Example 14.4, we chose an arbitrary M to demonstrate the recall of $B = (0.7, 0.3, 0.4)$, given

$A = (0.2, 0.3, 0.7, 1)$. Now employing Eq. (14.5) for framing M , Now presenting A to recall B' , we obtain

$$\begin{aligned} A \circ M &= (0.2 \ 0.3 \ 0.7 \ 1) \circ \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 \\ 0.7 & 0.3 & 0.4 \\ 0.7 & 0.3 & 0.4 \end{pmatrix} \\ &= (0.7 \ 0.3 \ 0.4) = B \end{aligned}$$

$$\begin{aligned} M \circ B^T &= \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 \\ 0.7 & 0.3 & 0.4 \\ 0.7 & 0.3 & 0.4 \end{pmatrix} \circ \begin{pmatrix} 0.7 \\ 0.3 \\ 0.4 \end{pmatrix} \\ &= (0.2, 0.3, 0.7, 0.7) \\ &\neq A \end{aligned}$$

i.e.

$$M = \begin{pmatrix} a_1 \wedge B \\ a_2 \wedge B \\ \vdots \\ a_n \wedge B \end{pmatrix} = (b_1 \wedge A^T, b_2 \wedge A^T, \dots, b_p \wedge A^T) \quad (14.7)$$

Here,

$$a_i \wedge B = (\min(a_1, b_1), \min(a_1, b_2), \dots, \min(a_1, b_p)) \quad (14.8)$$

and

$$b_1 \wedge A^T = \begin{pmatrix} \min(b_1, a_1) \\ \min(b_1, a_2) \\ \vdots \\ \min(b_1, a_n) \end{pmatrix}$$

But in the reverse direction while attempting to recall A given B , the output is The fuzzy Hebb matrix M illustrates two properties:

Property 1 The i th row of M equals the pairwise minimum of a_i , with the output pattern B . Also, the j th column of M equals the pairwise minimum of b_j with the input pattern A .

It can be seen that if A and B are such that for some $k, l, ak = 1$ or $bl = 1$

then the k th row of M is equal to B and the l th column of M equals A . In general if ak and bl are at least as large as that of every bj or ai respectively then the k th row of matrix M equals B and the l th column of M equals A .

Property 2 If there is any row of M which resembles B then the recall is successful in the forward direction

-
-
-

i.e.

$$M = \begin{pmatrix} 0.2 \\ 0.3 \\ 0.5 \\ 0.6 \end{pmatrix} \circ (0.1, 0.4, 0.5)$$

$$= \begin{pmatrix} 0.1 & 0.2 & 0.2 \\ 0.1 & 0.3 & 0.3 \\ 0.1 & 0.4 & 0.5 \\ 0.1 & 0.4 & 0.5 \end{pmatrix}$$

Now,

$$\begin{aligned} A \circ M &= (0.2, 0.3, 0.5, 0.6) \begin{pmatrix} 0.1 & 0.2 & 0.2 \\ 0.1 & 0.3 & 0.3 \\ 0.1 & 0.4 & 0.5 \\ 0.1 & 0.4 & 0.5 \end{pmatrix} \\ &= (0.1, 0.4, 0.5) = B \end{aligned}$$

$$\begin{aligned} B \circ M^T &= (0.1, 0.4, 0.5) \circ \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.3 & 0.4 & 0.4 \\ 0.2 & 0.3 & 0.5 & 0.5 \end{pmatrix} \\ &= (0.2, 0.3, 0.5, 0.5) \\ &\neq A \end{aligned}$$

i.e.

$$A \circ M = B \quad (14.9a)$$

Also, if there is any column of M which resembles A then the recall is successful in the backward direction

i.e.

$$B \circ M^T = A \quad (14.9b)$$

Example 14.6

Consider the fuzzy sets

$A = (0.2, 0.3, 0.5, 0.6)$ and $B = (0.1, 0.4, 0.5)$ Using Eq. (14.5), $M = ATB$

In the forward direction when A is presented, B is correctly recalled.

However, in the reverse direction when B is presented to recall A , Note, that property 2 is violated in this example. While there is a row in M

$$M = \begin{pmatrix} 0.3 & 0.2 & 0.1 \\ 0.4 & 0.2 & 0.1 \\ 0.5 & 0.2 & 0.1 \\ 0.6 & 0.2 & 0.1 \end{pmatrix}$$

◦

◦

$$h(A) = \max_{1 \leq i \leq n} \mu_A(x_i)$$

◦

◦

which resembles B , there is no column in M which resembles A . Hence, the recall is successful in the forward direction but not in the reverse direction.

Example 14.7

Consider the fuzzy sets

$A = (0.3, 0.4, 0.5, 0.6)$ and $B = (0.6, 0.2, 0.1)$ The correlation matrix M

Forward recall yields

$$A M = (0.6, 0.2, 0.1)$$

$= B$

Reverse recall yields

$$B \text{ } MT = (0.3, 0.4, 0.5, 0.6)$$

$= A.$

Here, with property 2 being completely satisfied, successful recalls are made in both directions.

The accuracy of recall in fuzzy Hebb FAMs using correlation minimum encoding depends on the heights of A and B , i.e. $h(A)$, $h(B)$. The *height* $h(A)$ of a fuzzy set A is the maximum fit value or membership value of A , i.e.

(14.10)

Also, a fuzzy set is normal if $h(A) = 1$. (14.11) i.e. there is atleast one ai with $h(ai) = 1$.

If the fuzzy sets A and B are normal, i.e. $h(A) = h(B) = 1$ then a perfect recall is ensured.

Also, if $A \text{ } M = B' \neq B$ then it holds that $B' \neq B$. Similarly if $B \text{ } MT = A' \neq A$ then $A' \neq A$

i.e. if $A' \neq A$ where $A' = (a'1, a'2, \dots, a'n)$ and $A = (a1, a2, \dots, an)$ then $a'i \leq ai$ for each i and

$$M = \begin{pmatrix} 0.2 & 0.3 & 0.3 \\ 0.2 & 0.4 & 0.4 \\ 0.2 & 0.5 & 0.5 \\ 0.2 & 0.5 & 0.6 \end{pmatrix}$$

o

◦

$$\begin{pmatrix} 0.3 & 0.3 & 0.3 \\ 0.4 & 1 & 0.6 \\ 0.4 & 0.8 & 0.6 \\ 0.4 & 0.7 & 0.6 \end{pmatrix}$$

◦

◦

◦

◦

◦

◦

$a' k < ak$ for atleast one k .

Example 14.8

For the fuzzy sets $A = (0.3, 0.4, 0.5, 0.6)$ and $B = (0.2, 0.5, 0.1)$, $h(A) = 0.6$ and $h(B) = 1$. Therefore, B alone is a normal fuzzy set.

Computing M yields

Attempting to recall B is unsuccessful since

$$A M = (0.2, 0.5, 0.6)$$

$$= B' \neq B$$

This is so since $h(A) \neq h(B)$. Also observe that $B' \neq B$.

Example 14.9

For the two normal fuzzy sets $A = (0.3, 1, 0.8, 0.7)$ and $B = (0.4, 1, 0.6)$ since $h(A) = h(B) = 1$, we must have successful recalls in both directions. Here, $M = ATB =$

Recalling B results in

$$A M = (0.4, 1, 0.6) = B$$

and recalling A results in

$$B AT = (0.3, 1, 0.8, 0.7) = A.$$

The results of the *correlation minimum bidirectional FAM theorem* are: If $M = ATB$ then

(i) $A M = B$ iff $h(A) \geq h(B)$ (ii) $B MT = A$ iff $h(B) \geq h(A)$ (iii) $A' M \neq B$ for any A'

(iv) $B' MT \neq B$ for any B' (14.12) Correlation product
encoding provides an alternative fuzzy Hebbian

$$\begin{aligned} M &= A^T B && (14.13) \\ \text{i.e. } m_{ij} &= (a_i, b_j) \\ \text{Thus, } M &= \begin{pmatrix} a_1 B \\ a_2 B \\ \vdots \\ a_n B \end{pmatrix} = (b_1 A^T, b_2 A^T, \dots, b_m A^T) \end{aligned}$$

$$\begin{aligned}
M = A^T B &= \begin{pmatrix} 0.2 \\ 0.3 \\ 0.7 \\ 1 \end{pmatrix} (0.7, 0.3, 0.4) \\
&= \begin{pmatrix} 0.14 & 0.06 & 0.08 \\ 0.21 & 0.09 & 0.12 \\ 0.49 & 0.21 & 0.28 \\ 0.7 & 0.3 & 0.4 \end{pmatrix}
\end{aligned}$$

(i) $A \circ M = B$ iff $h(A) = 1$
(ii) $B \circ M^T = A$ iff $h(B) = 1$
(iii) $A' \circ M \subset B$ for any A'
(iv) $B' \circ M^T \subset A$ for any B'
(14.14)

encoding scheme. In this scheme, we frame M the correlation matrix as

Example 14.10

Let $A = (0.2, 0.3, 0.7, 1)$ and $B = (0.7, 0.3, 0.4)$, then the matrix M encoding the FAM rule (A, B) using correlation product encoding is given by The following results are supported by the correlation product bidirectional FAM theorem

If $M = ATB$ and A, B are non null fit vectors then,

$$\max_{1 \leq k \leq N} M_k,$$

$$\begin{aligned}
A_1 &= (0.3, 0.2, 0.1), & B_1 &= (0.2, 0.1) \\
A_2 &= (0.4, 0.3, 0.7), & B_2 &= (0.6, 0.7) \\
A_3 &= (0.7, 0.4, 0.1), & B_3 &= (0.5, 0.3)
\end{aligned}$$

$$M_1 = \begin{pmatrix} 0.2 & 0.1 \\ 0.2 & 0.1 \\ 0.1 & 0.1 \end{pmatrix} \quad M_2 = \begin{pmatrix} 0.4 & 0.4 \\ 0.3 & 0.3 \\ 0.6 & 0.7 \end{pmatrix} \quad M_3 = \begin{pmatrix} 0.5 & 0.3 \\ 0.4 & 0.3 \\ 0.1 & 0.1 \end{pmatrix}$$

Now,

$$M = \max(M_1, M_2, M_3)$$

$$= \begin{pmatrix} 0.5 & 0.4 \\ 0.4 & 0.3 \\ 0.6 & 0.7 \end{pmatrix}$$

14.4 FAM INVOLVING A RULE BASE

Let us suppose there are N FAM rules $(A_1, B_1), (A_2, B_2), \dots, (A_N, B_N)$, in other words, a *rule base*. Making use of the fuzzy Hebb encoding scheme mentioned in Eq. (14.5) we obtain N FAM matrices $M_i, i = 1, 2, \dots, N$. Now if we were to frame a single correlation matrix M by superimposing N different matrices using the scheme

$$M =$$

(14.15)

the superimposition fails to recall the vectors correctly since M is the matrix $AT^T B$ where A, B are the pointwise maximum of the respective N fit vectors A_k, B_k .

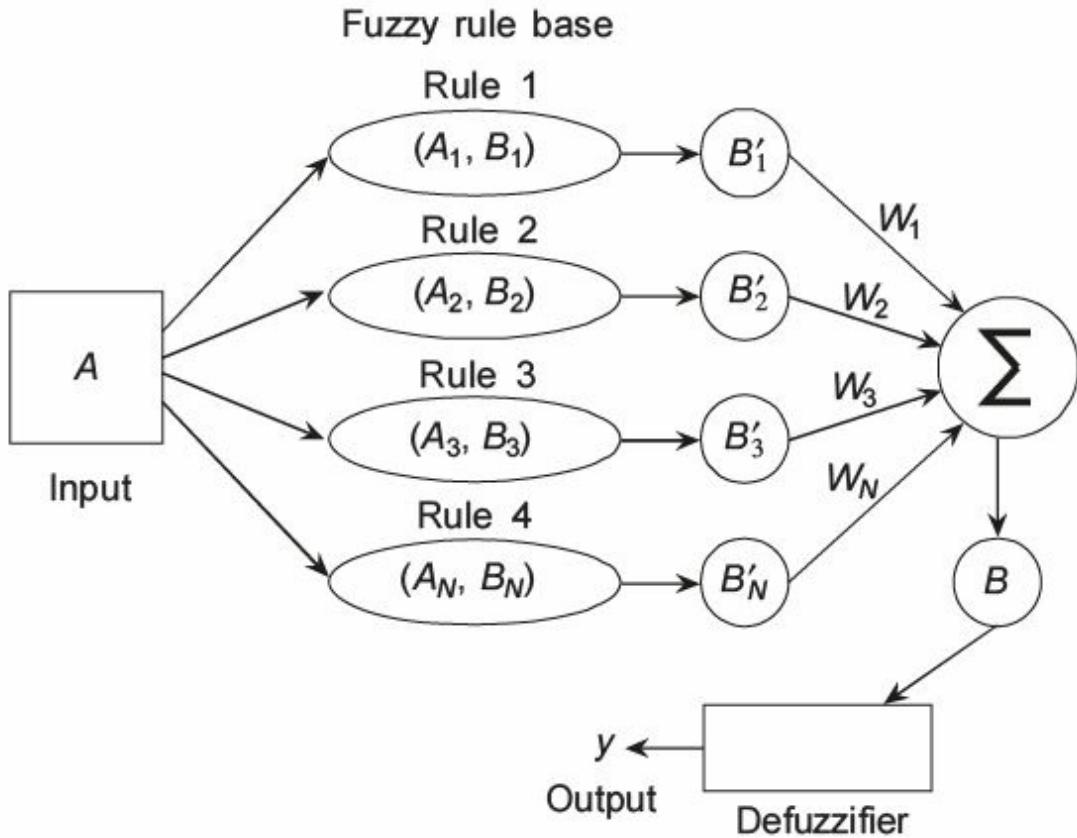
Example 14.11

Consider the three FAM rules $(A_i, B_i), i = 1, 2, 3$ as Computing M_1, M_2, M_3 using the Eq. (14.5) results in Let us submit A_1, A_3 to recall B_1, B_3

$$A_1 \circ M = (0.3, 0.3) \neq B_1$$

$$A_3 \circ M = (0.5, 0.4) \neq B_3$$

$$\sum_{k=1}^N w_k B'_k$$



The fuzzy solution approach is therefore to associate the input vector A with each of the N FAM matrices M_i and additively superimpose the N

recalled vectors B'_i $i = 1, 2, \dots, N$. Thus, here the recalled vectors are superimposed rather than the N matrices M_i . The input vector A activates N different rules in parallel but to varying degrees. Thus, in the case of A partially activating A_i , the output B_i , will also only partially resemble B'_i .

The recalled vector B equals the weighted sum of the individual recalled vectors B'_i k . i.e.

$$B =$$

(14.16)

where the weights w_k indicate the credibility or the strength of the k th FAM rule (A_k, B_k). In most practical applications we choose $w_1 = w_2 = \dots$

$wN = 1$. The recalled vector B is the normalized sum of the fit vectors $B' k$.

Now we defuzzify the output B to result in a single crisp value of the output universe of discourse $Y = \{y_1, y_2, \dots, y_p\}$. Any of the defuzzification schemes could be employed. Recollect that Example 14.3 demonstrated this procedure for a rule base with two FAM rules. A general FAM system architecture for a rule base is illustrated in Fig. 14.3.

Fig. 14.3 FAM system architecture for a rule base.

If X_1 is A_1 and X_2 is $A_2 \dots X_l$ is A_l

then Y_1 is B_1 and Y_2 is $B_2 \dots Y_p$ is B_p

or

If X_1 is A_1 or X_2 is A_2 or ... X_l is A_l

then Y_1 is B_1 or Y_2 is $B_2 \dots Y_p$ is B_p

or

If X_1 is A_1 or X_2 is A_2 or ... X_l is A_l

then Y_1 is B_1 and Y_2 is $B_2 \dots Y_p$ is B_p

14.5

FAM

RULES

WITH

MULTIPLE

ANTECEDENTS/CONSEQUENTS

FAM systems which encode single associations can be extended to encode rules with multiple antecedents and consequents.

Thus, rules of the type

and so on, can be encoded in a FAM.

For instance, consider the rule *if X 1 is A 1 and X 2 is A 2 then Y is B*. This rule denotes an association ($A_1, A_2; B$) where A_1, A_2 are the antecedent fuzzy sets and B is the consequent fuzzy set. The universes of discourse for the fuzzy variables X_i and Y are defined beforehand. Infact, each of the universes of discourse could comprise a group of fuzzy set values $A_1, A_2, A_3, \dots, A_k$.

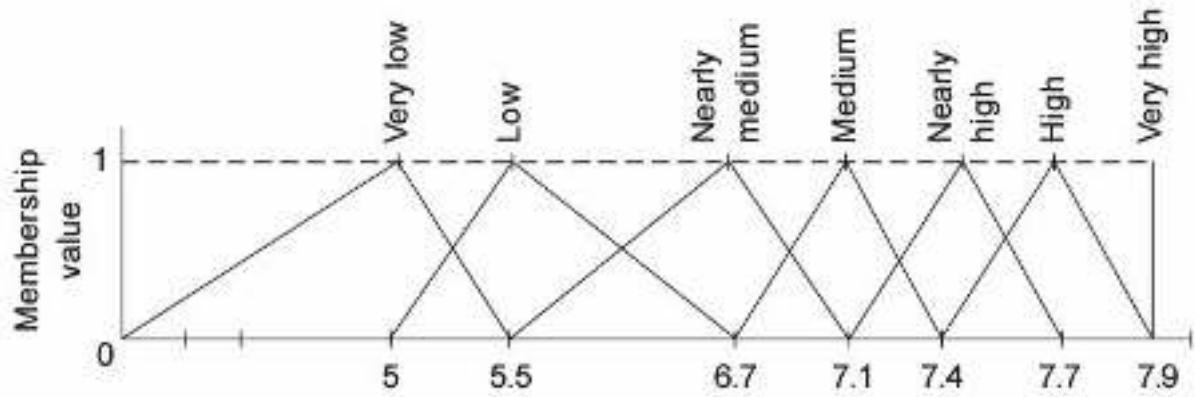
Similarly, a different group of fuzzy set values could define the universe of discourse of the output fuzzy set.

Example 14.12

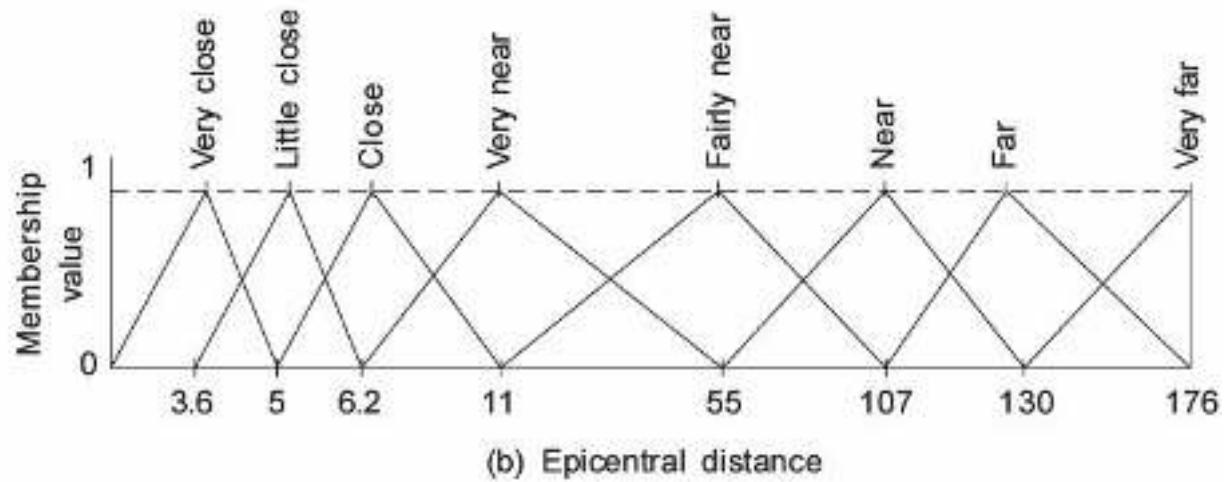
Consider the following multi-antecedent FAM rule.

If the earthquake magnitude is very low and the epicentral distance is very close and the peak ground acceleration/spectral intensity is very high then the damage magnitude is low.

Here, earthquake magnitude (X_1), epicentral distance (X_2), peak ground acceleration/spectral intensity (X_3) and damage membership (Y) are the fuzzy set variables. The fuzzy sets defined are illustrated in Fig. 14.4.



(a) Earthquake magnitude



(b) Epicentral distance

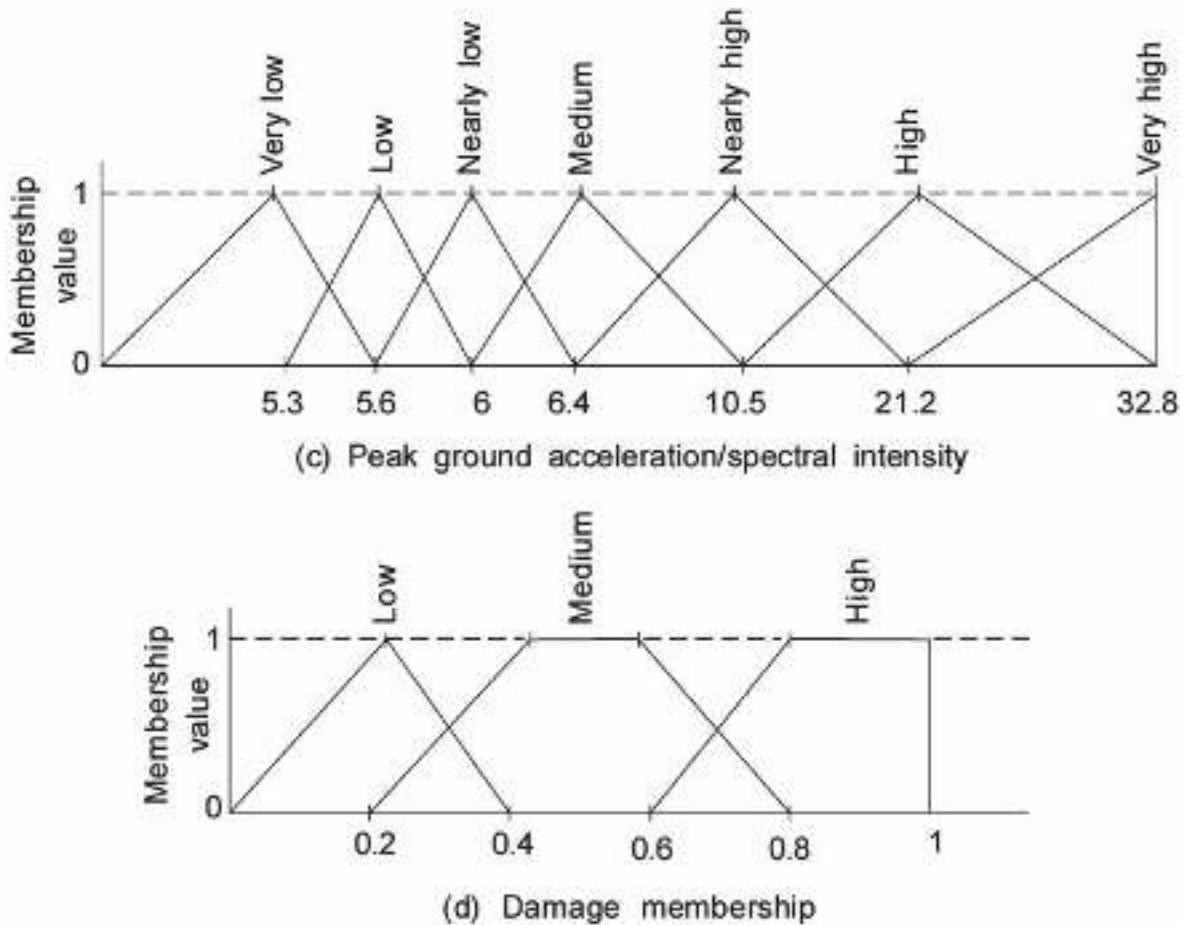


Fig. 14.4 Fuzzy sets for the rule defined in Example 14.12.

Observe that the respective universes of discourse for the fuzzy variable X_1 , X_2 , X_3 and Y comprise the library of fuzzy sets as follows: *Earthquake magnitude*

{Very low, low, nearly medium, medium, nearly high, high, very high}

Epicentral distance

{Very close, little close, close, very near, fairly near, near, far, very far}

Peak ground acceleration/spectral intensity

{Very low, low, nearly low, medium, nearly high, high, very high}

Damage membership

{Low, medium, high}

A FAM rule base with two antecedents and a single consequent is defined here as:

If X is A_i and Y is B_j then Z is C_k $i = 1, 2, \dots, p; j = 1, 2, \dots, q; k = 1, 2, \dots, r$

	B_1	B_2	B_3	...	B_j	...	B_q
A_1							C_2
A_2		C_l					
A_3							
:							
A_i						C_k	
:							
A_p		C_1					

This can be represented conveniently using a matrix representation as shown in Fig. 14.5.

Fig. 14.5 FAM matrix representation.

In the case of multiantecedent rules, Mamdani (1977) and other investigators have suggested using multidimensional matrices for the FAM

representation. But for most practical applications multidimensional matrices are inconvenient. Instead, technologists find the graphical inference method elegant in usage. However in this case, it is necessary that compound associations of the FAM rule base are decomposed into simpler associations.

14.5.1 Decomposition Rules

The decomposition of compound associations in FAM rules involving multiple antecedents/consequents, into its simpler constituents is enumerated

as:

(a) Compound rule

If X is A then Z is C or W is D

Decomposed rule

If X is A then Z is C

If X is A then W is D

(b) Compound rule

If X is A then Z is C and W is D

Decomposed Rule

If X is A then Z is C

If X is A then W is D

(c) Compound rule

If X is A or Y is B then Z is C

Decomposed rule

If X is A then Z is C

If Y is B then Z is C

(d) Compound rule

If X is A and Y is B then Z is C

Decomposition of the rule in this case is not possible.

It may be verified that the rules of decomposition are based on the laws of propositional calculus (refer Chapter 7).

Example 14.13

Consider the FAM rule base

$R\ 1 : \text{If } X \text{ is } A\ 1 \text{ and } Y \text{ is } B\ 1 \text{ then } Z \text{ is } C\ 1$

$R\ 2 : \text{If } X \text{ is } A\ 2 \text{ and } Y \text{ is } B\ 2 \text{ then } Z \text{ is } C\ 2 \text{ or } Z \text{ is } C\ 3$

$R\ 3 : \text{If } X \text{ is } A\ 3 \text{ or } Y \text{ is } B\ 3 \text{ then } Z \text{ is } C\ 3$

We proceed to demonstrate the applications of the graphical inference method on the above rule base. Decomposing the rule base, we get $R'1 : \text{If } X \text{ is } A\ 1 \text{ and } Y \text{ is } B\ 1 \text{ then } Z \text{ is } C\ 1$

$R'2 : \text{If } X \text{ is } A\ 2 \text{ and } Y \text{ is } B\ 2 \text{ then } Z \text{ is } C\ 2$

$R'3 : \text{If } X \text{ is } A\ 2 \text{ and } Y \text{ is } B\ 2 \text{ then } Z \text{ is } C\ 3$

$R'4 : \text{If } X \text{ is } A\ 3 \text{ then } Z \text{ is } C\ 3$

$R'5 : \text{If } Y \text{ is } B\ 3 \text{ then } Z \text{ is } C\ 3$

Let a, b be the inputs to the FAM rule base. Figure 14.6(a) illustrates the propagation of membership functions when a, b are presented. Note that inputs a, b fire rules R'

2

1, $R'2$ and $R'3$ to a degree $\min(m\ 11, m\ 12), \min(m\ 1, m\ 2$

3

3

) and $\min(m\ 1, m\ 2)$ respectively. The aggregate of the outputs and the subsequent defuzzification using centroid method have been shown in Fig.

14.6(b). The final crisp output is shown as z .

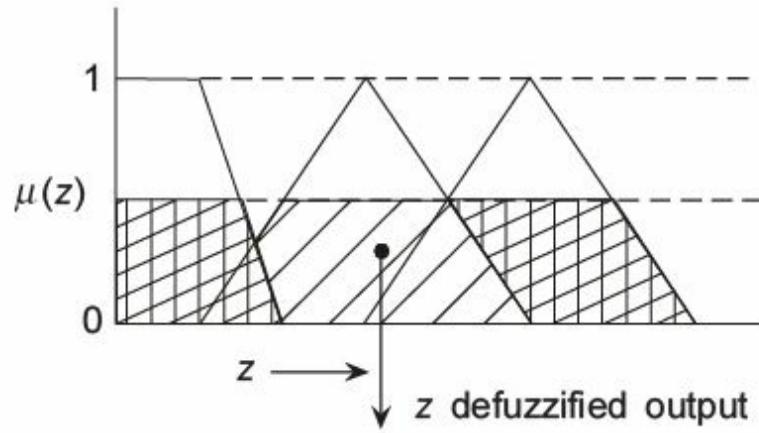
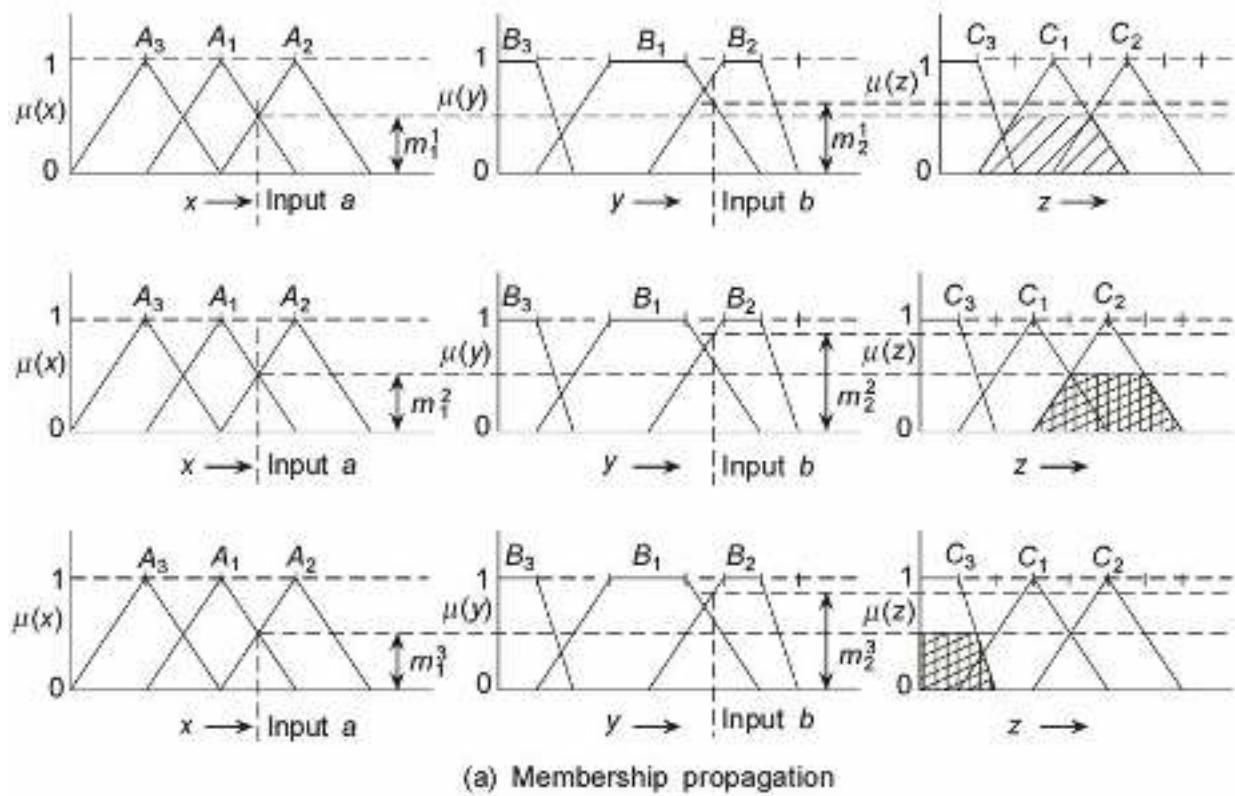
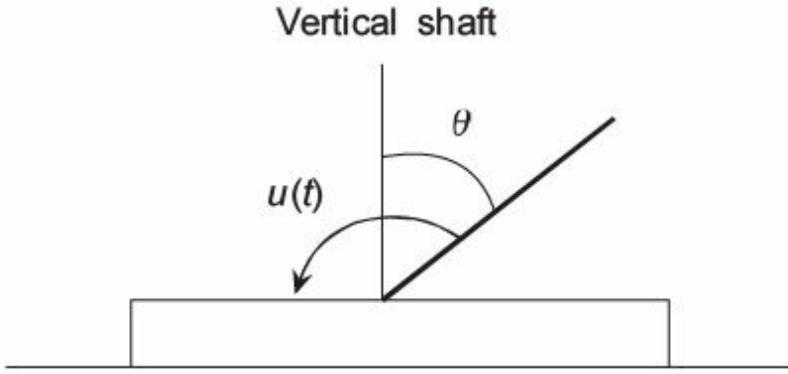


Fig. 14.6 Graphical inference method on multiantecedent FAM rules.



$$\begin{aligned}x_1^{k+1} &= x_1^k + x_2^k \\x_2^{k+1} &= x_1^k + x_2^k - u^k\end{aligned}\quad (14.17)$$

14.6 APPLICATIONS

In the section, we discuss two classical problems to demonstrate the application of FAM and its graphical method of inference.

14.6.1 Balancing an Inverted Pendulum

The problem of Inverted Pendulum is a classical fuzzy control problem. Here, the problem is to adjust a motor to balance the inverted pendulum in two dimensions. The problem comprises two fuzzy state variables, namely angle θ (fuzzy state variable x_1) which the pendulum shaft makes with the vertical and the angular velocity $\Delta\theta$ (fuzzy state variable x_2). Here, if $\theta = 0$ then the shaft is in the vertical position. θ is positive if the shaft is to the right of the vertical and negative if it is to the left. The instantaneous angular velocity $\Delta\theta$

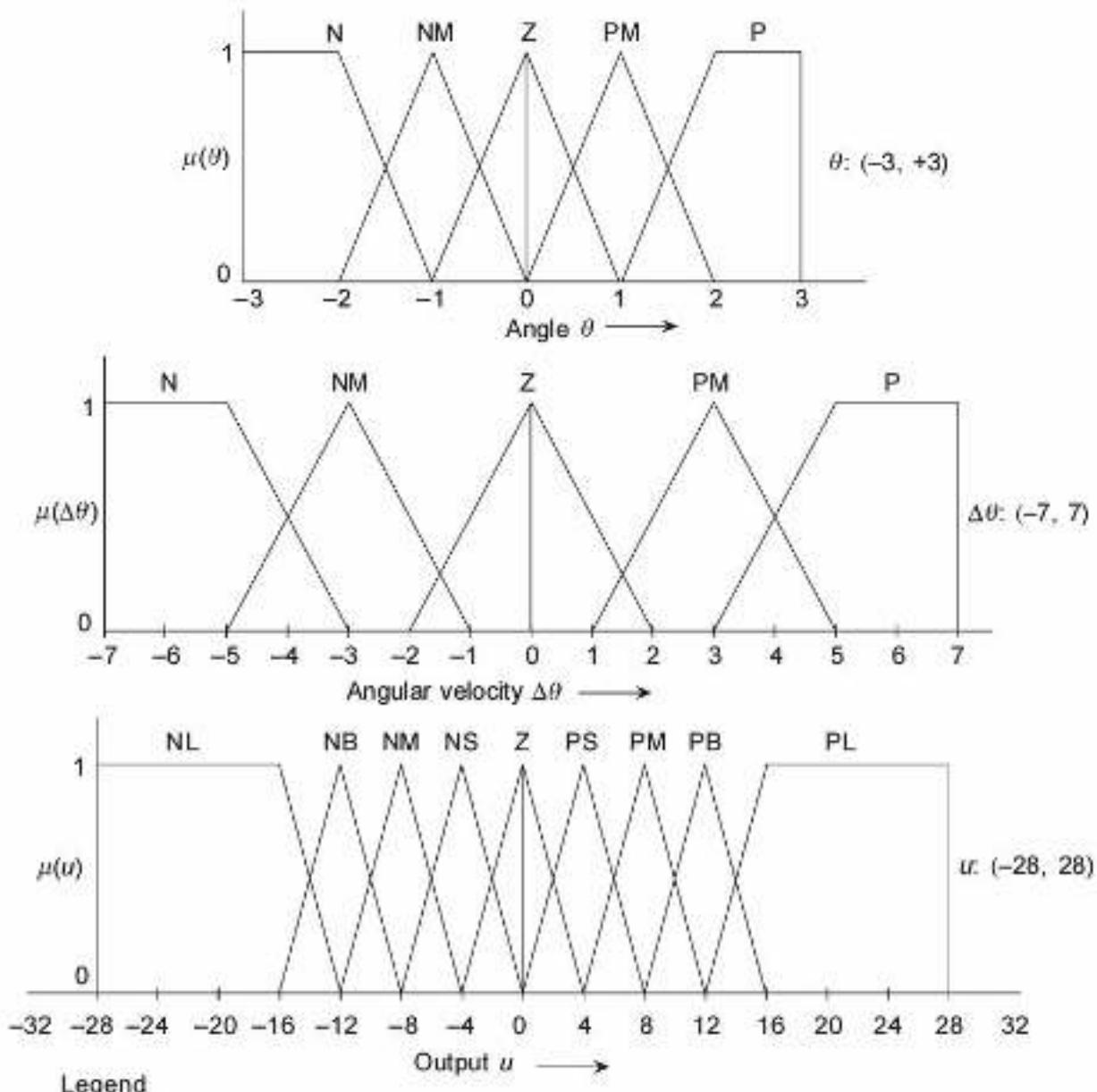
is approximated as the difference between the present angle measurement θ_t and the previous measurement θ_{t-1} .

The control fuzzy variable is $u(t)$ which is the torque applied to the pole located at the tip point of the pendulum in the anticlockwise direction. The control action can be positive or negative depending on whether the pendulum falls to the left or right of the vertical respectively. If the pendulum

staff is vertical then the control action should be zero. Figure 14.7 illustrates a model of the inverted pendulum.

Fig. 14.7 Model of an inverted pendulum.

The linearized discrete time state space equations for the inverted pendulum problem represented as matrix difference equations are



Legend

N Negative	NM Negative medium	Z Zero
NL Negative large	NB Negative big	NS Negative small
PS Positive small	PB Positive big	PL Positive large
PM Positive medium	P Positive	

We restrict the universes of discourse to small intervals, namely $[-3, 3]$

degrees for the angle θ , $[-7, 7]$ degrees per second for the angular velocity $\Delta\theta$

and $[-28, 28]$ for the output action u . Since the two fuzzy state variables and the output variable move over a positive, zero and negative scale, we quantize their universes of discourse into overlapping fuzzy sets as given in Fig. 14.8.

Fig. 14.8 Fuzzy set descriptions for the inverted pendulum problem.

$\Delta\theta \backslash \theta$	N	NM	Z	PM	P
N	NL		NB		Z
NM			NM	Z	
Z	NB	NM	Z		PB
PM		Z			PM
P	Z		PB	PM	PL

The fuzzy rules associated with the problem are shown in the FAM table presented in Fig. 14.9. We have chosen a restricted FAM representation.

Fig. 14.9 FAM matrix representation for the inverted pendulum problem.

We now begin a simulation of the control problem using the graphical method of inference. Let us start with the initial conditions $\theta = 0.5^\circ$, $\Delta\theta = -1.5$ degrees per second. We demonstrate the simulation for three cycles, $k = 0, 1$, and 2 . In the first cycle for $k = 0$, the initial conditions fire the following rules:

R 1 : If θ is Z and $\Delta\theta$ is Z then u is Z.

R 2 : If θ is Z and $\Delta\theta$ is NM then u is NM.

R 3 : If θ is PM and $\Delta\theta$ is NM then u is Z.

Figure 14.10(a) illustrates the membership propagation for the inputs x_0

$$1 = \theta$$

$$= 0.5 \text{ and } x_0$$

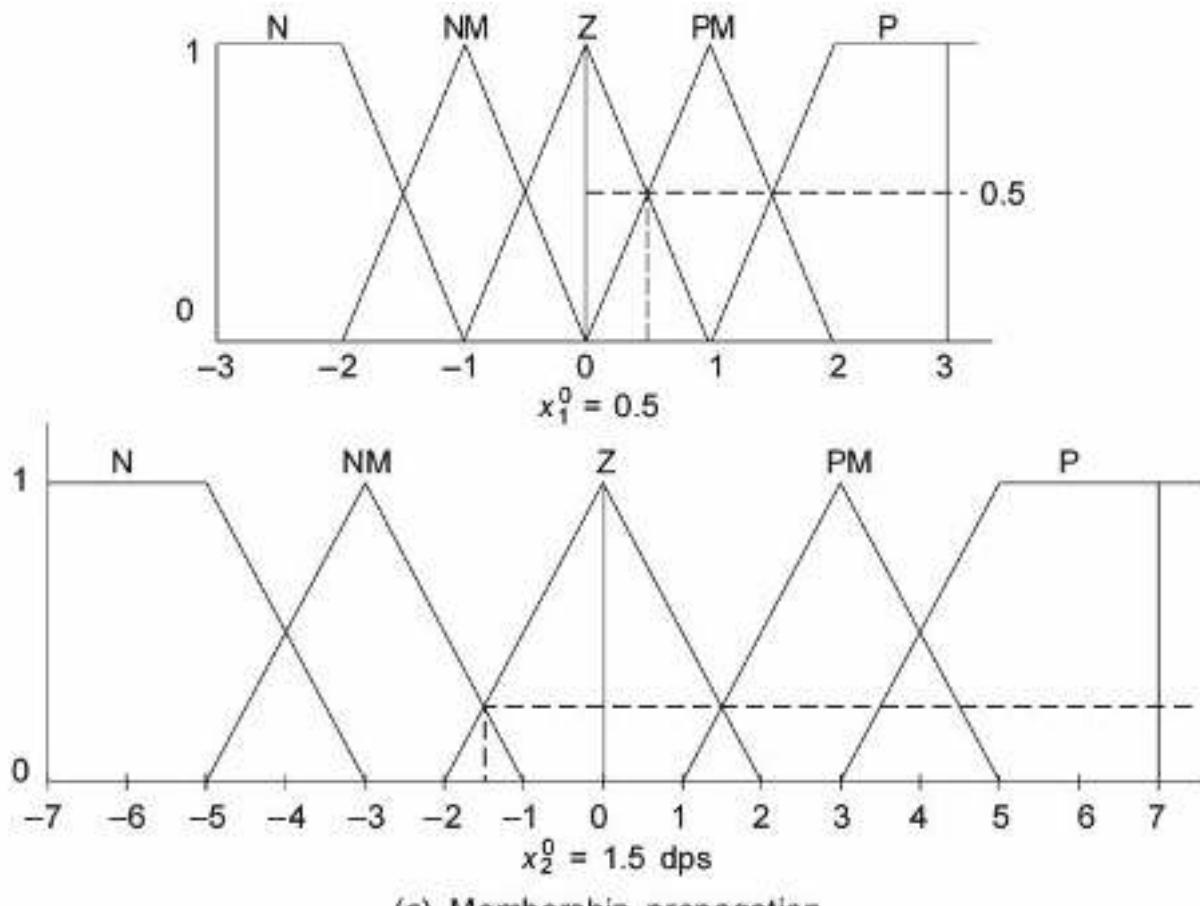
$$2 = \Delta\theta$$

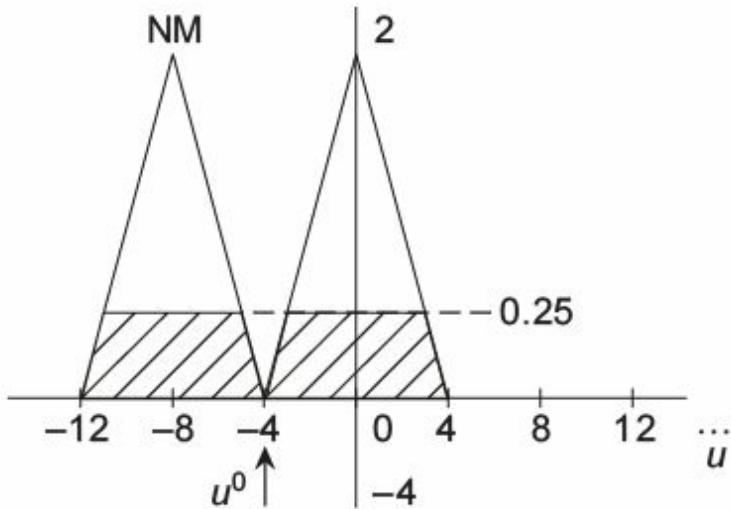
$= -1.5$ degrees/second. Observe that x_0

1 triggers the fuzzy sets Z and PM

while x_0

2 triggers NM and Z . Figure 14.10(b) shows the union of the truncated fuzzy consequents and the defuzzified output using the centroid method. Here the output at the end of the simulation cycle ($k = 0$) is $u(0) = -4$.





(b) Defuzzification using centroid method.

$$x_1^{(1)} = x_1^{(0)} + x_2^{(0)} = 0.5 - 1.5 = -1$$

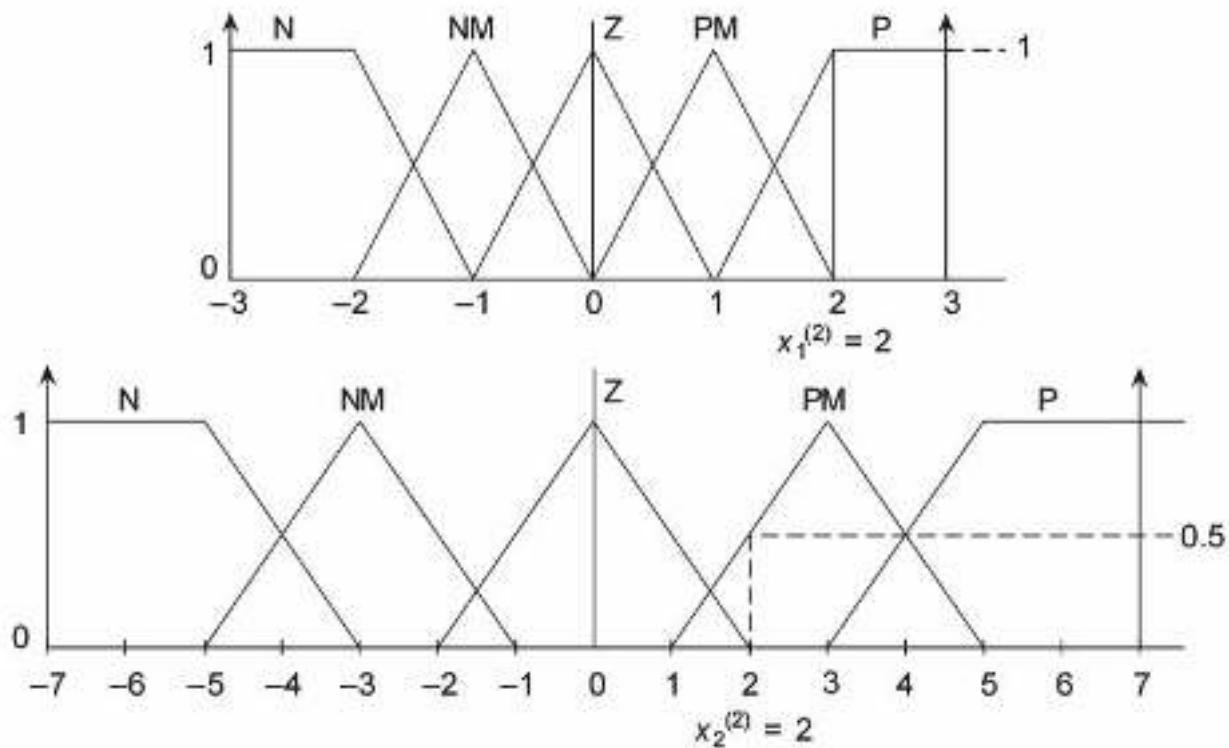
$$x_2^{(1)} = x_1^{(0)} + x_2^{(0)} - u^{(0)} = 0.5 - 1.5 - (-4) = 3$$

With $x_1^{(1)} = -1$ and $x_2^{(1)} = 3$, we proceed to the next cycle.

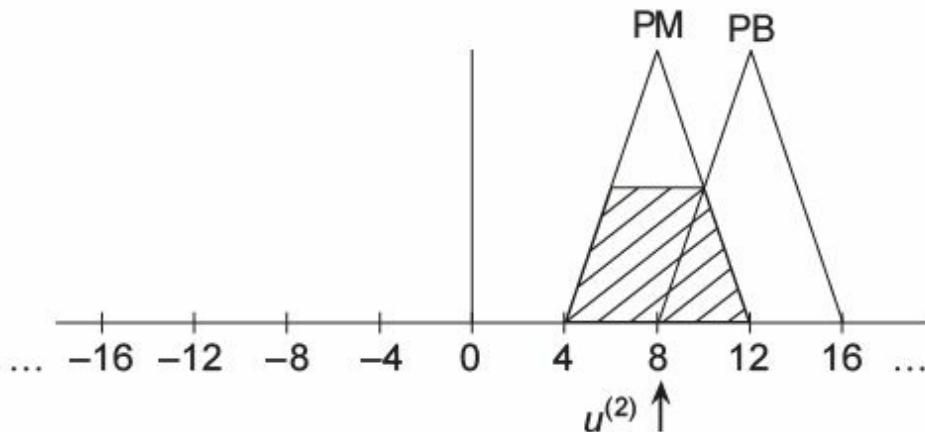
Fig. 14.10 Inverted pendulum problem: results at the end of simulation cycle $k = 0$.

We substitute the values obtained in the $k = 0$ th cycle in Eq. (14.17) to obtain the initial conditions for the next cycle ($k = 1$).

The only rule fired is



(a) Membership propagation



(b) Defuzzification using centroid method

If θ is NM and $\Delta\theta$ is PM then u is Z.

Figure 14.11(a) shows the membership propagation and Fig. 14.11(b) the defuzzified output value. The output obtained is $u(1) = 0$.

Fig. 14.11 Inverted pendulum problem: results at the end of the simulation cycle $k = 1$.

We substitute $x(1)$

(1)

1

, $x(2)$

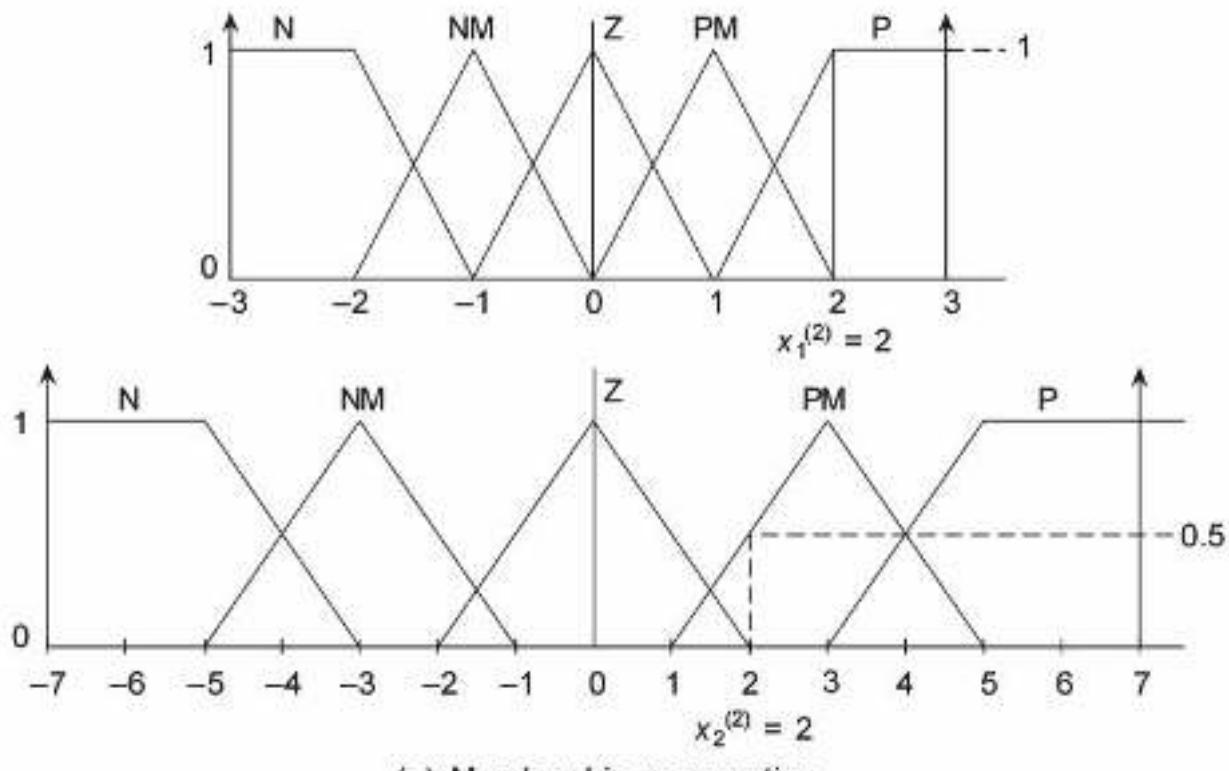
and $u(1)$ in Eq. (14.17) to obtain the initial conditions for the next simulation cycle ($k = 2$).

We get

$x(2)$

1

= 2,



$x(2)$

2

= 2

The rules fired are

If θ is P and $\Delta\theta$ is PM then u is PM.

If θ is P and $\Delta\theta$ is Z then u is PB.

Figure 14.12(a) shows the membership propagation and Fig. 14.12(b) the defuzzified output,

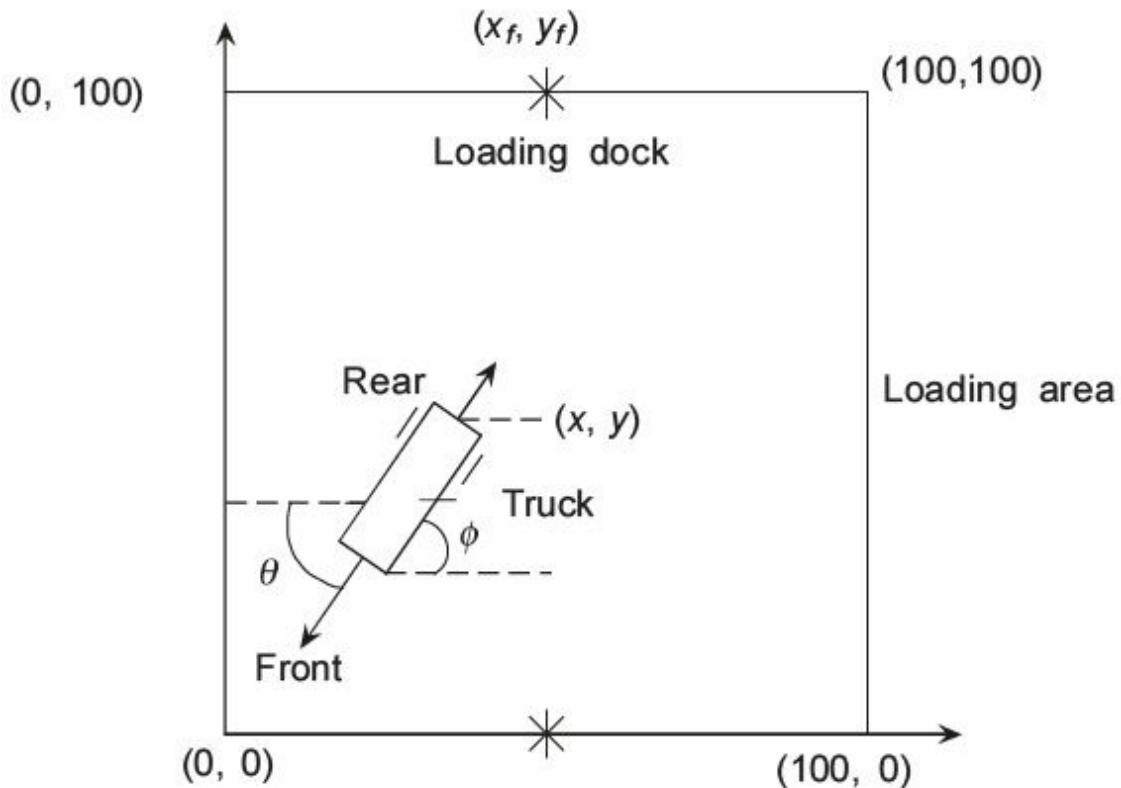
$u(2) = 8$. Thus, each simulation cycle beginning from $k = 0$ proceeds using the x_1, x_2 , values obtained in the previous cycle. We conclude the simulation cycles here. But in reality, the cycles may have to be repeated many number of times.

Fig. 14.12 Inverted pendulum problem: results at the end of the simulation cycle $k = 2$.

14.6.2 Fuzzy Truck Backer-upper System

In this section we demonstrate the application of FAM inference procedure for the problem of *Fuzzy Truck Backer-upper system* (Nguyen and Widrow, 1989) as discussed by Seong-Gon Kong and Kosko (1994).

Figure 14.13 shows the truck and the loading area. Here, the three state variables φ , x , and y determine the position of the truck. φ is the angle of the



truck with the horizontal and the coordinate pair (x, y) specifies the position of the rear center of the truck.

The objective is to make the truck arrive at the loading dock at a right angle, i.e. $\theta_f = 90^\circ$ and in such a way that the rear center (x, y) is aligned with (x_f, y_f) , the position of the loading dock. Here, only backing up motion of the truck is considered. The truck moves backward every stage by a certain distance. The loading zone and the loading dock are marked as shown in the figure.

Fig. 14.13 Fuzzy truck backer-upper system.

The output to be computed is the steering angle θ that will back up the truck to the loading dock from any given position and from any angle in which it is currently positioned in the loading zone. The input variables have been reduced to φ —the angle of the truck and x —the

x coordinate of its position. The y coordinate has been ignored assuming enough clearance between the truck and the loading dock.

The fuzzy set values of the input output variables and their ranges have been presented in Table 14.2.

Table 14.2 Fuzzy set values and ranges of input-output variables of the fuzzy truck backer-upper system Input/output variable

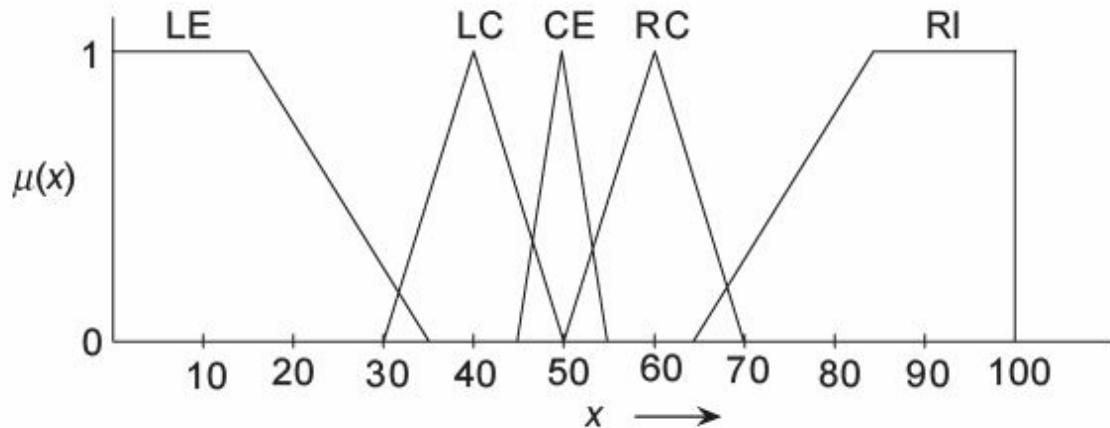
Fuzzy set values

Range

RB : Right below

LB : Left below

RU : right upper



ϕ

RV : Right vertical

$-90 \leq \phi \leq 270$

VE : Vertical

LV : Left vertical

LU : Left upper

LE : Left

RI : Right

x

LC : Left centre

$0 \leq x \leq 100$

CE : Center

RC : Right centre

NB : Negative big

NM : Negative medium

NS : Negative small

θ

ZE : Zero

$-30 \leq \theta \leq 30$

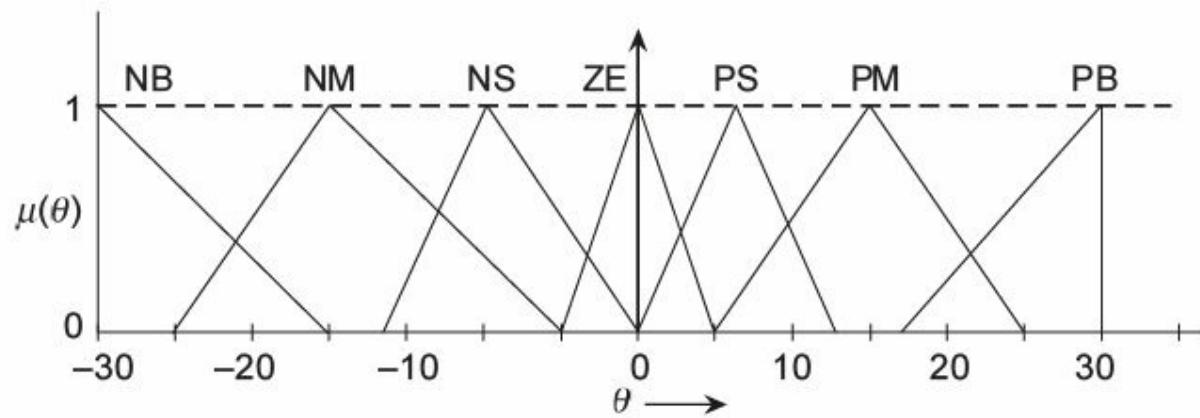
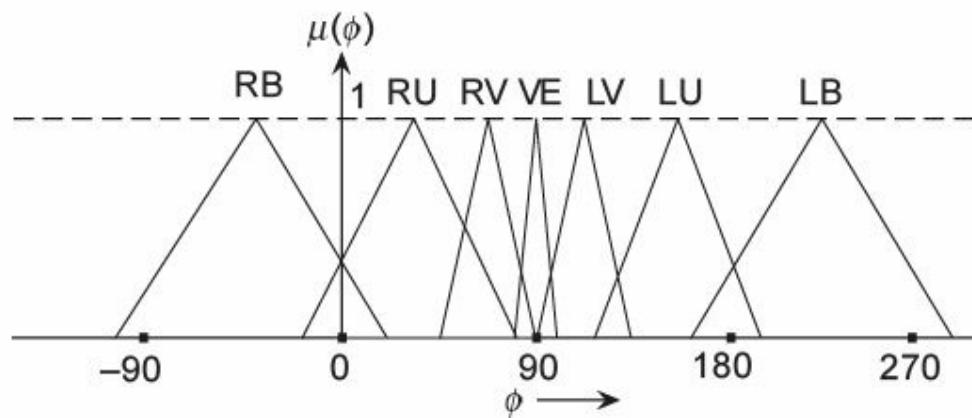
PS : Positive small

PM : Positive medium

PB : Positive big

Figure 14.14 illustrates the fuzzy sets of the input/output variables. The FAM bank associated with the truck backer-upper system is shown in Fig.

14.15. There are 35 FAM rules associated.



		x				
		LE	LC	CE	RC	RI
ϕ	RB	PS	PM	PM	PB	PB
	RU	NS	PS	PM	PB	PB
	RV	NM	NS	PS	PM	PB
	VE	NM	NM	ZE	PM	PM
	LV	NB	NM	NS	PS	PM
	LU	NB	NB	NM	NS	PS
	LB	NB	NB	NM	NM	NS

Fig. 14.14 Fuzzy membership functions for the fuzzy truck backer-upper system.

Fig. 14.15 FAM table of the fuzzy truck backer-upper system.

Making use of the graphical inference procedure and centroid method of defuzzification, the approximate contour of the plot of the truck backing for an initial condition $(x, \phi) = (20, 30)$ is as shown in Fig. 14.16.

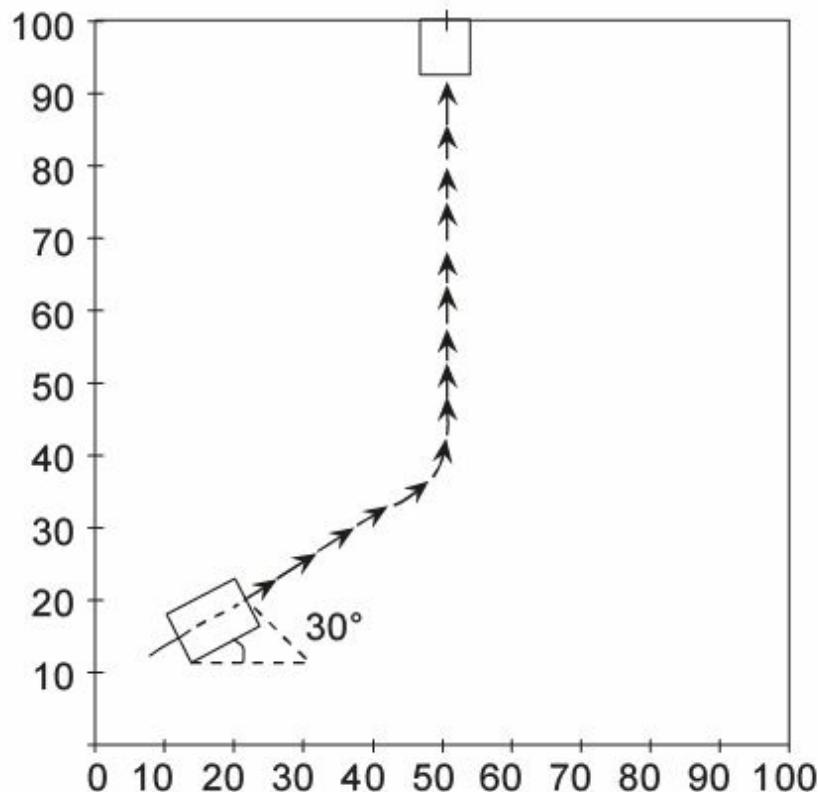


Fig. 14.16 Truck trajectories (approximate contour) for an initial position $(x, \phi) = (20, 30)$.

SUMMARY

Fuzzy associative memories (FAMs) are fuzzy systems which map fuzzy sets to fuzzy sets and behave like associative memories.

A FAM encodes fuzzy rules with single associative or compound associations e.g.

If X is A then Y is B.

If X₁ is A₁ and X₂ is A₂ ... and X_n is A_n then Y₁ is B₁ and Y₂ is B₂ ...

and Y_p is BP.

The graphical method of inference is an elegant method to infer an output *B* given an input *A* for a single association FAM system comprising a bank of *M* rules. Correlation matrix encoding could be employed for inference in fuzzy Hebb FAMs. But the accuracy of recall depends on the heights of the fuzzy sets *A*, *B* involved in the association, i.e. $h(A)$ should be equal to $h(B)$. Besides, this method is not suitable for application on a FAM bank of *M* rules.

In the case of a FAM system with multiple antecedents/consequents, the graphical method of inference can be successfully employed after the rules have been simplified using the rules of decomposition.

The application of FAM is demonstrated on two classical problems, namely balancing an inverted pendulum and fuzzy truck backer-upper system.

PROGRAMMING ASSIGNMENT

P14.1 Aircraft landing control problem (Timothy Ross, 1997) The problem deals with the simulation of the final descent and landing approach of

an aircraft. The two input state space variables are the height *h* of the aircraft above

the ground and the vertical velocity *v* of the aircraft. The output variable is the control force *f*.

The control equations for this problem are

$$\begin{aligned} vi + 1 &= vi + fi \\ hi + 1 &= hi + vi \end{aligned} \quad (i)$$

where vi , hi and fi are the values acquired in the previous simulation cycle and $vi + 1$, $hi + 1$ are the new values.

Tables P14.1(a), (b), (c) show the membership values for the fuzzy sets of height, vertical velocity, and output force.

Table P14.1(a) Membership values for height

0	0
100	0
200	0
300	0
400	0
500	0
600	0
700	0
800	0
900	0
1000	0
Large (L)	1
0	0
0	0
0	0
0	0

0

0

0.2

0.4

0.6

0.8

1

Medium (M)

0

0

0

0

0.2

0.4

0.6

0.8

1

0.8

0.6

Small (S)

0.4

0.6

0.8

1

0.8

0.6

0.4

0.2

0

0

0

Near zero (NZ)

1

0.8

0.6

0.4

0.2

0

0

0

0

0

0

...

Table P14.1(b) Membership values for velocity

-30

-25

-20

-15

-10

-5

0

5

10

15

20

25

30

Up large (UL)

0

0

0

0

0

0

0

0

0

0.5

1

1

1

Up small (US)

0

0

0

0

0

0

0.5

1

0.5

0

0

0

Zero (Z)

0

0

0

0

0

0.5

1

0.5

0

0

0

0

0

Table P14(d) FAM table for aircraft landing simulation

Height \ Velocity	DL	DS	Z	US	UL
L	Z	DS	DL	DL	DL
M	US	Z	DS	DL	DL
S	UL	US	Z	DS	DL
NZ	UL	UL	Z	DS	DS

Down small (DS)

0

0

0

0.5

1

0.5

0

0

0

0

0

0

0

Down large (DL)

1

1

1

0.5

0

0

0

0

0

0

0

0

0

...

Table P14.1(c) Membership values for control force

-30

-25

-20

-15

-10

-5

0

5

10

15

20

25

30

Up large (UL)

0

0

0

0

0

0

0

0

0

0.5

1

1

1

Up small (US)

0

0

0

0

0

0

0

0.5

1

0.5

0

0

0

Zero (Z)

0

0

0

0

0

0.5

1

0.5

0

0

0

0

0

Down small (DS)

0

0

0

0.5

1

0.5

0

0

0

0

0

0

0

Down large (DL)

1

1

1

0.5

0

0

0

0

0

0

0

0

0

The FAM table is shown in Table P14.1(d).

Assume the initial conditions to be

Height $h_0 = 900$ ft

Velocity $v_0 = -18$ ft/sec

(a) Trace the simulation cycles for five steps.

(b) Write the program to compute the control force starting from the stated initial conditions for N number of simulations cycles. Choose N to be a large number. Repeat the simulation for different initial conditions.

(c) For the values of h and v acquired in the simulation experiment, plot h vs v to get the profile of the descent.

SUGGESTED FURTHER READING

Adaptive FAM (AFAM) (Kosko, 1994) is a time varying FAM which provides a mapping between fuzzy sets that vary with time. Fu-lai Chung and Tong Lee (1994) have proposed a high capacity FAM model called FRM

(Fuzzy Relational Memory). Timothy Ross (1997) discusses interesting applications of FAM especially using the graphical method of inference.

REFERENCES

Mamdani, E.H. (1977), Application of Fuzzy Logic to Approximate Reasoning using Linguistic Synthesis, *IEEE Trans on Computers*, Vol. C-26, No. 12, pp. 1182–1191, December.

Nguyen, D. and B. Widrow (1989), The Truck Backer-Upper: An Example of Self-learning in Neural Networks, *Proc. of Intl. Joint Conf. on Neural Networks (IJCNN-89)*, Vol. II,

pp. 357–363.

Seong-Gon Kong and Bart Kosko (1994), Comparison of Fuzzy and Neural Truck Backer-Upper Control Systems, in *Neural Networks and Fuzzy Systems: A Dynamical System Approach to Machine Intelligence*, by Bart Kosko, Prentice-Hall of India.

Timothy, J. Ross (1997), *Fuzzy Logic with Engineering Applications*, McGraw Hill, 1997.

Chapter15

Fuzzy Logic Controlled Genetic

Algorithms

Almost all the computing, including computational machines and finite element analysis are considered as *hard computing*. They are based on mathematical approaches to problem solving and they imbibe their basic characteristics from mathematics. On the other hand, *soft computing* methods are based on biological approaches to problem solving, where mathematics does not play as central a role as it does in engineering problem solving methods. Hard computing software considers both input and output to be precise to within round off. In fact, there may not be much use for high degree of precision in most engineering problems since for example, material parameters cannot be determined with a high degree of precision whereas soft computing methods have inherited imprecision tolerance and non universality from biological systems. An added attraction of soft computing is due to the imprecision tolerance and random initial state of the soft computing tools.

This introduces a random variability in the model of the mathematical systems, very similar to random variability, which exists in the real systems.

15.1 SOFT COMPUTING TOOLS

15.1.1 Fuzzy Logic as a Soft Computing Tool

The realization of uncertainty, vagueness and ambiguity in the world has led to the concept of fuzziness. In the last three decades, significant progress has been made in the development of fuzzy sets, fuzzy logic theory, and their use in engineering applications. The successful application of fuzzy sets and fuzzy logic can be attributed to the fact that fuzzy theory reflects the true situation in the real world, where human thinking is dominated by approximate reasoning logic. This is suited for applications where the ability to model real-world problems in precise mathematical forms is difficult. The foundation of fuzzy logic is fuzzy set theory, first proposed by Bellman and Zadeh (1970), Wang and Wang (1985a, b), Soh and Yang (1996), Yang and Soh (2000) and Rao (1987), applied fuzzy optimization techniques. In a traditional optimization techniques, the constraints must strictly be satisfied.

However, it is not reasonable to discard those designs that slightly violate one or more constraints during the early design stage. These complexities and uncertainties encountered in the optimization and design of real structures provide the main motivation for the fuzzy integrated system. Hence, fuzzy logic can be considered as a soft computing tool.

15.1.2 Genetic Algorithm as a Soft Computing Tool

As seen in Chapters 8 and 9, genetic algorithm is a computational model based on natural evolution (Holland, 1975). A system to be optimized is represented by a binary string which encodes the parameters of the system. A population of strings with initial random parameters is used. A number of generations are simulated with operators representing the major elements of evolution such as competition, fitness based selection, recombination, and mutation. The whole process is highly random. However, the evolutionary process lead to filter individuals in the population closer to satisfying the objective function of the optimization problem. Genetic algorithms have all the characteristics of soft computing. The methodology is highly robust and imprecision tolerant. If a unique optimum exists, the procedure approaches it through gradual improvement of the fitness and if the optimum is not unique,

the method will approach one of the optimum solutions.

$$C_{ij}^L \leq C_{ij}(R, A, T) \leq C_{ij}^U$$

15.2 PROBLEM DESCRIPTION OF OPTIMUM DESIGN

In case of civil engineering or machine tool structure, the process of simultaneous sizing, geometry, and topology can be stated as follows.

Find the particular set of sizing variables A , geometric variables R , and topological variable T such that the structural weight $W(R, A, T)$

$$\rightarrow \min \quad (15.1)$$

subject to

(15.2)

$$i = 1, 2, \dots, n \quad j = 1, 2, \dots, n$$

in which A and R are vectors of member cross-sectional area and joint coordinates respectively

(as discussed in Chapter 9). T represents the set of existing members, L and U

superscripts denote the lower and upper bounds respectively, $C_{ij}(R, A, T)$ specify the constraints that limit the relevant design variable domains, nc are total number of constraint types, and nj are total number of constraints of j th constraint type.

The constraint types may include the constraints of

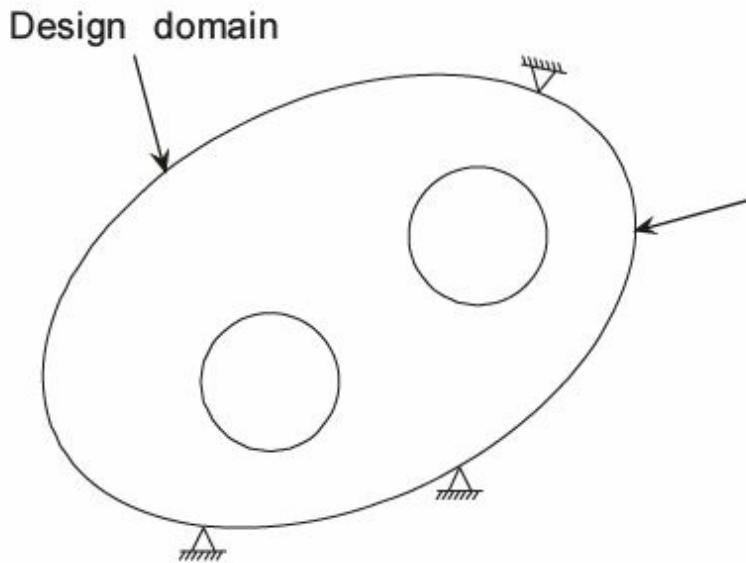
1. member cross-sectional areas,
2. joint coordinates,
3. member allowable stresses,
4. joint allowable displacements,

5. member buckling strength,

6. member length, and

7. member slenderness (L/r).

The optimum design problem described here refers to the problem of finding an optimal structure within a predefined design domain, satisfying the design constraints and loading and support conditions. The general design is shown in Fig. 15.1. The structure can be evolved within the design domain or inside the design domain. The structure can have any number of nodes and any number of elements. Some nodes may be fixed to satisfy the



requirements of loading and supporting, and the others can occupy any position within the design domain. The objective of the optimum design is to minimize the weight of the structure. The formulation for this problem can be expressed in terms of Eqs. (15.1) and (15.2).

Fig. 15.1 General design domain.

$$\tilde{C}_{ij}$$

$$\tilde{C}_{ij}^L$$

$$\tilde{C}_{ij}^U$$

$$\tilde{C}_{ij}^{Le}$$

$$\tilde{C}_{ij}^{Ue}$$

$$\tilde{C}_{ij}^L - \tilde{C}_{ij}^{Le} \leq C_{ij} \leq \tilde{C}_{ij}^U + \tilde{C}_{ij}^{Ue}$$

15.3 FUZZY CONSTRAINTS

The classical or crisp set and mathematical logic divide the world into “yes” or “no”, “white” or “black”, and “true” or “false” as discussed in crisp logic.

On the other hand, fuzzy sets deal with the objects that are a matter of degree with all possible grades of truth between “yes” or “no” and the various shades of colours between “white” and “black”. Fuzzy set and fuzzy logic has been discussed in detail in Chapters 6 and 7. Herein, we will discuss the relevance of fuzzy set to structural optimization.

The fuzzy set theory (Zadeh, 1987) has a function that admits a degree of membership in the set from complete exclusion (0) to absolute inclusion (1).

Such a function is called a membership function $\mu_i(y)$ of the object ‘y’ in the fuzzy set \tilde{A} : $\mu_i(y): R^n \rightarrow [0, 1]$. The membership represents a certain degree of belonging of the object in the fuzzy sets. The transition from not belonging to belonging is gradual, which gives one or some means of handling vagueness. Fuzzy sets thus, overcome a major weakness of crisp sets. Fuzzy sets do not have an arbitrarily established boundary to separate the members form non-members.

An inequality constraint C_{ij} in Eq. (15.2) for optimum design problems can be defined to a fuzzy constraint

with an α membership degree as shown in

Fig. 15.2(a) . In Fig. 15.2(a),

and

are respectively the lower and upper

bounds of the corresponding object C_{ij} and

and

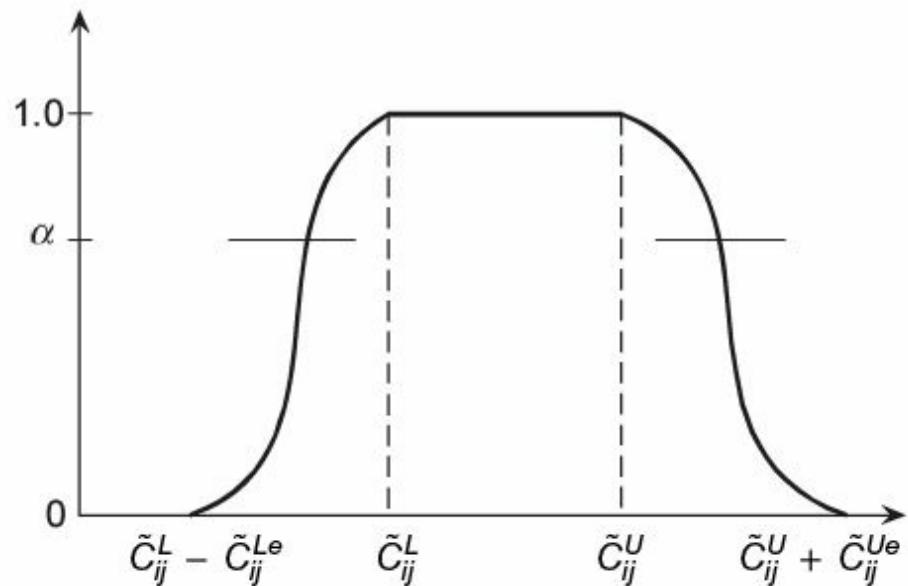
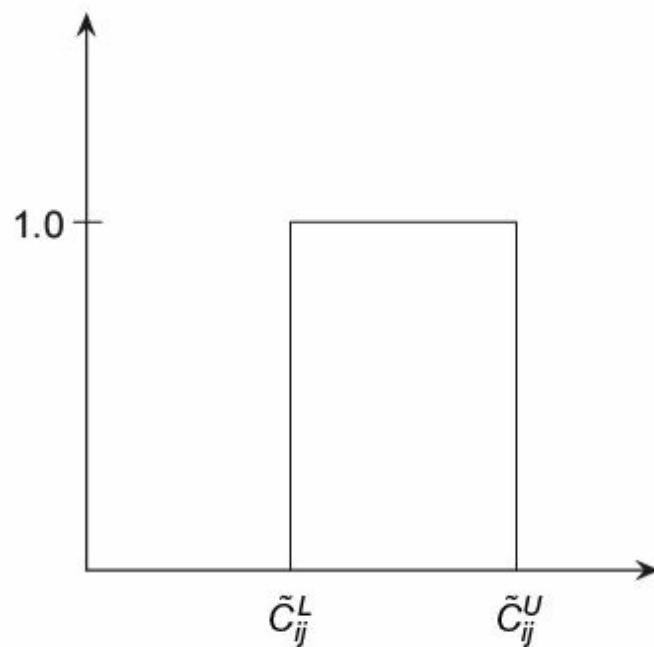
are zero. This fuzzy

constraint becomes a “hard constraint” or a crisp set as shown in Fig. 15.2(b). At the same time, the value α also gives us a description about how the constraint is satisfied. If $\alpha = 1$, the constraint is fully satisfied and if zero, the C_{ij} is not satisfied. Furthermore, its value between 0 and 1 implies that the constraint is satisfied to the relevant degree. Hence, on the fuzzy set theory introduced above, Eqs. (15.1) and (15.2) can be transformed into the following fuzzy optimization problem as

$$W(R, A, T) \rightarrow \min \quad (15.3)$$

subject to

$$(15.4)$$

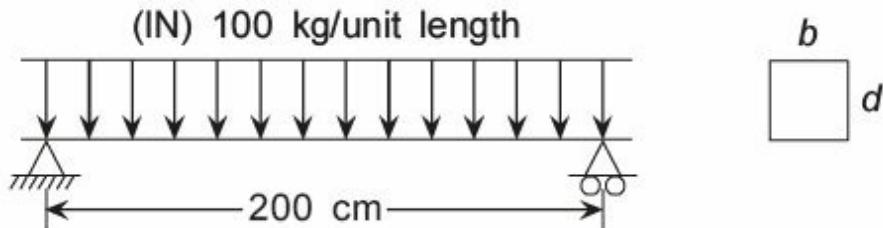

 \tilde{C}_{ij}


In order to understand the principles, let us discuss an optimization problem with fuzzy hard constraint or crisp set.

Fig. 15.2(a) Fuzzy constraint

with a membership degree.

Fig. 15.2(b) Fuzzy hard constraint.



$$\frac{6pl}{8bd^2}$$

$$\sqrt{\frac{3750}{b}}$$

$$C_2 = \delta = \frac{5pl^4}{384E(bd^3/12)} \leq 0.2, \quad d \leq \sqrt[3]{\frac{62500}{b}}$$

15.4 ILLUSTRATIONS

15.4.1 Optimization of the Weight of A Beam

Consider a beam of span $l = 200$ cm carrying a load of $P = 100$ N/unit length with $E = 2 \times$

106 N/cm². It is proposed to optimize the weight to meet deflection limit of 0.2 cm and stress of 800 N/cm². The beam is of rectangular cross-section with dimensions $b \times d$ (see Fig. 15.3) .

Fig. 15.3 Simply supported beam.

Solution

$$P = \text{load/unit length} = 100; \quad l = 200;$$

Weight of the beam, $W = \rho Lbd$

Since ρ, l are constants

$$\min \rightarrow W(b, d) = bd$$

C 1 = stress = $\lambda \leq 800 \text{ N/cm}^2$

C 2 = deflection = $\delta \leq 0.2 \text{ cm}$

C 1 = $\sigma =$

$$\leq 800, \text{ or } bd \leq 3750, \text{ or } d =$$

The d vs b curve is shown as d_1 curve in Fig. 15.4.

The d versus b curve is shown as d_2 curve in Fig. 15.4. The optimum value is at the point A where $b = 13.6 \text{ cm}$ and $d = 16 \text{ cm}$ and gives $W = bd =$

225.76.

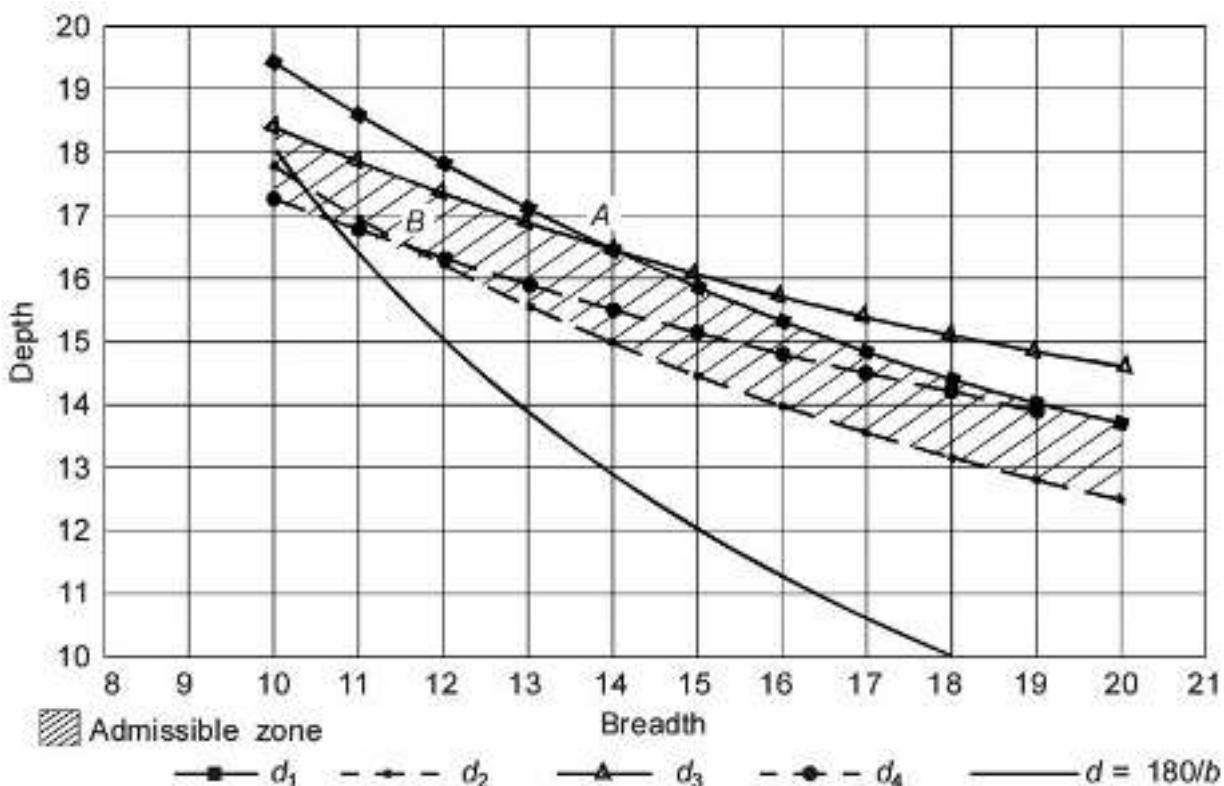


Fig. 15.4 Fuzzy optimization.

If imprecise information, i.e. fuzzy is available with respect to deflection or stress limits, it is necessary to find out how optimum solution changes for any variation of constraints. Assuming that the stress limits and deflection limits are raised by 20%, i.e. $\sigma^* = 960$; $\delta^* = 0.24$ for the new problem, $d =$

$(3125/b)1/2$ for stress limit denoted by d_3 curve and $d = (52083.33/b)1/3$ for deflection limit denoted by d_4 curve can be drawn as shown in Fig. 15.4. The optimum value is at B for which $b = 11.3$; $d = 16.6$ giving $W = bd = 187.58$.

The cost function curve for $bd = 180$ is also shown in Fig. 15.4. Now it is very clear if the limits of stress and deflection are imprecise to within 20%, a new optimization solution is obtained as $b = 11.3$; and $d = 16.6$. Here, it may be noted that both constraints control the design and the end result indicates how far the weight objective function sets increase due to fuzzy constraints.

Even though in the above example “crisp set” or fuzzy “hard” constraint is considered, nevertheless this example explains the method of treating fuzzy constraints in an optimization problem.

15.4.2 Optimal Mix Design for High Performance Concrete

High performance concrete (HPC) is cement concrete in which each

$$W = \left(\sum_{i=1}^{mg} \text{cost}_i \times X_i \right) \Bigg/ \sum_{i=1}^{mg} X_i$$

ingredient performs effectively to combine towards fresh concrete as well as hardened concrete properties. HPC consists of all ingredients of conventional concrete with chemical admixtures as super plasticizers and admixtures like fly ash and silica fume. The following materials are used in HPC: 1. Cement —43 grade (ordinary Portland cement)

2. Fine aggregate—naturally available sand
3. Coarse aggregate—locally available blue granite
4. Mineral admixtures—silica fume imported from Australia
5. Chemical admixtures—super plasticizer (Conplast SP 337)

A general discrete sizing structural optimization problem is posed as $W = F(x)$ (15.5)

subject to

$$gj(x) \leq 0 \text{ for } j = 1, 2, \dots, NC \quad (15.6) \text{ satisfying}$$

$$Xi_{\min} \leq Xi \leq Xi_{\max} \text{ for } i = 1, 2, \dots, mg \quad (15.7)$$

Here, NC and mg represent the number of constraints and the number of independent design variables and $W = F(X)$ is the objective function. gj is the j th constraint and the inequalities in

Eq. (15.7) are known as *side constraints* on the design variables.

In case of high performance concrete, W is the cost of the mix per unit weight and is called *cost function* given by

(15.8)

It is necessary to design HPC mix for the strength of 120 MPa and for a slump of 120 mm. Considering $\pm 10\%$ tolerance, the constraint equation is written as

$$108 \leq f \leq 132 \quad (15.9a)$$

$$108 \leq s \leq 132 \quad (15.9b)$$

where f is the strength and s the slump given by (obtained by training 23 data using single hidden neural network with single hidden neuron)

$$\ln\left(\frac{f/160}{(1.387 - f/160)}\right)$$

$$\ln\left(\frac{s/200}{(1.164 - s/200)}\right)$$

$$\sum_{j=1}^2 C_j$$

$$(X_i)_{\text{inc}} = \frac{\{(X_i)_{\max} - (X_i)_{\min}\}}{2^{nb} - 1}$$

$$= -0.7388 X 1 - 0.6813 X 2 + 3.330 X 3 \\ + 0.00429 X 4 + 0.3616 X 5 + 0.4141 \quad (15.10a)$$

$$= 0.863 X 1 + 0.686 X 2 - 2.3558 X 3$$

+

$$0.00862 X 4$$

+

$$0.19856 X 5$$

-

$$3.039 \quad (15.10b)$$

The constraint equation for strength is written as

$$C 1 = 0 \quad \text{if } 108 < f < 132$$

$$C 1 = (1 - f/108) \quad \text{if } f < 108$$

$$C_1 = (f/132 - 1) \quad \text{if } f > 132$$

Similarly, the constraint equation for slump is written as

$$C_2 = 0 \quad \text{if } 108 < s < 132$$

$$C_2 = (1 - s/108) \quad \text{if } s < 108$$

$$C_2 = (s/132 - 1) \quad \text{if } s > 132$$

For GA (genetic algorithm), the constrained problem is transformed into unconstrained problem as explained in Chapters 8 and 9 as

$$\Phi(X) = F(X) \{1 + pC\}$$

where

$$C =$$

where p can be taken as 10 for all practical purposes.

Assume, we represent the constituents by 5-bit strings. (X_i)min is represented by 00000 and (X_i)max is represented by 11111. The increment (X_i)inc in the design variable is calculated as

where nb is the number of bits and here $nb = 5$. To get the corresponding proportion of the constituent of bit string of 10110, the decoded value is $1 \times 24 + 0 \times 23 + 1 \times 22 + 1 \times 21 + 0 \times 20 = 22$

and the proportion of the constituents corresponding to 10110 is given by $X_i = (X_i)$ min + 22(X_i)inc

To start the algorithm, the initial population is created randomly. If the mix consists of five constituents and any individual in the population represents five bits, each constituent contains 25 bits. The objective function and constraints are calculated for every individual and genetic algorithm is applied as explained in Chapter 9 and the optimal mix is arrived at as
Cement = 1

Sand = 1.36

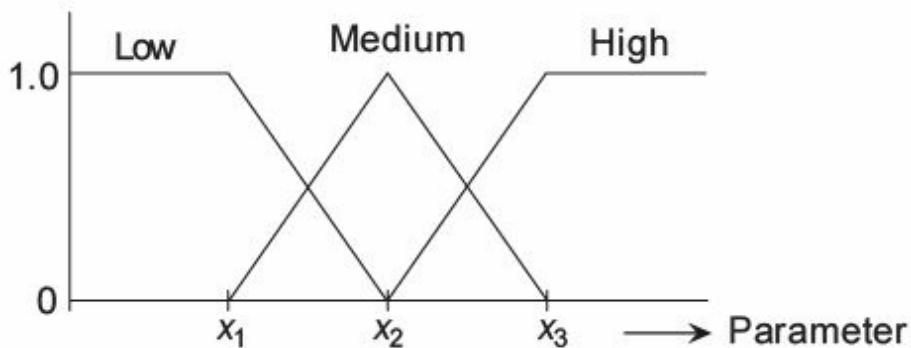
Coarse aggregate = 2.95

Water cement ratio = 0.4190

Silica fume = 21.3%

Superplasticizer = 4.03

The above mix gives the strength of 129 MPa and a slump of 125 mm and the cost of concrete mix/unit volume is given as Rs 3.50.



15.5 GA IN FUZZY LOGIC CONTROLLER DESIGN

For optimal control problems, fuzzy logic techniques are primarily applied since quick control strategy is needed and imprecise and qualitative definition of action plans are available. While designing an optimal fuzzy controller, one has to look for two primary activities.

1. Find optimal membership functions for control and action variable.
2. Find an optimal set of rules between control and action variable.

In the above two cases, GAs have been suitably used. Figure 15.5 shows typical membership functions for a variable (control or action) having three choices low, medium, and high. Since the maximum membership function value of these choices is always one, the abscissae marked by X_i are usually chosen by the user. These abscissae can be treated as variables in GA and an

optimization problem can be posed to find these variables for minimizing or maximizing a control strategy such as time of variable operation, product quality, and others. Consider an example given by Deb (1999) to illustrate how GA can be uniquely applied to the above problem. Let us assume that there are two control variables (temperature and humidity) and there are three operations for each, low, medium, and high. There is one action variable (water jet flow rate) which also takes three choices low, medium, and high.

Considering individual effect of each control variable separately, there are total of $4 \times 4 - 1 = 15$ combinations of control variables possible as shown in Table 15.1.

Fig. 15.5 Fuzzy membership function and typical variables used for optimal design.

Table 15.1 Action variable for a string representing a fuzzy rule base

Humidity

Temperature

Low

Medium

High

No action

Low

High

Medium

—

Medium

Medium

Low

—

Medium

Medium

High

Medium

High

—

—

No action

—

—

High

—

Thus, finding an optimal rule base is equivalent to finding the four operations (fourth operation—not known) or the action variable for each combination of the control variables. A GA with a string length of 15 and with a ternary coding can be used to represent the rule base for this problem.

Considering real values 1 to 4 for representation as 1—Low, 2—Medium, 3

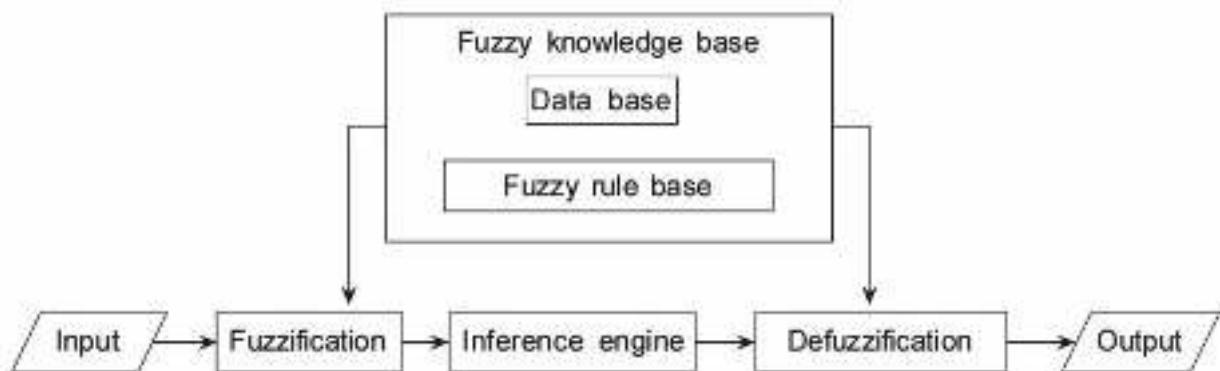
—High,

4—No action, thereby signifying the absence of the corresponding combination of action variables in the rule base. Table 15.1 shows the rule base and this can be represented by the following string.

3 1 2 4 2 4 3 4 4 2 4 3 2 2 4

Although this rule base may not be the optimal one, GA can process a population of such rule bases and finally find the optimal rule base. Once the rows present in the rule base are determined from the string user, defined fixed membership functions can be used to simulate the underlying process.

The objective function can be evaluated and the usual single point cross over and a mutation operator (one allele mutating to one of three other alleles) can be used with this coding. GA can find the optimal number of rules to solve the problem. If one wants to use binary strings instead of ternary strings and two bits are used to represent each of four operations, a total of 30 bits is necessary to represent a rule base. This kind of technique has been used to design fuzzy logic controller for mobile robot navigation among dynamic obstacles (Deb et al., 1998). Both optimum membership function determination and optimal rule base tasks can be achieved simultaneously by using a concatenation of two codings mentioned above. A part of the overall string will represent the abscissae of the control variables and the rest of the string will represent the rules present in the rule base. Fitness is calculated as explained above.



15.6 FUZZY LOGIC CONTROLLER

Soh and Yang (1996), Yang and Soh (2000) have investigated fuzzy based structural optimization using GA. Using the approach of Soh and Yang as seen in Section 15.4, a fuzzy logic controller (FLC) a rule based system incorporating the flexibility of human decision making is used for fuzzy structural optimization. The fuzzy functions are intended to represent a human expert's conception of the linguistic terms, thus giving an approximation of the confidence with which precise numeric value is described by a linguistic label.

15.6.1 Components of Fuzzy Logic Controller (FLC)

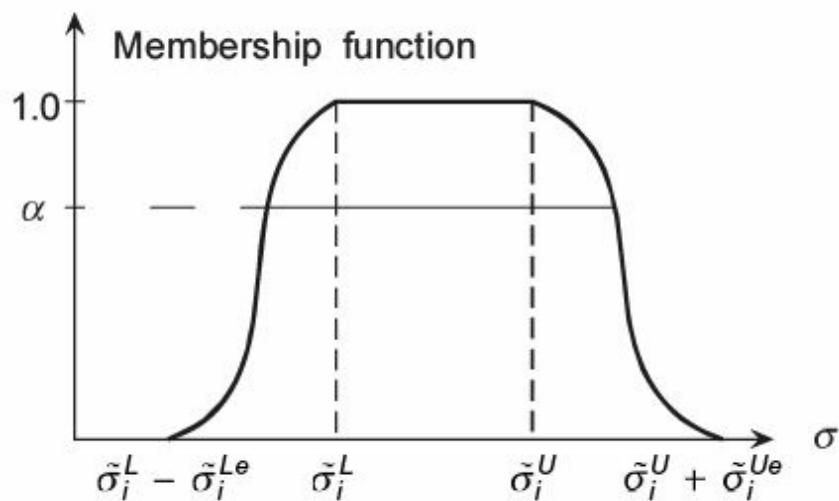
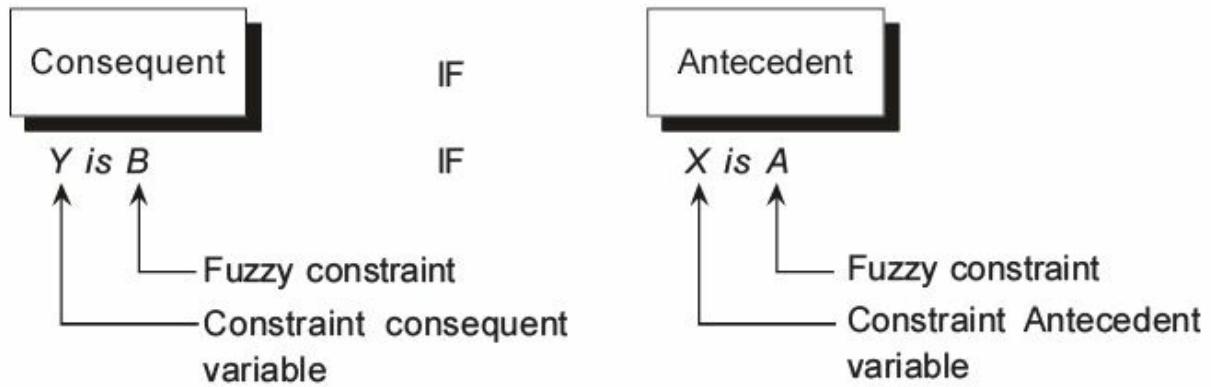
As shown in Fig. 15.6, fuzzy logic controller process is divided into three stages.

- (a) Fuzzification—To calculate fuzzy input (i.e. to evaluate the input variables with respect to corresponding linguistic terms in the condition side).
- (b) Fuzzy inference—To calculate fuzzy output (i.e. to evaluate the activation strength of every rule base and combine their action sides).
- (c) Defuzzification—To calculate the actual output (i.e. to convert the fuzzy output into precise numerical value).

Fig. 15.6 Framework of fuzzy logic controller (FLC).

15.6.2 Fuzzy IF-THEN Rules

Fuzzy rules take the form IF (conditions) and THEN (actions), where



conditions and actions are linguistic variables, respectively. An example of Fuzzy IF-THEN rule is given below.

Increase interest rates slightly if unemployment is low and inflation is moderate.

Increase interest rates sharply if unemployment is low and inflation is moderate but rising sharply.

Decrease interest rates slightly if unemployment is low but increasing and inflation rate is low and stable.

The primary format of IF-THEN rules is given in Fig. 15.7

Fig. 15.7 Format of IF-THEN rule.

Fig. 15.8 Fuzzy constraint for stress.

Example

Volume is small if pressure is high.

Usually in civil engineering, most of these specifications in codes and the functional requirements set by the users must be given in natural language to describe the expert's knowledge of design modifications.

In usual structural optimization, the stress constraint written as

$$\begin{aligned} \sigma_i^L &\leq \sigma_i \leq \sigma_i^U \\ \tilde{\sigma}_i^L - \tilde{\sigma}_i^{Le} &\tilde{\leq} \sigma_i \leq \tilde{\sigma}_i^U + \tilde{\sigma}_i^{Ue} \\ &\tilde{\leq} \\ (15.11) \end{aligned}$$

is a member stress constraint, where σ_i is the stress in member ‘ i ’ and σ_i^L and σ_i^U are the lower and upper bounds of the allowable stress. But in case of fuzzy optimization, the Eq. (15.11) is replaced as

(15.12)

with relevant α membership degree similar to Fig. 15.2 as in Fig. 15.8.

Here, the symbol means fuzzy variable operator and α represents a series of linguistic variables that means “very very small”, “small”, “medium”,

“large”, “very large”, and “very very large” and so forth. According to Zadeh (1987), the following seven fuzzy variables are usually in the study of structural optimization as negative large (NL), negative medium (NM), negative small (NS), zero (ZE), positive small (PS), positive medium (PM), and positive large (PL). They are defined by the membership functions as shown in Fig. 15.9 . For the convergence of implementation, seven fuzzy variables are assigned seven integer reference numbers, namely

-3, -2, -1, 0, 1, 2, 3 respectively.

If the allowable stress is 140 MPa and the tolerance for each unit is 5Mpa, X axis is also given in terms of stress values. Similarly, one can define seven fuzzy membership functions for displacements as well as for any other variable.

As explained before, heuristic fuzzy rules can be written as

Rule 1: IF the maximum of violated member stress constraints is PS and all the displacement constraints are inactive

THEN the change of the corresponding member cross sectional area is PS.

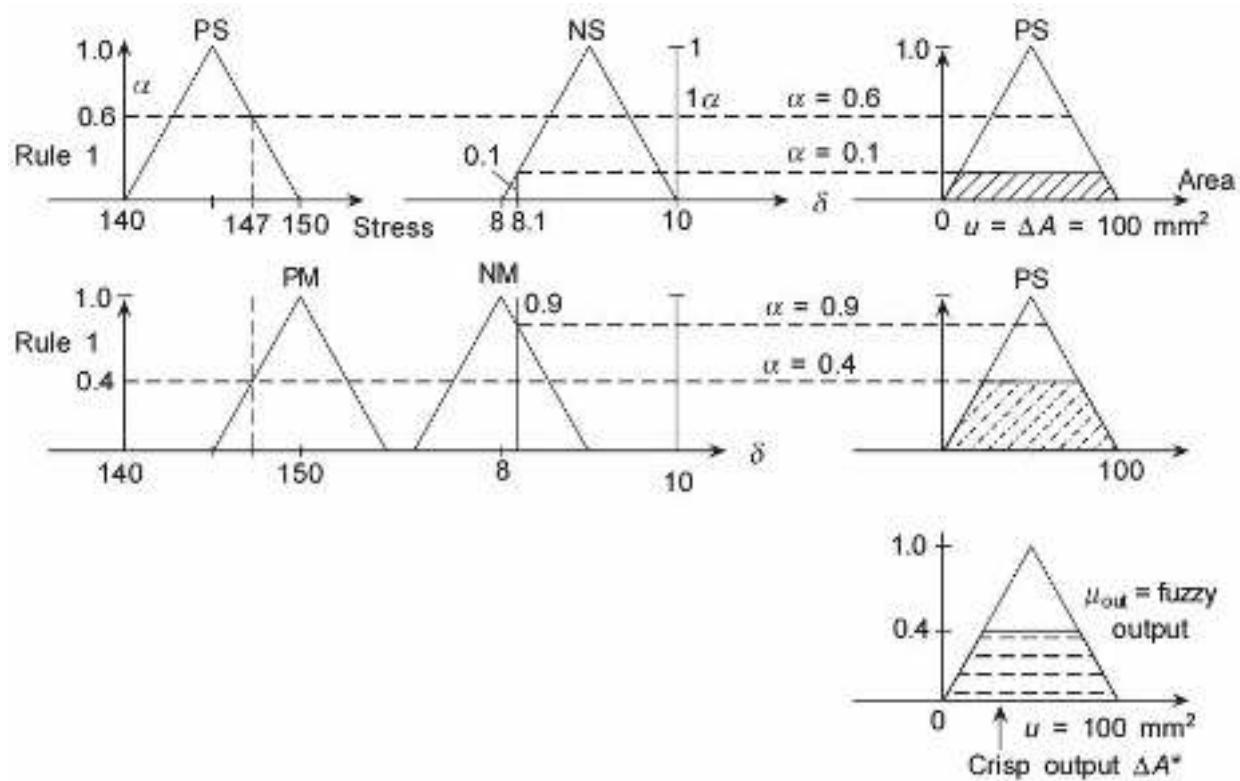
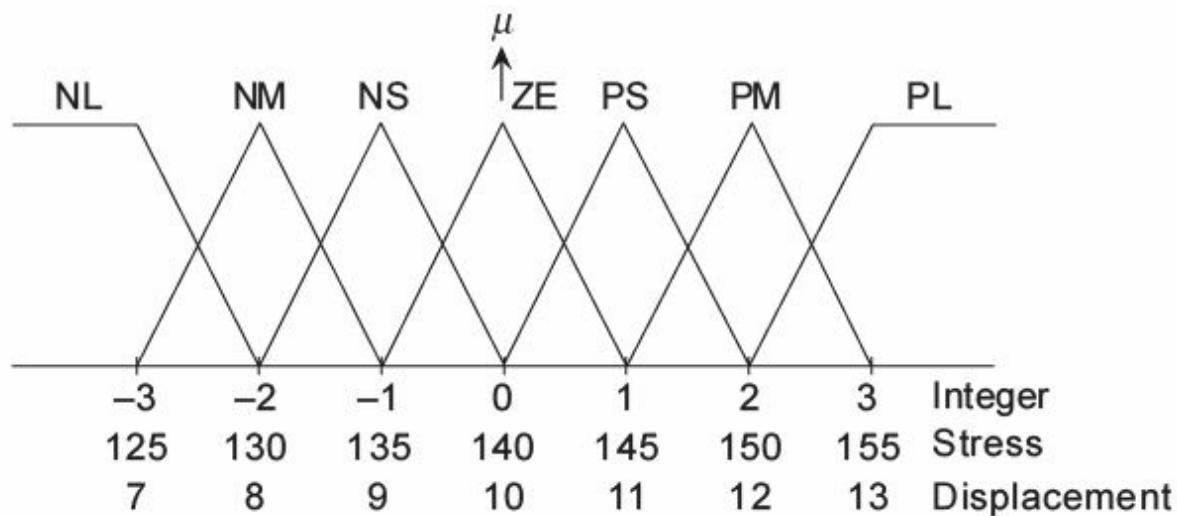
Rule 2: IF all constraints are inactive and the minimum of member stress constraints is NL

THEN the change of the corresponding member cross-sectional area is NS.

As an input, the constraint C_{ij} is usually classified as 1. active for ZE,
2. inactive for NL, NM, and NS, and
3. violated for PS, PM, and PL.

On the other hand for the output, the modification of the member cross-sectional areas has the possibilities, NL, NM, NS, ZE, PS, PM, and PL.

Fuzzy controller inputs are usually crisp numbers. Fuzzy inputs may also

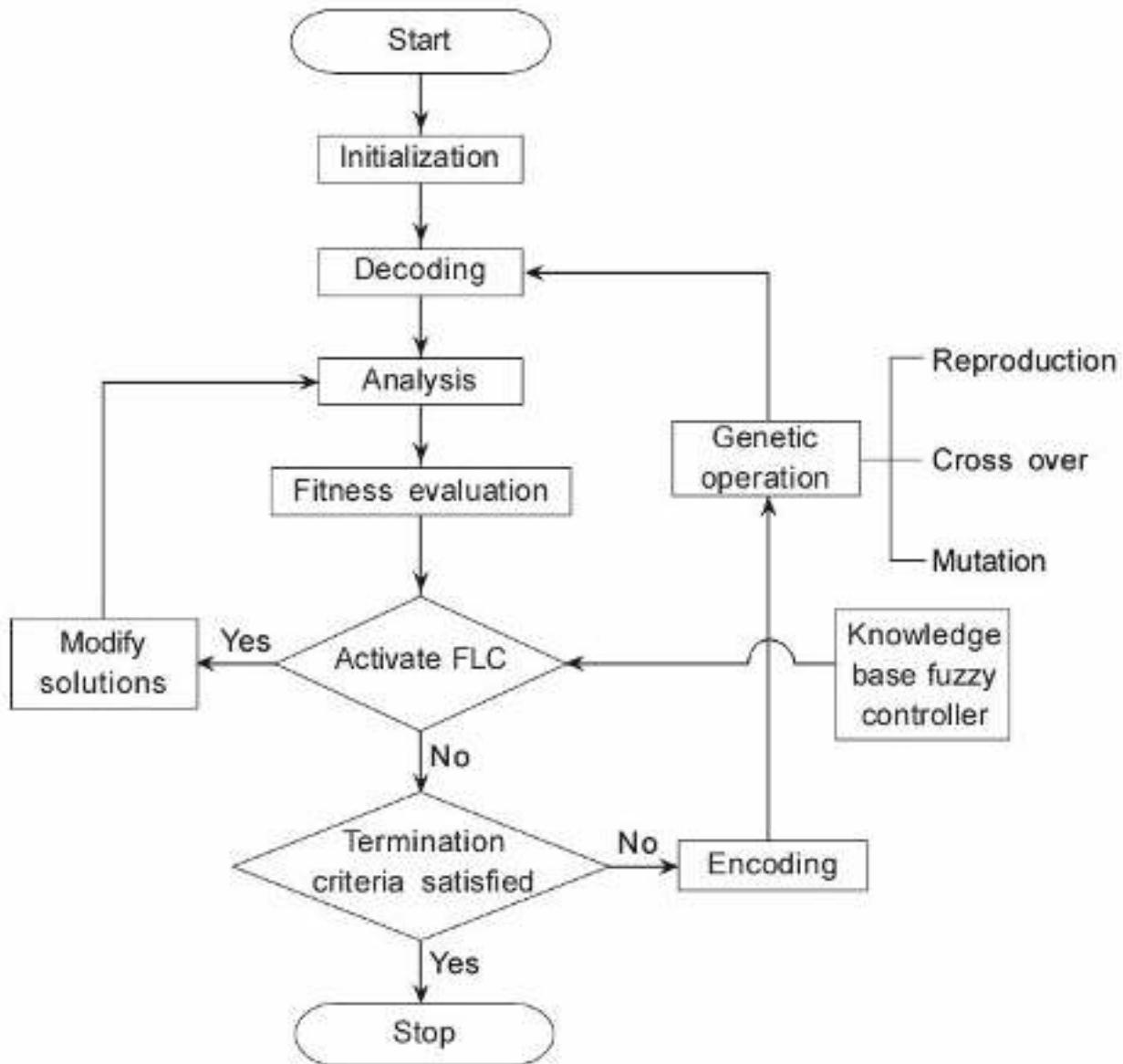


be considered in the case of uncertain or noisy measurements and crisp numbers may be defuzzified. Figure 15.10 shows the application of Rule 1.

The degree of fulfilment (DOF) of Rule 1 is 0.4. The total fuzzy output μ_{out} is the union of the two outputs shown in Fig. 15.9. At this point we need to defuzzify and obtain the crisp value for ΔA^* representative of μ_{out} as explained in Chapters 6 and 7.

Fig. 15.9 Membership function for fuzzy variable.

Fig. 15.10 Evaluation of the Rule 1.



15.7 FLC-GA BASED STRUCTURAL OPTIMIZATION

First, coding scheme is to be defined and the initial population is produced.

The computation with genetic operators is used to evaluate fitness function with respect to the objective function. Figure 15.11 shows the FLC-GA based optimization procedure. Using FLC we can get the expert's experience in

fuzzy rule base of FLC. Hence, the search can react optimum solution quickly. As a result, computing time is very much reduced. The predefined probability and fuzzy representation of design constraints causes FLC to reduce the risk of premature problem solution caused by improper rule.

Fig. 15.11 Flow chart of FLC GA based optimization.

15.8 APPLICATIONS

15.8.1 Optimum Truss

Yang and Soh (2000) have found out the optimum truss structure within the given design domain as shown in Fig. 15.12 . All the truss members are selected from a set of 30 standard sections

(i.e. W 14 × 22 through W14 × 426). $E = 201 \text{ GPa}$, $f_y = 248.8 \text{ MPa}$, and $\rho = 78.51 \text{ kN/cu m}$, allowable tensile stress $< 0.6 f_y$, allowable slenderness ratio L/r is 300 for tension members and 200 for compression members, the length of the members $5 \text{ m} < L < 35 \text{ m}$, the deflection

$\delta < L/1000$ (i.e 70 mm), and the allowable buckling stress must be determined from buckling consideration. Coordinates of the joints must be randomly selected with a step of 0.5 m. Soh and Yang (1996) used a population size of 2000, maximum generation of runs 100, and the probabilities of reproduction, cross over and mutation 0.1, 0.8 and 0.1 respectively. The solutions obtained by the authors are given in Table 15.2.

Table 15.2 Optimal results

Member

Section

Member

Section

Position

Value

1–2

W14 × 74

2–5

W14 × 61

X5

3.5

2–3

W14 × 109

2–6

W14 × 61

Y5

6.0

3–4

W14 × 132

3–6

W14 × 74

X6

12.5

4–9

W14 × 211

4–7

W14 × 82

Y6

9.0

1–5

W14 × 132

4–8

W14 × 61

X7

23.5

5–6

W14 × 132

W

451.63 kN

Y7

10.0

6–7

W14 × 176

Popl size

2000

X8

35.0

7–8

W14 × 211

Generation

50

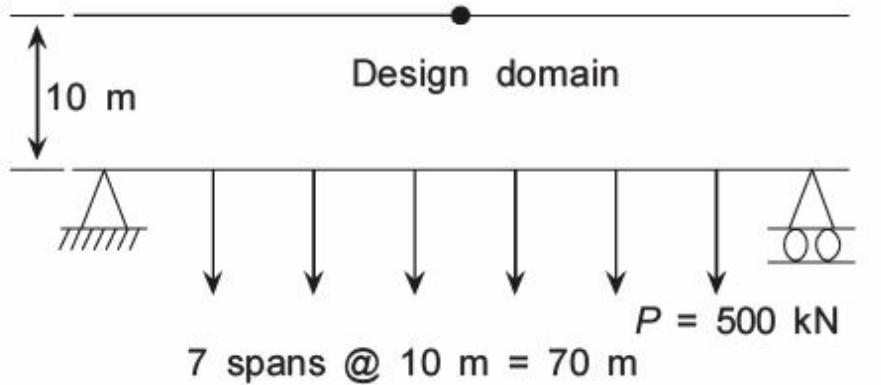
Y8

10.0

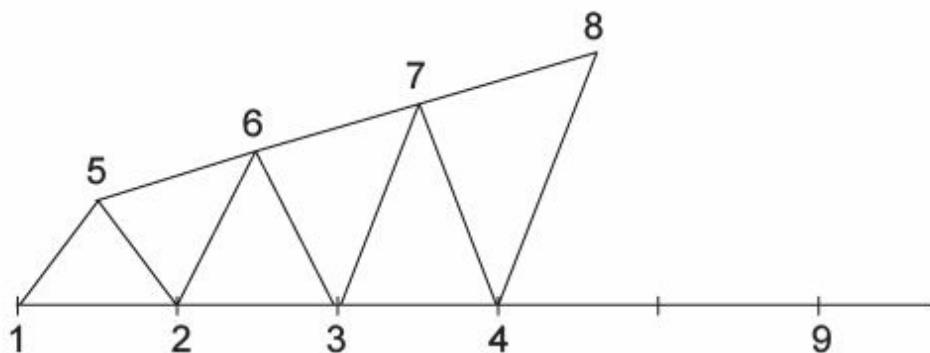
Iteration

100,000

.



(a) Design domain



(b) Optimal truss

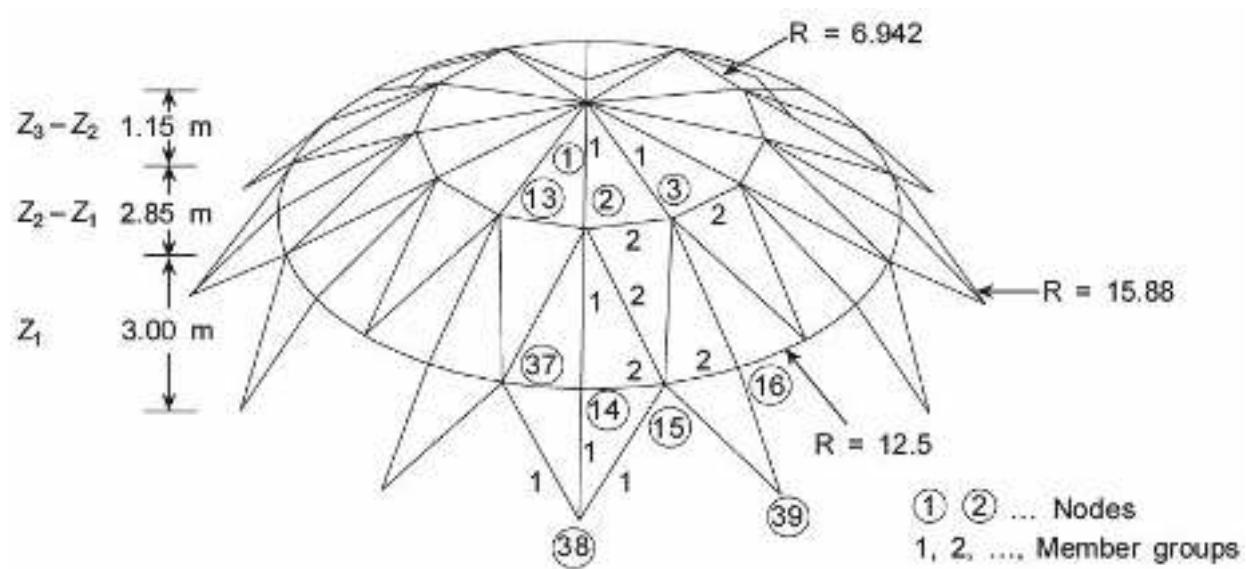
Fig. 15.12 Optimal truss configuration.

15.8.2 112 Bar Dome Space Truss

Soh and Yang (1996) have analysed 112 bar space truss dome shown in Fig.

15.13. For clarity, the joint members are circled and the members of the dome have been categorised into two groups. The structure has been subjected to a vertical loading at each unsupported joint. The detailed data are 5 kN at node 1, 0.4 kN at nodes 17, 23, 29, and 35, 1.2 kN at nodes 16, 18, 22, 24, 28, 30, 34 and 36 and 2 kN for other joints. All loads are acting downwards. All the joints were allowed to move vertically within the limit defined by fundamental requirements. Thus, there are total of five design variables, which include two sizing variables and three independent coordinate variables (Z_1, Z_2, Z_3) as shown in Fig. 15.13 . The objective is to minimize the weight of the structure subjected to constraints in stress,

displacement, and buckling. The properties and allowable values are given in Table 15.3.



$$\sigma_i^b = \frac{k\pi^2 E}{\lambda_i^2}$$

$$\left\{ \begin{array}{l} \sigma_i^b = \sigma_y \left(1 - \frac{\lambda_i^2}{2C^2} \right) / n \\ \text{where } n = \frac{5}{3} + \frac{3\lambda_i}{8c} - \frac{\lambda_i^3}{8C^3} \end{array} \right.$$

Fig. 15.13 Bar dome.

Table 15.3 Data for example—112 bar dome space truss

Young's modulus

210 GPa

Allowable stress

165 MPa

Density

100 kN/cu m

Allowable deflection

± 20 mm at joints

1, 17, 23, 29, 35

Lower and upper area size

150–1000 sq mm

The buckling stress constraint σ_{bi} if member ‘ i ’ is computed as: For $\lambda_i > C$, i.e. elastic buckling

(15.13a)

For $\lambda_i < C$, i.e. plastic buckling

(15.13b)

where $\lambda_i = Li / ri$; $r = 0.4993 A^{0.6777}$ for tubular sections; σ_y is the yield stress given as

$$\sigma_y = \sigma_{all}/0.6$$

$$k = 12/13$$

$$C = \sqrt{\frac{2\pi^2 E}{\sigma_y}}$$

An optimum shape design having minimum steel weight of 33.278 kN as shown in

Table 15.4 was obtained after 41 generations than compared with the steel weight of 34.51 kN using pure GA approach. With 72 generations, the fuzzy-GA hybrid approach given by Soh and Yang (1996) has resulted in 43% reduction in required number of iterations and 3.57% reduction in weight.

Table 15.4 Optimum results for 112 bar dome

Design variable

Results obtained by Yang and Soh (1996)

A1

597.8

A2

538.33

Z1

2.85 m

Z2

6.11 m

Z3

7.45 m

W

33.27 kN

Reduction in weight compared with ordinary GA

3.57%

SUMMARY

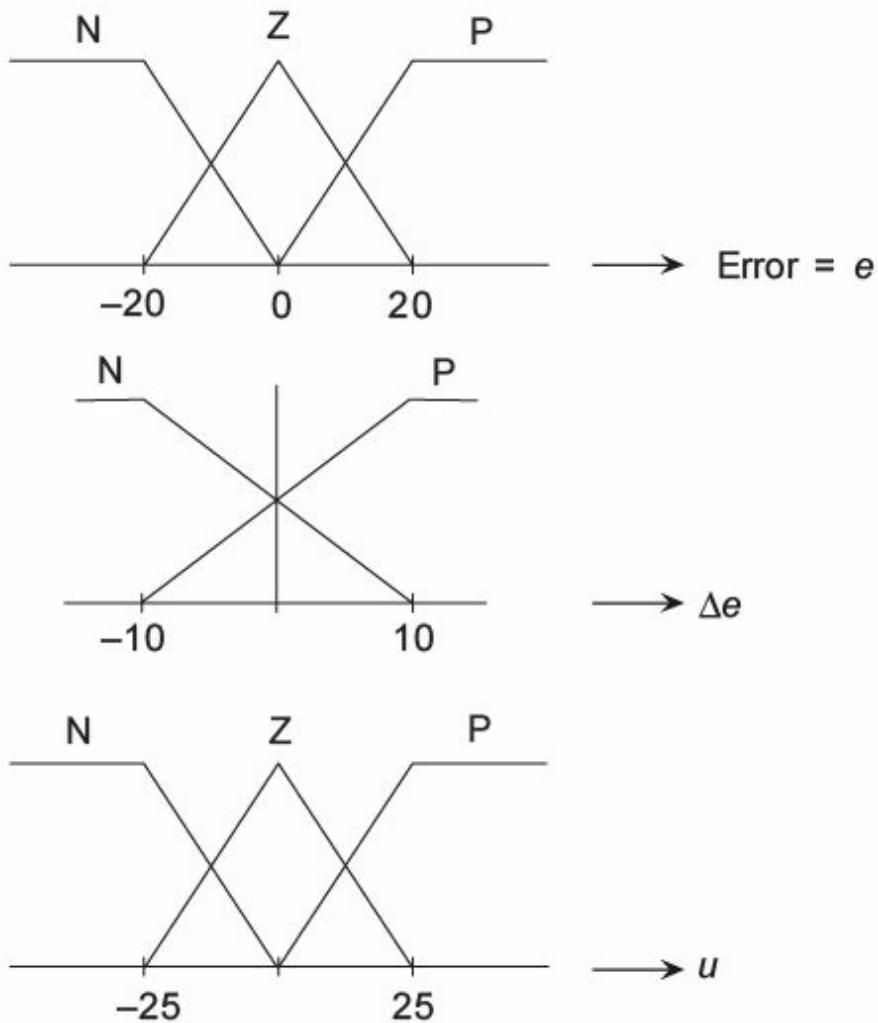
The hybrid fuzzy-GA approach produces the least weight design of structures with an optimal shape, not by intuition but by the automated GA based simulation procedure coupled with expert knowledge and experience.

The imprecise and vague information in structural design process, especially design constraints, are dealt with using fuzzy set theory.

The hybrid approach is able to integrate expert knowledge and experience with the GA search procedure by using an FLC.

GA search procedure can be guided in a more intelligent way by the artificial intelligence based simulation and hence this has potential in shape optimization.

Illustrative examples show that the proposed method can reduce the required computational time and enhance the search efficiency of pure GA.



PROGRAMMING ASSIGNMENT

P15.1 Consider Fig. P15.1. The fuzzy control system uses inputs of e , Δe , and output variable u . Determine output u for $e = 100\%$ and $\Delta e = -2\%$.

Use fuzzy IF-THEN rules as given in

Table P15.1.

Fig. P15.1

TABLE P15.1 IF-THEN RULE

e

Δe

THEN U

N

N

P

N

P

P

IF

Z

N

Z

Z

P

Z

P

N

N

P

P

N

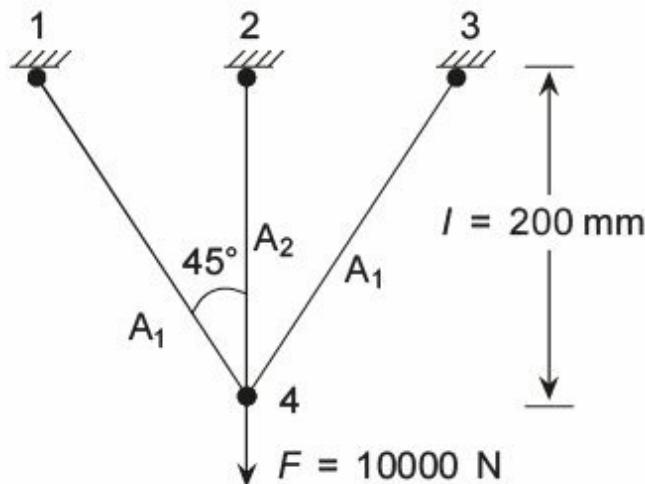
Consider a three bar truss shown in Fig. P15.2 . The data assumed is $E = 200 \text{ GPa}$, maximum stress 147.15 MPa , and the maximum displacement 5

$$\delta = \frac{Fl}{E(0.707A_1 + A_2)}$$

the stress

$$\sigma_1 = \frac{0.5F}{(0.707A_1 + A_2)}$$

$$\sigma_2 = 2\sigma_1$$



mm. The truss is subjected to vertical load of 10 kN at node 4 downwards.

Perform fuzzy genetic optimization to find out the areas of inclined member and vertical member (the structure is symmetric). The minimum and maximum areas to be used are 120 sq mm and 400 sq mm respectively. The deflection at the node 4 due to vertical load can be taken as

Fig. P15.2.

Tolerance for each stress unit can be taken as 5 MPa and for displacement as 0.5 mm .

The following rules can be used.

Rule 1: IF the maximum violated member stress constraint is PS and all the displacement constraints are inactive

THEN the change of the corresponding member cross sectional area is PS.

Rule 2: IF all displacement constraints are inactive and the minimum of member constraint is NL

THEN the change of the corresponding cross sectional area is NS.

Each unit of increase in cross-sectional area can be assumed as 10 sq mm.

Write a program of FLC-GA and get the optimal design.

(Assume for input: ZE—active; NL, NM, NS—inactive; PS, PM, PL—
violated, and for output all possibilities—NL, NM, NS, PS, PM, PL, ZE)

SUGGESTED FURTHER READING

Kruse, R., J. Gebhardt, and F. Klawonn (1994), *Foundations of Fuzzy Systems*, Wiley, Chichester, UK.

Rao, S.S., K. Sundararaju, B.G. Prakash, and C. Balakrishna (1990), Fuzzy Goal Programming Approach for Structural Optimization, *AIAA, J.* 30(5), pp. 1425–1432.

Wang, G.Y., and W.Q. Wang (1985b), Fuzzy Optimum Design of Aseismic Structures, *Earthquake Engg. and Structural Design*, 13(6), pp. 827–837.

REFERENCES

Bellman, R.E. and L.A. Zadeh (1970), Decision Making in a Fuzzy Environment, *Management Science*, 17(4), pp. 141–164.

Deb, K., D.K. Prathihar, and A. Ghosh (1998), Learning to Avoid Moving Obstacles Optimally for Mobile Robots using Genetic Fuzzy Approach, *Parallel Problem Solving from Nature*, (Eds). A.E. Eisen, T. Bach, M.

Schoenauer, and H.P. Schwefel, 5, pp. 583–592.

Deb, K. (1999), An Introduction to Genetic Algorithms, *Sadhana*, Vol. 24, Parts 4 & 5, August & October 1999, pp. 293–315.

Holland, J.H. (1975), Adaptation in Natural and Artificial Systems (Ann Arbor. University of Michigan Press).

Rao, S.S. (1987), Optimum Design of Structures in a Fuzzy Environment, *AIAA, J.*, 25(12),

pp. 1633–1636

Soh, C.K. and J. Yang (1996), Fuzzy Controlled Genetic Algorithm Search for Shape Optimization, ASCE, *Joul. of Computing in Civil Engg.* , Vol. 10, No. 2, April, pp. 143–150.

Wang, G.Y. and W.Q. Wang (1985a), Fuzzy Optimum Design of Structures, *Engg. Optimization*, Reading, U.K. 8(4), pp. 291–300.

Wang, G.Y. and W.Q. Wang (1985b), Fuzzy Optimum Design of Aseismic Structures, *Earthquake Engineering and Structural Design*, 13(6), pp.

827–837.

Yang, Y. and C.K. Soh (2000), Fuzzy Logic Integrated Genetic Programming for Optimization and Design, ASCE, *Joul. of Computing in Civil Engg.*, Vol. 14. No. 4, October, pp. 249–254.

Zadeh, L.A. (1987), *Fuzzy Sets and Application*, John Wiley & Sons, New York, USA.

Index

Activation function, 14

ADALINE network, 28

Adaptive backpropagation, 75

Adaptive resonance theory (ART), 5, 117

ART2, 140

network, 125

Alleles, 229

Analog pattern, 140

Angle ply laminates, 272

ANSYS, 83

Ant colony optimization, 226

Artificial intelligence, 1

Artificial neuron model, 13

Associative memory

- dynamic, 88
- nonrecurrent, 88
- recurrent, 88
- static, 87

Attentional layer, 117

Attentioned system, 128

Augmented backpropagation, 77

Autoassociative memory, 87, 88

Autocorrelators, 88, 89

Average fitness, 271

Backpropagation, 34

Backpropagation networks, 5

BAM energy function, 92

BAM operations, 91

Binary encoding, 234

Bit-wise operator, 263

Boltzmann constant, 245

Boltzmann selection, 242

Bottom-up weights, 127

Buckling strength, 418

Cellular automata, 226

Character recognition, 145

Chinese character, 151

Chromosomes, 4, 306

Classification of soil, 64, 146

Comparison layer, 129

Complement coding, 359, 360

Complement operator, 263

Composite laminates, 272

Conscious networks, 81

Constraints, 278

Continuous inversion, 259

Control parameters, 266

Convergence, 271

Correlation matrix encoding, 393

Correlation minimum encoding, 395

Cost function, 422

Crisp relations, 179–181

Crisp set

operations 161–163

properties, 163

Cross-over, 4, 254, 425

rate, 258

Cumulative probability, 243

Darwinian theory, 237

Decomposition rules, 404

Deletion, 253

Dominance, 253

Earthquake damage evaluation, 347, 353

Electrical load forecasting, 323

Elitism, 248

Error, 45

Euclildead norm, 45

Evolution strategy, 226

Exclusive OR (XOR), 264

Exponential BAM, 99–101

Fabric defect identification, 107, 110

FAM bank of rules, 390

FAM rule base, 400, 403, 404, 405

Fast learning, 127

FEAST, 285

FEAST-C, 274

Feature extraction, 372

Fine tuning, 81

Fitness function, 4, 228, 237, 309

Fixed increment learning algorithm, 27

Fuzzy ART, 126

Fuzzy ARTMAP, 358

Fuzzy associative memory, 6, 389, 390

Fuzzy BP

architecture, 331

inference, 339

learning, 333

training, 336

Fuzzy constant, 419

Fuzzy genetic hybrid, 6

Fuzzy Hebb FAMs, 395, 414

Fuzzy inference, 4

Fuzzy logic, 417

controller, 424

Fuzzy neuron, 330

Fuzzy relations, 3, 182

Fuzzy rule base, 4

Fuzzy set

height, 398

membership, 3, 169

operations, 3, 171–176
properties, 176

Genetic algorithm based backpropagation
coding, 306
convergence, 313
fitness function, 309
reproduction, 311
weight extraction, 308

GA with memory, 288

Gene, 229

Generation, 241

gap, 249

Genetic algorithm, 225, 228

Genetic fuzzy, 2

Genetic inheritance operators, 4

Genome, 229

Genotype, 228

Graphical method of inference, 392, 406, 408, 414

Hard computing, 417

Hard hitting threshold network, 35

Hetero-associative memory, 87, 88

Heterocorrelators, 88, 91

Hexadecimal encoding, 234

High performance concrete, 422

Hooke–Jeeves method, 287

Hopfield associative memory, 89

Hot extrusion of steel, 65

Human brain, 12

Hybrid systems

- auxiliary, 298
- definition, 298
- embedded, 299
- sequential, 298

Hyperbolic tangent function, 16

Image recognition application, 370

Inference session, 56

Inheritance operator, 253

Inventory, 225

Inversion, 253, 259

Inverted pendulum balancing, 389, 404, 414

Isodata, 117

Job shop scheduling, 289

Journal bearing, 59

K-factor design, 322

K-means algorithm, 117

Knapsack problem, 231

Knowledge base evaluation, 347, 348

Learning coefficient, 74

Learning epoch, 46

Linear programming, 225

Linearly separable, 25

Liquid limit, 64

Local optimization, 283

Locus, 229

LR type fuzzy numbers

operations, 330

trapezoidal, 329

triangular, 329

Machine learning, 291

MADALINE network, 28–30

Masking, 264

Mass inversion, 259

Match tracking, 359, 360

Mating, 253

MATLAB, 83

Matrix cross-over, 257

McCulloch Pitt's model, 16

Membership function, 424

Migration, 253

Mix design, 422

Moment based invariants, 372

Momentum, 52

coefficient, 73

Multilayer feedforward, 18, 34

Multilayer perceptron, 41

Multilevel optimization, 289

Multiple training encoding, 87, 95, 96

Multipoint cross-over, 255

Mutation, 4, 229, 425

rate, 261

Neural networks
applications, 30
architecture types, 2, 16, 17, 18
characteristics, 2, 19
history, 22
learning methods, 2, 3, 19, 20
neuron structure, 13
taxonomy, 21
Neuro fuzzy hybrid, 2, 5, 6
NEURONET, 59
Nonlinear programming, 225
Non-traditional search, 250
Normalization, 360
Objective function, 228
OR operator, 264
Pareto-optimal solution, 289
Penalty function, 278
Perceptron, 23–27, 35
Permutation encoding, 235
Plastic limit, 64

Plasticity, 125

Population clustering, 245

Queuing, 225

Radial basis function, 38

Rank selection, 242, 247

Recognition layer, 127

Recognition of characters, 105

Recurrent, 18

networks, 81

Regeneration, 260

Reproduction, 4, 242, 311

Resonance, 117, 129, 361

Robotics, 291

Roulette wheel, 242

Satellite images, 150

Scheduling, 225

Schema, 271

Segregation, 253, 260

Selective operation, 242

Selective pressure, 245

Sequential learning, 80

Sharing, 253

fitness, 289

Shift operator, 264

Sigmoidal function, 15

Sigmoidal gain, 40, 74

Signal processing, 291

Signum function, 15

Simplified ART, 126

Simplified BAM, 101–104

Simplified Fuzzy ARTMAP, 6

inference, 364

training, 362

Simulated annealing, 227

Single association FAM, 390

Single layer feedforward network, 17

Single site, 254

Slenderness, 418

Slow learning, 127

Soft computing, 1, 417

Sommerfeld number, 60

Speciation, 253

Squashed-S function, 40

Stability, 125

-plasticity, 11, 12

Steel dome, 284

Steepest descent, 47

Step function, 14

Stochastic, 226

remainder, 245

String's fitness, 237

Survival of the fittest, 4

Swarm intelligence, 227

Symbolic logic, 3

Synapses, 43

Three-bar truss, 277

Threshold value, 70

Thresholding function, 14, 15

Time tabling, 289

Top-down weights, 127

Tournament selection, 212, 245

Translocation, 253

Transportation, 225

Travelling salesman, 289

Tree encoding, 236

Truck backer-upper system, 389, 411, 414

Two bar pendulum, 238

Two point cross-over, 255

Two-third rule, 129

Unconstrained optimization, 230

Uniform cross-over, 256

Unipolar multimodal, 38

Value encoding, 236

VCdim, 73

Vector quantization, 117, 118

Vigilance parameter, 359, 360

Vigilance test, 126

Winner take all, 129

XOR problem, 26

Document Outline

- [NEURAL NETWORKS, FUZZY LOGIC, AND GENETIC ALGORITHMS: Synthesis and Applications](#)
- [Copyright](#)
- [Dedication](#)
- [Table of Contents](#)
- [Preface](#)
 - [ORGANIZATION](#)
- [1. Introduction to Artificial Intelligence Systems](#)
 - [1.1 Neural Networks](#)
 - [1.2 Fuzzy Logic](#)
 - [1.3 Genetic Algorithms](#)
 - [1.4 Structure of This Book](#)
 - [SUMMARY](#)
 - [REFERENCES](#)
- [Part 1: Neural Networks](#)
- [2. Fundamentals of Neural Networks](#)
 - [2.1 Basic Concepts of Neural Networks](#)
 - [2.2 Human Brain](#)
 - [2.3 Model of an Artificial Neuron](#)
 - [2.4 Neural Network Architectures](#)
 - [2.4.1 Single Layer Feedforward Network](#)
 - [2.4.2 Multilayer Feedforward Network](#)
 - [2.4.3 Recurrent Networks](#)
 - [2.5 Characteristics of Neural Networks](#)
 - [2.6 Learning Methods](#)
 - [2.7 Taxonomy of Neural Network architectures](#)
 - [2.8 HISTORY OF NEURAL NETWORK RESEARCH](#)
 - [2.9 Early Neural Network Architectures](#)
 - [2.9.1 Rosenblatt's Perceptron](#)
 - [XOR Problem](#)
 - [Algorithm 2.1](#)
 - [2.9.2 ADALINE Network](#)
 - [2.9.3 MADALINE Network](#)

- [2.10 Some Application Domains](#)
- [SUMMARY](#)
- [PROGRAMMING ASSIGNMENT](#)
- [SUGGESTED FURTHER READING](#)
- [REFERENCES](#)
- [3. Backpropagation Networks](#)
 - [3.1 ARCHITECTURE OF A BACKPROPAGATION NETWORK](#)
 - [3.1.1 The Perceptron Model](#)
 - [3.1.2 The Solution](#)
 - [3.1.3 Single Layer Artificial Neural Network](#)
 - [3.1.4 Model for Multilayer Perceptron](#)
 - [3.2 BACKPROPAGATION LEARNING](#)
 - [3.2.1 Input Layer Computation](#)
 - [3.2.2 Hidden Layer Computation](#)
 - [3.2.3 Output Layer Computation](#)
 - [3.2.4 Calculation of Error](#)
 - [3.2.5 Training of Neural Network](#)
 - [3.2.6 Method of Steepest Descent](#)
 - [3.2.7 Effect of Learning Rate '\\$eta\\$\\$'](#)
 - [3.2.8 Adding a Momentum Term](#)
 - [3.2.9 Backpropagation Algorithm](#)
 - [Algorithm 3.1 \(Backpropagation Learning Algorithm\)](#)
 - [3.3 ILLUSTRATION](#)
 - [3.4 APPLICATIONS](#)
 - [3.4.1 Design of Journal Bearing](#)
 - [3.4.2 Classification of Soil](#)
 - [3.4.3 Hot Extrusion of Steel](#)
 - [3.5 EFFECT OF TUNING PARAMETERS OF THE BACKPROPAGATION NEURAL NETWORK](#)
 - [3.6 SELECTION OF VARIOUS PARAMETERS IN BPN](#)
 - [3.6.1 Number of Hidden Nodes](#)
 - [3.6.2 Momentum Coefficient \\$alpha\\$\\$](#)
 - [3.6.3 Sigmoidal Gain \\$lambda\\$\\$](#)
 - [3.6.4 Local Minima](#)
 - [3.6.5 Learning Coefficient \\$eta\\$\\$](#)

- [3.7 VARIATIONS OF STANDARD BACKPROPAGATION ALGORITHM](#)
 - [3.7.1 Decremental Iteration Procedure](#)
 - [3.7.2 Adaptive Backpropagation \(Accelerated Learning\)](#)
 - [3.7.3 Genetic Algorithm Based Backpropagation](#)
 - [3.7.4 Quick Prop Training](#)
 - [3.7.5 Augmented BP Networks](#)
 - [3.7.6 Sequential Learning Approach for Single Hidden Layer Neural Networks](#)
- [3.8 RESEARCH DIRECTIONS](#)
 - [3.8.1 New Topologies](#)
 - [3.8.2 Better Learning Algorithms](#)
 - [3.8.3 Better Training Strategies](#)
 - [3.8.4 Hardware Implementation](#)
 - [3.8.5 Conscious Networks](#)
- [SUMMARY](#)
- [PROGRAMMING ASSIGNMENT](#)
- [REFERENCES](#)
- [4. Associative Memory](#)
 - [4.1 AutoCorrelators](#)
 - [4.2 HeteroCorrelators: Kosko's Discrete BAM](#)
 - [4.2.1 Addition and Deletion of Pattern Pairs](#)
 - [4.2.2 Energy Function for BAM](#)
 - [4.3 WANG ET AL.'S MULTIPLE TRAINING ENCODING STRATEGY](#)
 - [Algorithm 4.1 \(Wang et al.'s Multiple Training Encoding Strategy.\)](#)
 - [4.4 EXPONENTIAL BAM](#)
 - [4.4.1 Evolution Equations](#)
 - [4.5 Associative Memory for real-coded pattern pairs](#)
 - [4.5.1 Input Normalization](#)
 - [4.5.2 Evolution Equations](#)
 - [Algorithm 4.2 \(Simplified Bi-directional Associative Memory\)](#)
 - [4.6 Applications](#)
 - [4.6.1 Recognition of Characters](#)

- [4.6.2 Fabric Defect Identification](#)
 - [4.7 RECENT TRENDS](#)
 - [SUMMARY](#)
 - [PROGRAMMING ASSIGNMENT](#)
 - [REFERENCES](#)
- [5. Adaptive Resonance Theory](#)
 - [5.1 INTRODUCTION](#)
 - [5.1.1 Cluster Structure](#)
 - [5.1.2 Vector Quantization](#)
 - [FOR THRESHOLD DISTANCE OF 2](#)
 - [FOR THRESHOLD DISTANCE OF 4.5](#)
 - [5.1.3 Classical ART Networks](#)
 - [5.1.4 Simplified ART Architecture](#)
 - [5.2 ART1](#)
 - [5.2.1 Architecture of ART1](#)
 - [5.2.2 Special Features of ART1 Models](#)
 - [5.2.3 ART1 Algorithm](#)
 - [Algorithm 5.1 \(Art1 Algorithm\)](#)
 - [5.2.4 Illustration](#)
 - [5.3 ART2](#)
 - [5.3.1 Architecture of ART2](#)
 - [5.3.2 ART2 Algorithm](#)
 - [Algorithm 5.2 \(ART2 Algorithm\)](#)
 - [5.3.3 Illustration](#)
 - [5.4 APPLICATIONS](#)
 - [5.4.1 Character Recognition Using ART1](#)
 - [5.4.2 Classification of Soil \(Rajasekaran et al., 2001\)](#)
 - [5.4.3 Prediction of Load from Yield Patterns of Elastic-Plastic Clamped Square Plate](#)
 - [Output of the Example 5.4](#)
 - [5.4.4 Chinese Character Recognition—Some Remarks](#)
 - [5.5 Sensitiveness of Ordering of Data](#)
 - [SUMMARY](#)
 - [PROGRAMMING ASSIGNMENT](#)
 - [SUGGESTED FURTHER READING](#)
 - [REFERENCES](#)

- [Part 2: FUZZY LOGIC](#)
- [6. Fuzzy Set Theory](#)
 - [6.1 FUZZY VERSUS CRISP](#)
 - [6.2 CRISP SETS](#)
 - [6.2.1 Operations on Crisp Sets](#)
 - [6.2.2 Properties of Crisp Sets](#)
 - [6.2.3 Partition and Covering](#)
 - [6.3 FUZZY SETS](#)
 - [6.3.1 Membership Function](#)
 - [6.3.2 Basic Fuzzy Set Operations](#)
 - [6.3.3 Properties of Fuzzy Sets](#)
 - [6.4 CRISP RELATIONS](#)
 - [6.4.1 Cartesian Product](#)
 - [6.4.2 Other Crisp Relations](#)
 - [6.4.3 Operations on Relations](#)
 - [6.5 FUZZY RELATIONS](#)
 - [6.5.1 Fuzzy Cartesian Product](#)
 - [6.5.2 Operations on Fuzzy Relations](#)
 - [SUMMARY](#)
 - [PROGRAMMING ASSIGNMENT](#)
 - [SUGGESTED FURTHER READING](#)
 - [REFERENCE](#)
- [7. Fuzzy Systems](#)
 - [7.1 CRISP LOGIC](#)
 - [7.1.1 Laws of Propositional Logic](#)
 - [7.1.2 Inference in Propositional Logic](#)
 - [7.2 PREDICATE LOGIC](#)
 - [7.2.1 Interpretations of Predicate Logic Formula](#)
 - [7.2.2 Inference in Predicate Logic](#)
 - [7.3 Fuzzy Logic](#)
 - [7.3.1 Fuzzy Quantifiers](#)
 - [7.3.2 Fuzzy Inference](#)
 - [7.4 FUZZY RULE BASED SYSTEM](#)
 - [7.5 Defuzzification](#)
 - [7.6 Applications](#)
 - [7.6.1 Greg Viot's Fuzzy Cruise Controller](#)

- [7.6.2 Air Conditioner Controller](#)
 - [SUMMARY](#)
 - [PROGRAMMING ASSIGNMENT](#)
 - [SUGGESTED FURTHER READING](#)
 - [REFERENCES](#)
- [Part 3: GENETIC ALGORITHMS](#)
- [8. Fundamentals of Genetic Algorithms](#)
 - [8.1 GENETIC ALGORITHMS: HISTORY](#)
 - [8.2 BASIC CONCEPTS](#)
 - [8.2.1 Biological Background](#)
 - [8.3 CREATION OF OFFSPRINGS](#)
 - [8.3.1 Search Space](#)
 - [8.4 WORKING PRINCIPLE](#)
 - [8.5 ENCODING](#)
 - [8.5.1 Binary Encoding](#)
 - [8.5.2 Octal Encoding \(0 to 7\)](#)
 - [8.5.3 Hexadecimal Encoding \(0123456789ABCDEF\)](#)
 - [8.5.4 Permutation Encoding](#)
 - [8.5.5 Value Encoding](#)
 - [8.5.6 Tree Encoding](#)
 - [8.6 FITNESS FUNCTION](#)
 - [8.7 REPRODUCTION](#)
 - [8.7.1 Roulette-wheel Selection](#)
 - [8.7.2 Boltzmann Selection](#)
 - [8.7.3 Tournament Selection](#)
 - [8.7.4 Rank Selection](#)
 - [8.7.5 Steady-state Selection](#)
 - [8.7.6 Elitism](#)
 - [8.7.7 Generation Gap and Steady-state Replacement](#)
 - [SUMMARY](#)
 - [PROGRAMMING ASSIGNMENT](#)
 - [REFERENCES](#)
- [9. Genetic Modelling](#)
 - [9.1 INHERITANCE OPERATORS](#)
 - [9.2 CROSS OVER](#)
 - [9.2.1 Single-site Cross Over](#)

- [9.2.2 Two-point Cross Over](#)
 - [9.2.3 Multi-point Cross Over](#)
 - [9.2.4 Uniform Cross Over](#)
 - [9.2.5 Matrix Cross Over \(Two-dimensional Cross Over\)](#)
 - [9.2.6 Cross Over Rate](#)
- [9.3 INVERSION AND DELETION](#)
 - [9.3.1 Inversion](#)
 - [9.3.2 Deletion and Duplication](#)
 - [9.3.3 Deletion and Regeneration](#)
 - [9.3.4 Segregation](#)
 - [9.3.5 Cross Over and Inversion](#)
- [9.4 MUTATION OPERATOR](#)
 - [9.4.1 Mutation](#)
 - [9.4.2 Mutation Rate Pm](#)
- [9.5 BIT-WISE OPERATORS](#)
 - [9.5.1 One's Complement Operator](#)
 - [9.5.2 Logical Bit-wise Operators](#)
 - [9.5.3 Shift Operators](#)
- [9.6 BIT-WISE OPERATORS USED IN GA](#)
- [9.7 GENERATIONAL CYCLE](#)
- [9.8 CONVERGENCE OF GENETIC ALGORITHM](#)
- [9.9 APPLICATIONS](#)
 - [9.9.1 Composite Laminates](#)
 - [9.9.2 Constrained Optimization](#)
- [9.10 MULTI-LEVEL OPTIMIZATION](#)
- [9.11 REAL LIFE PROBLEM](#)
- [9.12 DIFFERENCES AND SIMILARITIES BETWEEN GA AND OTHER TRADITIONAL METHODS](#)
- [9.13 ADVANCES IN GA](#)
- [SUMMARY](#)
- [PROGRAMMING ASSIGNMENT](#)
- [SUGGESTED FURTHER READING](#)
- [SOME USEFUL WEBSITES](#)
- [REFERENCES](#)
- [Part 4: HYBRID SYSTEMS](#)

- [10. Integration of Neural Networks, Fuzzy Logic, and Genetic Algorithms](#)
 - [10.1 HYBRID SYSTEMS](#)
 - [10.1.1 Sequential Hybrid Systems](#)
 - [10.1.2 Auxiliary Hybrid Systems](#)
 - [10.1.3 Embedded Hybrid Systems](#)
 - [10.2 NEURAL NETWORKS, FUZZY LOGIC, AND GENETIC](#)
 - [10.2.1 Neuro-Fuzzy Hybrids](#)
 - [10.2.2 Neuro-Genetic Hybrids](#)
 - [10.2.3 Fuzzy-Genetic Hybrids](#)
 - [10.3 PREVIEW OF THE HYBRID SYSTEMS TO BE DISCUSSED](#)
 - [10.3.1 Genetic Algorithm based Backpropagation Network](#)
 - [10.3.2 Fuzzy-Backpropagation Network](#)
 - [10.3.3 Simplified Fuzzy ARTMAP](#)
 - [10.3.4 Fuzzy Associative Memories](#)
 - [10.3.5 Fuzzy Logic Controlled Genetic Algorithms](#)
 - [SUMMARY](#)
 - [REFERENCES](#)
- [11. Genetic Algorithm Based Backpropagation Networks](#)
 - [11.1 GA BASED WEIGHT DETERMINATION](#)
 - [11.1.1 Coding](#)
 - [11.1.2 Weight Extraction](#)
 - [11.1.3 Fitness Function](#)
 - [11.1.4 Reproduction](#)
 - [11.1.5 Convergence](#)
 - [11.2 APPLICATIONS](#)
 - [11.2.1 K-factor Determination in Columns](#)
 - [11.2.2 Electrical Load Forecasting](#)
 - [SUMMARY](#)
 - [PROGRAMMING ASSIGNMENT](#)
 - [SUGGESTED FURTHER READING](#)
 - [REFERENCES](#)
- [12. Fuzzy Backpropagation Networks](#)
 - [12.1 LR-TYPE FUZZY NUMBERS](#)
 - [12.1.1 Operations on LR-type Fuzzy Numbers](#)

- [12.2 FUZZY NEURON](#)
- [12.3 FUZZY BP ARCHITECTURE](#)
- [12.4 LEARNING IN FUZZY BP](#)
- [12.5 INFERENCE BY FUZZY BP](#)
 - [Algorithm 12.2](#)
- [12.6 APPLICATIONS](#)
 - [12.6.1 Knowledge Base Evaluation](#)
 - [12.6.2 Earthquake Damage Evaluation](#)
- [SUMMARY](#)
- [PROGRAMMING ASSIGNMENT](#)
- [REFERENCES](#)
- [13. Simplified Fuzzy ARTMAP](#)
 - [13.1 FUZZY ARTMAP: A BRIEF INTRODUCTION](#)
 - [13.2 SIMPLIFIED FUZZY ARTMAP](#)
 - [13.2.1 Input Normalization](#)
 - [13.2.2 Output Node Activation](#)
 - [13.3 WORKING OF SIMPLIFIED FUZZY ARTMAP](#)
 - [13.4 Application: Image Recognition](#)
 - [13.4.1 Feature Extraction—Moment Based Invariants](#)
 - [13.4.2 Computation of Invariants](#)
 - [13.4.3 Structure of the Simplified Fuzzy ARTMAP based](#)
 - [13.4.4 Experimental Study](#)
 - [13.5 RECENT TRENDS](#)
 - [SUMMARY](#)
 - [PROGRAMMING ASSIGNMENT](#)
 - [REFERENCES](#)
- [14. Fuzzy Associative Memories](#)
 - [14.1 FAM—AN INTRODUCTION](#)
 - [14.2 SINGLE ASSOCIATION FAM](#)
 - [14.2.1 Graphical Method of Inference](#)
 - [14.2.2 Correlation Matrix Encoding](#)
 - [14.3 Fuzzy Hebb FAMs](#)
 - [14.4 FAM INVOLVING A RULE BASE](#)
 - [14.5 FAM RULES WITH MULTIPLE ANTECEDENTS/CONSEQUENTS](#)
 - [14.5.1 Decomposition Rules](#)

- [14.6 APPLICATIONS](#)
 - [14.6.1 Balancing an Inverted Pendulum](#)
 - [14.6.2 Fuzzy Truck Backer-upper System](#)
- [SUMMARY](#)
- [PROGRAMMING ASSIGNMENT](#)
- [SUGGESTED FURTHER READING](#)
- [REFERENCES](#)
- [15. Fuzzy Logic Controlled Genetic Algorithms](#)
 - [15.1 SOFT COMPUTING TOOLS](#)
 - [15.1.1 Fuzzy Logic as a Soft Computing Tool](#)
 - [15.1.2 Genetic Algorithm as a Soft Computing Tool](#)
 - [15.2 PROBLEM DESCRIPTION OF OPTIMUM DESIGN](#)
 - [15.3 FUZZY CONSTRAINTS](#)
 - [15.4 ILLUSTRATIONS](#)
 - [15.4.1 Optimization of the Weight of A Beam](#)
 - [15.4.2 Optimal Mix Design for High Performance Concrete](#)
 - [15.5 GA IN FUZZY LOGIC CONTROLLER DESIGN](#)
 - [15.6 FUZZY LOGIC CONTROLLER](#)
 - [15.6.1 Components of Fuzzy Logic Controller \(FLC\)](#)
 - [15.6.2 Fuzzy IF-THEN Rules](#)
 - [15.7 FLC-GA BASED STRUCTURAL OPTIMIZATION](#)
 - [15.8 APPLICATIONS](#)
 - [15.8.1 Optimum Truss](#)
 - [15.8.2 112 Bar Dome Space Truss](#)
 - [SUMMARY](#)
 - [PROGRAMMING ASSIGNMENT](#)
 - [SUGGESTED FURTHER READING](#)
 - [REFERENCES](#)
- [Index](#)