Unit 1

1

INTRODUCTION TO SOFT COMPUTING

Unit Structure

1.0 Objectives

In this chapter, we will try to learn what is soft computing, difference between hard computing and soft computing and reason for why soft computing evolved. At the end, some application of soft computing will be discussed.

1.1 Computational Paradigm

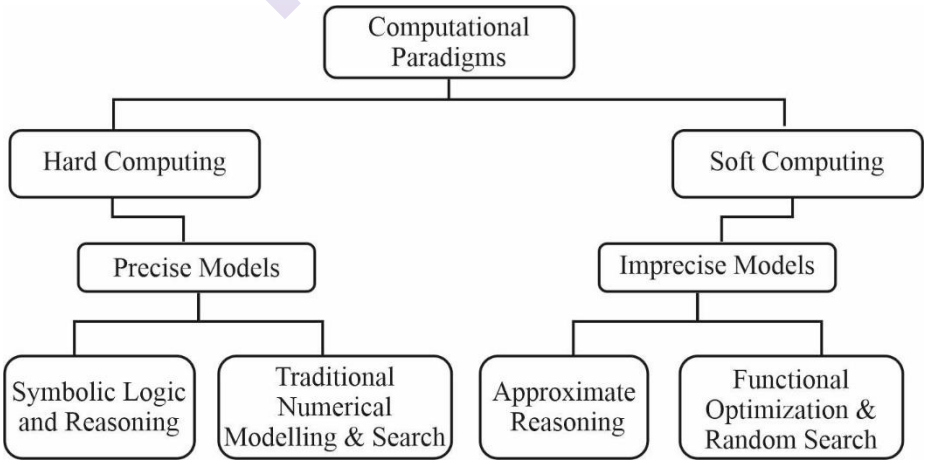


Figure 1.1: Computational Paradigms

Computational paradigm is classified into two viz: Hard computing and soft computing. Hard computing is the conventional computing. It is based on the principles of precision, certainty, and inflexibility. It requires mathematical model to solve problems. It deals withs the precise models. This model is further classified into symbolic logic and reasoning, and traditional numerical modelling and search methods. The basic of traditional artificial intelligence is utilised by these methods. It consumes a lot of time to deal with real life problem which contains imprecise and uncertain information. The following problems cannot accommodate hard computing techniques:

1. Recognition problems
2. Mobile robot co-ordination, forecasting
3. Combinatorial problems

Soft computing deals with approximate models. This model is further classified into two approximate reasoning, and functional optimization & random search methods. It handles imprecise and uncertain information of the real world. It can be used in all industries and business sectors to solve problems. Complex systems can be designed with soft computing to deal with the incomplete information, where the system behaviour is not completely known or the existence of measures of variable is noisy.

1.1.1 Soft Computing v/s Hard Computing

Hard Computing	Soft Computing
It uses precisely stated analytical model.	It is tolerant to imprecision, uncertainty, partial truth and approximation.
It is based on binary logic and crisp systems.	It is based on fuzzy logic and probabilistic reasoning.
It has features such as precision and categoricity.	It has features such as approximation and dispositionality.
It is deterministic in nature.	It is stochastic in nature.
It can work with exact input data.	It can work with ambiguous and noisy data.
It performs sequential computation.	It performs parallel computation.
It produces precise outcome.	It produces approximate outcome.

1.2 Introduction to Soft Computing

The real-world problems require systems that combines knowledge, techniques, and methodologies from various source. These systems should possess humanlike expertise within specific domain, adapt themselves and learn to do better in the changing environments and explain how they make decisions or take actions.

Natural language is used by human for reasoning and drawing conclusion. In conventional AI, the human intelligent behaviour is expressed in the language form or symbolic rules. It manipulates the symbols on the assumption that such behaviour can be stored in symbolically structured knowledge base known as physical symbol system hypothesis.

"Basically, Soft Computing is not a homogenous body of concepts & techniques. Rather, it is partnership of distinct methods that in one way or another conform to its guiding principle. At this juncture, the dominant aim of soft computing is to exploit the tolerance for imprecision and uncertainty to achieve tractability, robustness and low solutions cost. The principal constituents of soft computing are fuzzy logic, neurocomputing, and probabilistic reasoning, with the latter subsuming genetic algorithms, belief networks, chaotic systems, and parts of learning theory. In partnership of fuzzy logic, neurocomputing, and probabilistic reasoning, fuzzy logic is mainly concerned with imprecision and approximate reasoning; neurocomputing with learning and curve-fitting; and probabilistic reasoning with uncertainty and belief propagation."

-Zadeh (1994)

Soft computing combines different techniques and concepts. It can handle imprecision and uncertainty. Fuzzy logic, neurocomputing, evolutionary and genetic programming, and probabilistic computing are fields of soft computing. Soft computing is designed to model and enable solutions to real world problems, which cannot be modelled mathematically. It does not perform much symbolic manipulation.

The main computing paradigm of soft computing are: Fuzzy systems, Neural Networks and Genetic Algorithms.

- Fuzzy set are for knowledge representation via fuzzy If – Then rules.
- Neural network for learning and adaptivity and
- Genetic algorithm for evolutionary computation.

To achieve close resemblance with human like decision making, soft computing aims to exploit the tolerance for approximation, uncertainty, imprecision, and partial truth.

- Approximation: the model has similar features but not same.
- Uncertainty: the features of the model may not be same as that of the entity/belief.
- Imprecision: the model features (quantities) are not same as that the real ones but are close to them.

1.3 Premises of Soft Computing

- The real-world problems are imprecise and uncertain.
- Precision and certainty carry a cost.
- There may not be precise solutions for some problems.

1.4 Guidelines of Soft Computing

The guiding principle of soft computing is to exploit the tolerance for approximation, uncertainty, imprecision and partial truth to achieve tractability, robustness and low solution cost. Human mind is the role model for soft computing.

1.5 Uncertainty of AI

- Objective (features of whole environment)
  - There are lot of uncertainty in the world. We have limited capabilities to sense these uncertainties.
- Subjective (features of interaction with concrete environment
  - For the same/similar situation people may have different experiences. This experience maps on the features of semantics of different languages.

1.6 Application of Soft Computing

The application of soft computing has proved following advantages:

- The application that cannot be modelled mathematically can be solved.
- Non-linear problems can be solved.
- Introducing human knowledge such as cognition, understanding, recognition, learning and other into the field of computing.

Few applications of soft computing are enlisted below:

- **Handwritten Script Recognition using Soft Computing:**

  It is one of the demanding parts of computer science. It can translate multilingual documents and sort the various scripts accordingly. Block -level technique concept is used by the system to recognize the script from several script document given. To classify the script according to their features, it uses Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT) together.

- **Image Processing and Data Compression using Soft Computing:**

  Image analysis is the high-level processing technique which includes recognition and bifurcation of patterns. It is one of the most important parts of the medical field. The problem of computational complexity and efficiency in the classification can be easily be solved using soft computing techniques. Genetic algorithms, genetic programming, classifier systems, evolutionary strategies, etc are the techniques of soft computing that can be used. These algorithms give the fastest solutions to pattern recognition. These help in analysing the medical images obtained from microscopes as well as examine the X-rays.

- **Use of Soft Computing in Automotive Systems and Manufacturing:**

  Automobile industry has also adapted soft computing to solve some of the major problems.

  Classic control methods is built in vehicles using the Fuzzy logic techniques. It takes the example of human behavior, which is described in the forms of rule – "If-Then "statements.

  The logic controller then converts the sensor inputs into fuzzy variables that are then defined according to these rules. Fuzzy logic techniques are used in engine control, automatic transmissions, antiskid steering, etc.

- **Soft Computing based Architecture:**

  An intelligent building takes inputs from the sensors and controls effectors by using them. The construction industry uses the technique of DAI (Distributed Artificial Intelligence) and fuzzy genetic agents to provide the building with capabilities that match human intelligence. The fuzzy logic is used to create behaviour-based architecture in intelligent buildings to deal with the unpredictable nature of the environment, and these agents embed sensory information in the buildings.

- **Soft Computing and Decision Support System:**

  Soft computing gives an advantage of reducing the cost of the decision support system. The techniques are used to design, maintain, and maximize the value of the decision process. The first application of fuzzy logic is to create a decision system that can predict any sort of risk. The second application is using fuzzy information that selects the areas which need replacement.

- **Soft Computing Techniques in Power System Analysis:**

  Soft computing uses the method of Artificial Neural Network (ANN) to predict any instability in the voltage of the power system. Using the ANN, the pending voltage instability can be predicted. The methods which are deployed here, are very low in cost.

- **Soft Computing Techniques in Bioinformatics:**

  The techniques of soft computing help in modifying any uncertainty and indifference that biometrics data may have. Soft computing is a technique that provides distinct low-cost solutions with the help of algorithms, databases, Fuzzy Sets (FSs), and Artificial Neural Networks (ANNs). These techniques are best suited to give quality results in an efficient way.

- **Soft Computing in Investment and Trading:**

  The data present in the finance field is in opulence and traditional computing is not able to handle and process that kind of data. There are various approaches done through soft computing techniques that help to handle noisy data. Pattern recognition technique is used to analyse the pattern or behaviour of the data and time series is used to predict future trading points.

Summary

In this chapter, we have learned that the soft computing is the partnership of multiple techniques which helps to accomplish a particular task. The real-world problem that contains uncertain and imprecise information can be solved using soft computing techniques.

Review Questions

1. What is computational paradigm?
2. State difference between hard computing and soft computing?
3. Write a short note on soft computing.
4. What are the premises and guiding principle of soft computing techniques?
5. Give any three applications of soft computing.

Bibliography, References and Further Reading

- https://www.coursehero.com/file/40458824/01-Introduction-to-Soft-Computing-CSE-TUBEpdf/
- https://techdifferences.com/difference-between-soft-computing-and-hard-computing.html
- https://www.researchgate.net/profile/Mohamed_Mourad_Lafifi/post/Soft_Computing_Applications/attachment/5b8ef4933843b0067537cb3b/AS%3A667245734817800%401536095188583/download/Soft+Computing+and+its+Applications.pdf
- https://wisdomplexus.com/blogs/applications-soft-computing/
- Artificial Intelligence and Soft Computing, by Anandita Das Battacharya, SPD 3rd, 2018
- Principles of Soft Computing, S.N. Sivanandam, S.N.Deepa, Wiley, 3rd , 2019
- Neuro-fuzzy and soft computing, J.S.R. Jang, C.T.Sun and E.Mizutani, Prentice Hall of India, 2004

❖❖❖

2

TYPES OF SOFT COMPUTING TECHNIQUES

Unit Structure

2.0 Objectives

The objective of this chapter is to give the overview of various soft computing techniques.

2.1 Types of Soft Computing Techniques

Following are the various techniques of soft computing:

1. Fuzzy Computing
2. Neural Network
3. Genetic Algorithms
4. Associative memory
5. Adaptive Resonance Theory

6. Classification
7. Clustering
8. Probabilistic Reasoning
9. Bayesian Network

All the above techniques are discussed in brief in the below sections.

2.2 Fuzzy Computing

The knowledge that exists in real world is vague, imprecise, uncertain, ambiguous, or probabilistic in nature. This type of knowledge is also known as fuzzy knowledge. Human thinking and reasoning frequently involves fuzzy information.

The classical computing system involves two valued logic (true/false, 1/0, yes/no). This system sometimes may not be able to answer some questions as human does, as they do not have complete true answer. The computing system is not just expected to give answers like human but also describe the reality level calculated with the imprecision and uncertainty of the facts and rules applied.

Lofti Zadeh observed that the classical computing system was not capable to handle subjective data representation or unclear human ideas. In 1965, he introduced fuzzy set theory as the extension of classical set theory where elements have degrees of memberships. It allows to determine the distinctions among the data that is neither true nor false. It is like process of human thinking like very hot, hot, warm, little warm, cold, too cold.

In classical system, 1 represents absolute truth value and 0 represents absolute false value. But in the fuzzy system, there is no logic for absolute truth and absolute false value. But in fuzzy logic, there is intermediate value too present which is partially true and partially false.

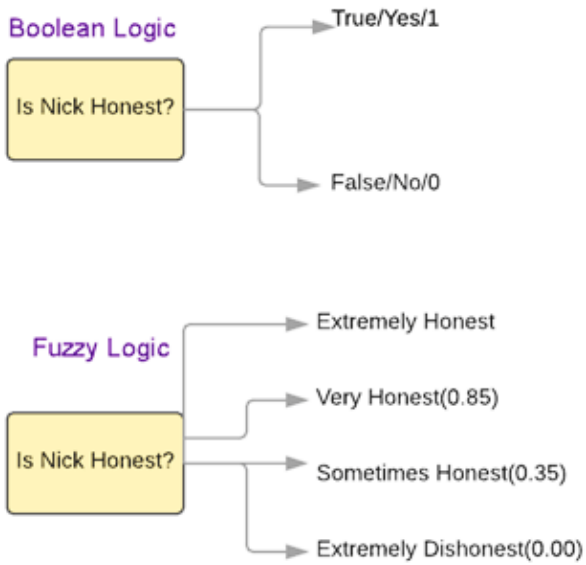


Fig 2.1: Fuzzy logic with example

Fuzzy Logic Architecture:

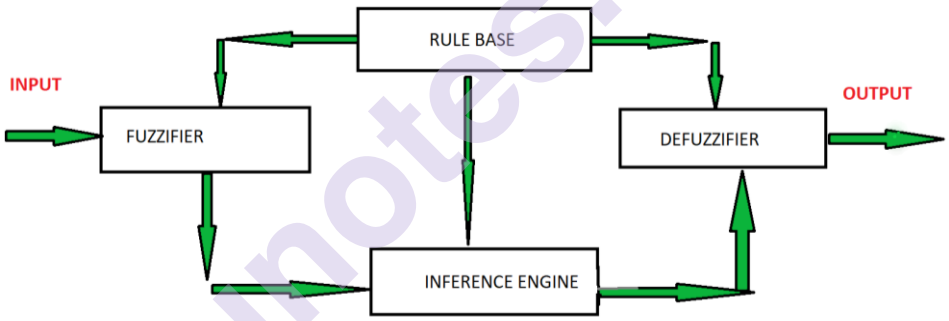


Fig 2.2: Fuzzy Logic Architecture

Fuzzy logic architecture mainly constitutes of following four components:

- **Rule base**: It contains the set of rules. The If-then conditions are provided by the experts to govern the decision-making system. These conditions are based on linguistic information.
- **Fuzzification**: It converts the crisp numbers into the fuzzy sets. The crisp input is measured by the sensors and passed into the control system for processing.
- **Inference engine:** It determines the matching degree of the current fuzzy input with respect to each rule and decides which rules are to be fired according to the input field. Next, the fired rules are combined to form the control actions.
- **Defuzzification:** The fuzzy set obtained from the inference engine is converted into the crisp value.

Characteristics of fuzzy logic:

1. It is flexible and easy to implement.
2. It helps to represent the human logic.
3. It is highly suitable method for uncertain or approximate learning.
4. It views inference as a process of propagating elastic constraints.
5. It allows you to build nonlinear functions of arbitrary complexity.

When not to use fuzzy logic:

1. If it is inconvenient to map an input space to an output space.
2. When the problem can be solved using common sense.
3. When many controllers can do the fine job, without the use of fuzzy logic.

Advantages of Fuzzy Logic System:

- Its structure is easy and understandable.
- It is used for commercial and practical purposes.
- It helps to control machines and consumer products.
- It offers acceptable reasoning. It may not offer accurate reasoning.
- In data mining it helps you to deal with uncertainty.
- It is mostly robust as no precise inputs are required.
- It can be programmed to in the situation when feedback sensor stops working.
- Performance of the system can be modified or altered by using inexpensive sensors to keep the overall system cost and complexity low.
- It provides a most effective solution to complex issues.

Disadvantages of Fuzzy Logic System:

- The results of the system may not be widely accepted as the fuzzy logic is not always accurate.
- It does not have the capability of machine learning as-well-as neural network type pattern recognition.
- Extensive testing with the hardware is needed for validation and verification of a fuzzy knowledge-based system.
- It is difficult task to set exact, fuzzy rules and membership functions.

Application areas of Fuzzy Logic:

- Automotive Systems: Automatic Gearboxes, Four-Wheel Steering, Vehicle environment control.
- Consumer Electronic Goods: Photocopiers, Still and video cameras, television.
- Domestic Goods: Refrigerators, Vacuum cleaners, Washing Machines.
- Environment Control: Air conditioners, Humidifiers.

2.3 Neural Computing

Artificial Neural Network (ANN) also known as neural network is the concept inspired from human brain and the way the neurons in the human brain works. It is computational learning system that uses a network of functions to understand and translate a data input of one form into another form. It contains large number of interconnected processing elements called as neuron. These neurons operate in parallel and are configured. Every neuron is connected with other neurons by a connection link. Each connection is associated with weights which contain information about the input signal.

Components of Neural Networks:

1. Neuron model: The information process unit of ANN.

   Neuron model consist of the following:

   a. Input

   b. Weight

   c. Activation functions

2. Architecture: The arrangement of neurons and links connecting neurons, where every link.

   Following are the different ANN architecture:

   a. Single layer Feed forward Network

   b. Multi-layer Feed forward Network

   c. Single node with its own feedback

   d. Single layer recurrent network

   e. Multi-layer recurrent network

3. A learning algorithm: For training ANN by modifying the weights in order to model a particular learning task correctly on the training examples.

   Following are the different types of learning algorithm:

   a. Supervised Learning

   b. Unsupervised Learning

   c. Reinforcement Learning

Applications of Neural Network:

1. Image recognition
2. Pattern recognition
3. Self-driving car trajectory prediction
4. Email spam filtering
5. Medical diagnosis

2.4 Genetics Algorithms

Genetic Algorithms initiated and developed in the early 1970's by John Holland are unorthodox search and optimization algorithms, which mimic some of the process of natural evolution. Gas perform directed random search through a given set of alternative 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 object function which is usually referred to as a fitness function.

Biological Background:

All living organism consist of cell. In each cell, there is a set of chromosomes which are strings of DNA and serves as a model of the organism. A chromosomes consist of genes of blocks of DNA. Each gene encodes a particular pattern. Basically, it can be said that each gene encodes a traits.

Steps involved in the genetic algorithm:

- Initialization: Define the population for the problem.
- Fitness Function: It calculates the fitness function for all the chromosomes in the population.
- Selection: Two fittest chromosomes are selected for the producing the offspring.

- Crossover: Information in the two chromosomes is exchanged to produce the new offspring.
- Mutation: It is the process of promoting diversity in the populations.

Benefits Of Genetic Algorithm

- Easy to understand.
- We always get an answer and the answer gets better with time.
- Good for noisy environment.
- Flexible in forming building blocks for hybrid application.
- Has substantial history and range of use.
- Supports multi-objective optimization.
- Modular, separate from application.

Application of Genetic Algorithm:

- Recurrent Neural Network
- Mutation testing
- Code breaking
- Filtering and signal processing

2.5 Associative Memory

An associative memory is a content-addressable structure that maps a set of input patterns to a set of output patterns. The associative memory are of two types : auto-associative and hetero-associative.

An **auto-associative memory** retrieves a previously stored pattern that most closely resembles the current pattern. In a **hetero-associative memory**, the retrieved pattern is, in general, different from the input pattern not only in content but possibly also in type and format.

Description of Associative Memory:

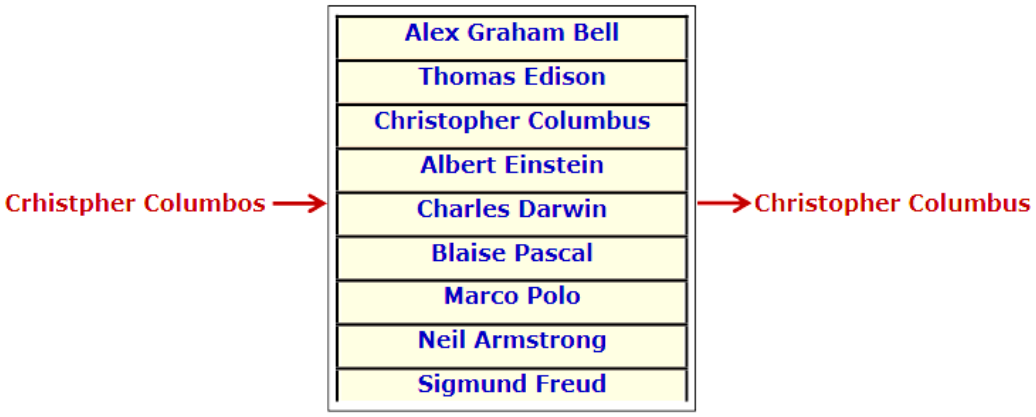


Fig 2.3: A content-addressable memory, Input and output

A content-addressable memory is a type of memory that allows, the recall of data based on the degree of similarity between the input pattern and the patterns stored in memory. It refers to a memory organization in which the memory is accessed by its content and not or opposed to an explicit address in the traditional computer memory system. This type of memory allows the recall of information based on partial knowledge of its contents.

The simplest artificial neural associative memory is the linear associator. The other popular ANN models used as associative memories are Hopfield model and Bidirectional Associative Memory (BAM) models.

2.6 Adaptive Resonance Theory

ART stands for "Adaptive Resonance Theory", invented by Stephen Grossberg in 1976. ART encompasses a wide variety of neural networks, based explicitly on neurophysiology. The word "Resonance" is a concept, just a matter of being within a certain threshold of a second similarity measure. The basic ART system is an unsupervised learning model, like many iterative clustering algorithms where each case is processed by finding the "nearest" cluster seed that resonate with the case and update the cluster seed to be "closer" to the case. If no seed resonate with the case, then a new cluster is created.

Grossberg developed ART as a theory of human cognitive information processing. The emphasis of ART neural networks lies at unsupervised learning and self-organization to mimic biological behavior. Self-organization means that the system must be able to build stable recognition categories in real-time. The unsupervised

learning means that the network learns the significant patterns based on the inputs only. There is no feedback. There is no external teacher that instructs the network or tells which category a certain input belongs. The basic ART system is an unsupervised learning model.

The model typically consists of:

- a comparison field and a recognition field composed of neurons,
- a vigilance parameter, and
- a reset module.

Comparison field and Recognition field:

- The Comparison field takes an input vector (a 1-D array of values) and transfers it to its best match in the Recognition field; the best match is, the single neuron whose set of weights (weight vector) matches most closely the input vector.
- Each Recognition Field neuron outputs a negative signal(proportional to that neuron's quality of match to the input vector) to each of the other Recognition field neurons and inhibits their output accordingly.
- Recognition field thus exhibits lateral inhibition, allowing each neuron in it to represent a category to which input vectors are classified.

Vigilance parameter:

- It has considerable influence on the system memories:
  - higher vigilance produces highly detailed memories,
  - lower vigilance results in more general memories

Reset module:

- After the input vector is classified, the Reset module compares the strength of the recognition match with the vigilance parameter.
  - If the vigilance threshold is met, then training commences.
  - Else, the firing recognition neuron is inhibited until a new input vector is applied.

Training ART-based Neural Networks:

- Training commences only upon completion of a search procedure. What happens in this search procedure :

- The Recognition neurons are disabled one by one by the reset function until the vigilance parameter is satisfied by a recognition match.
- If no committed recognition neuron's match meets the vigilance threshold, then an uncommitted neuron is committed and adjusted towards matching the input vector.

Methods of Learning:

- Slow learning method: here the degree of training of the recognition neuron's weights towards the input vector is calculated using differential equations and is thus dependent on the length of time the input vector is presented.
- Fast learning method: here the algebraic equations are used to calculate degree of weight adjustments to be made, and binary values are used.

Types of ART Systems:

- **ART 1**: The simplest variety of ART networks, accept only binary inputs.
- **ART 2** : It extends network capabilities to support continuous inputs.
- **Fuzzy ART** : It Implements fuzzy logic into ART's pattern recognition, thus enhances generalizing ability. One very useful feature of fuzzy ART is complement coding, a means of incorporating the absence of features into pattern classifications, which goes a long way towards preventing inefficient and unnecessary category proliferation.
- **ARTMAP** : Also known as Predictive ART, combines two slightly modified ARTs , may be two ART-1 or two ART-2 units into a supervised learning structure where the first unit takes the input data and the second unit takes the correct output data, then used to make the minimum possible adjustment of the vigilance parameter in the first unit in order to make the correct classification.

2.7 Classification

Classification is supervised learning. Classification algorithms is used to predict the categorical values. Training is provided to identify the category of new observations. The program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Classes are also called as target/labels or categories.

Classification algorithms:

- Logistic Regression
- Naïve Bayes
- K-Nearest Neighbour
- Decision tree
- Random Forest

Application of Classification:

- Email Spam Detection
- Speech Recognition
- Identification of Cancer tumour cells
- Biometric Identifications

2.8 Clustering

Clustering is type of unsupervised learning method. In this learning we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Its task is to divide the population or data points into several groups. Data points in the same group are similar to the other data point in the same group and dissimilar to the data points in other groups.

Why Clustering?

Clustering determines the grouping among the unlabelled data present. There is no criteria for a good clustering. It depends on the criteria that the user fits the need of the user.

Clustering Methods:

- Density-Based Methods
- Hierarchical Based Methods
    - Agglomerative (bottom up approach)
    - Divisive (top down approach)
- Partitioning Methods
- Grid-based Methods

Applications of Clustering in different fields

- Marketing
- Biology
- Insurance
- City Planning
- Earthquake studies

2.9 Probabilistic Reasoning

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty. We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance. In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.
- In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:
    - Bayes' rule
    - Bayesian Statistics

2.10 Bayesian Networks

Bayesian network is also known Bayesian belief network, decision network or Bayesian Model. It deals with the probabilistic events and solves a problem which has uncertainty.

Bayesian networks are a type of probabilistic graphical model that uses Bayesian inference for probability computations. Bayesian networks aim to model conditional dependence, and therefore causation, by representing conditional dependence by edges in a directed graph. Through these relationships, one can efficiently conduct inference on the random variables in the graph through the use of factors.

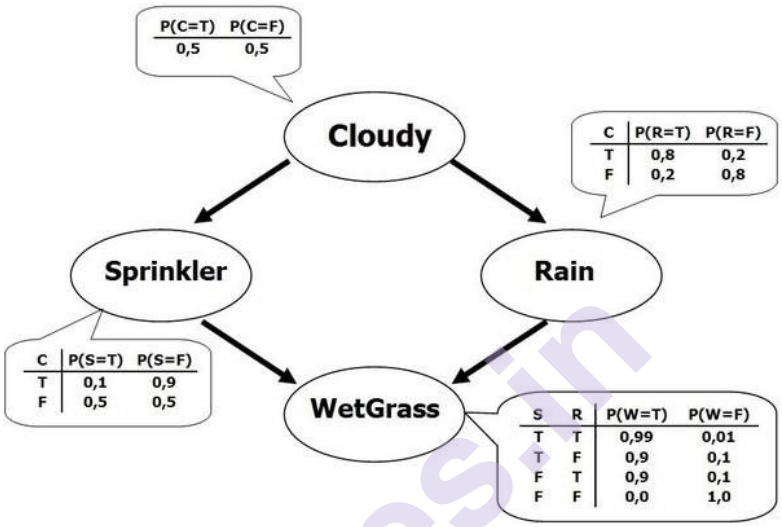


Fig 2.4: Bayesian Network example

A Bayesian network is a **directed acyclic graph** in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable. Formally, if an edge (A, B) exists in the graph connecting random variables A and B, it means that P(B|A) is a **factor** in the joint probability distribution, so we must know P(B|A) for all values of B and A in order to conduct inference.

The Bayesian network has mainly two components:

- Causal Component
- Actual numbers

Each node in the Bayesian network has condition probability distribution P(Xi |Parent(Xi)), which determines the effect of the parent on that node.

Applications of Bayesian Networks:

- Medical Diagnosis
- Management efficiency
- Biotechnology

Summary

In this chapter we have learned different techniques used in soft computing. Fuzzy system can be used when we want to deal with uncertainty and imprecision. Adaptivity and learning abilities in the system can be build using neural computing. To find the better solution to the problem, genetic algorithms can be applied. The pattern can be retrieved from the memory based on the content and not based on address is called associative memory. Find the input patterns closest resemblances in the memory can also be done with the adaptive resonance theory. Classification is based on supervised learning usually used for predictions and clustering is based on unsupervised learning. Probabilistic reasoning and Bayesian Networks are based on the probability of the event occurring.

Review Questions

1. Write a short note on fuzzy system.
2. What is artificial neural network? Explain its components and learning methods.
3. Write a short note on genetic algorithms.
4. Explain the working of Adaptive Resonance Theory.
5. Write a short note on associative memory.
6. Compare classification technique with clustering technique.
7. Write a short note on probabilistic reasoning.
8. Write a short note on Bayesian Networks.

Bibliography, References and Further Reading

- https://www.coursehero.com/file/40458824/01-Introduction-to-Soft-Computing-CSE-TUBEpdf/
- https://www.geeksforgeeks.org/fuzzy-logic-introduction/
- https://www.guru99.com/what-is-fuzzy-logic.html
- https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_fuzzy_logic_systems.htm

- https://deepai.org/machine-learning-glossary-and-terms/neural-network
- https://www.javatpoint.com/bayesian-belief-network-in-artificial-intelligence
- https://www.javatpoint.com/probabilistic-reasoning-in-artifical-intelligence#:~:text=Probabilistic%20reasoning%20is%20a%20way,logic%20to%20handle%20the%20uncertainty
- https://www.geeksforgeeks.org/clustering-in-machine-learning/
- https://www.javatpoint.com/classification-algorithm-in-machine-learning
- https://www.geeksforgeeks.org/genetic-algorithms/
- Artificial Intelligence and Soft Computing, by Anandita Das Battacharya, SPD 3rd, 2018
- Principles of Soft Computing, S.N. Sivanandam, S.N.Deepa, Wiley, 3rd , 2019
- Neuro-fuzzy and soft computing, J.S.R. Jang, C.T.Sun and E.Mizutani, Prentice Hall of India, 2004

❖❖❖

UNIT 2

3

INTRODUCTION TO ARTIFICIAL NEURAL NETWORK & SUPERVISED LEARNING NETWORK I

Unit Structure

3.0 Objectives

1. The fundamentals of artificial neural network
2. Understanding between biological neuron and artificial neuron
3. Working of a basic fundamental neuron model.
4. Terminologies and terms used for better understanding of Artificial Neural Network
5. The basics of supervised learning and perceptron learning rule
6. Overview of adaptive and multiple adaptive linear neurons

3.1 Basic Concept

Neural networks are information processing systems that are implemented to model the working of the human brain. It is more of a computational model used to perform tasks in a better optimized way than the traditional systems. The essential properties of biological neural networks are considered in order to understand the information processing tasks. This indeed will allow us to design abstract models of artificial neural networks which can be simulated and analyzed.

3.1.1 Introduction to Artificial Neural Network

Artificial Neural Network (ANN) is an information processing system that possesses characteristics with biological neural networks. ANNs consists of large number of highly interconnected processing elements called nodes or units or neurons. These neurons operate in parallel. Every neuron is connected to the other neuron through the communication link with assigned weights which contain information about the input signal. These processing elements are called neurons or artificial neurons.

3.1.2 Overview of Biological Neural Network

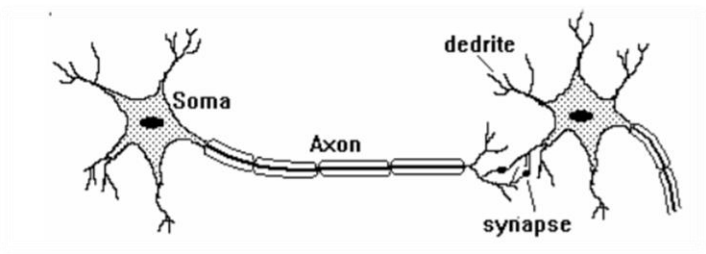


Fig 3.1: Schematic diagram of a Neuron
(Image courtesy: Ugur Halici Lecture notes)

The fact that the human brain consists of large number of neurons with numerous interconnections that processes information. The term neural network is usually referred to the biological neural network that processes and transmits information. The biological neurons are part of the nervous system.

The biological neuron consists of three major parts

1. Soma or Cell body- contains the cell nucleus. In general, processing occurs here
2. Dendrites- branching fibres that protrude from the cell body or soma. The nerve is connected to the cell body.
3. Axon- It carries the impulses of the neuron. It carries information away from the soma to other neurons.
4. Synapse- Each strand of an axon terminates into a small bulb-like organ called synapse. It is through synapse the neuron introduces its signals to other neurons.

Working of the neuron

1. Dendrites receive activation signal from other neurons which is the internal state of every neuron
2. Soma processes the incoming activity signals and convert its into output activation signals.
3. Axons carry signals from the neuron and sends it to other neurons.
4. Electric impulses are passed between the synapses and the dendrites. The signal transmission involves a chemical process called neuro-transmitters.

3.1.3 Human Brain v/s Artificial Neural Network

Comparison between biological and artificial neurons based on the following criteria

1. Speed – Signals in human brain move at a speed dependent on the nerve impulse. The biological neuron is slow in processing as compared to the artificial neural networks which are modelled to process faster.
2. Processing- The biological neuron can perform massive parallel operations simultaneously. A large number of simple units are organized to solve problems independently but collectively. The artificial neurons also respond in parallel but do not execute programmed instructions.

3. Size and Complexity- The size and complexity of the brain is comparatively higher than that of artificial neural network. The size and complexity of an ANN is different for different applications
4. Storage Capacity – The biological neuron stores the information in its interconnection and in artificial neuron it is stored in memory locations.
5. Tolerance- The biological neuron has fault tolerant capability but artificial neuron has no tolerant capability. Biological neurons considers redundancies whereas artificial neurons cannot consider redundancies.
6. Control mechanism- There is no control unit to monitor the information processed in to the network in biological neural networks whereas in artificial neuron model all activities are continuously monitored by a control unit.

3.1.4 Characteristics of Artificial Neural Networks

1. It is a mathematical model consists of computational elements implemented neurally.
2. Large number of highly interconnected processing elements known as neurons are prominent in ANN
3. The interconnections with their weights are associated with neurons.
4. The input signals arrive at the processing elements through connections and weights.
5. ANNs collective behavior is characterized by their ability to learn, recall and generalize from the given data.
6. A single neuron carries no specific information.

3.1.5 How a simple neuron works?

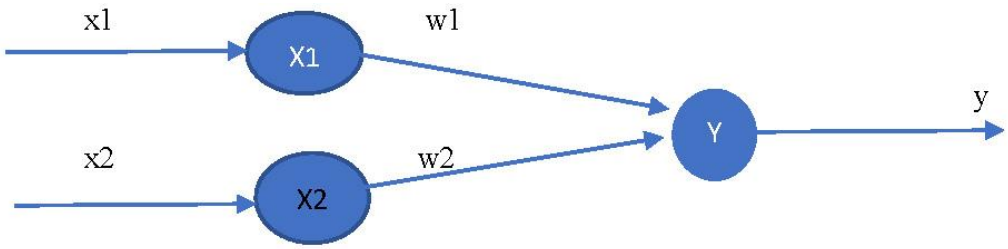


Fig 3.2 Architecture of a simple artificial neural net

From the given figure above, there are two input neurons X1 and X2 transmitting signal to the output neuron Y for receiving signal.

The input neurons are connected to the output neurons over a weighted interconnection links w1 and w2.

For above neuron architecture , the net input has to be calculated in the way.

yin = x1w1+x2w2

where x1 and x2 are the activations of the input neurons X1 and X2 . The output yin of the output neuron Y can be obtained by applying activations over the net input .

y =f(yin)

Output = Function (net input calculated)

The function to be applied over the net input is called activation function .

3.2 Basic Models of Artifical Neural Network

The models of ANN are specified by the three basic entities

1. The model's synaptic interconnections
2. The learning rules adopted for updating and adjusting the connection weights
3. The activation functions

3.2.1. The model's synaptic interconnections

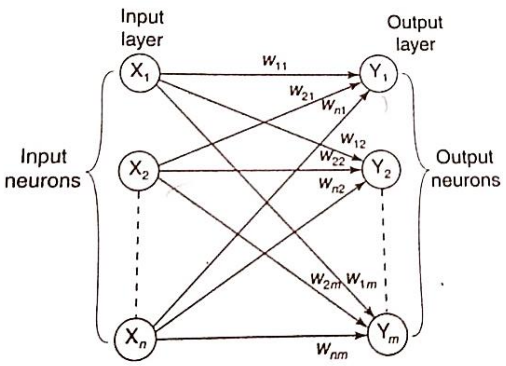
ANN consists of a set of highly interconnected neurons connected through weights to the other processing elements or to itself. The arrangement of these processing elements and the geometry of their interconnections are important for ANN. The arrangement of neurons to form layers and the connection pattern formed within and between layers is called the network architecture.

There are five basic neuron connection architectures.

1. Single-layer feed-forward network
2. Multilayer feed-forward network
3. Single node with its own feedback
4. Single-layer recurrent network
5. Multi-layer recurrent network

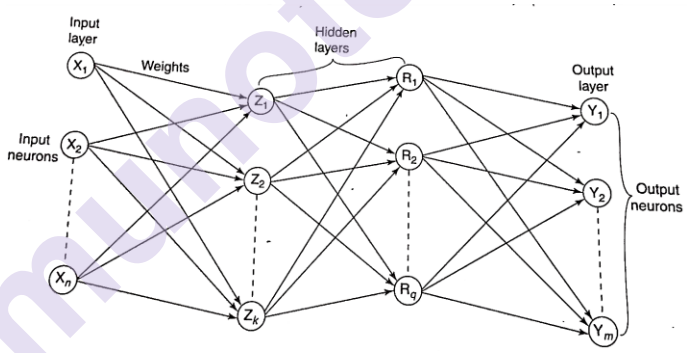
1. Single-layer feed-forward network

It consists of a single layer of network where the inputs are directly connected to the output, one per node with a series of various weights.



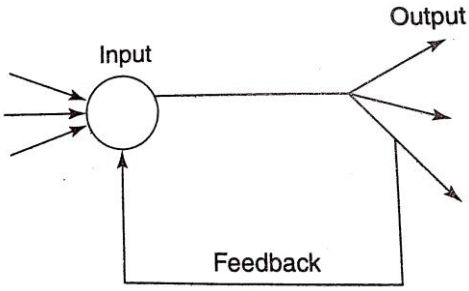
2. Multi-layer feed-forward network

It consists of multi layers where along with the input and output layers, there are hidden layers. There can be zero to many hidden layers. The hidden layer is usually internal to the network and has no direct contact with the environment.



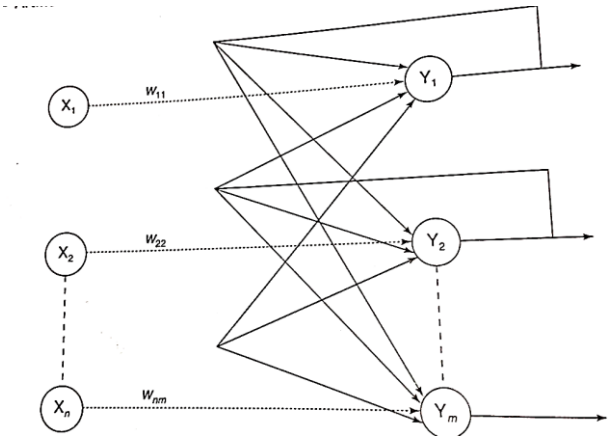
3. Single node with own feedback

The simplest neural network architecture giving feedback to itself with a single neuron.



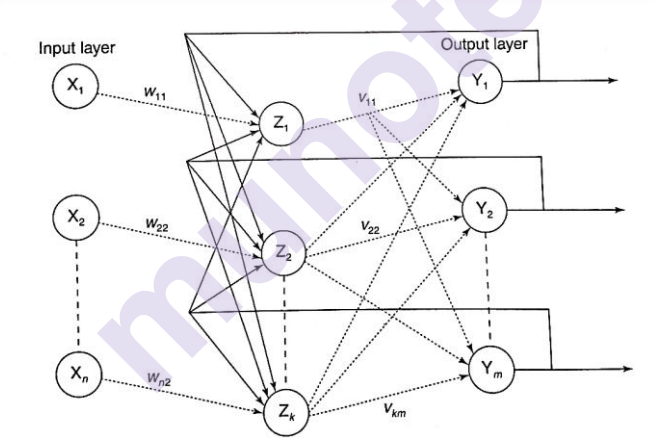
4. Single-layer recurrent network

A single-layer network with a feedback directed back to itself or to other processing element or both.



5. Multilayer recurrent network

A recurrent network has at least a feedback in place. The processing elements output can be directed back to the nodes in the previous layer.



3.2.2. Learning

The most important part of ANN is it capability to train or learn. It is basically a process by means of which a neural net adapts for adjusting or updating the connection weights in order to receive a desired response.

Learning in ANN is broadly classified into three categories

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

1. Supervised Learning

In Supervised learning, it is assumed that the correct target output values are known for each input pattern. In this learning, a supervisor or teacher is needed for error minimization. The difference between the actual and desired output vector is minimized using the error signal by adjusting the weights until the actual output matches the desired output.

2. Unsupervised Learning

In Unsupervised learning, the learning is performed without the help of a teacher or supervisor. In the learning process, the input vectors of similar type are grouped together to form clusters. The desired output is not given to the network. The system learns on its own with the input patterns.

3. Reinforcement Learning

The Reinforcement learning is a form of Supervised learning as the network receives feedback from its environment. Here the supervisor does not present the desired output but learns through the critic information.

3.2.3 Activation Function

An activation function f is applied over the net input to calculate the output of an ANN. The choice of activation functions depends on the type of problems to be solved by the network.

The most common functions are

1. Identity function- It is a linear function. It is defined as f(x) = x for all x
2. Binary step function: The function can be defined as

$$f(x) = \begin{cases} 1 \text{ if } x >= \theta \\ 0 \text{ if } x < \theta \end{cases}$$

Here,θ represents the threshold value.

3. Bipolar Step function: The function can be defined as

$$f(x) = \begin{cases} 1 \text{ if } x >= \theta \\ -1 \text{ if } x < \theta \end{cases}$$

Here,θ represents the threshold value

4. Sigmoidal functions: These functions are used in back-propagation nets.

They are of two types:

Binary Sigmoid function: It is known as unipolar sigmoid function.

It is defined by the equation

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

Here, $\boldsymbol{\lambda}$ is the steepness parameter. The range of the sigmoid function is from 0 to 1

Bipolar Sigmoid function: This function is defined as

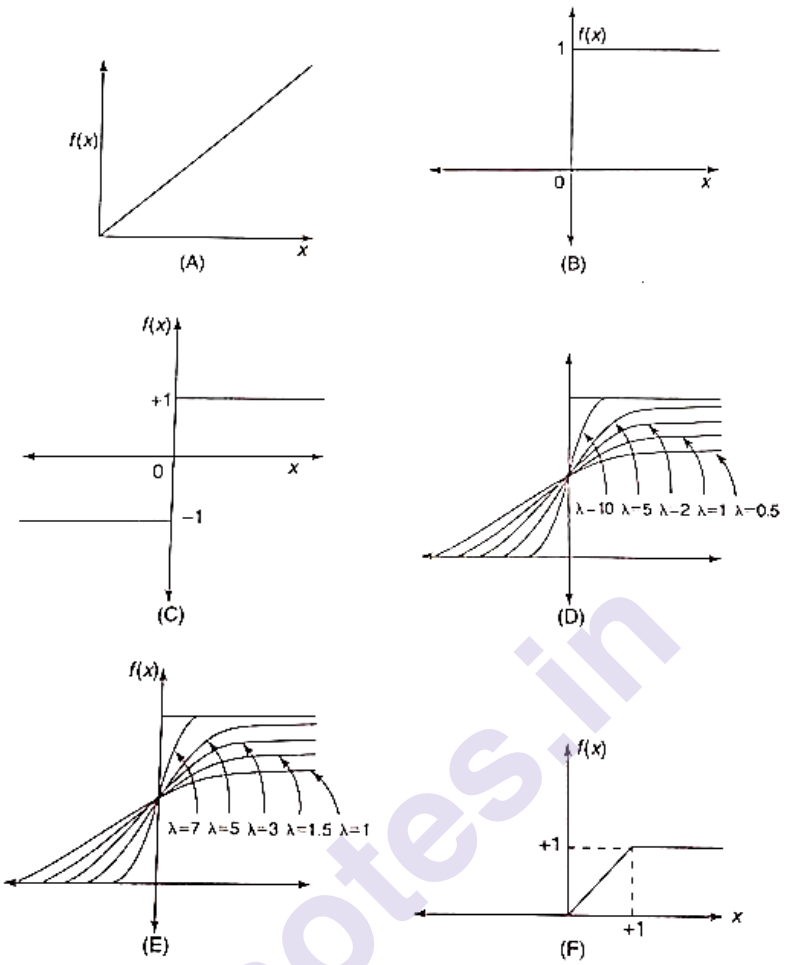
$$f(x) = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$$

Here, $\boldsymbol{\lambda}$ is the steepness parameter. The range of the sigmoid function is from -1 to +1

5. Ramp function: The ramp function is defined as

$$f(x) = \begin{cases} 1 & \text{if } x > 1 \\ x & \text{if } 0 \leq x \leq 1 \\ 0 & \text{if } x < 0 \end{cases}$$

The graphical representation is shown below for all the activation functions



Depiction of activation functions: (A) identity function; (B) binary step function; (C) bipolar step function; (D) binary sigmoidal function; (E) bipolar sigmoidal function; (F) ramp function.

3.3 Terminologies of ANN

3.3.1 Weights

Weight is a parameter which contains information about the input signal. This information is used by the net to solve a problem.

In ANN architecture, every neuron is connected to other neurons by means of a directed communication link and every link is associated with weights.

W_{ij} is the weight from processing element 'i' source node to processing element 'j' destination node.

3.3.2 Bias (b)

The bias is a constant value included in the network. Its impact is seen in calculating the net input. The bias is included by adding a component $x_0 = 1$ to the input vector X.

Bias can be positive or negative. The positive bias helps in increasing the net input of the network. The negative bias helps in decreasing the net input of the network.

3.3.3. Threshold (θ)

Threshold is a set value used in the activation function. In ANN, based on the threshold value the activation functions are defined and the output is calculated.

3.3.3 Learning Rate (α)

The learning rate is used to control the amount of weight adjustment at each step of training. The learning rate ranges from 0 to 1. It determines the rate of learning at each time step.

3.4 McCulloch- Pitts Neuron (MP neuron model)

MP neuron model was the earliest neural network model discovered by Warren McCulloch and Walter Pitts in 1943.It is also known as Threshold Logic Unit.

The M-P neurons are connected by directed weighted paths. The activation of this model is binary. The weights associated with the communication links may be excitatory (weight is positive) or inhibitory (weight is negative). Each neuron has a fixed threshold and if the net input to the neuron is greater than the threshold then the neuron fires otherwise it will not fire.

3.5 Concept of Linear Separability

Concept: Sets of point in 2-D space are linearly separable if the points can be separated by a straight line

In ANN, linear separability is the concept wherein the separation is based on the network response being positive or negative. A decision line is drawn to separate positive and negative responses. The decision line is called as linear-separable line.

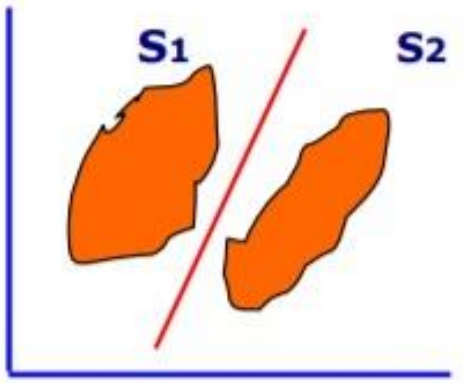


Fig 3.3: Linear Separable Patterns

The linear separability of the network is based on the decision-boundary line. If there exists weights for which the training data has correct response ,+ 1 (positive) ,it will lie on one side of the decision boundary line and all other data on the other side of the boundary line. This is known as linear separability.

3.6 Hebb Network

Hebb or Hebb learning rule stated by Donald Hebb in 1949 states that, the learning is performed by the change in the synaptic gap. Explaining further, he stated “When an axon of cell A is near enough to excite cell B, and repeatedly takes place in firing it, some growth or metabolic change takes place in one or both the cells such that A’s efficiency, as one of the cells firing B, is increased”.

In Hebb learning, if two interconnected neurons are ‘ON’ simultaneously then the weights associated with these neurons can be increased by changing the strength in the synaptic gap.

The weight update is given by

$W_i\,(new) = w_i\,(old) + x_i y$

Flowchart of Training algorithm,

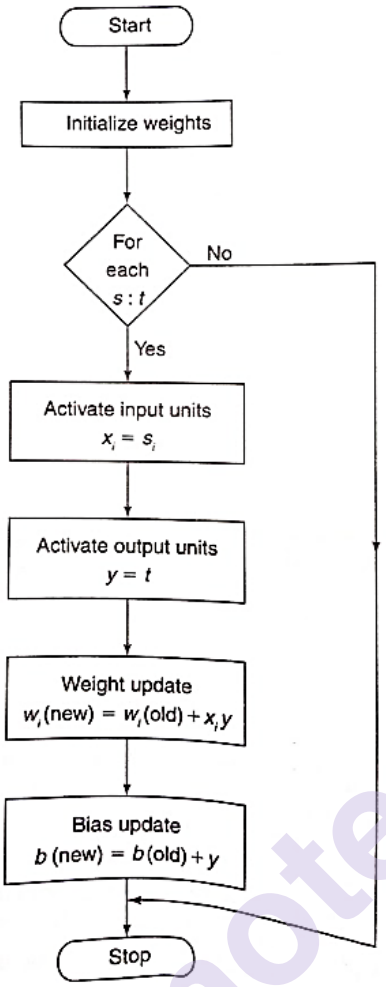


Fig 3.4: Flowchart of Hebb training algorithm

3.7 Perceptron Networks

Perceptron Networks are single-layer feed forward networks. They are the simplest perceptron,

Perceptron consists of three units – input unit (sensory unit), hidden unit (associator unit) and output unit (response unit). The input units are connected to the hidden units with fixed weights having values 1, 0 or -1 assigned at random. The binary activation function is used in input and hidden unit. The response unit has an activation of 1, 0 or -1. The output signal sent from the hidden unit to the output unit are binary.

The output of the perceptron network is given by y =f(yin) where yin is the activation function.

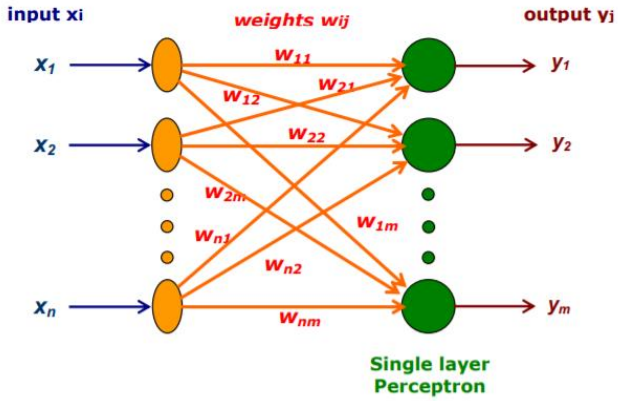


Fig 3.5: Perceptron model

Perceptron Learning algorithm

The training of perceptron is a supervised learning algorithm. The algorithm can be used for either bipolar or binary input vectors, fixed threshold and variable bias.

The output is obtained by applying the activation function over the calculated net input.

The weights are adjusted to minimize error when the output does not match the desired output.

Step 0: Initialize the weights and the bias (for easy calculation they can be set to zero). Also initialize the learning rate $\alpha (0 < \alpha \leq 1)$. For simplicity α is set to 1.

Step 1: Perform Steps 2–6 until the final stopping condition is false.

Step 2: Perform Steps 3–5 for each training pair indicated by $s:t$.

Step 3: The input layer containing input units is applied with identity activation functions:

$$x_i = s_i$$

Step 4: Calculate the output of the network. To do so, first obtain the net input:

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

where "n" is the number of input neurons in the input layer. Then apply activations over the net input calculated to obtain the output:

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

Step 5: *Weight and bias adjustment:* Compare the value of the actual (calculated) output and desired (target) output.

If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else we have

$$w_i(\text{new}) = w_i(\text{old})$$
$$b(\text{new}) = b(\text{old})$$

Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again from Step 2.

3.8 Adaptive Linear Neuron (ADALINE)

It is a network with a single linear unit. The linear activation functions are called linear units. In this, the input-output relationship is linear. Adaline networks are trained using the delta rule.

Adaline is a single-unit neuron, which receives input from several units and also from one unit, called bias. An Adeline model consists of trainable weights. The inputs are of two values (+1 or -1) and the weights have signs (positive or negative).

Initially random weights are assigned. The net input calculated is applied to a quantizer transfer function (possibly activation function) that restores the output to +1 or -1. The Adaline model compares the actual output with the target output and with the bias and the adjusts all the weights.

3.8.1 Training Algorithm

The Adaline network training algorithm is as follows:

Step0: weights and bias are to be set to some random values but not zero. Set the learning rate parameter α.

Step1: perform steps 2-6 when stopping condition is false.

Step2: perform steps 3-5 for each bipolar training pair s:t

Step3: set activations foe input units i= 1 to n.

Step4: calculate the net input to the output unit.

Step5: update the weight and bias for i=1 to n

Step6: if the highest weight change that occurred during training is smaller than a specified tolerance then stops the training process, else continue. This is the test for the stopping condition of a network.

3.8.2 Testing Algorithm

It is very essential to perform the testing of a network that has been trained. When the training has been completed, the Adaline can be used to classify input patterns. A step function is used to test the performance of the network. The testing procedure for the Adaline network is as follows:

Step0: initialize the weights. (The weights are obtained from the training algorithm.)

Step1: perform steps 2-4 for each bipolar input vector x.

Step2: set the activations of the input units to x.

Step3: calculate the net input to the output units

Step4: apply the activation function over the net input calculated.

3.9 Multiple Adaptive Linear Neurons (Madaline)

It consists of many adalines in parallel with a single output unit whose value is based on certain selection rules. It uses the majority vote rule. On using this rule, the output unit would have an answer either true or false.

On the other hand, if AND rule is used, the output is true if and only if both the inputs are true and so on.

The training process of Madaline is similar to that of Adaline

3.9.1 Architecture

It consists of "n" units of input layer and "m" units of Adaline layer and "1" unit of the Madaline layer. Each neuron in the Adaline and Madaline layers has a bias of excitation "1". The Adaline layer is present between the input layer and the Madaline layer; the Adaline layer is considered as the hidden layer.

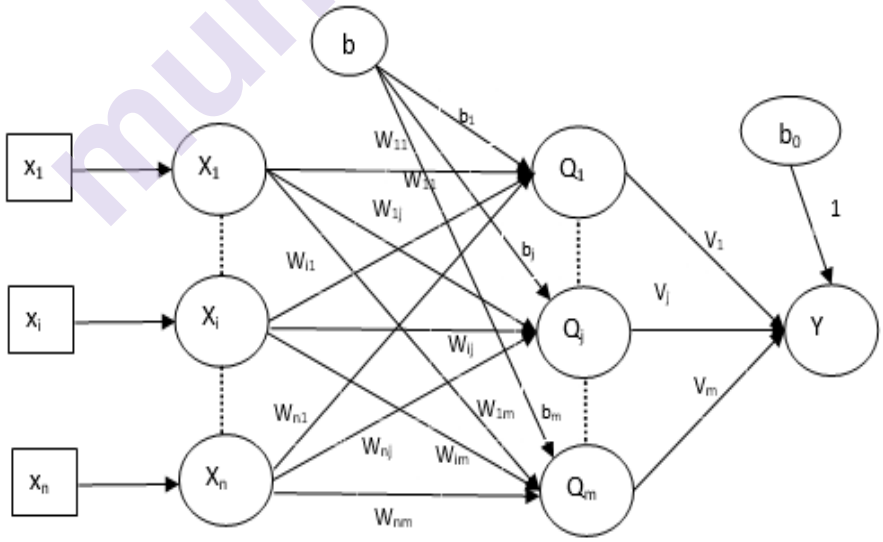


Fig 3.6: Architecture of Madaline layer

Review Questions

1. Define the term Artificial Neural Network.
2. List and explain the main components of biological neuron.
3. Mention the characteristics of an artificial neural network.
4. Compare the similarities and differences between biological and artificial neuron.
5. What are the basic models of an artificial neural network?
6. List and explain the commonly used activation functions.
7. Define the following
    a. Weights
    b. Bias
    c. Threshold
    d. Learning rate
8. Write a short note on McCulloch Pitts Neuron model.
9. Discuss about the concept of liner separability.
10. State the training algorithm used for the Hebb learning networks.
11. Explain perceptron network.
12. What is Adaline? Draw the model of an Adaline network.
13. How is Madaline network formed?

REFERENCES

1. "Principles of Soft Computing", by S.N. Sivanandam and S.N. Deepa, 2019, Wiley Publication, Chapter 2 and 3
2. http://www.sci.brooklyn.cuny.edu/ (Artificial Neural Networks, Stephen Lucci PhD)
3. Related documents, diagrams from blogs, e-resources from RC Chakraborty lecture notes and tutorialspoint.com.

❖❖❖

Unit 2

4

SUPERVISED LEARNING NETWORK II AND ASSOCIATIVE MEMORY NETWORK

Unit Structure

4.0 Objectives

1. To understand Back-propagation networks used in real time application.
2. Theory behind radial basis network and its activation function
3. Special supervised learning networks such as time delay neural networks, functional link networks, tree neural networks and wavelet neural networks
4. Details and understanding about associative memory and its types
5. Hopfield networks and its training algorithm.
6. An overview of iterative auto associative and temporal associative memory

4.1 Backpropagation networks

It is applied to multi-layer feed forward networks consisting of processing elements with different activation functions. The networks associated with back propagation learning algorithm is known as Back propagation networks. It uses gradient descent method to calculate error and propagate it back to the hidden unit.

The training at BPN is performed in three stages

1. The feed-forward of the input training pattern
2. The calculation and back-propagation of the error
3. Weight updates

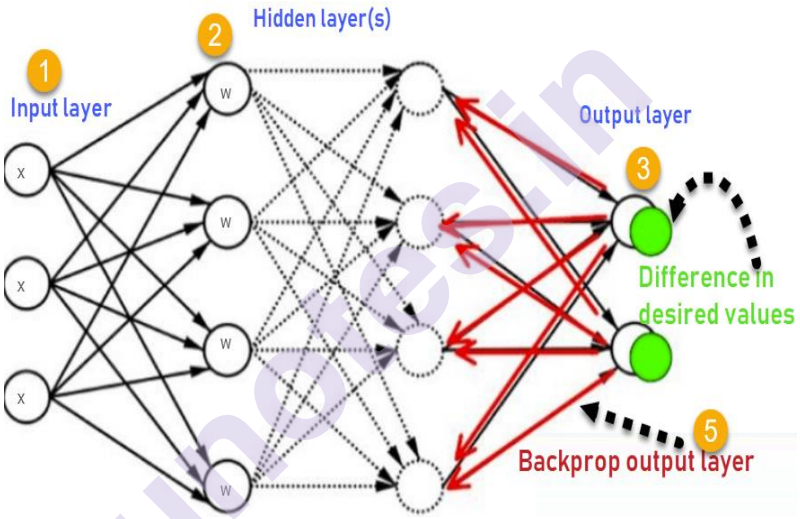


Fig. 4.1:Architecture of Backpropagation network (Image:guru99.com)

1. A back-propagation neural network is a multilayer, feed-forward neural network consisting of an input layer, a hidden layer and output layer.
2. The neurons present in the hidden and output layers have activation with always value 1.
3. The bias also acts as weights.
4. During the learning phase, signals are sent in the reverse direction.
5. The output obtained can be either binary or bipolar.

4.2 Radial Basis Function network

The radial basis function is a classification and functional approximation neural network. It uses non-linear activation functions like sigmoidal and Gaussian

functions. Since radial basis functions have only one hidden layer, the convergence of optimization is much faster.

1. The architecture consists of two layers.

2. The output nodes form a linear combination of the basis functions computed by means of radial basis function nodes. Hidden layer generates a signal corresponding to an input vector in the input layer, and corresponding to this signal, network generates a response.

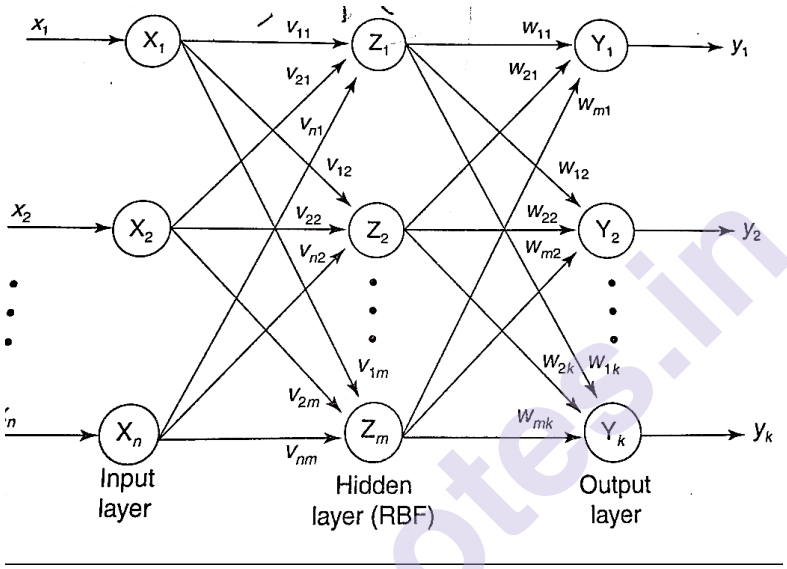


Fig.4.2: Architecture of Radial Basis functions

4.3 Time Delay Neural Networks

Time delay networks are basically feed-forward neural networks except that the input weights has a tapped delay line associated to it.In TDNN, when the output is being fed back through a unit delay into the input layer, the net computed is equivalent to an infinite impulse response filter.

A neuron with a tapped delay line is called a Time delay neural network unit and a network which consists of TDNN units is called a Time delay neural network. Application od TDNN is speech recognition.

4.4 Functional Link networks

Functional link networks is a specifically designed high order neural networks with low complexity for handling linearly non-separable problems. It has no hidden layers. This model is useful for learning continuous functions.

The most common example of linear non-separability is XOR problem.

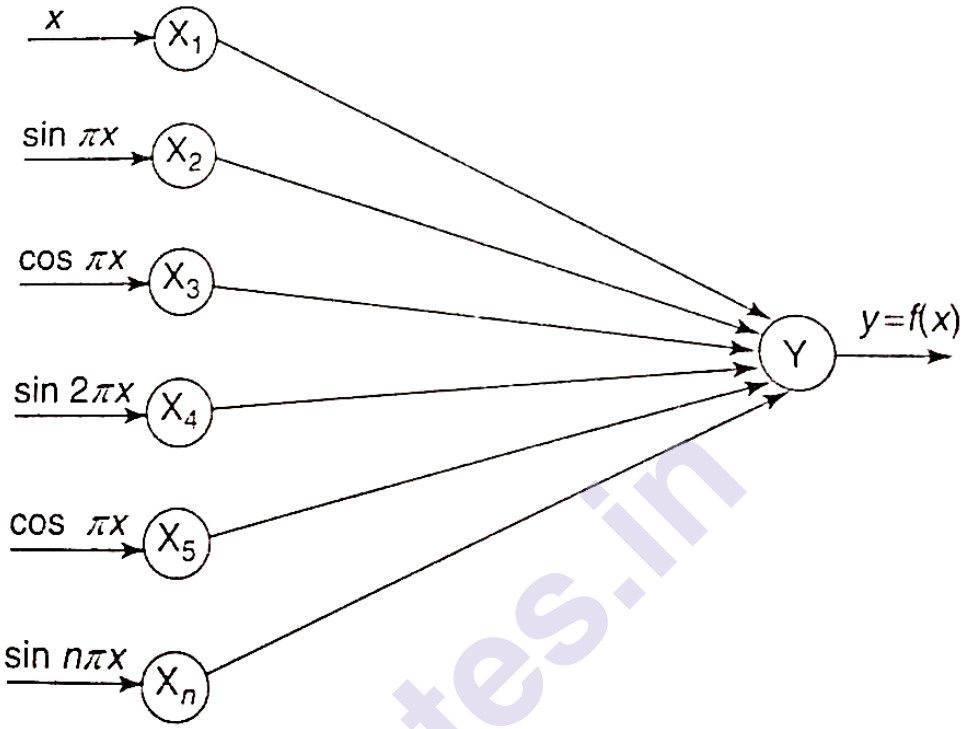


Fig 4.3: Functional line network model with no hidden layer

4.5 Tree Neural Networks

These networks are basically used for pattern recognition problems. It uses multilayer neural network at each decision-making node of a binary classification for extracting a non-linear feature.

The decision nodes are circular nodes and the terminal nodes are square nodes. The splitting rule decides whether the pattern moves to the right or left.

The algorithm consists of two phases

1. The growing phase- A large tree is grown in this phase by recursively finding the rules of splitting until all the terminal nodes have nearly pure membership or else it can split further.
2. Tree pruning phase- To avoid overfilling/overfitting of data, a smaller tree is selected or it is pruned.

Example- Tree neural networks can be used for waveform recognition problem.

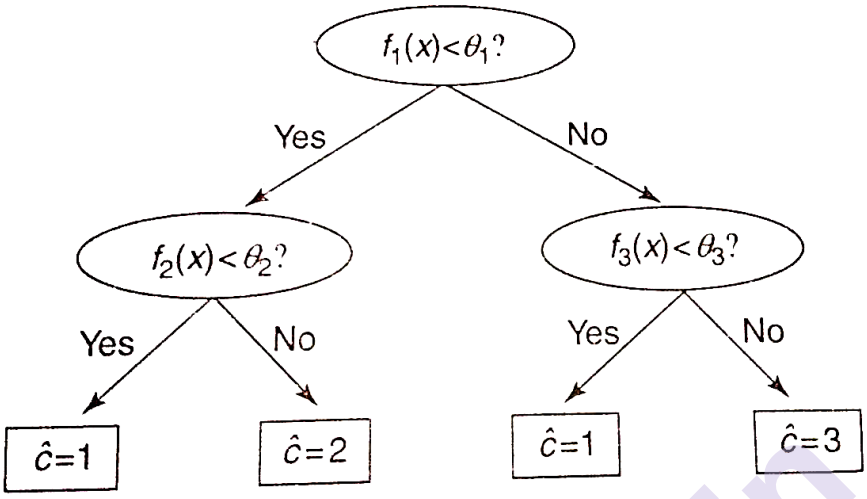


Fig 4.4: Binary Classification tree

4.6 Wavelet Neural Networks

These networks work on wavelet transform theory. It is useful for functional approximation through wavelet decomposition. It consists of rotation, dilation, translation and if the wavelet lies on the same line then it is called wavelon instead of a neuron.

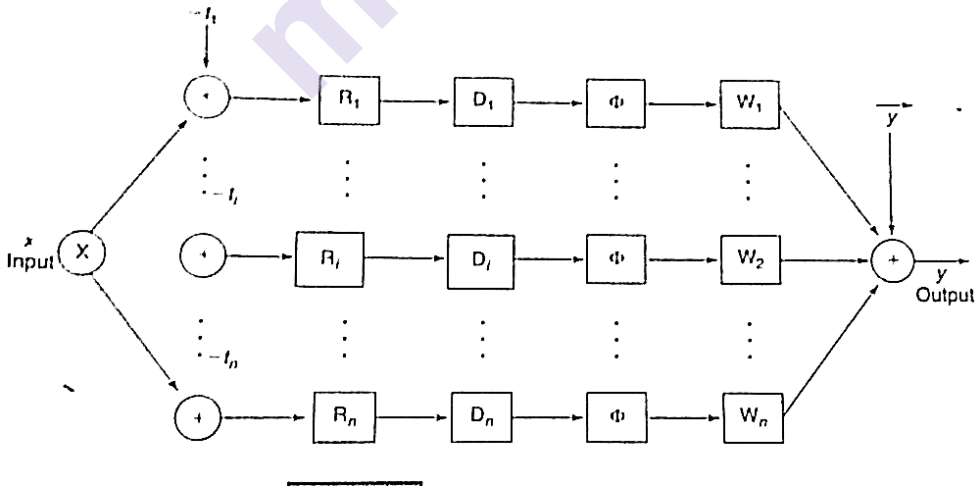


Fig 4.5: Wavelet Neural network with translation, rotation, dilation and wavelon

4.7 Associative Memory Networks-Overview

1. An associative memory is a content addressable memory structure that maps the set of input patterns to the output patterns. It can store a set of patterns as memories. The recall is through association of the key pattern with the help of information memorized. Associative memory makes a parallel search with a stored data file. The concept behind this type of search is to retrieve the stored data either completely or partially.

2. A content-addressable structure refers to a memory organization where the memory is accessed by its content. The associative memory is of two types autoassociative memory and heteroassociative memory which are single-layer nets where the weights are determined by the net output which is stored as a pattern. The architecture of the associative net is either feed-forward or iterative.

4.8 Autoassociative Memory Network

1. In this network, training input and target output vectors are same.
2. Determination of weight is called storing of vectors.
3. Weight is set to zero.
4. It increases net ability to generalize
5. The net's performance is based on its ability to reproduce a stored pattern from a noisy input.

Architecture

For an autoassociative net, the training input and target output vectors are the same. The input layer consists of n input units and the output layer also consists of n output units. The input and output layers are connected through weighted interconnections.

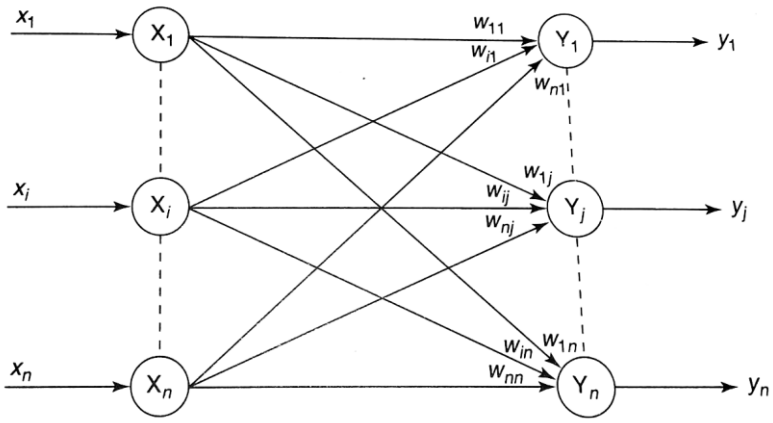


Fig 4.6: Autoassociative network

Step 0: Initialize all the weights to zero,

$$w_{ij} = 0 \quad (i = 1 \text{ to } n, \quad j = 1 \text{ to } n)$$

Step 1: For each of the vector that has to be stored perform Steps 2–4.

Step 2: Activate each of the input unit,

$$x_i = s_i \ (i = 1 \text{ to } n)$$

Step 3: Activate each of the output unit,

$$y_j = s_j \ (j = 1 \text{ to } n)$$

Step 4: Adjust the weights,

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j$$

The weights can also be determined by the formula

$$W = \sum_{p=1}^{P} s^T(p) s(p)$$

4.9 Heteroassociative memory network

1. In this network, the training input and the target output vectors are different.
2. The determination of weights is done by either using Hebb rule or delta rule.
3. The net finds an appropriate output vector, corresponds to an input vector x, that may be either one of the stored patterns or a new pattern.

Architecture

The input layer consists of n number of input units and the output layer consists of m number of output units. There is a weighted connection between the input and output layers. Here, the input and output are not correlated with each other.

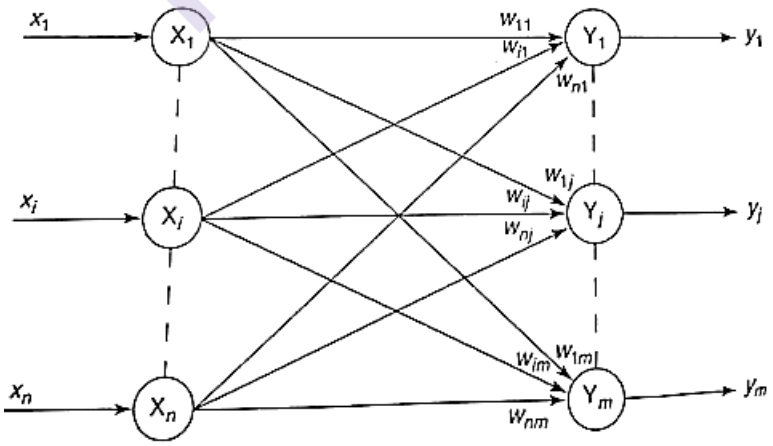


Fig 4.7: Heteroassociative network

4.10 Bidirectional Associative Memory (BAM)

1. The BAM network performs forward and backward associative searches for stored stimulus responses.
2. It a type of recurrent heteroassociative pattern matching network that encodes using Hebbian learning rule.
3. BAM neural nets can respond either ways from input and output layers.
4. It consists of two layers of neurons which are connected by directed weight path connections.
5. The network dynamics involves two layers of interaction until all the neurons reach equilibrium.

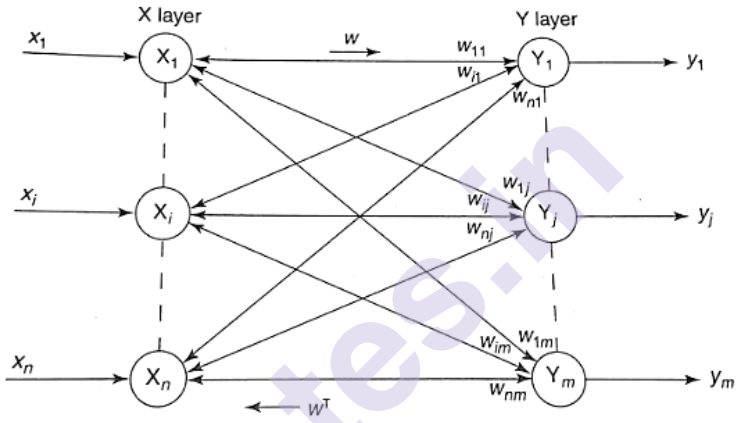


Fig: 4.8 Bidirectional associative memory net

4.11 Hopfield Networks

1. These networks were developed by John. J. Hopfield.
2. Through his work, he promoted construction of the hardware chips.
3. These networks are applied in associative memory and optimization problems.
4. They are basically of two types -discrete and continuous Hopfield networks.

Discrete Hopfield networks- The Hopfield networks is an autoassociative fully interconnected single-layer feedback network with fixed weights.
It works in discrete fashion. The network takes two-valued inputs -binary or bipolar. In this network, only one unit updates its activation at a time.
The usefulness of content addressable memory is realized by discrete Hopfield net.

Continuous Hopfield networks- In this network, time is considered to be a continuous variable. These networks are used for solving optimization problems like travelling salesman problems. These networks can be realized as an electronic circuit. The nodes of these Hopfield networks have continuous graded output. The total energy of the network decreases continuously with time.

QUSETIONS

1. Define Content addressable memory
2. What are the two main types of associative memory?
3. What are Back Propagation networks?
4. Explain the architecture and working of Radial basis function networks.
5. What is Bidirectional associative memory network?
6. Write a short note on Hopfield network.

REFERENCES

1. **“**Principles of Soft Computing”, by S.N. Sivanandam and S.N. Deepa, 2019, Wiley Publication, Chapter 3 and Chapter 4.
2. Related documents, diagrams from blogs, e-resources from RC Chakraborty lecture notes.

❖❖❖

Unit 3

5

UNSUPERVISED LEARNING

Unit Structure

5.0 Introduction

In this learning, there exists no feedback from the system (environment) *w* indicate the desired outputs of a network. The network by itself should discover any relationships of interest, such as features, patterns, contours, correlations or categories, classification in the input data, and thereby translate the discovered relationships into outputs. Such networks are also called self-organizing networks. An unsupervised learning can judge how similar a new input pattern is to typical patterns already seen, and the network gradually learns what similarity is; the network may construct a set of axes along which to measure similarity to previous patterns, i.e., it performs principal component analysis, clustering, adaptive vector quantization and feature mapping.

For example, when net has been trained to classify the input patterns into any one of the output classes, say, P, Q, R, S or T, the net may respond to both the classes, P and Q or R and S. In the case mentioned, only one of several neurons should fire,

i.e., respond. Hence the network has an added structure by means of which the net is forced to make a decision, so that only one unit will respond. The process for achieving this is called competition. Practically, considering a set of students, if we want to classify them on the basis of evaluation performance, their score may be calculated, and the one whose score is higher than the others should be the winner. The same principle adopted here is followed in the neural networks for pattern classification. In this case, there may exist a tie; a suitable solution is presented even when a tie occurs. Hence these nets may also be called competitive nets, the extreme form of these competitive nets is called winner-take-all.

The name itself implies that only one neuron in the competing group will possess a nonzero output signal at the end of competition.

There exist several neural networks that come under this category. To list out a few: Max net, Mexican hat, Hamming net, Kohonen self-organizing feature map, counter propagation net, learning vector quantization (LVQ) and adaptive resonance theory (ART).

The learning algorithm used ·m most of these nets is known as Kohonen learning. In this learning, the

units update their weights by forming a new weight vector, which is a linear combination of the old weight vector and the new input vector. Also, the learning continues for the unit whose weight vector is closest to the input vector. The weight updation formula used in Kohonen learning for output cluster unit j is given as

$$w_j(\text{new}) = w_{\Theta j}(\text{old}) + \alpha \left[x - w_{\Theta j}(\text{old})\right]$$

where x is the input vector; $w_{\Theta j}$ the weight vector for unit j; α the learning rate whose value decreases

monotonically as training continues. There exist two methods to determine the winner of the network during competition. One of the methods for determining the winner uses the square of the Euclidean distance between the input vector and weight vector, and the unit whose weight vector is at the smallest Euclidean distance from the input vector is chosen as the winner. The next method uses the dot product of the input vector and weight vector. The dot product between the input vector and weight vector is nothing but the net inputs calculated for the corresponding duster units. The unit with the largest dot product *is* chosen as the winner and the weight updation is performed over it because the one with largest

ot product corresponds to the smallest angle between the input and weight vectors, f both are of unit length.

.1. Fixed Weight Competitive Nets

hese competitive nets arc those where the weights remain fixed, even during aining process. The idea of competition is used among neurons for enhancement f contrast in their activation functions. These are

Iaxnet, Mexican hat and Hamming net.

Iaxnet

he Maxnet serves as a sub net for picking the node whose input is larger.

rchitecture of Maxnet

he architecture of Maxnet is shown in Figure 5·1, where fixed symmetrical eights are present over the

eighted interconnections. The weights between the neurons are inhibitory and ixed. The Maxnet with this structure can be used as a subnet to select a particular ode whose net input is the largest.

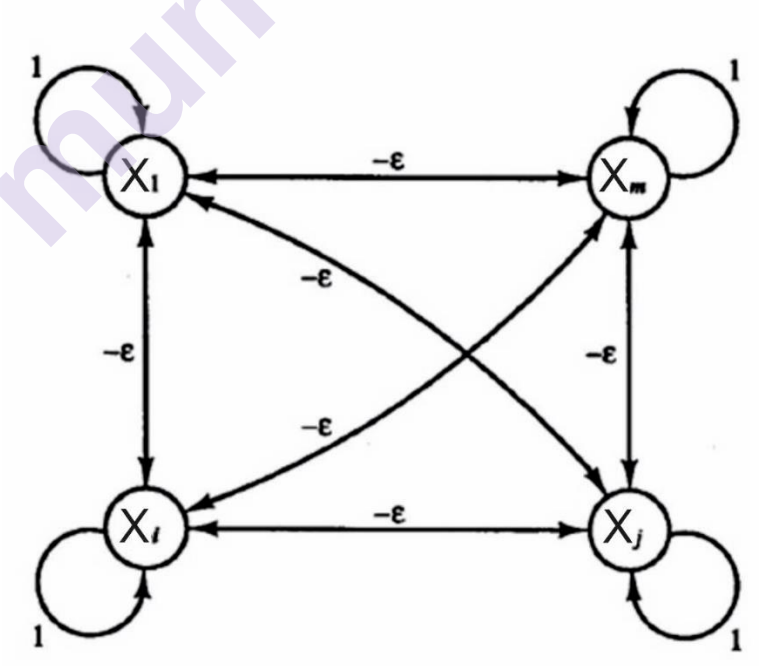


Figure 5.1 Maxnet Structure

Testing/Application Algorithm of Maxnet:

Step
0: Initial weights and initial activations are ser. The weight is set as $[0 < \varepsilon < 1/m]$, where "m^n is the total number of nodes. Let

$$x_j(0) = \text{ input to the node } X_j$$

and

$$w_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\varepsilon & \text{if } i \neq j \end{cases}$$

Step 1: Perform Steps $2 - 4$, when stopping condition is false. Step 2: Update the activations of each node. For $j = 1$ to m,

$$x_j(n \in w) = f\left[x_j(0.d) - \varepsilon \sum_{i \neq j} x_k(old)\right]$$

Step
3: Save the acrivarions obtained for use in the next iteration. For $j = 1$ to m,
$x_j(\text{ oid }) = x_j$ (new)
Step 4: Finally, test the stopping condition for convergence of the network. The following is the stopping condition: If more than one node has a nonzero activation, continue; else stop.

5.2 Mexican Hat Net

In 1989, Kohonen developed the Mexican hat network which is a more generalized contrast enhancement

network compared to the earlier Maxner. There exist several "cooperative neighbors" (neurons in close proximity) to which every other neuron is connected by excitatory links. Also, each neuron is connected over inhibitory weights to a number of" competitive neighbors" {neurons present farther away). There are several oilier fanher neurons ro which the connections between the neurons are nor established. Here, in addition to the connections within a particular laye·r Of neural net, the neurons also receive some other external signals.

This interconnection pattern is repeated for several other neurons in the layer.

5.2.1 Architecture of Mexican Hat Net

The architecture of Mexican hat is shown in Figure 5·2, with the interconnection pattern for node Xi. The

neurons here are arranged in linear order; having positive connections between Xi and near neighboring units, and negative connections between Xi and farther away neighboring units. The positive connection region is called region of cooperation and the negative connection region is called region of competition. The size of these regions depends on the relative magnitudes existing between the positive and negative weights and also on the topology of regions such as linear, rectangular, hexagonal grids, ere. In Mexican Hat, there exist two symmetric regions around each individual neuron.

The individual neuron in Figure 5-2 is denoted by Xi. This neuron is surrounded by other neurons Xi+ 1,

Xi-1, Xi+2, *Xi-2,* .... The nearest neighbors to the individual neuron Xi are Xi+1, Xi-1. Xi+2• and Xi-2·

Hence, the weights associated with these are considered to be positive and are denoted by WI and w2. The

farthest neighbors m the individual neuron Xi are taken as Xi+3 and Xi-3, the weights associated with these are negative and are denoted by w3. It can be seen chat Xi+4 and Xi-4 are not connected to the individual neuron Xi, and therefore no weighted interconnections exist between these connections. To make it easier, the units present within a radius of 2 [query for unit] to the unit Xi are connected with positive weights, the units within radius 3 are connected with negative weights and the units present further away from radius 3 are not connected in any manner co the neuron Xi.

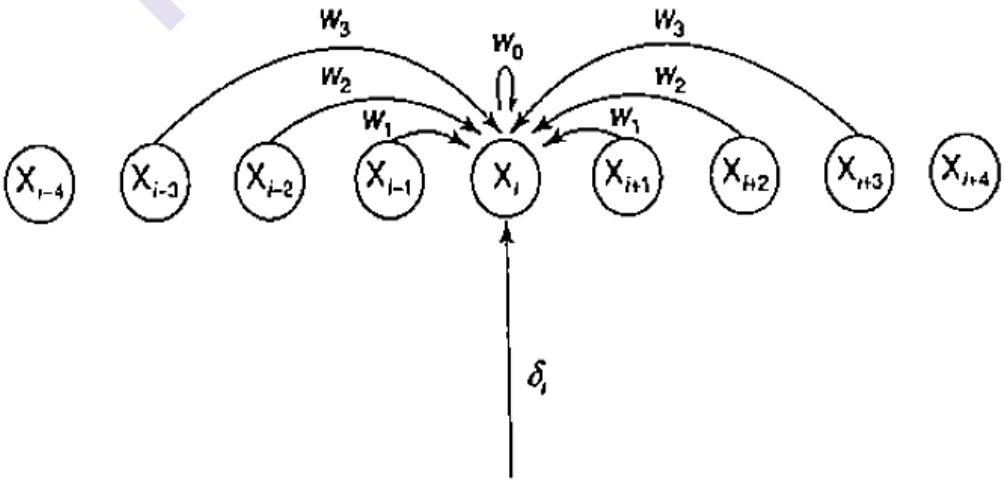


Figure 5.2 Structure of Maxican Hat

5.2.3 Flowchart of Mexican Hat Net

The flowchart for Mexicann hat is shown in Figure 5-3. This dearly depicts the flow of the process performed in Mexican Hat Network.

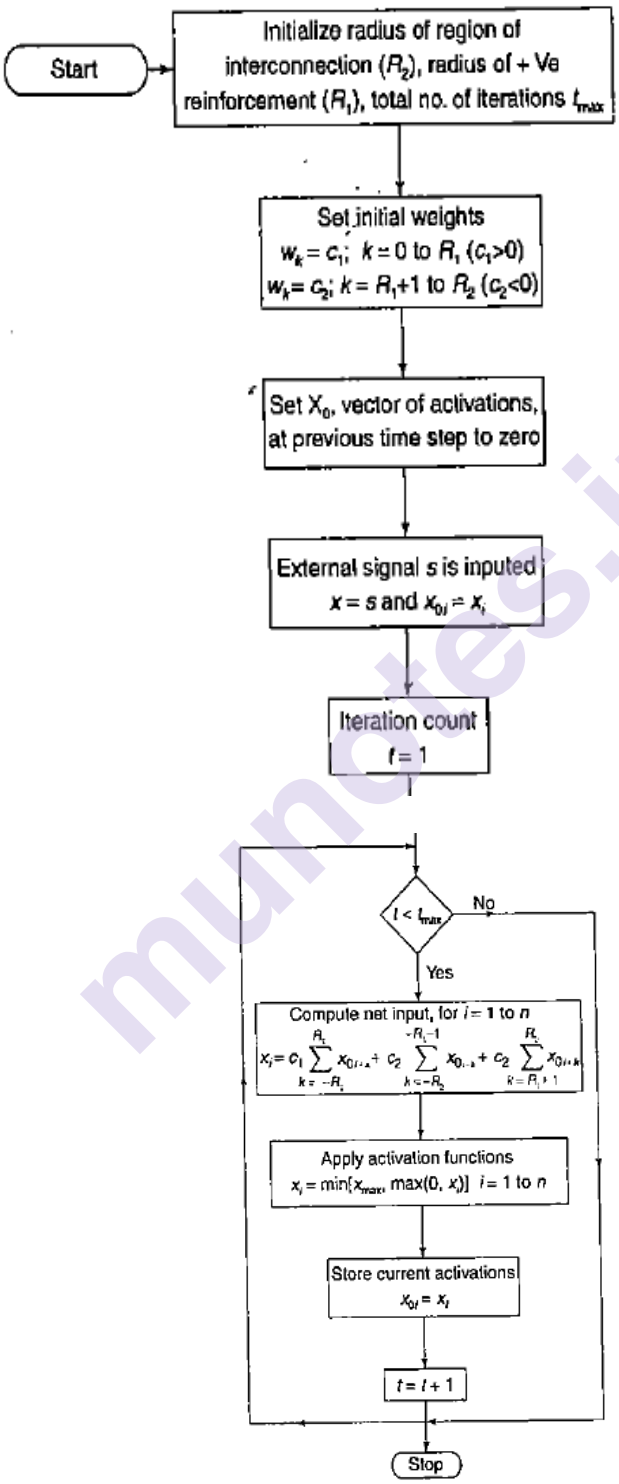


Figure 5.3. Flowchart of Mexican Hat

5.2.3 Algorithm of Mexican Hat Net:

The various parameters used in the training algorithm are as shown below.

R_2 = radius of rcgions of interconnections

X_{i+k} and X_{f-k} are conniected to the individual units X_i for $k = 1$ to R_2.

R_1 = adrus of tegion with positive reinforcement $(R_1 < R_2)$

W_k = weight berween X_i and the unis X_{i+k} and X_{i-k}

$0 \leqslant k \leqslant R_1,\ w_k =$ positive
$R_1 \leqslant k \leqslant R_2,\ w_k =$ negative
$t =$ external input signal
$x =$ vector of accivation
$x_0 =$ vecior of activations at previous time step

$t_{\max}$ = total number of iterations of contmst enhancemen.

Here the iteration is started only with the incoming of the external signal presented to the network.

Step 0: The parameters R_1, R_2, t_{max} are initialized accordingly. Initialize weights as

$w_k = c_1$ for $k = 0, \ldots, R_1$ (where $c_1 > 0$)
$w_k = c_2$ for $k = R_1 + 1, \ldots, R_2$ (where $c_2 < 0$)

Initialize $x_0 = 0$.

Step 1: Input the external signal s:

$$x = s$$

The activations occurring are saved in array x_0. For $i = 1$ to n,

$$x_{0i} = x_i$$

Once activations are stored, set iteration counter $t = 1$.

Step 2: When t is less than $t_{\max}$, perform Steps 3-7.

Srep 3: Calculace net input. Fot $i = 1$ to n,

$$x_i = c_1 \sum_{k=-R_1}^{R_1} x_{0_{i+}} + c_{k=-R_2}^{-R_1-1} x_{0_{i+k}} + c_2 \sum_{k=R_1+1}^{R_2} x_{0_{i+k}}$$

Step 4: Apply the activation function. For $i = 1$ to n,

$$x_i = \text{m}\ \ [x_{\max}, \text{m}\ \ (0, x_i)]$$

Step 5: Save the current activations in x_0, i.e., for $i = 1$ to n,

$$x_{0i} = x_i$$

Step 6: Increment the iteration counter:

$$t = t + 1$$

Step 7: Test for stopping condition. The following is the stopping condition: If $t < t_{\text{Ex.}}$ then continue Else stop. The positive reinforcement here has the capacity to increase the activation of units with larger initial activations and the negative reinforcement has the capacity' to reduce the activation of unis with smaller initial activations. The activation function used here for unit X_i at a particular time instant $''t''$ is given by

$$x_i(\lambda) = f\left[s_i(t) + \sum_k w_k x_{i+k} + k(t-1)\right]$$

The terms present within the summation symbol are the weighted signals that arrived from other units α the previous time step.

5.3 Hamming Network

The Hamming network selects stored classes, which are at a maximum Hamming distance (H) from the

noisy vector presented at the input (Lippmann, 1987). The vectors involved in this case are all binary and

bipolar. Hamming network is a maximum likelihood classifier that determines which of several exemplar

vectors (the weight vector for an output unit in a clustering net is exemplar vector or code book vector for the pattern of inputs, which the net has placed on that duster unit) is most similar to an input vector (represented *as* an n~tuple). The weights of the net are determined by the exemplar vectors. The difference between the tom! number of components and the Hamming distance between the vecrors gives the measure of similarity between the input vector and stored exemplar vcctors.lt is already discussed the Hamming distance between the two vectors is the number of components in which the vectors differ.

Consider two bipolar vectors *x* and y; we use a relation

$x \cdot y = a - d$

where *a* is the number of components in which the vectors agree, *d* the number of components in which the vectors disagree. The value *"a - d"* is the Hamming distance existing between two vectors. Since, the total number of components is n, we have,

$$n = a + d$$

i.e., $$d = n - a$$

On simplification, we get

$$\begin{aligned} x \cdot y &= a - d \\ x \cdot y &= a - (n - a) \\ x \cdot y &= 2a - n \\ 2a &= x \cdot y + n \\ a &= \frac{1}{2}(x \cdot y) + \frac{1}{2}(n) \end{aligned}$$

From the above equation, it is clearly understood that the weights can be set to one-half the exemplar vector and bias can be set initially to *n/2*. By calculating the unit with the largest net input, the net is able to locate a particular unit that is closest to the exemplar. The unit with the largest net input is obtained by the Hamming net using Maxnet as its subnet.

5.3.1. Architecture of Hamming Network:

The architecture of Hamming network is shown in Figure 5-4. The Hamming network consists of two layers. The first layer computes the difference between the total number of components and Hamming distance between the input vector *x* and the stored pattern of vectors in the feed-forward path. The efficient response in this layer of a neuron is the indication of the minimum Hamming distance value between the input and the category, which this neuron represents. The second layer of the Hamming network is composed of Maxnet (used as a subnet) or a Winner-take-all network which is a recurrent network The Maxnet is found to suppress the values at Maxnet output nodes except the initially maximum output node of the first layer.

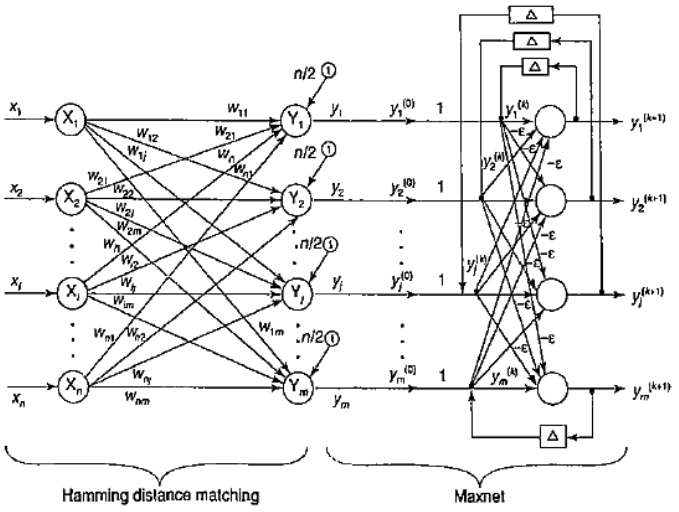


Figure 5.4 Structure of Hamming Network

5.3.2 Testing Algorithm of Hamming Network:

The given bipolar input vector is *x* and for a given set of *"m"* bipolar exemplar vectors say e(l),.

e(j), ... , e(m), the Hamming network is used to determine the exemplar vector that is closest m the input

vector *x.* The net input entering unit Yj gives the measure of the similarity between the input vector and

exemplar vector. The parameters used here are the following:

n = number of input units (number of components of *input-output* vector)

m= number of output units (number of components of exemplar vector)

e(j)= jth exemplar vector, i.e.,

e(j) = [e1 (j), ... , e_j(j), ... , e_n(j)]

The testing algorithm for the Hamming Net is as follows:

Step 0: Initialize the weights. For $i = 1$ ro n and $j = 1$ ro m,

$$w_{ij} = \frac{e_i(j)}{2}$$

Initialize the bias for storing the $'m^n$ exemplar vectors. For $j = 1$ to m,

$$b_j = \frac{n}{2}$$

Step 1: Perform Steps 2-4 for each inpuc vector x.

Step 2: Calculate the net input to each unit Y_j, i.e.,

$$y_{inj} = b_j + \sum_{i=1}^{\omega} x_i w_{ij},\ j = 1 \text{ to } m$$

Step 3: Initialize the activations for Maxnet, i.e.,

$$y_j(0) = y_{inj},\ j = 1 \text{ to } m$$

Step 4: Maxnet is found to iterate for finding the exemplar that best matches the inpur patterns.

5.4 Kohonen Self-Organizing Feature Maps

Feature's mapping is a process which converts the patterns of arbitrary dimensionality into a response of one- or two-dimensional arrays of neurons, i.e. it converts a wide pattern space into a typical feature space. The network performing such a mapping is called feature map. Apart from its capability to reduce the higher dimensionality, it has to preserve the neighborhood relations of the input patterns, i.e. it has to obtain a topology preserving map. For obtaining such feature maps, it is required to find a self-organizing array which consist of neurons arranged in a one-dimensional array or a two-dimensional array. To depict this, a typical network structure where each component of the input vector x is connected to each of nodes is shown in Figure 5-5.

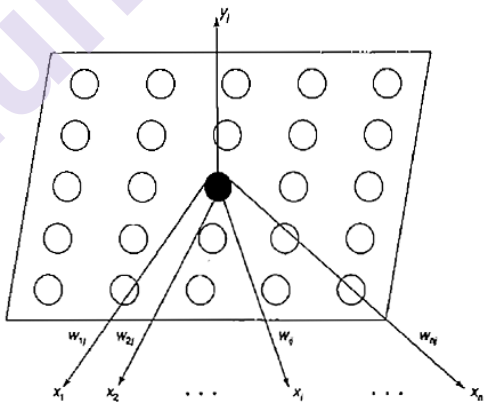


Figure 5.5 One-dimensional Feature mapping network

On the other hand, if the input vector is two-dimensional, the inputs, say *x(a,* b), can arrange themselves

in a two-dimensional array defining the input space *(a,* b) as in Figure 5-6. Here, the two layers are fully connected.

The topological preserving property is observed in the brain, bur nor found in any other artificial neural network.

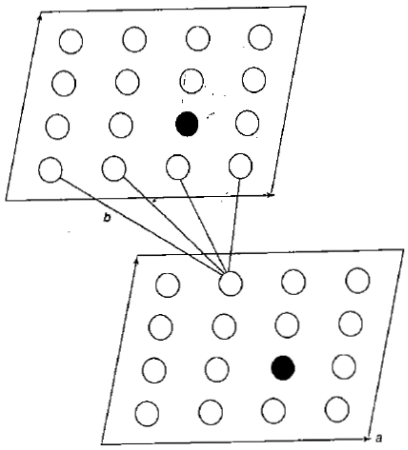


Figure 5.6. Two dimensional feature mapping network

5.4.1 Architecture of Kohonen Self-Organizing Feature Maps

Consider a linear array of cluster units as in Figure 5-7. The neighborhoods of the units designated by "o" of radii Ni(k1), Ni(k2) and Ni(k,), *k1* > *k,* > *k,,* where *k1* = 2, *k2* = 1, *k3* = 0.

For a rectangular grid, a neighborhood (Ni) of radii k1, k2, and k3 is shown in Figure 5-8 and for a

hexagonal grid the neighborhood is shown in Figure 5-9. In all the three cases (Figures 5-7-5-9), the unit with "#" symbol is the winning unit and the other units are indicated by "o." In both rectangular and hexagonal grids, *k1* >k2 > *k3,* where k1 = 2, *k2* = 1, *k3* = 0.

For rectangular grid, each unit has eight nearest neighbors but there are only six neighbors for each unit in

the case of a hexagon grid. Missing neighborhoods may just be ignored. A typical architecture of Kohonen self-organizing feature map (KSOFM) is shown in Figure 5-10.

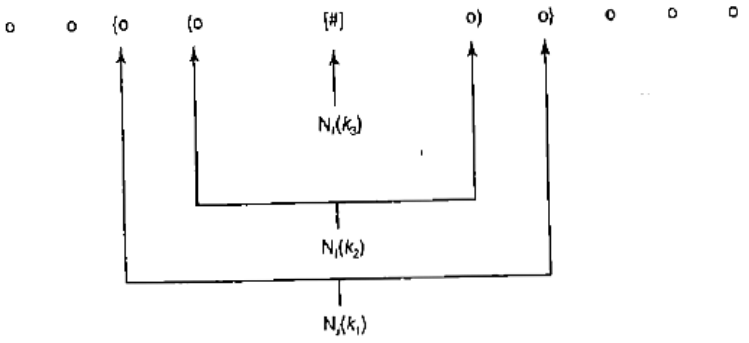
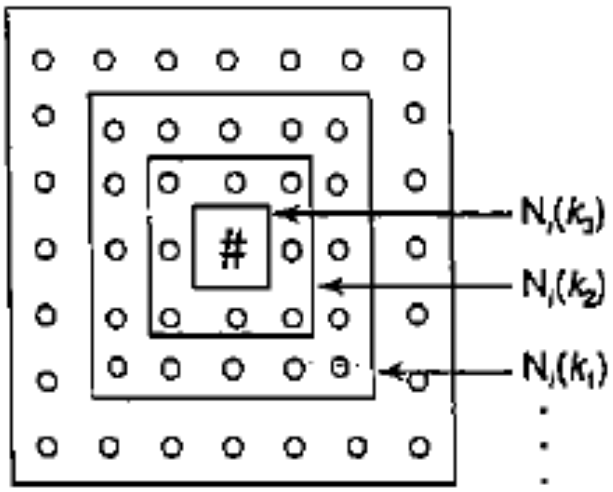


Figure 5.7. Linear array of cluster units



Fifure 5.8.Rectanguler grid

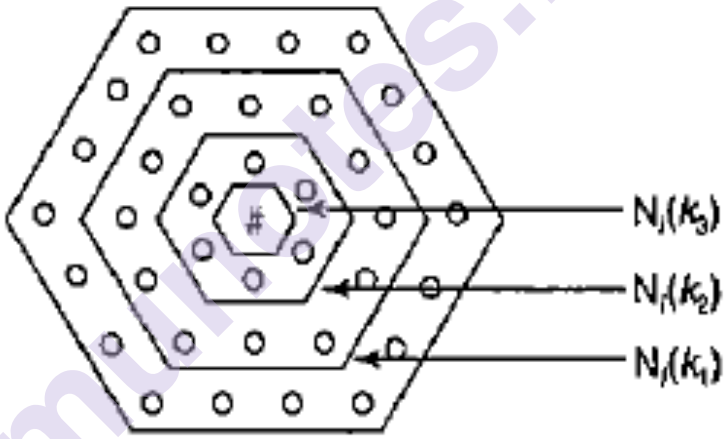


Figure 5.9. Hexagonal grid

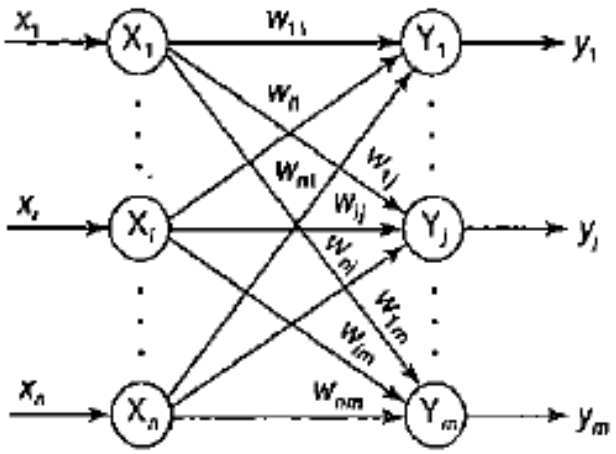


Figure 5.10. Kohonen self organizing feature map architecture

Flowchart of Kohonen Self-Organizing Feature Maps

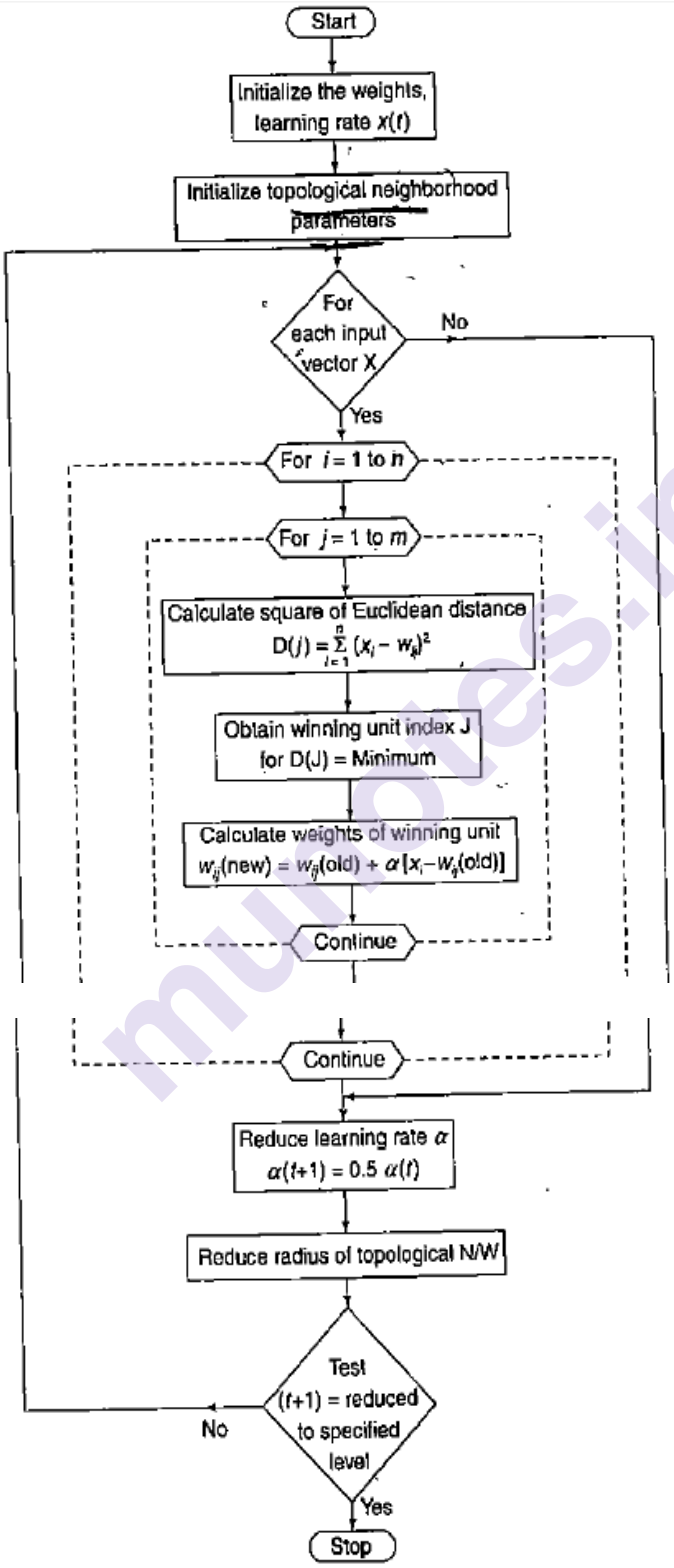


Figure 5.11. Flowchart for training process of KSOFM

5.4.2. Training Algorithm of Kohonen Self-Organizing Feature Maps:

Step 0: - Initialize the weights w_{ij} : Random values may be assumed. They can be chosen as the same range of values as the component if input vector. If information related to distribution of clusters is known, the initial weights. can bet taken to reflect that prior knowledge.

- Set topological neighborhood parameters: As clustering progresses, the radius of the neighborhood Decreases
- Initialize the learning rate α : It should be a slowly decreasing function of time.

Step 1: Perform Steps $2 - 8$ when stopping condition is false.

Step 2; Perform Steps 3-5 for each input vector x.

Step 3: Compute the square of the Euclidean distance, i.e., for each $j = 1$ to m,

$$D(j) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(x_i - w_{ij}\right)^2$$

Step 4: Find the winning unit index J, so that D(J) is minimum. (In Steps 3 and 4 , dot product method can also be used to find the winner, which is basically the calculation of net input, and the winner will be the one with the largest dot product.)

Step 5: For all units j within a specific neighborhood of J and for all i, calculate the new weights:

$$w_{jj}(\text{new}) = w_{ij}(\text{o} \mid \text{d}) \pm \alpha_0\left[x_i - w_{ij}(\text{ old })\right]$$

Or

$$w_{ij}(\text{ new }) = (1 - \alpha) w_{ij}(\text{ old }) + \alpha x_i$$

Step 6: Update the learning rate α using the formula $\alpha(t + 1) = 0.5\alpha(t)$.

Step 7: Reduce radius of topological neighborhood at specified time intervals.

Step 8 : Test for stopping condition of the network

5.5. Kohonen Self-Organizing Motor Map :

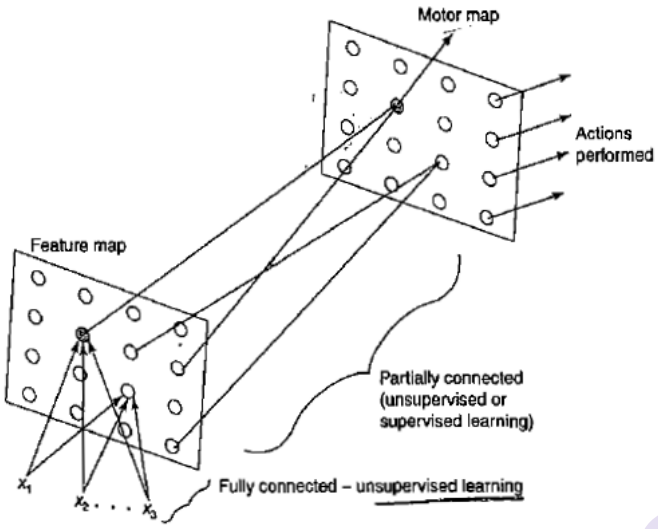


Figure 5.12. Architecture of kohonen self organizing motor map

The extension of Kohonen feature map for a multilayer network involve the addition of an association layer to the output of the self-organizing feature map layer. The output node is found to associate the desired output values with certain input vectors. This type of architecture is called as Kohonen self-organizing motor map and layer that is added is called a motor map in which the movement command,

are being mapped into two-dimensional locations of excitation. The architecture of KSOMM is shown in

Figure 5-12. Here, the feature map is a hidden layer and this acts as a competitive network which classifies the input vectors.

5.6 Learning Vector Quantization (LVQ)

LVQ is a process of classifying the patterns, wherein each output unit represents a particular class. Here, for each class several units should be used. The output unit weight vector is called the reference vector or code book vector for the class which the unit represents. This is a special case of competitive net, which uses supervised learning methodology. During the training the output units are found to be positioned to approximate the decision surfaces of the existing Bayesian classifier. Here, the set of training patterns with known classifications is given to the network, along with an initial distribution of the reference vectors. When the training process is complete, an LVQ net is found to classify an input vector by assigning it to the

same class as that of the output unit, which has its weight vector *very close to the* input vector. Thus LVQ is a classifier paradigm that adjusts the boundaries between categories to minimize existing misclassification. LVQ is used for optical character recognition, converting speech mro phonemes and other application as well.

5.6.1. Architecture of LVQ:

Figure 5-13 shows the architecture of LVQ. From Figure 5-13 it can be noticed that there exists input layer with *"n"* unit; and output layer with *"m"* units. The layers are found to be fully interconnected with weighted linkage acting over the links.

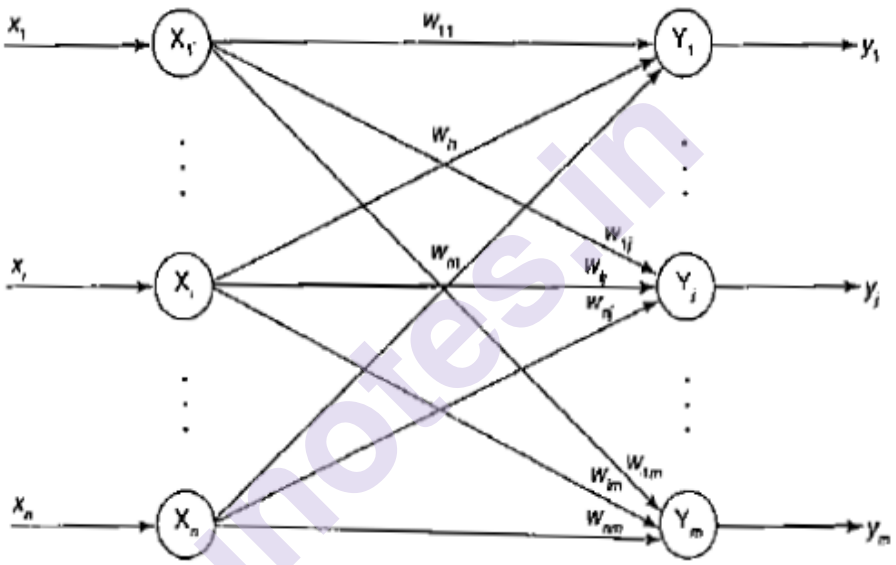


Figure 5.13. Architecture of LVQ

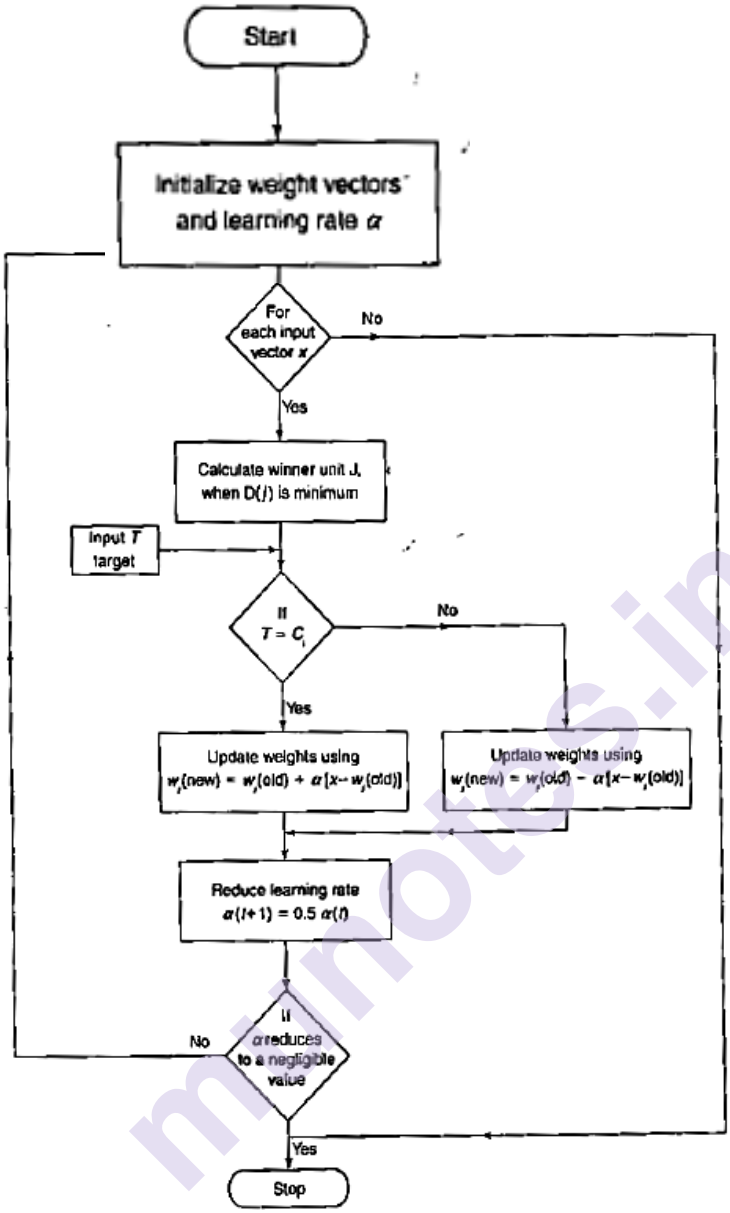
5.6.2. Flowchart of LVQ:

The parameters used for the training process of a LVQ include the following:

$x =$ taaining vector $(x_1, \dots, x_i, \dots, x_n)$
$T =$ category or class for the training vector x
$w_j =$ weight vector for jh outpus unit $(z_{1j}, \dots, w_{ij}, \dots, w_{vj})$

$c_j =$ cluster or class or category associated with jh output unit.

The Euclidean distance of jh outpui unit is $D(j) = \sum(x_i - w_{ij})^2$. The flowchart indicaring the flow of training process is shown in Figure $5 - 14$.



5.6.3. Training Algorithm of LVQ:

Step 0: Initialize the reference vectors. This can be done using the following steps.

- From the given sec of training vectors, take the first " m " (number of clusters) training vectors and use them as weighc vectors, the remaining vectors can be used for training.
- Assign the initial weights and classifications random.1y.
- K -means chustering mechod.

Set initial learning rate α.

Step1: Perform Steps $2-6$ if the stopping condition is false.

Step 2: Perform Steps 3-4 for each training input vector x.

Step 3: Calculate the Euclidean distance; for $i = 1$ to $n, j = 1$ to m,

$$D(j) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(x_i - w_{ij}\right)^2$$

Find the winning unit index J, when $D(J)$ is minimum.

Step 4: Update the weights on the winning unit, w, using the following conditions.

If $T = q$, then $u_j(new) = u_j(old) + \alpha\left[x - w_j(old\}\right]$

If $T \neq q$, then $u_j(new) = ug(old) - \alpha\left[x - u_j(0]d)\right]$

Step 5: Reduce the learning rate α.

Step 6: Test for the stopping condition of the training process.

(The stopping conditions may be fixed number of epochs or if learning rate has reduced to a negligible value.)

5.7 Counter propagation Networks

They are multilayer networks based on the combinations of the input, output and clustering layers. The applications of counter propagation nets are data compression, function approximation and pattern association. The counter propagation network is basically constructed from an instar-outstar model. This model is a three-layer neural network that performs input-output data mapping, producing an output vector yin response to an input vector *x,* on the basis of competitive learning. The three layers in an instar-outstar model are the input layer, the hidden (competitive) layer and the output layer. The connections between the input layer and the competitive layer are the instar structure, and the connections existing between the competitive layer and the output layer are the outstar structure.

There are two stages involved in the training process of a counter propagation net. The input vectors are

clustered in the first stage. Originally, it is assumed that there is no topology included in the counter propagation network. However, on the inclusion of a linear

topology, the performance of the net can be improved. The dusters are formed using Euclidean distance method or dot product method. In the second stage of training, the weights from the cluster layer units to the output units are tuned to obtain the desired response.

There are two types of counter propagation nets:

(i) Full counter propagation net

(ii) Forward-only counter propagation net

5.7.1. Full Counter propagation Net:

Full counter propagation net (full CPN) efficiently represents a large number of vector pairs *x:y* by adaptively constructing a look-up-table. The approximation here is x*.y*, which is based on the vector pairs x*:y,* possibly with some distorted or missing elements in either vector or both vectors. The network is defined to approximate a continue function, defined on a compact set A. The full CPN works best if the inverse function f^{-1} exists and is continuous. The vectors *x* and *y* propagate through the network in a counter flow manner to yield output vectors x* and y*, which are the approximations of *x* and y, respective. During competition, the winner can be determined either by Euclidean distance or by dot product method. In case of dot product method, the one with the largest net input is the winner. Whenever vectors are to be compared using the dot product metric, they should be normalized. Even though the normalization can be performed without loss of information by adding an extra component, yet to avoid the complexity Euclidean distance method can be used. On the basis of this, direct comparison can be made between the full CPN and forward-only CPN.

For continuous function, the CPN is as efficient as the back-propagation net; it is a universal continuous function approximate. In case of CPN, the number of hidden nodes required to achieve a particular level

of accuracy is greater than the number required by the back-propagation network. The greatest appeal of

CPN is its speed of learning. Compared to various mapping networks, it requires only fewer steps of training to achieve best performance. This is common for any hybrid learning method that combines unsupervised learning (e.g., instar learning) and supervised learning (e.g., outsrar learning).

As already discussed, the training of CPN occurs in two phases. In the input phase, the units in the duster

layer and input layer are found to be active. In CPN, no topology is assumed for the cluster layer units; only the winning units are allowed to learn. The weight pupation learning rule on the winning duster units is

$$v_{ij}(\text{new}) = v_i(0ld) + \alpha[x_i - v_{ij}(old)], \quad i = 1 \text{ to } n$$
$$w_{k/}(\text{new}) = w_{kj}(old) + \beta(y_k - w_b(old)], \quad k = 1 \text{ to } m$$

In the second phase of training, only the winner unit J remains active in the cluster layer. The weights between the winning cluster unit J and the output units are adjusted so that the vector of activations of the units in the Y-output layer is y* which is an approximation to the input vector *y* and X* which is an approximation to the input vector *x.* The weight updating for the units in the Y-output and X-output layers are

$$u_{jk}(new) = u_{jk}(old) + a[y_k - u_{jk}(old)], \; k = 1 \text{ to } m$$
$$t_j(new) = t_j(old) + b[x_i - t_j(old)], \; i = 1 \text{ to } n$$

5.7.2. Architecture of Full Counter propagation Net

The general structure of full CPN is shown in Figure 5-15. The complete architecture of full CPN is shown in Figure 5-16.

The four major components of the instar-outstar model are the input layer, the instar, the competitive layer and the outstar. For each node *i* in the input layer, there is an input value $x_{i;}$. An instar responds maximally to the input vectors from a particular duster. All the instar are grouped into a layer called the competitive layer.

Each of the instar responds maximally to a group of input vectors in a different region of space. This layer of instars classifies any input vector because, for a given input, the winning instar with the strongest response identifies the region of space in which the input vector lies. Hence, it is necessary that the competitive layer single outs the winning instar by setting its output to a nonzero value and also suppressing the other outputs to zero. That is, it is a winner-take-all or a Maxnet-type network. An outstar model is found to have all the nodes in the output layer and a single node in the competitive layer. The outstar looks like the fan-out of a node. Figures 5-17 and 5-18 indicate the units that are active during each of the two phases of training a full CPN.

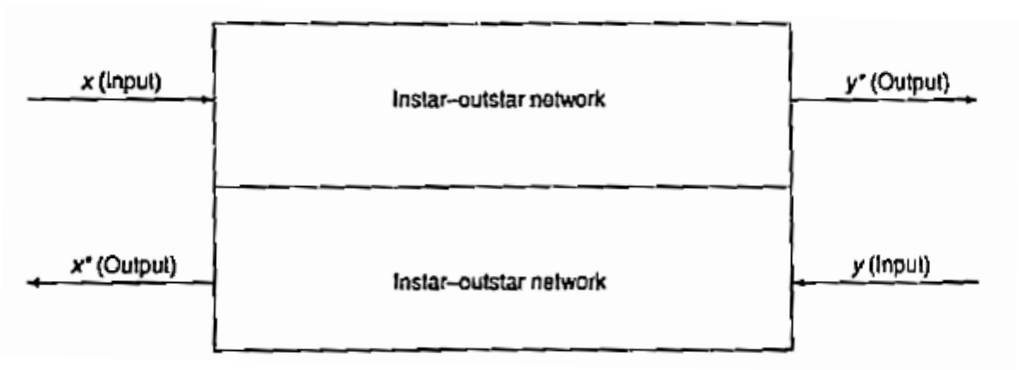


Figure 5.15.General Structure of full CPN

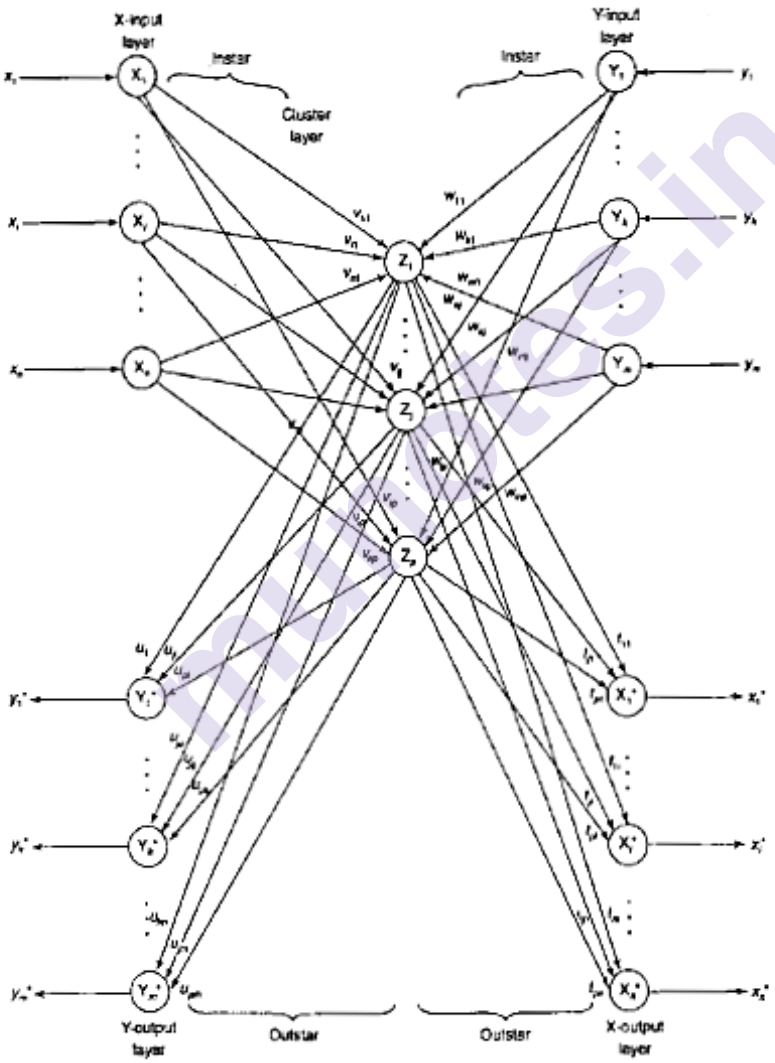


Figure 5.16. Architecture of full CPN

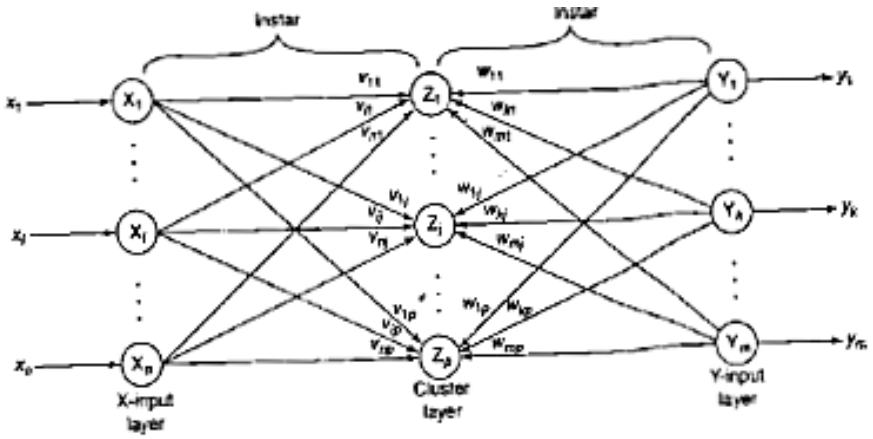


Figure 5.17 First phase of training of full CPN

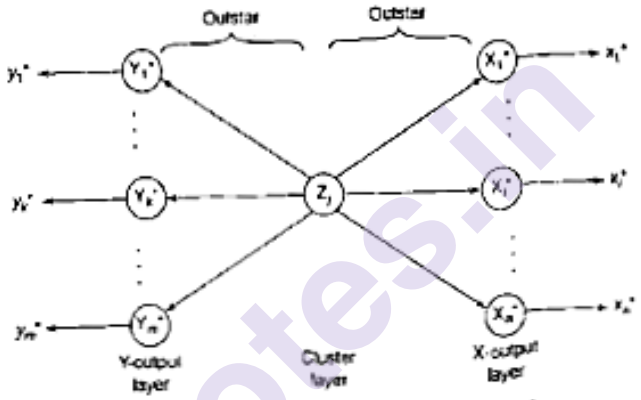


Figure 5.18 Second phase of training of full CPN

5.7.3. Training Algorithm of Full Counter propagation Net:

Step 0: Set the initial weighrs and the initial learning rate.

Step 1: Perform Sreps $2-7$ if stopping condition is folse for phase I training.

Step 2: For each of the training input vector pair $x: y$ presented, perform Steps $3-5$.

Step 3: Make the X-input layer activations to vector X. Make the Y-inpur layer acrivations to vector Y.

Step 4: Find the winning cluster unit. If dot product method is used, find the cluster unit z_j with target net inpur: for $j = 1$ to p.

$$s_{nj} = \sum_{i=1}^{n} x_i v_{ij} + \sum_{k=1}^{m} \gamma_k w_{k_j}$$

If Euclidean distance merhod is used, find the cluster unis z_1 whore squared distance from input vecrors is the smallest:

$$D_j = \sum_{i=1}^{n} \left(x_i - v_{ij}\right)^2 + \sum_{k=1}^{im} \langle \gamma_k - u^n k_i \rangle^2$$

If there occurs a tie in case of selection of winner unit, the unit with the smallest index is the winner. Take the winner unit index as J.

Step 5: Update the weights over the calculated winner unit z_j.

Step 6: Reduce the learning rates.

$$\alpha(t+1) = 0.5\alpha(t): \ \beta(t+1) = 0.5\beta(t)$$

Step 7: Test stopping condition for phase I training.

Step 8: Perform Steps 9-15 when stopping condition is false for phase II training.

Step 9: Perform Steps $10 - 13$ for each training input pair $x; y$. Here α and β are small constant values.

Step 10: Make the X-input layer activations to vector x. Make the Y-input layer activations to vectot y.

Step 11: Find the winning cluster unit (use formulas from Step 4). Take the winner unit index as j.

Step 12: Update the weights entering into unit 3).

$$\text{For } i = 1 \text{ to } n, \ v_{ij}(\text{ new }) = v_{ij}(\text{old}) + \alpha\left[x_i - v_{ij}(\text{old})\right]$$
$$\text{For } k = 1 \text{ to } m, \ w_{kj}(\text{new}) = w_{k/}(\text{old}) + \beta[y_k - w_k(\text{old})]$$

Step 13: Update the weights from unit z_j to ghe outpur layers.

$$\text{For } i = 1 \text{ to } n, \ c_j(\text{ new }) = t_j(\text{ old }) + b\left[x_i - t_j(o]d\right)\right]$$
$$\text{For } k = 1 \text{ to } m, \ u_{jk}(\text{ new }) = u_{jk}(\text{ old }) + a\left[y_k - u_{jk}(\text{ old })\right]$$

Step 14: Reduce the learning rates a and b.

$$a(t+1) = 0.5a(t); \ b(t+1) = 0.5b(t)$$

Step 15: Test stopping condition for phase II training.

5.7.4. Testing Algorithm of Full Counter propagation Net:

Step 0: Initialize the weights (from training algorithm).

Step 1: Perform Steps 2-4 for each input pair X: Y.

Step 2: Ser X-input layer activations to vector X. Ser Y-input layer activarions to vector Y.

Step 3: Find the cluster unir z_j that is closest to the input pair.

Step 4: Calculate approximations to x and y :

$$x_j^* = t_{ji}; y_k^* = u_{jk}$$

5.7.5. Forward Only Counter propagation Net:

A simplified version of full CPN is the forward-only CPN. The approximation of the function *y* = f(x) but not of *x = f(y)* can be performed using forward-only CPN, i.e., it may be used if the mapping from *x* to y is well defined but mapping from *y* to *x is* not defined. In forward-only CPN only the x-vectors are used to form the clusters on the Kohonen units. Forward-only CPN uses only the *x* vectors to form the clusters on the Kohonen units during first phase of training.

In case of forward-only CPN, first input vectors are presented to the input units. The cluster layer units compete with each other using winner-take-all policy to learn the input vector. Once entire set of training vectors has been presented, there exist reduction in learning rate and the vectors are presented again, performing several iterations. First the weights between the input layer and duster layer are trained. Then the weights between the cluster layer and output layer are trained. This is a specific competitive network, with target known. Hence, when each input vector is presented m the input vector, its associated target vectors are presented to the output layer. The winning duster unit sends its signal to the output layer. Thus each of the output unit has a computed signal *(w_{jk})* and die target value *(y_k).* The difference between these values is calculated; based on this, the weights between the winning layer and output layer are updated. The weight updation from input units to cluster units is done using the learning rule given below:

For i= 1 to n,

$$\cdot\, v_i(\text{ new }) = v_i \text{ fold }) + \alpha\left[x_i - v_{ij}(\text{ old })\right] = (1-\alpha)v_j(\text{ old }) + \alpha x_i$$

The weight updation from cluster units to output units is done using following the learning rule: For $k = 1$ to m,

$$w_{jk}(\text{ new }) = v_{jk}(\text{ old }) + a\left[y_k - w_{jk}(old)\right] = (1-a)w_{jk}(\text{ old }) + ay_k$$

The learning rule for weight updation from the duster units to output units can be written in the form of delta rule when the activations of the cluster units (z_j) are included, and is given as

$$w_{jk}(\text{ new }) = w_{jk}(0 \mid d) + nz_j\left[y_k - w_{jk}(\text{ old })\right\}$$

where

$$z_j = \begin{cases} 1 & \text{if } j = J \\ 0 & \text{if } j \neq J \end{cases}$$

This occurs when w_{jk} is interprered as the computed output (i.e., $y_k = w_{jk}$). In the formulation of forward-only CPN also, no topological structure was assumed.

5.7.6. Architecture of Forward Only Counter propagation Net:

Figure 5-20 shows the architecture of forward-only CPN. It consists of three layers: input layer, cluster (competitive) layer and output layer. The architecture of forward-only CPN resembles the back-propagation network, but in CPN there exists interconnections between the units in the duster layer (which are nor connected in Figure 5-20). Once competition is completed in a forward-only CPN, only one unit will be active in that layer and it sends signal to the output layer. As inputs are presented m the network, the desired outputs will also be presented simultaneously.

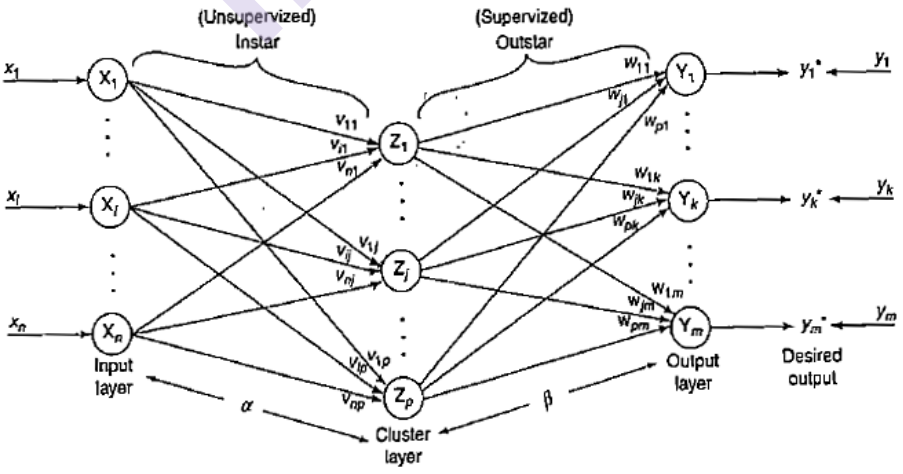


Figure 5.19 Architecture of forward only CPN

5.7.8. Training Algorithm of Forward Only Counter propagation Net:

Step 0: Initialize the weights and learning races.

Step 1: Perform Steps 2-7 when stopping condition for phase I training is false.

Step 2: Perform Steps 3-5 for each of training input X.

Step 3: Set the X-input layer acrivations to vector X.

Step 4: Compute the winning cluster unit (J). If dot product mechod is used, find the cluster unit zy Step wich the largest net input:

$$z_{inj} = \sum_{k=1}^{n} x_i v_{ij}$$

If Euclidean distance is used, find the cluster unit z_j square of whose distance from the input pattetn is smallest:

$$D_j = \sum_{i=1}^{n} \left(x_i - v_{ij}\right)^2$$

If there exists a tie in the selection of winner unit, the unit with the smallest index is chosen as the winner.

Step 5: Perform weight updation for unit z_j. For $i = 1$ to n,

$$v_{ij}(\text{ new }) = v_{ij}(\text{ old }) + \alpha\left[x_i - v_{ij}(\text{ old })\right]$$

Step 6: Reduce learning mte α

$$\alpha(t+1) = 0.5\alpha(t)$$

Step 7: Test the stopping condition for phase I training.

Step 8: Perform Steps $9 - 15$ when stopping condition for phase II training is false. (Set α a small constant value for phase II training.)

Step 9: Perform Steps 10-13 for each tmining input pait x.. .

Step 10: Ser X-input layer activations to vector X. Set Y-output layer activations to vector Y.

Step 11: Find the winning cluster unit (J) [use formulas as in Step 4].

Step 12: Update the weights into unit z_j. For $i = 1$ to n,

$$v_{ij}(\text{ new }) = v_{ij}(\text{ old }) + \alpha[x_i - v_i \text{ (old) } |$$

Step 13: Update the weights from unit z) to the output units. For $k = 1$ to m,

$$w_{jk}(\text{ new }) = w_{jk}(\text{ old }) + \beta\left[\eta_k - w_{jk}(\text{ old })\right]$$

Step 14: Reduce learning rate β, i.e.,

$$\beta(t+1) = 0.5\beta(t)$$

Step 15: Test the stopping condition for phase II training.

5.7.9. Testing Algorithm of Forward Only Counter propagation Net:

Step 0: Set initial weights. (The initial weights here are the weights obtained during training.)

Step 1: Present input vector X.

Step 2: Find unit J that is closest to vector X.

Step 3: Set activations of output units:

$$y_k = w_{jk}$$

5.8 Adaptive Resonance Theory Network

The adaptive resonance theory (ART) network, developed by Steven Grossberg and Gail Carpenter (1987), is consistent with behavioral models. This is an unsupervised learning, based on competition, that finds categories autonomously and learns new categories if needed. The adaptive resonance model was developed to solve the problem of instability occurring in feed-forward systems. There are two types of ART: ART 1 and ART 2. ART 1 is designed for clustering binary vectors and ART 2 is designed to accept continuous-valued vectors. In both the ners, input patterns can be presented in any order. For each pattern, presented to the network, an appropriate cluster unit is chosen and the weighs of the cluster unit are adjusted to let the cluster unit learn the pattern. This network controls the degree of similarity of the patterns placed on the same cluster units. During training, each training pattern may be presented several times. It should be noted that the mput patterns should not be presented on the same cluster unit, when it is presented each time. On the basis of this, the stability of the net is defined as that wherein a pattern is not presented o previous cluster units.

The adaptive resonance theory (ART) network, developed by Steven Grossberg and Gail Carpenter (1987), is consistent with behavioral models. This is an unsupervised learning, based on competition, that finds categories auconomously and learns new categories if needed. The adapdive resonance model was developed to solve the problem of instability oceutring in feed-forward systems. There are two types of ART: ART 1 and ART 2. ART 1 is designed for clustering binary vectors and ART 2 is designed to accept continuous-valued vectors. In both the ners, input patterns can be presented in any order. For each pattern, presented to the network, an appropriate cluster unit is chosen and the weighs of the cluster unit are adjusted to let the cluster unit learn the pattern. This network controls the degree of similarity of the patterns placed on the same cluster units. During training, each training pattern may be presented several times. It should be noted that the input patterns should not be presented on the same cluster unit, when it is presented each time. On the basis of this, the stability of the net is defined as that wherein a pattern is not presented (o previous cluster units The stability may be achieved by reducing the learning rates. The ability of the network to respond to a new pattern equally at any stage of learning is called as plastic: ART nets are designed to possess the properties, stability and plasticity. The key concept of ART is that the stability plasticity can be resolved by a system in which the network includes bottom-up (input-output) competitive learning combined with top-down (output-input) learning. The instability of instar-outstar networks could be solved by reducing the learning rate gradually to zero by freezing the learned categories. Buc, at this point, the net may lose its plasticity or the ability to react to new data. Thus it is difficult to possess both stability and plasticity. ART networks are designed particularly to resolve the stability-plasticity dilemma, that is, they are stable to preserve significant past learning but nevertheless remain adaptable to incorporate new information whenever it appears.

5.8.1. Fundamental architecture of ART-

Three groups of neurons reused to build an ART network. These include:

1. Input processing neurons (F1 layer).
2. Clustering units (F2 layer).
3. Control mechanism (controls degree of similarity of patterns placed on the same duster

The processing neuron (F_1) layer consists of two portions: Input portion and interface portion input portion may perform some processing based on the inputs it receives. This is especially performed in the case of ART 2 compared to ART 1.

The interface portion of the F_1 layer combines the input from input portion of F_1 and F_2 layers for comparing the similarity of the input signal with the weight vector for the interface portion 25 F (b).

There exist two sets of weighted interconnections for controlling the degree of similarity between the units in the interface portion and the cluster layer. The bottom-up weights are used for the connection from F_1(b) layer to F_2 tayer and are represented by $\delta_{ij}(f$ th F_1 unit to jhF_2 unit). The iop-down weights are used for the connection from F_2 layer to F_1(b) layer and are repiesented by $t_{\bar{\mu}}(j$ th F_2 unit to i th F_1 anic). The competitive Jayer in this cose is the cluster layct and the duster unit wich largest net input is the victim to learn the input pattern, and the activations of all other F_2 urnis are mate zero The interface units combinc the data from input and cluster layer units. On the basis of the similarity between the top-down weight vector and input vector, the cluster unit may be allowed to learn the input pattern. This decision is done by-esset mechanism unit on the basis of the signals receives from interface portion and input portion of the F_1 layer. When duster unit is not allowed to learn, it is inhibited and a new cluster unit is selected as the victim.

5.8.2. Fundamental algorithm of ART-

Step 0: initialize the necessary parameters.

Step 1: Perform Steps $2-9$ when stopping condition is false.

Step 2: Perform Steps $3-8$ for each input vector.

Step 3: F_1 layer processing is done.

Step 4: Perform Steps $5-7$ when teset condition is true.

Step 5: Find the victim unit to learn the current input pattern. The victim unit is going to be the F_2 unit (that is nor inhibited) with the largest input.

Step 6: F_1 (b) units combine their inputs from F_1 (a) and F_2.

Step7: Test for reset condition. Step If reset is true, then the current victim unit is rejected (inhibited); go to Step 4. If reser is false, then che carrent victim unit is accepted for learning; go to next step (Step 8).

Step 8: Weight updation is performed.

Step 9: Test for stopping condition.

Adaptive resonance theory 1 (ART 1) network is designed for binary input vectors. As discussed generally, the ART 1 net consists of two fields of units-input unit (F_1 unit) and output unit (F_2 unit)-aiong with the reser control unit for controlling the degree of similarity of patterns placed on the same cluster unit. There exist two sets

of weighted interconnection patch between F_1 and F_2 layers. The supplemental unic present in the net provides the efficient neural control of the leatning process. Carpenter and Grossberg have designed ART 1 network as a real-time system. In ART 1 network, ic is not necessary to present an input pattern in a particular order; it can be presented in any order. ART 1 network can be practically implemented by analog circuits governing the differential equations, i. Q. the bottom-up and top down weights are controlled by differential equations.)ART 1 network runs throughout autonomously. It does nor require any external control signals and can run stably with infinite patterns of input data.

ART 1 network is trained using fast learning method, in which the weights reach equilibrium during each learning trial. During this resonance phase, the activations of F units do not change; hence the equilibrium weights can be determined exactly The ART 1 network performs well with perfect binary input patterns, but is sensitive to noise in the input dara. Hence care should be taken to handle the noise.

5.8.3. Fundamental architecture of ART1-

The ART 1 network is made up of two units:

1 Computational units.

2 Supplemental units.

In this section we will discuss in detail about these two units.

Computational units

The computational unit for ART 1 consists of the following:

1 Input units (F_1 unit – both input portion and interface portion).

2 Cluster units (F_2 unit – outpuc unit),

Reset control unit (controls degree of similarity of patterns placed on same cluster). The basic architecture of ART I (computational unit) is shown in Figure 5-22. Here each unit present in the input portion of F_1 layer ($\mathrm{i,e.,F_1(a)}$ layer unic) is connected to the respective unic in the interface portion of E layer (i.e., $F_1(\mathrm{b})$ layer unit). Reset control unit has connections from each f$F_1(\mathrm{a})$ and $F_1(\mathrm{b})$ units. Also, each unit in $F_1(\mathrm{b})$ layer is connected through two weighted interconnection pachs to each unic in F_2 layer and the reser control unit is connected to every F_2 unit.The X_i unit of F_1(b) layer is connected to Y_j unit of F_2 layer through bottom-up weight (b_{ij}) and the Y_j unit of F_2 is connected to X_i unit of F_1 through top-down weights

(t_{ji}). Thus ART 1 includes a bottom-up competitive learning system combined with a top-down outstar learning system. In Figure $5-22$ for simplicity only the weighted interconnections b_{ij} and t_{ji} are shown, the other units' weighted interconnections are in a similar way. The cluster layer (F_2 layer) unit is a competitive layer, where only the uninhibited node with the largest net input has nonzero activation.

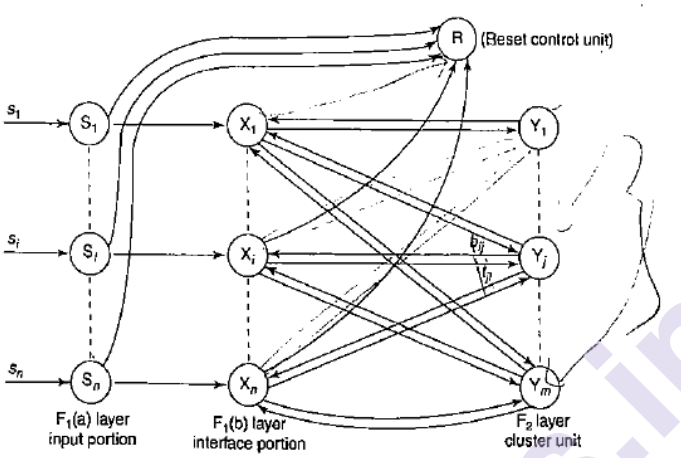


Figure 5.22 Basic architecture of ART 1

5.8.4. Training Algorithm of ART1-

Step 0: initialize the parameters:

$$\text{and } 0 < \rho \leq 1$$

Initialize the weights:

$$0 < b_{ij}(0) < \frac{\alpha}{\alpha - 1 + n} \text{ and } t_{ji}(0) = 1$$

Step 1: Perform Steps 2-13 when stopping condition is false.

Step 2: Perform Steps $3-12$ for each of the training input.

Step 3: Set activations of all F_2 units to zero. Set the activations of $F_1(2)$ units to input vectors.

Step 4: Calculate the norm of Σ

$$\| s \| = \sum_j s_i$$

Step 5: Send input signal from F_1 (a) layer to F_1 (b) byer:

$$x_1 = s_i$$

Step 6: for each F_2 pode thar is not inhibited, the following rule should hold: If $y; \neq -1$, then $\bar{y}_j = \sum b_{ij x_i}$

Step 7: Perform Steps $8 - 11$ when reset is true.

Step 8 : Find J for $y_j \geq y_j$ for all nodes j. If $y_j = -1$, then all the nodes are inhibited and note that this pattern cannot be clustered.

Step 9: Recalculate activation X of F_1(b) :

$$x_i = s_i t_j$$

Step 10: Calculate the norm of vector x.

$$\| x \| = \sum_i x_i$$

Step 11: Test for reset condition. If $\| x \| / \| s \| < \rho$, then inhibit node $J, y_j = -1$. Go back to step 7 again. Else if $\| x \| / \| s \| \geq \rho$, then procced to the next step (Step 12).

Step 12: Perform weight updation for node J. (fast learning):

$$b_{ij}(\text{ new }) = \frac{\alpha x_i}{\alpha - 1 + \| x \|}$$

$$\sqrt{t_j i}\ (\text{new})\ = x_i\}$$

Step 13: Test for stopping condition. The following may be the stopping conditions:

a. No change in weights.

b. No reset of units.

c. Maximum number of epochs reached.

5.8.5. Adaptive Resonance Theory 2 (ART2):

Adaptive resonance theory 2 (ART 2) is for continuous-valued input vectors. In ART 2 network complexity is higher than ART 1 network because much processing is needed in F 1 layer. ART 2 network was developed by Carpenter and Grossberg

in 1987. ART 2 network was designed to self-organize recognition categories for analog as well as binary input sequences. The major difference between ART l and ART 2 networks is the input layer. On the basis of the stability criterion for analog inputs, a three-layer feedback system in the input layer of ART 2 network is required: A bottom layer where the input patterns are read in, a top layer where inputs coming from the output layer are read in and a middle layer where the top and bottom patterns are combined together to form a marched pattern which is then fed back to the top and bottom input layers. The complexity in the F1 layer is essential because continuous-valued input vectors may be arbitrarily dose together. The F1 layer consists of normalization and noise suppression parameter, in addition to comparison of the bottom-up and top-down signals, needed for the reset mechanism.

The continuous-valued inputs presented to the ART 2 network may be of two forms. The first form

is a "noisy binary" signal form, where the information about patterns is delivered primarily based on the

components which are "on" or "off," rather than the differences existing in the magnitude of the components chat are positive. In this case, fast learning mode is best adopted. The second form of patterns are those, in which the range of values of the components carries significant information and the weight vector for a cluster is found to be interpreted as exemplar for· the patterns placed-on chat unit. In this type of pattern, slow learning mode is best adopted. The second form of data is "truly continuous."

5.8.6. Fundamental architecture of ART2-

A typical architecture of ART 2 network is shown in Figure $5-25$. From the figure, we can notice that F_1 layer consists of six types of units - W, X, U, V, P, Q- and there are " n " units of each type. In Figure $5-25$, only one of these units is shown. The supplemental parc of the connection is shown in Figure $5-26$.

The supplemental unit " N′′ between units W and X receives signals from all " W" units, computes the no run of vector w and sends this signal to each of the X units. This signal is inhibitory signal. Each of this $(X_1, \ldots, X_i, \ldots, X_n)$ also receives excicatory signal from the corresponding W unit. In a similar way, there exists supplemental units between U and V, and P and Q, performing the same operation as done between W and X. Each X unit and Q unit is connecred to V unit. The connections between P_j of the F_1 layer and Y_j of the F_2 layer show the weighted

interconnections, which multiplies the signals transmitted over those pachs. The winning F_2 unics' activation is $d(0 < d < 1)$. There exists normalization between W and X, V and U_1 and P and Q. The noimalization is performed approximately to unit length.

The operations performed in F_2 layer are same for both ART 1 and ART 2. The units in F_2 layer compete with each other in a winner-take-all policy to learn each input pattern. The testing of reset condition differs for ART 1 and ART 2 networks. Thus, in ART 2 network, some processing of the input vector is necessary because the magnitudes of the real valued input vectors may vary more than for the binary input vectors.

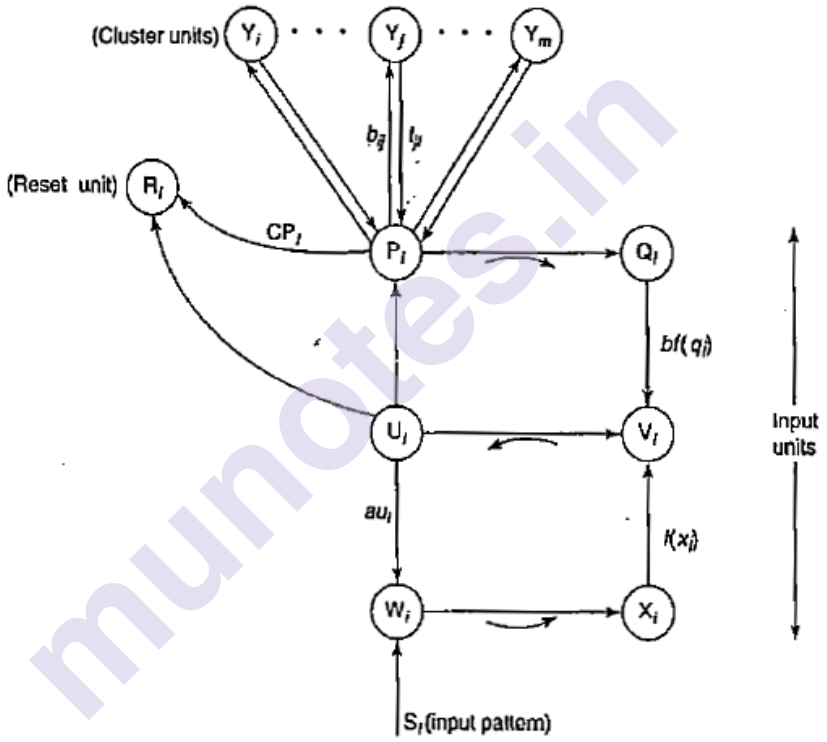


Figure 5.25. Architecture of ART2 network

5.8.7.Training Algorithm of ART2:

Step 0: Initialize the following parameters: $a, b, c, d, e, \alpha, \rho, \theta$. Also, specify the number of epochs of training (nep) and number of learning iterations (nit).

Step 1: Perform Steps 2-12 (nep) times.

Step 2: Perform Steps $3 - 11$ for each input vector s.

Step 3: Update F_1 unit activations:

$$u_i = 0;\ w_i^2 = s_i;\ P_i = 0;\ q_i = 0;\ v_i = f(x_i)$$
$$x_i = \frac{s_i}{e + \| s \|}$$

Update F_1 unit activations again:

$$u_j = \frac{v_i}{e + \| v \|};\ w_i = s_i + au_i;$$
$$P_i = u_i;\ x_i = \frac{w_i}{e + \| w \|'};$$
$$q_i = \frac{p_i}{e + \| p \|};\ v_i = f(x_i) + bf(q_i)$$

In ART 2 networks, norms are calculated as the square root of the sum of the squares of the respective values.

Step 4: Calculate signals to F_2 units:

$$y_j = \sum_{j=1}^{n} b_{ij} p_i$$

Step 5: Perform Steps 6 and 7 when reset is true.

Step 6: Find F_2 unit Y_j wich largest signal J is defined such that $y_j \geq y_j, j = 1$ (o m).

Step 7: Check for reser:

$$u_i = \frac{v_i}{c + \| v \|};\ P_i = u_i + dt_j;\ r_i = \frac{w_i + cP_i}{e + \| u \| \div c \| p \|}$$

If $\| r \| < (\rho - e)$, then $y_J = -1$ (inhibit J). Reser is true; perform Step 5 .

If $\| r \| \geq (\rho - e)$, then

$$w_i = s_i + au_i;\ x_i = \frac{w_i}{e + \| w \|}:$$
$$q_i = \frac{p_i}{e + \| p \|};\ v_i = f(x_i) + bf(q_i)$$

Reset is false. Proceed to Step 8.
Step 8: Perform Steps 9-l 1 for specified number of learning interactions.

Step 9: Update the weights for winning unit J:

$$t_{ii} = \alpha d u_i + \{[1 + \alpha d(d-1)\}\}t_j$$
$$b_{ij} = \alpha d u_i + \{[1 + \alpha d(d-1)]\}b_{ij}$$

Step 10: Update F_ acrivations:

$$u_i = \frac{v_i}{c + \| v \|}: \quad w_i = s_i + a u_i;$$
$$P_i = u_i + d t_{ji}; \quad x_i = \frac{w_i}{e + \| w \|};$$
$$q_i = \frac{P_i}{e + \| p \|}; \quad v_i = f(x_i) + b f(q_i)$$

Step 11: Check for the stopping condition of weight updating.

Step 12: Check for the stopping condition for number of epochs.

Review Questions:

1. Explain the concept of Unsupervised Learning.
2. Write a short note on Fixed Weight Competitive Nets
3. Explain Algorithm of Mexican Hat Net
4. What is mean by Hamming Network
5. Explain the Architecture of Hamming Network
6. Write a short note on Kohonen Self-Organizing Feature Maps
7. Write a short note on Learning Vector Quantization (LVQ)
8. Explain Counter propagation Networks
9. What is mean by Adaptive Resonance Theory Network

Reference

1. “Principles of Soft Computing”, by S.N. Sivanandam and S.N. Deepa, 2019, Wiley Publication, Chapter 2 and 3

2. http://www.sci.brooklyn.cuny.edu/ (Artificial Neural Networks, Stephen Lucci PhD)

3. Related documents, diagrams from blogs, e-resources from RC Chakraborty lecture notes and tutorialspoint.com.

❖❖❖

6

SPECIAL NETWORKS

Unit Structure

6.1. Simulated Annealing Network

The concept of simulated annealing has it origin in the physical annealing process performed over metals and other substances. In metallurgical annealing, a metal body is heated almost to its melting point and then cooled back slowly to room temperature. This process eventually makes the metal's global energy function reach an absolute minimum value. If the metal's temperature is reduced quickly, the energy of the metallic lattice will be higher than this minimum value because of the existence of frozen lattice dislocations that would otherwise disappear due to thermal agitation. Analogous to the physical annealing behaviour, simulated annealing can make a system change its state to a higher energy state having a chance to jump from local minima or global maxima. There exists a *cooling procedure* in the simulated annealing process such that the system has a higher

probability of changing to an increasing energy state in the beginning phase of convergence. Then, as time goes by, the system becomes stable and always moves in the direction of decreasing energy state as in the case of normal minimization produce.

With simulated annealing, a system changes its state from the original state SA^{old} to a new stare SA^{new} with a probability P given by

$$P = \frac{1}{1 + \exp(-\Delta E/T)}$$

where $\Delta E = E^{old} - E^{new}$ (energy change = difference in new energy and old energy) and T is the nonnegative parameter (acts like temperature of a physical system). The probability P as a function of change in energy (ΔE) obtained for different values of the remperature T is shown in Figure $6-1$. From Figure $6-1$, it can be noticed that the probability when $\Delta E > 0$ is always higher than she probability when $\Delta E < 0$ for any remperature.

An optimization problem seeks to find some configuration of parameters $\dot{X} = (X_1, \ldots, X_n)$, hat minimizes some function $f(X)$ called cost function. In an arcificial neural network, configuration parameters are associated with the set of weights and the cost function is associated with the error function.

The simulated annealing concept is used in statistical mechanics and is called Metropolis algorithm. *As* discussed earlier, this algorithm is based on a material that anneals into a solid as temperature is slowly decreased. To understand this, consider the slope of a hill having local valleys. A stone is moving down the hill. Here, the local valleys are local minima, and the bottom of the hill is going to be the global or universal minimum. It is possible that the stone may stop at a local minimum and never reaches the global minimum. In neural nets, this would correspond to a set of weights that correspond to that of local minimum, but this is nm the desired solution. Hence, to overcome this kind of situation, simulated annealing perturbs the stone such that if it is trapped in a local minimum, it escapes from it and continues falling till it reaches its global minimum (optimal solution). At that point, further perturbations cannot move the stone to a lower position.

Figure 6-2 shows the simulated annealing between a stone and a hill.

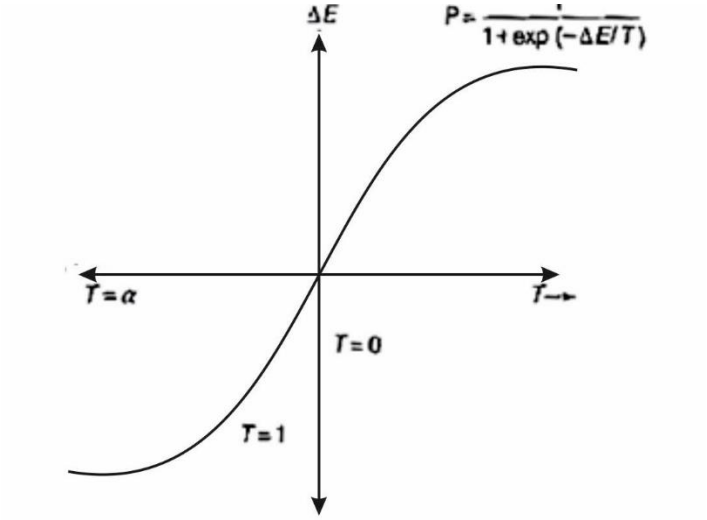


Figure 6.1 Probability "P" as a function in energy(AE) for different values of temperature T

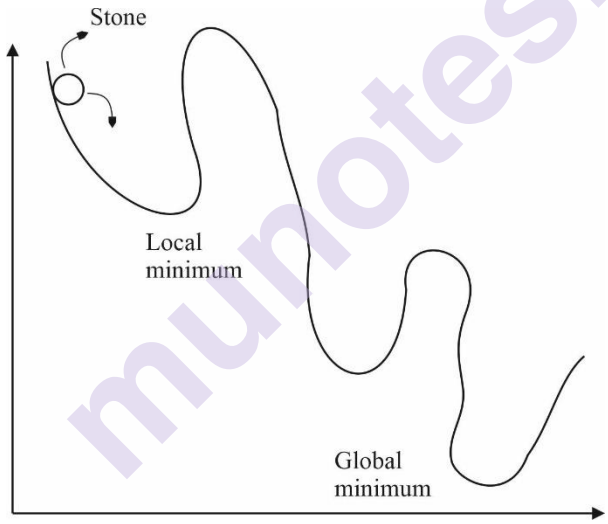


Figure 6.2 Simulated annealing stone and hill

components required for annealing algorithm are the following

A basic system configuration: The possible solution of a problem over which we search for a best (optimal) answer. (In a neural ner, this is optimum steady-state weight.)

The move set: A ser of allowable moves thar permit us to escape from local minima and reach all possible configurations.

A cost function associated with the error function.

4 A cooling schedule: Starting of the cost function and rules to determine when it should be lowered and by how much, and when annealing should be terminated.
Simulated annealing networks can be used to make a network converge to its global minimum.

6.2. Boltzmann Machine

The early optimization technique used in artificial neural networks is based on the Boltzmann machine. When the simulated annealing process is applied *w* the discrete Hopfield network, it becomes a Boltzmann machine. The network is configured as the vector of the states of the units, and the stares of the units are binary valued with probabilities state transition. The Boltzmann machine described in this section has fixed weights *wij.* On applying the Boltzmann machine to a constrained optimization problem, the weights represent the constraints of the problem and the quantity to be optimized. The discussion here is based on the fact of maximization of a consensus function (CF).

The Boltzmann machine consists of a set of units (Xi, and Xj) and a set of bi-directional connections between pairs of units. This machine can be used as an associative memory. If the units *Xi;* and Xj are connected, then $w_{ij} \neq 0$. There exists symmetry in the weighted interconnections based on the directional nature. It can be represented as $w_{ij}=w_{ji}$. There also may exist a self-connection for a unit *(w_{ij}).* For unit *Xi,* its State x_i; may be either 1 or 0. The objective of the neural net is to maximize the CF given by

$$\mathrm{CF} = \sum_{i} \sum_{j \leq i} w_{ij} x_i x_j$$

The maximum of the CF can be obtained by letting each unit attempt to change its state (alter between " 1^n and " 0 " or " 0 " and "1"). The change of sate can be done either in parallel or sequencial manner. However, in this case ali the description is based on sequential manner. The consensus change when unit X_i changes its state is given by

$$\Delta CF(i) = (1 - 2x_i)\left(w_{ij} + \sum_{j \neq i} w_{ij} x_i\right)$$

where x_i is the current srate of unit X_i. The variation in coefficient $(1 - 2x_i)$ is given by

$$(1 - 2x_i) = \begin{cases} +1, & X_i \text{ is currently off} \\ -1, & X_i \text{ is currently on} \end{cases}$$

f unit X_i were to change its activations, then the resulting change in the CF can be obtained from the information that is local to unit X_i. Generally, X_i does not change ts stare, but if the states are changed, then this increases the consersus of the net. The probability of the network that accepts a change in the state for unit X_i is given by

$$\mathrm{AF}(i, T) = \frac{1}{1 + \exp\left[-\Delta \mathrm{CF}(i)/T\right]}$$

where T (temperature) is the controlling parameter and it will gradually decrease as the CF reaches the maximum value. Low values of T are acceptable because hey increase rhe net consensus since the net accepts a change in state. To help the net not to stick with the local maximum, probabilistic functions are used widely.

6.2.1. Architecture of Boltzmann Machine

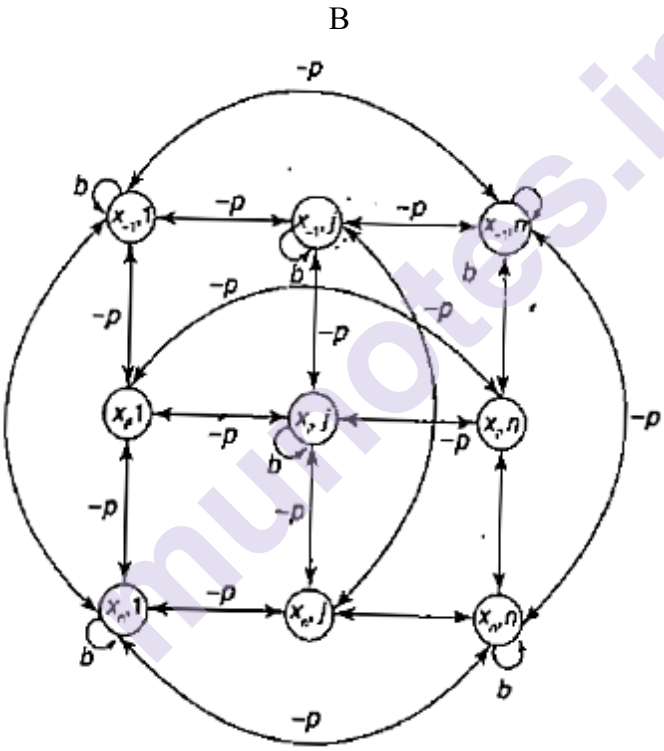


Figure 6.3 Architecture of Boltzmann machine

6.2.2. Testing Algorithm of Boltzmann Machine

Step 0: Initialize the weights representing the constraints of the problem. Also nitialize control parameter T and activate the units.

Step 1: When stopping condition is false, perform Steps 2-8.

Step 2: Perform Steps $3 - 6n^2$ rimes. (This forms an epoch.)

Step 3: Integers I and J are chosen random values berween 1 and n. (Unit $U_{1,j}$ is the current victim to change its state.)

Step 4: Calculate the change in consensus:

$$\Delta CF = \left(1 - 2X_{l,j}\right)\left[w(I,J:I,J) + \sum_{i,j\neq} \sum_{1,j} v(i,j:I,J)X_{i,J}\right]$$

Step 5: Calculate the probability of acceptance of the change in state:

$$\mathrm{AF}(T) = 1/1 + \exp\left[-(\Delta \mathrm{CF}/T)\right]$$

Step 6: Decide whether to accept the change or not. Les R be a random number between

0 and 1. If $R <$ AF, accept the change:

$X_{L,J} = 1 - X_{l,j}$ (This changes the scate $\mathrm{U}_{L,J}$.) If $R \geq AF$, reject the change.

Step 7: Reduce the control parameter T. T (new) $= 0.95T$ (old)

Step 8: Test for stopping condition, which is:

If the temperature reaches a specified value or if there is no change of state for specified number of epochs then stop, else continue.

6.3. Gaussian Machine

Gaussian machine is one which includes Boitzmann machine, Hopfield net and other neural networks. The Gaussian machine is based on the following three parameters:

(a) a slope parameter of sigmoidal function α,

(b) a time step Δt, (c) temperacure T. The steps involved in the operation of the Gaussian net are the following:

Step 1: Compute the net input to unit X_i :

$$\mathrm{nec}_i = \sum_{j=1}^{N} w_{ij}v_j + \theta_i + \epsilon$$

where θ; is the rhreshold and $\in$ the random noise which depends on temperature T.

Step 2: Change the activity level of unit X_i :

$$\frac{\Delta x_i}{\Delta t} = -\frac{x_i}{t} + \mathrm{net}_i$$

Step 3: Apply the activation function:

$$v_i = f(x_i) = 0.5[1 + \tanh(x_i)]$$

The binary step function corresponds to $\alpha = \infty$ (infinity).

The Gaussian machine with $T = 0$ corresponds the Hopfield net. The Bolamann machine can be obtained by setting $\Delta t = \mathrm{r} = 1$ to get

$$\Delta x_i = -x_i + \text{ net }_i$$

$$\text{or } x_i \text{ (new)} = \text{ net }_i = \sum_{j=1}^{N} iv_{ij}v_j + \theta_i + \epsilon$$

The approximate Boltzmann acceptance function is obtained by integrating the Gaussian noise distribution

$$\int_0^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{(x - x_i^2)}{2\sigma^2} dx \approx \text{AF } (r, T) = \frac{1}{1 + \exp(-x_i lT)}$$

where $x_i = \Delta CF(i)$. The noise which is found to obey a logistic rather than a Gaussian distribution produces a Gaussian machine that is identical to Boltzmann machine having Metropolis acceptance function, i.e., the output set to 1 with probability,

$$\text{AF}(i, \text{T}) = \frac{1}{1 + \exp(-x_i/T)}$$

$$\Delta x_i = -x_i + \text{ net }_i$$

6.4. Cauchy Machine

Cauchy machine can be called fast simulated annealing, and it is based on including more noise to the net input for increasing the likelihood of a unit escaping from a neighbourhood of local minimum. Larger changes in the system's configuration can be obtained due to the unbounded variance of the Cauchy distribution. Noise involved in Cauchy distribution is called "coloured noise" and the noise involved in the Gaussian distribution is called "white noise." By setting $\Delta t = \tau = 1$, the Cauchy machine can be extended into the Gaussian machine, to obtain

$$\Delta x_i = -x_i + \text{net}_i$$

$$\text{or } x_i \text{ (new)} = \text{ net }_i = \sum_{j=1}^{N} w_{ij}v_j + \theta_i + \epsilon$$

The Cauchy acceptance function can be obtained by integrating the Cauchy noise distribution:

$$\int_0^{\infty} \frac{1}{\pi} \frac{Tdx}{T^2 + (x - x_i)^2} = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x_i}{T}\right) = \text{AF } (i, T)$$

where $x_i = \Delta CF(t)$. The cooling schedule and temperature have to be considered in both Cauchy and Gaussian machines.

6.5. Probabilistic Neural Net

The probabilistic neural net *is* based on the idea of conventional probability theory, such as Bayesian classification and other estimators for probability density functions, to construct a neural net for classification. This net instantly approximates optimal boundaries between categories. It assumes that the training data are original representative samples. The probabilistic neural net consists of two hidden layers as shown in Figure 6-4. The first hidden layer contains a dedicated node for each training pattern amd the second hidden layer contains a dedicated node for each class. The two hidden layers are connected on a class-by-class basis, that is, the several examples of the class in the first hidden layer are connected only to a single machine unit in the second hidden layer.

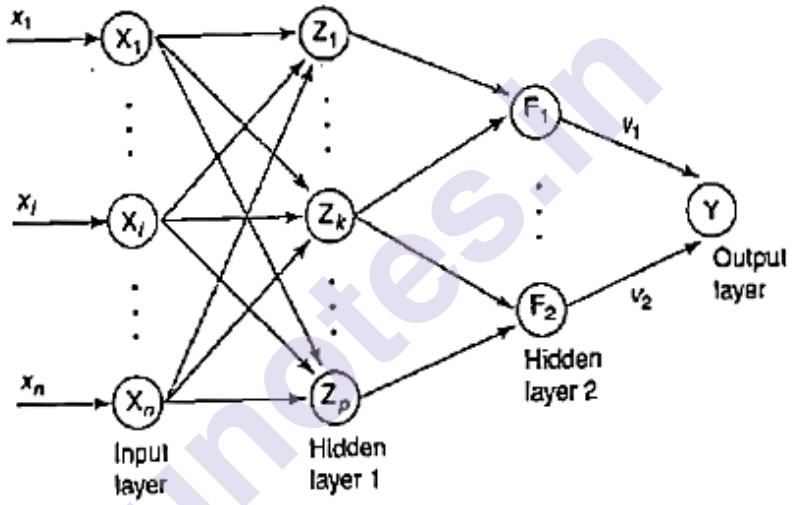


Figure 6.4. Probabilistic neural network

The algorithm for the construction of the net is as follows:

Step 0: For each training input pattern $x(p), p = 1$ to P, perform Steps 1 and 2.

Step 1: Create pattern unit z_k (hidden-layer-l unit). Weight vecror for unit z_k is given by

$$w_k = x(p)$$

Unit z_k is either z -class- 1 unit or z -class- 2 unic.

Step 2: Connect the hidden-layer- 1 unit to the hidden-layer- 2 unic.

If $x(p)$ belongs to class 1, then connect the hidden layer unic z_k ro the hidden layer unit F_1.

Otherwise, connect pattern hidden layer unit z_k to the hidden layer unit F_2.

6.6. Cascade Correlation Network:

Cascade correlation is a network which builds its own architecture as the training progresses. Figure 6-5 shows the cascade correlation architecture. The network begins with some inputs and one or more output nodes, but it has no hidden nodes. Each and every input is connected to every output node. There may be linear units or some nonlinear activation function such as bipolar sigmoidal activation function in the output nodes. During training process, new hidden nodes are added to the network one by one. For each new hidden node, the correlation magnitude between the new node's output and the residual error signal is maximized. The connection is made to each node from each of the network's original inputs and also from every pre-existing hidden node. During the time when the node is being added to the network, the input weights of the hidden nodes are-frozen, and only the output connections are trained repeatedly. Each new node thus adds a new one-node layer to the network.

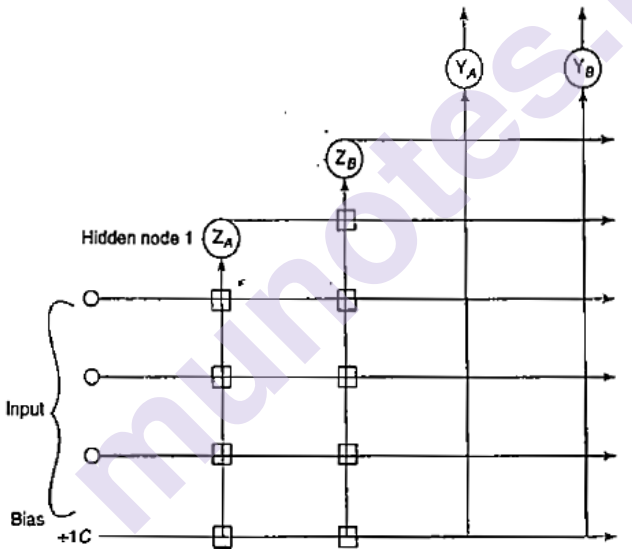


Figure 6.5. Cascade architecture after two hidden nodes have been added

In Figure 6-5, the vertical lines sum all incoming activations. The rectangular boxed connections are frozen and "0" connections are trained continuously. In the beginning of the training, there are no hidden nodes, and the network is trained over the complete training set. Since there is no hidden node, a simple learning rule, Widrow-Hofflearning rule, is used for training. After a certain number of training cycles, when there is no significant error reduction and the final error obtained is unsatisfactory, we try to reduce the residual errors further by adding a new hidden node. For performing this task, we begin with a candidate node that receives trainable input connections from the network's external inputs and from all pre-

existing hidden nodes. The output of this candidate node is not yet connected to the active network. After this, we run several numbers of epochs for the training set. We adjust the candidate node's input weights after each -epoch to maximize C which is defined as

$$C = \sum_i \mid \sum_j (v_j - \vec{v})(E_{j,i} - \bar{E}_o)$$

where i is the network output at which error is measured, j the raining partern, v the candidate node's output value, E_o the residual output error at node $o, \bar{v}$ the value of y averaged over all parterns, $\overline{E_o}$ the value of E_o

averaged over all patterns. The value " C'' ' measures the correlation berween the candidate node's oucput value and the calculated residual output error. For maximizing C, the gradient $\partial d\ \partial w_i$ is obrained as

$$\frac{\partial c}{\partial w_i} = \sum_{j,i} \sigma_i (E_{j,i} - \bar{E}_i) d_j I_m$$

where σ_i is the sign of the correlation between the candidatc's value and output i; d_j the derivative for pattern j of the candidate node's activation function with respecc to sum of its inputs; $I_{m,j}$ the input the candidate node receives from node m for pattern j. When gradient $\partial d\ \partial w_i$ is calculated, perform gradient ascent to maximize C. As we are training only a single layer of weights, simple delta learning rule can be applied. When C stops improving, again a new candidate can be brought in as a node in the active network and its input weights are frozen. Once again, all the output weights are trained by the delta learning rule as done previously, and the whole cycle repeats itself until the error becomes acceptably small.

6.7. Cognitron Network:

The synaptic strength from cell X to cell Y is reinforced if and only if the following two conditions are true:

l. Cell X- presynaptic cell fires.

2. None of the postsynaptic cells present near cell Y fire stronger than Y.

The model developed *by* Fukushima was called cognitron as a successor to the perceptron which can perform cognizance of symbols from any alphabet after training. Figure 6-6 shows the connection between presynaptic cell and postsynaptic cell.

The cognitron network is a self-organizing multilayer neural network. Its nodes receive input from the defined areas of the previous layer and also from units within its own area. The input and output neural elements can rake the form of positive analog values, which are proportional to the pulse density of firing biological neurons. The cells in the cognitron model use a mechanism of shunting inhibition, i.e., a cell is bound in terms of a maximum and minimum activities and is driven toward these extremities. The area from which the cell receives input is called connectable area. The area formed by the inhibitory cluster is called the vicinity area. Figure 6. 7 shows the model of a cognitron. Since the connectable areas for cells in the same vicinity are defined to overlap, but are not exactly the same, there will be-a slight difference appearing between the cells which is reinforced so that the gap becomes more apparent. Like this, each cell is allowed to develop its own characteristics.

Cognitron network can be used in neurophysiology and psychology. Since this network closely resembles the natural characteristics of a biological neuron, this is best suited for various kinds of visual and auditory information processing systems. However, a major drawback of cognitron net is that it cannot deal with the problems of orientation or distortion. To overcome this drawback, an improved version called neocognitron was developed.

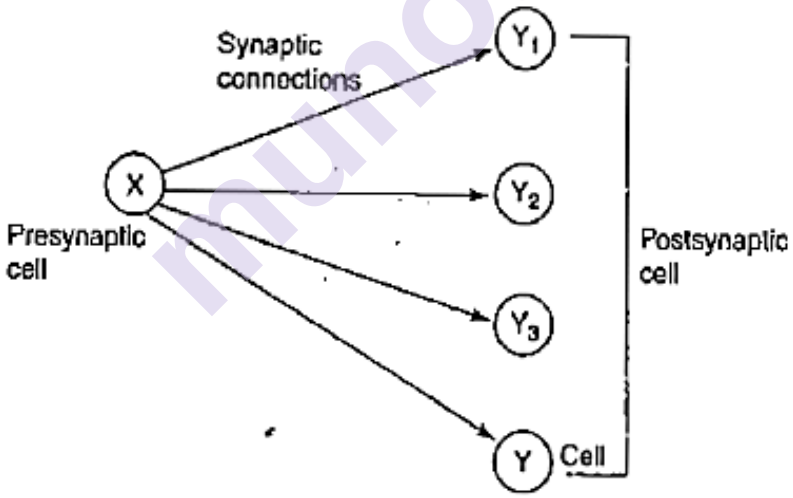


Figure 6.6 Connection between presynaptic cell and postsynaptic cell

Figure 6.7 Model of a cognitron network

6.8. Neocognitron Network

Neocognitron is a multilayer feed-forward network model for visual pattern recognition. It is a hierarchical net comprising many layers and there is a localized pattern of connectivity between the layers. It is an extension of cognitron network. Neocognitron net can be used for recognizing hand-written characters. A neocognitron model is shown in Figure 6·8.

The algorithm used in cognitron and neocognitron is same, except that neocognicron model can recognize patterns that are position-shifted or shape-distorted. The cells used in neocognitron are of two types:

1. *S·-cell:* Cells that are trained suitably to. respond to only certain features in the previous layer.
2. *C-cell·* A C-cell displaces the result of an S-cell in space, i.e., son of "spreads" the features recognized by the S-cell.

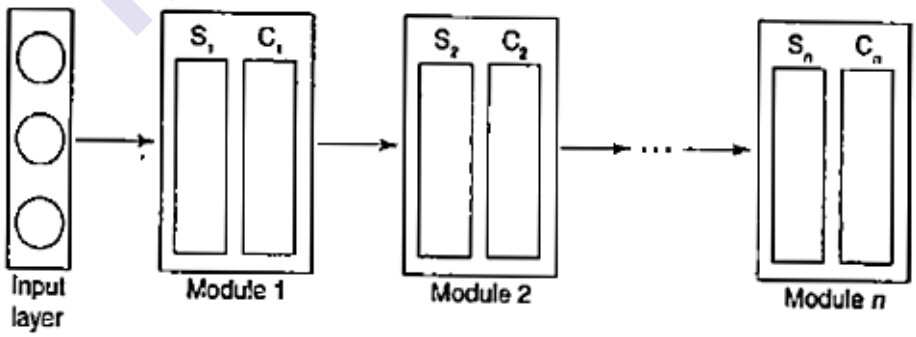


Figure 6.8 Neocognitron models

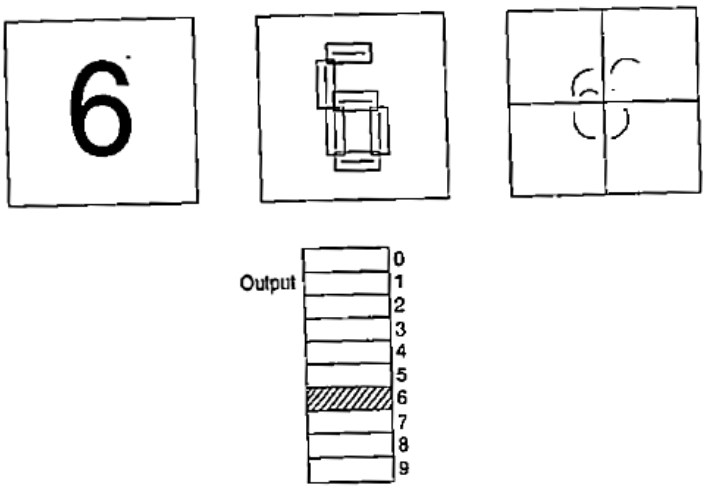


Figure 6.9 Sprcading effect in neocognitron

Neocognitron net consists of many modules with the layered arrangement of S-cells and C-cells. The S-cells receive the input from the previous layer, while C-cells receive the input from the S-layer. During training, only the inputs to the S-layer are modif1ed. The S-layer helps in the detection of spccif1c features and their complexities. The feature recognized in the S_1 layer may be a horizontal bar or a vertical bar but the feature in the Sn layer may be more complex. Each unit in the C-layer corresponds to one relative position independent feature. For the independent feature, C-node receives the inputs from a subset of S-layer nodes. For instance, if one node in C-layer detects a vertical line and if four nodes in the preceding S-layer detect a vertical line, then these four nodes will give the input to the specific node in C-layer to spatially distribute the extracted features. Modules present near the input layer (lower in hierarchy) will be trained before the modules that are higher in hierarchy, i.e., module 1 will be trained before module 2 and so on.

The users have to fix the "receptive field" of each C-node before training starts because the inputs to C-node cannot be modified. The lower level modules have smaller receptive fields while the higher level modules indicate complex independent features present in the hidden layer. The spreading effect used in neocognitron is shown in Figure 6-9.

6.9. Cellular Neural Network –

cellular neural network (CNN), introduced in 1988, is based on cellular automata, i.e., every cell in the network is connected only to its neighbour cells. Figures 6-10 (A) and (B) show 2 x 2 CNN and 3 x 3 CNN, respectively. The basic unit of a CNN is a cell. In Figures 6-10(A) and (B), C(l, l) and C(2, 1) are called as cells.

Even if the cells are not directly connected with each other, they affect each other indirectly due to propagation effects of the network dynamics. The CNN can be implemented by means of a hardware model. This is achieved by replacing each cell with linear capacitors and resistors, linear and nonlinear controlled sources, and independent sources. An electronic circuit model can be constructed for a CNN. The CNNs are used in a wide variety of applications including image processing, pattern recognition and array computers.

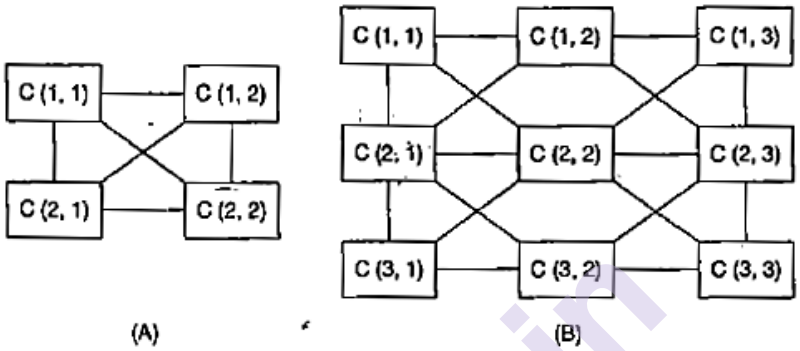


Figure 6.10 (A) A2*2CNN;(B) a 3*3 CNN

6.10. Optical Neural Networks

Optical neural networks interconnect neurons with light beams. Owing to this interconnection, no insulation is required between signal paths and the light rays can pass through each other without interacting. The path of the signal travels in three dimensions. The transmission path density is limited by the spacing of light sources, the divergence effect and the spacing, of detectors. *A$* a result, all signal paths operate simultaneously, and true data rare results are produced. In holograms with high density, the weighted strengths are stored.

These stored weights can be modified during training for producing a fully adaptive system. There are two classes of this optical neural network. They are:

1. electro-optical multipliers;

2. holographic correlators.

6.10.1. Electro-optical multipliers

Electro-optical multipliers, also called electro-optical matrix multipliers, perform matrix multiplication in

parallel. The network speed is limited only by the available electro-optical components; here the computation time is potentially in the nanosecond range. A

model of electro-optical matrix multiplier is shown in Figure 6-11.

Figure 6-11 shows a system which can multiply a nine-element input vector by a 9 X 7 matrix, which

produces a seven-element NET vector. There exists a column of light sources that passes its rays through

a lens; each light illuminates a single row of weight shield. The weight shield is a photographic film where transmittance of each square (as shown in Figure 6-11) is proportional to the weight. There is another lens that focuses the light from each column of the shield m a corresponding photoelectron. The NET is calculated as

$$\mathrm{NET}_k = \sum_i w_{ik} x_i$$

where NET_k is the net output of neuron *k;* w_{ik} the weight from neuron *i* to neuron *k;* x_i the input vector

component *i.* The output of each photo detector represents the dot product between the input vector and a

column of the weight matrix. The output vector set is equal to the produce of the input vector with weight

matrix. Hence, matrix multiplication is performed parallel. The speed is independent of the size of the array.

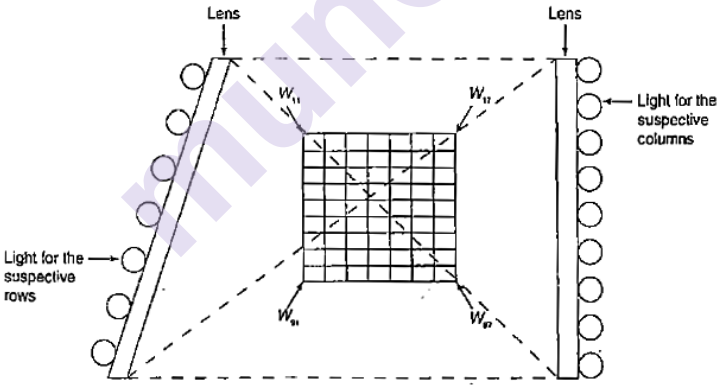


Figure 6.11 Elecrno-optical multiplier

6.10.2. Holographic Correlators

In holographic correlators, the reference images are stored in a thin hologram and are retrieved in a coherently illuminated feedback loop. The input signal, either noisy or incomplete, may be applied to the system and can simultaneously be correlated optically with all the stored reference images. These. correlations can be threshold and are fed back to the input, where the strongest correlation reinforces

the input image. The enhanced image passes around the loop repeatedly, which approaches the stored image more closely on each pass, until the system gets stabilized on the desired image. The best performance of optical correlators is obtained when they are used for image recognition. A generalized optical image recognition system with holograms is shown in Figure 6- 12.

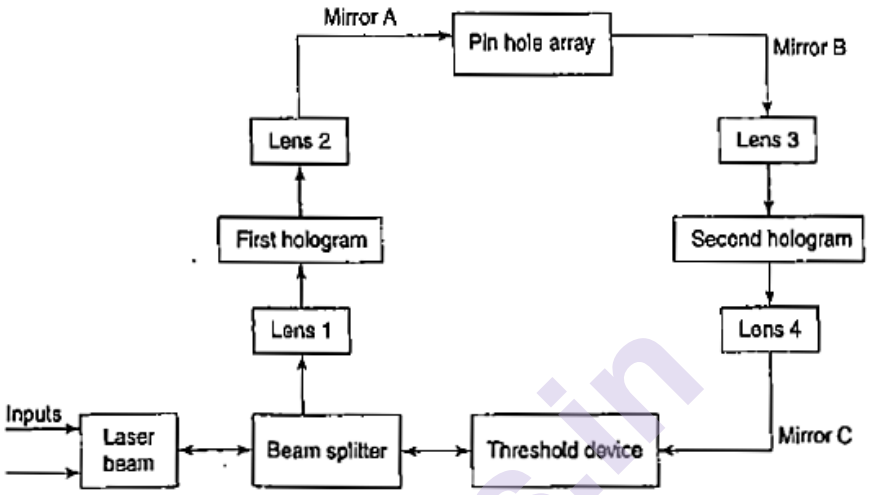


Figure 6.12 Optical image recognition system

The system input is an image from a laser beam. This passes through a beam splitter, which sends it to

the threshold device. The image is reflected, then gets reflected from the threshold device, passes back to the beam splitter, then goes to lens 1, which makes it fall on the first hologram. There are several stored images in first hologram. The image then gets correlated with each stored image. This correlation produces light patterns. The brightness of the patterns varies with the degree of correlation. The projected images from lens 2 and mirror A pass through pinhole array, where they are spatially separated. From this array, light patterns go to mirror B through lens 3 and then are applied to the second hologram. Lens 4 and mirror C then produce superposition of the multiple correlated images o1nto the back side of the threshold device.

The front surface of the threshold device reflects most strongly that pattern which is brightest on its rear surface. Its rear surface has projected on it the set of four correlations of each of the four stored images with the input image. The stored image that is similar to the input image possesses highest correlation. *This* reflected image again passes through the beam splitter and re-enters the loop for further enhancement. The system gets converged on the stored patterns most like the input pattern.

6.11. SPIKING NEURAL NETWORKS(SNN)

As it is well known that the biological nervous system has inspired the development of the artificial neural network models. On looking into the depth of working of biological neurons, it is noted that the working of these neurons and their computations are performed in temporal domain and the neuron firing depends on the timing between the spikes stimulated in the neurons of the brain. These fundamental biological understandings of the neuron operation lead the pathway to the development of spiking neural networks (SNN). SNNs fall under the category of third-generation neural networks and this is more closely related to the biological counterparts compared to the first- and second-generation neural networks. These developed spiking neural networks use transient pulses for performing the computations and require communications within the layers of the network designed. There exist different spiking neural models and their classification is based on their level of abstraction.

6.11.1. Architecture of SNN Model

Neurons in central nervous system communicate using short-duration electrical impulses called spikes or action potentials in which their amplitude is constant in the same structure of neurons. SNNs offer a biological plausible fast third-generation neural connectionist model. They derive their strength and interest from an accurate modelling of synaptic interactions between neurons, taking into account the time of spike emission. SNNs overcome the computational power of neural networks made of threshold or sigmoidal units. Based on dynamic event-driven processing, they open up new horizons for developing models with an exponential capacity of memorizing and a strong ability to fast adaptation.

Moreover, SNNs add a new dimension, the temporal axis, to the representation capacity and the processing abilities of neural networks. There are many different models one could use to model both the individual spiking neurons and also the nonlinear dynamics of the system. Neurons communicate with spikes, also known as action potentials. Since all spikes art identical (1-2 ms of duration and 100 mV of amplitude), the information is encoded by the liming of the spikes and not the spikes themselves. Basically, a neuron is divided into three parts: the dendrites, the soma and the axon. Generally speaking, the dendrites receive the input signals from the previous neurons. The received input signals are processed in the soma and the output signals are transmitted at the axon. The synapse is between every two neurons; if a neuron *j* sends a signal across the synapse to neuron i, the neuron that sends the signal is called *pre-synaptic neuron* and the neuron that receives the signal is called *post-synaptic neuron.* Every neuron is surrounded by positive and

negative ions. In the inner surface of the membrane there is an excess of negative charges and on the outer surface there is an excess of positive charges. Those charges create the membrane potential.

Each spiking neuron is characterized by a membrane potential. When the membrane potential reaches a critical value called threshold it emits an action potential, also known as a *spike* (Figure 7-1). A neuron is said to fire when its membrane potential reaches a specific threshold. When it fires, it sends a spike towards all other connected neurons. Its membrane potentials then reset and the neuron cannot fire for a short period of time, this time period *refractory period.* The output of a spiking neuron is therefore binary (spike or not) but it can be converted to continuous signal over time. Hence the activity of a neuron over a short period of lime is converted into a mean firing rate. The spikes are identical to each other and their form does not change as the signal moves from a pre-synaptic to a post-synaptic neuron. The firing time of a neuron is called *spike train.*

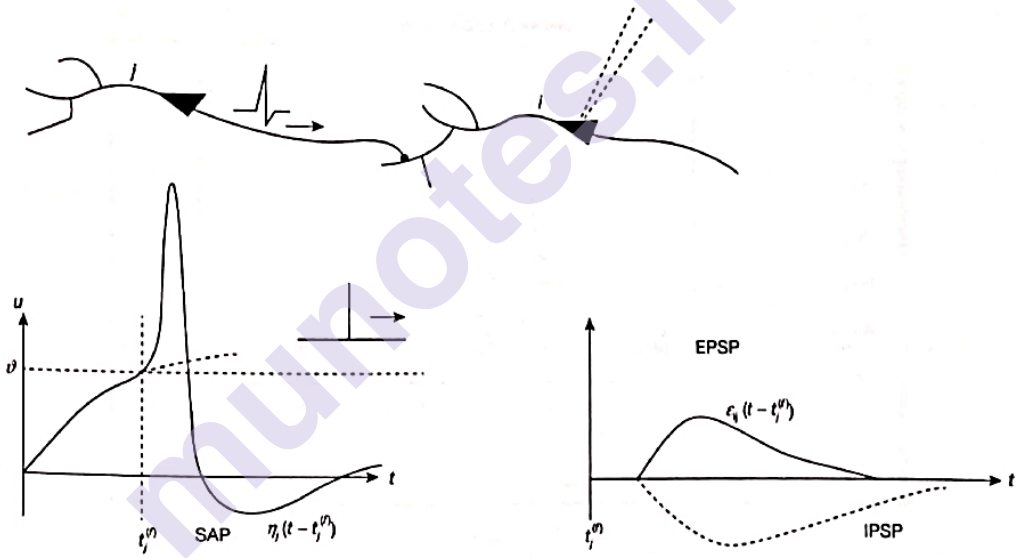


Fig-.6.13-SNN spikes: The membrane potential is increased and at time t(f) the membrane potential reaches the threshold so that a spike is emitted.

6.11.2. Izhikevich Neuron Model

The Izhikevich Neuron Model is defined by the following equation:

$$v' = 0.04v2 + 5v + 140 - u + I$$

$$u' = a(bv-u)$$

If v >= 30 mV, then $v = c$ and $u = u + d$. Here, / is the input, v is the neuron membrane voltage and u is the recovery variable of the activation of potassium *K* ionic currents and inactivation of sodium *Na* ionic currents. The model exhibits all known neuronal firing patterns with the appropriate values for the variables *a, b, c* and *d.*

1 The parameter *a* describes the time scale of the recovery variable *u.* Smaller values result in slower recovery. A typical value is $a = 0.02$.

2. The parameter *b* describes the sensitivity of the recovery variable *u* to the sub-threshold fluctuations of the membrane potential v. A typical value is *b* - 0.2.

3. The parameter c describes the after-spike reset value of the membrane potential v caused by the fast high-threshold *K* (potassium) conductance. A typical value for real neurons is $c = -65$ mV.

4. The parameter *d* describes the after-spike reset of the recovery variable *u* caused by slow high threshold *Na* (sodium) and *K* (potassium) conductance. A typical value of *d* is 2.

The IZ neuron uses voltage as its modelling variable. When the membrane voltage v(f) reaches 30 mV, a spike is emitted and the membrane voltage and the recovery variable are reset according to IZ neuron model equations. For I ms of simulation, this model takes 13 FLOPS. Figure 7-2 illustrates the IZ neuron model firing.

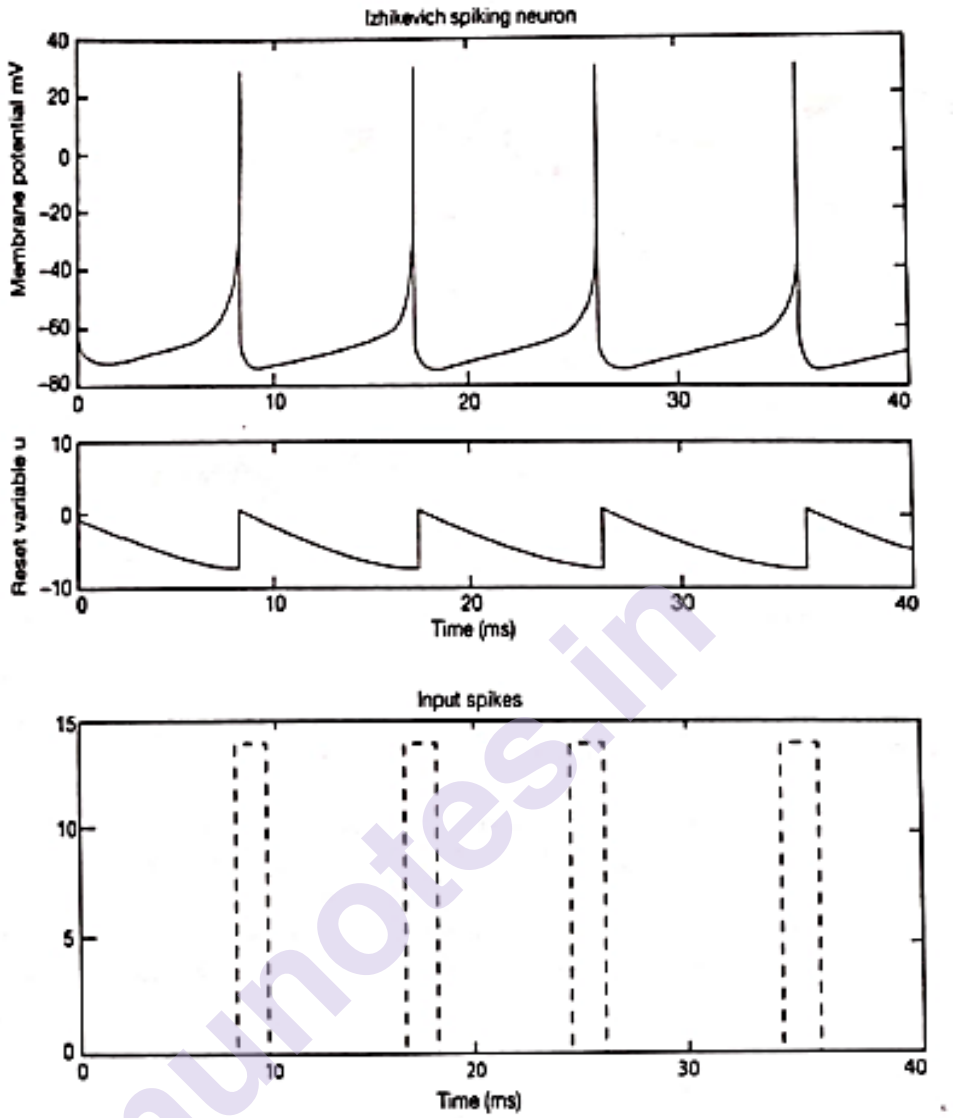


Fig- 6.14-The Izhikevich Spiking Neuron Model. In the top graph, there exists the membrane potential of the neuron. In the middle graph, there is the membrane recovery variable. Finally, the bottom plot represents the action pre-synaptic spikes.

The SNN with *N* neurons is assumed to be fully connected and hence the output of each neuron *I* is connected to every other neuron. The synaptic strength of these connections are given by the *N* x *N* matrix W where W[i, j] is the strength between the output of neuron *j* and the input of neuron i. Thus W[i, :] represents the synapses at the input of neuron *i,* whereas W[:, *j]* represents the synapse values connected to the outputs of neuron *j.* Each neuron has its own static parameters and varying state values. The set P represents the set of possible constant parameters and I is the set of neuron states. The set of possible inputs to the neurons is denoted by *R.*

The neuron updated function *f:(P,* S, *R)* -> *(S,* [0,1]) takes input parameters as the neuronal states and inputs and produces the next neuronal state and binary output.

Izhikevich's model uses a two-dimensional differential equation to represent the state of a single neuron i, namely, its membrane recovery variable u[i] and membrane potential v[i], that is (u[i], v[i]) ϵ S with a hard reset spike. Additional four parameters are used for the configuration of the neurons: *a* - time scale of u; *b* - sensitivity of *u; c* - value of v after the neuron is fired; *d* - value of *u* after the neuron is fired. Hence the neuron parameters are (a, *b, c, d)* ϵ *P,* These parameters can be tuned to represent different neuron classes. If the value of v[i| is above 30 mV, the output is set to 1 (otherwise it is 0) and the state variables are reset.

Izhikevich used a random input for each neuron in the range N(0,1), a zero mean and unit variance that is normally distributed. This input results in random number of neurons firing each time, depending not only on the intensity of the stimulus, but also on their randomly initialized parameters. After the input layer, one or more layers are connected in a feed-forward fashion. A spike occurs anytime the voltage reaches 30 mV. While the neurons communicate with spikes, the input current I_i of the neuron i is equal to

$$I_i = \sum_{j=1}^{n} w_{ij}\delta_j + \sum_{k=1}^{m} w_{ik} I_k(t)$$

where w_{ij} is weight of connection from node; to node i; w_{ik} is weight of connection from external input *k* to node i; $I_k(t)$ is binary external input *k;* δj is binary output of neuron *j* (0 or 1).

When the input current signal changes, the response of the Izhikevich neuron also changes, generating different firing rates. The neuron is responded during “T” ms with an input signal and it gets fired when its membrane potential reaches a specific value, generating an action potential (spike) or a train of spikes. The firing rate is evaluated as

$$\text{Firing rate} = \frac{\text{Number of spikes}}{T}$$

6.12. Encoding of Neurons in SNN

Spiking neural networks can encode digital and analogy information. The neuronal coding schemes are of three categories: *rate coding, temporal coding* and *population coding.* In rate coding, the information is encoded into the mean firing

rate of the neuron, which is also known as temporal average. In temporal coding, the information is encoded in the form of spike times. In population coding, a number of input neurons (population) are involved in the analog encoding and this produces different firing times. Commonly used encoding method is the population- based encoding.

In population encoding, analogy input values are represented into spike times using population coding. Multiple Gaussian receptive fields are used so that the input neurons encode an input value into spike times. The firing time is computed based on the intersection of Gaussian function. The centre of the Gaussian function is calculated using

$$\mu = I_{\min} + (2 * i - 3)/2 * (\mathrm{I}_{\max} - I_{\min})/(M - 2)$$

and the width is computed employing

$$\sigma = 1/\beta(I_{\max} - I_{\min})/(M - 2) \text{ where } 1 \leq \beta \leq 2$$

with the variable interval of $[I_{\min}, I_{\max}]$. The parameter " β " controls the width of each Gaussian receptive field.

6.12.1. Learning with Spiking Neurons

Similar to other supervised training algorithms, the synaptic weights of the network are adjusted iteratively in order to impose a desired input-output mapping to the SNN. Learning is performed through implementation of synaptic plasticity on excitatory synapses, The synaptic weights of the model, which are directly connected to the input pattern, determine the firing rate of the neurons. This means that the carried learning phase generates the desired behaviour by adjusting the synaptic weights of the neuron.

The neurons characterize sudden change of the membrane potential instantaneously prior to and subsequent to the firing. This potential behavioural feature leads to complexity in training SNNs. Some of the learning models include SpikeProp, spike-based supervised Hebbian learning, and ReSuMe and Spike time-dependent plasticity. Neurons can be trained to classify categories of input signals based on only a temporal configuration of spikes. The decision is communicated by emitting precisely timed spike trains associated with given input categories. Trained neurons can perform the classification task correctly.

The weights w between a pre-synaptic neuron *i* and a post-synaptic neuron *j* do not have fixed values. It has been proved through experiments that they change, and this affects the amplitude of the generated spike. The procedure of the weight

update is called learning process and it can be divided into two categories: supervised and unsupervised learning If the synaptic strength is increased then it is called long-term *potentiation* (LTP) and if the strength is decreased then it is called *long-term depression* (LTD).

6.12.2. Spike Prop Learning Algorithm

SNN employs spiking neurons as computational units which account to precise firing times of neurons for information coding. The information retrieval from the spike trains (neurons encode the information) are done by binary bit coding which is a population coding approach. This section presents the error-back propagation supervised learning algorithm as employed for the spiking neural networks.

Each SNN consists of a set of neurons (I, J), a set of edges ($E \subseteq I \times J$), input neurons $i \subseteq I$ and output neurons $j \subseteq J$. *For* each non-input neuron, $i \in I$, with threshold function V_{th} and potential *u(t),* each synapse $\{i, j\} \in E$ will have a response function ε_{ij} and weight w_{ij}. The structures of neurons tend to be fully connected feed forward neural network. The source neuron V will fire and propagate spikes along all directions. Formally, a spike train is defined as a sequence of pulses. Each target neuron w that receives a spike experiences an increase in potential at time *t*, similar as $w_{j,w} . \varepsilon_{j,w}$ (i-t).

The firing time of a neuron *i* is denoted as t where f = 1,2,3,... is the number of the spike. The objective is to train a set of target firing times t_{ft} and actual firing time t_a For a series of the input spike trains $S_{in}(t)$, a sequence of the target output spikes S (f) is obtained. The goal is to find a vector of the synaptic weights w such that the outputs of the learning neurons $S_{out}(t)$ are close to $S_t(t)$. Changing the weights of the synapses alters the timing of the output spike for a given temporal input pattern

$$S_1(t) = \sum_f \delta(t_e - t_f)$$

where $\delta(x)$ is the Dirac function, $\delta(x) = 0$ for $x \neq 0$ and $\int_{-}^{-} \delta(x)dx = 1$. Every pulse is taken as a single point in time. The objective is to train the desired target firing times $\{t_f\}$ and that of the actual firing times $\{t_a\}$. The least mean squares error function is chosen and is defined by

$$E = \frac{1}{2}\sum_{i \leqslant v} (t_\alpha - t_6)^2$$

In error-back propagation algorithm, each synaptic terminal is taken as a separate connection k from neuron i to j with weight $w_{\not} i\eta^*$ is the learning rate parameter. The basic weight adaptation functions for neurons in the output layer hidden layer are given by

$$\delta_j = \frac{\delta_{\xi}}{\delta_{\mathrm{u}}} \frac{\delta_{\mathrm{s}}}{\delta x_i(t_{\circ})} = \frac{(t_k - t_e)}{\sum_{i\epsilon t_j} \sum_1 w_{\phi i} \frac{\delta_{r'}(t)}{\delta t_d}}$$

$$\begin{aligned} \Delta w_{b,k} &= -\eta \frac{\delta E}{\delta w_{bk}} = -\eta y_{ik}(t_a) \cdot \delta_t \\ \delta_i &= \frac{\delta t_a}{\delta x_i(t_a)} \sum_{\mu \epsilon_i} \delta_j \frac{\delta x_1(t_a)}{\delta t_a} \\ \Delta w_{h,k} &= -\eta y_{hk}(t_{\alpha}) \cdot \delta_j \end{aligned}$$

The training process involves modifying the thresholds of the neuron firing and synaptic weights. The algorithmic steps involved in learning through Spike-Prop Algorithm are as follows:

6.12.3. Spike-Prop Algorithm

Step 1: The threshold is chosen and the weights are initialized randomly between 0 and 1.

Step ***2:*** In feed-forward stage, each input synapse receives input signal and transmits it to the next neuron (i.e., hidden units). Each hidden unit with SNN function calculated is sent to the output unit which in return calculates the spike function as the response for the given input. The firing time of a neuron t_a is found. The time to first spike of the output neurons is compared with that of the desired time t_{fi} of the first spike.

Step 3: Perform the error-back propagation learning process for all the layers. The equations are transformed to partial derivatives and the process is carried out.

Step 4: Calculate δ_j using actual and desired firing time of **each** output neuron.

Step 5; Calculate δ_i employing the actual and desired firing times of each hidden neuron and δ_j values.

Step 6: Update weights: For output layer, calculate each change in weight.

Step 7: Compute: New weight = Old weight + Δ w_{ijk}

Step 8: For hidden layer, calculate each change in weight.

Step 9: Compute new weights for the hidden layer. New weight = Old weight + Δ w_{hik}

Step 10: Repeat until the occurrence of convergence.

6.12.4. Spike Time-Dependent Plasticity (STOP) Learning

Spike time-dependent plasticity (STOP) is viewed as a more quantitative form of Hebbian learning. It emphasizes the importance of causality in synaptic strengthening or weakening. STDP is a form of Hebbian Learning where spike time and transmission are used in order to calculate the change in the synaptic weight of a neuron. When the pre-synaptic spikes precede post-synaptic spikes by tens of milliseconds, synaptic efficacy is increased. On the other hand, when the post-synaptic spikes precede the pre-synaptic spikes, the synaptic strength decreases. Furthermore, the synaptic efficacy Δw_{ij} is a function of the spike times of the pre-synaptic and post-synaptic neurons. This is called Spike *Timing*-Dependent Plasticity (STDP). The well-known STDP algorithm modifies the synaptic weights using the following algorithm

$$\Delta w = \begin{cases} A^{+}\exp\left(\Delta t/\tau^{+}\right) \text{if } \Delta t < 0 \\ -A^{-}\exp\left(-\Delta t/\tau\right) \text{ if } \Delta t \geq 0 \end{cases}$$

$$w_{\text{mev}} = \begin{cases} w_{\text{old}} + \eta\Delta w(w_{\text{mas}} - w_{\text{old}}) \text{ if } \Delta w \geq 0 \\ w_{\text{ous}} + \eta\Delta w(w_{o\Delta} - w_{\min}) \text{ if } \Delta t < 0 \end{cases}$$

Where $\Delta t = (t_{pre} - t_{post})$ the time delay between the pre synaptic spike and the post synaptic spike. If the pre-synaptic spike occurs before the post synaptic spike, the weight of the synapse should be increased. If the pre synaptic spike occurs after the post-synaptic spike, then the weight of the synapse gets reduced. STDP learning can be used for Inhibitory or excitatory neurons.

6.12.5. Convolutional neural network (CNN)

Convolutional neural network (CNN) is built up of one or more number of convolutional layers and after then it is trailed by one or more fully connected layers like feed forward networks. CNN architecture is designed to possess the structure of a two dimensional input image, that is, CNN's key advantage is that its input consists of images and this representation of images designs the architecture in a practical way. The neurons in CNN arc arranged in 3 dimensions: height, width, and depth. The information pertaining to "depth" is an activation volume and it represents the third dimension. This architectural design of CNN is carried out with the local connections and possesses weights which art subsequently followed by certain pooling operations. CNN's can be trained in an easy manner and these have minimal parameters for the same number of hidden units than that of the other fully interconnected networks considered for comparison, figure 7-3 shows the arrangement of neurons in three dimensions in a convolutional neural network. As a regular neural network, the convolutional neural network is also made up of

layers, and each and every layer transforms an input 3D volume to an output 3D volume along with certain differentiable activation functions with or without any parameters.

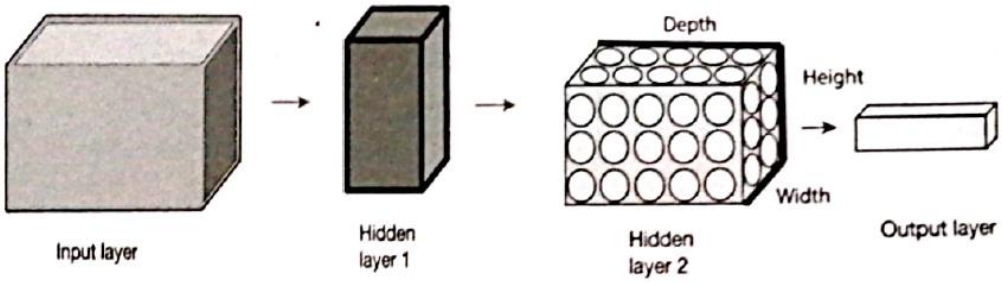


Figure 6.15 Arrangement of neurons in CNN model

6.12.6. Layers in Convolutional Neural Networks

It is well noted that the convolutional neural network is a sequence of layers and each and every layer in CNN perform transformation of one volume activations to the other by employing a differentiable function. CNN *consists* of three major layers:

1. Convolutional layer
2. Pooling layer
3. Fully interconnected layer (regular neural models like perceptron and BPN)

These layers exist between the input layer and output layer Input layer holds the input values represented by the pixel values of an image. Convolutional layer performs computation and determines output of a neuron that is *connected to* local regions in the input. The computation is done by performing dot product between their weights and a small region that is connected to the input volume. After then, an element wise activation function is applied wherein the threshold set to zero. Applying this activation function results no change in the size of the volume of the layers Pooling layer carries out the down sampling operation along with the spatial dimensions including width and height Regular fully connected layers perform computation of the class scores (belongs to the class or nut) and result m a specified volume size. In this manner, convolutional neural networks transform the original input layer by laser and result in the final scores. Pooling layer implements only a died function whereas convolutional and fully interconnected layer implements transformation on functions and as well on the weights and biases of the neurons.

Fundamentally, a convolutional neural network is none comprising a sequence of layers that transform the image volume into an output volume. Each of the designed layers in CNN is modelled to take an input 3 dimensional volume data and perform transformation to an output 3 dimensional data employing a differentiable function Here, the designed convolutional and fully inter connected layers possess parameters and the pooling layers do not possess a parameter.

6.12.7. Architecture of a Convolutional Neural Network

It is well known that CNN is made up of a number of convolutional and pooling (also called as sub-sampling) layers, subsequently followed by fully interconnected layers (at certain cases this layer becomes optional based on the application considered).

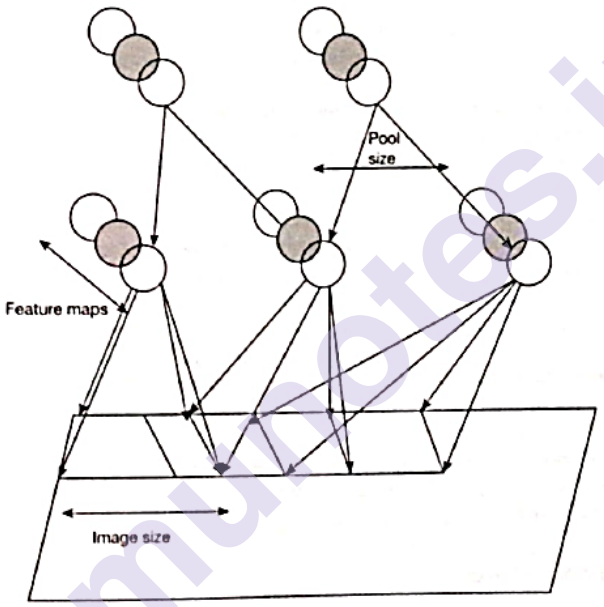


Figure 6.17 CNN with convolutional and pooling layers

The input presented to the convolutional layer is an *n* x *n x p* image where "*n*" is the height and width of an image and *"p"* refers to the number of channels (e g., an RGB image possess 3 channels and so *p* = 3). The convolutional layer to be constructed possesses *'m* filters of size r x r x *q,* where *"r"* tends to be smaller than the dimension of the image and *"q"* can be the same size as that of *"p"* or it can be smaller and vary for each of the filter. The filter size enables the design of locally connected structure which gets convolved with the image for producing "m" feature maps. The size of feature map will be "*n* - *r* + 1". Each of the feature maps then gets pooled (sub-sampled) based on maximum or average pooling over *r x r* connecting regions. The value of *"r"* is 2 for small images and 5 for larger images. A bias and a non-linear sigmoidal function can be applied to each of the feature

map before or after the pooling layer, figure *7*-4 shows the architecture of the convolutional neural network.

6.12.8. Designing the Layers in CNN Model

CNN b nude up of the three individual layers and this subsection presents the details on designing each of these lasers specifying their connectivity and hyper parameters.

1- Design of Convolutional Layer

The primary building block of convolutional neural network is the convolutional layer. The convolutional layer is designed to perform intense computations in a CNN model. Convolutional layer possess a set of trainable filters and every filter is spatially small (along the width and height) but noted to extend through the fullest depth of the input volume. When the forward pass gets initiated, each filter slides across the height and width of the input volume and the dot product is computed between the input at any position and that of the entries in the filter. When the filter slides across the height and weight of the input volume, a two-dimensional activation feature map is produced that gives the responses of that filter at every spatial position. The filters get activated when they come across certain type of visual features (like edge detection, color stain on the first layer, certain specific patterns or honeycomb existing on higher layers of the network) and the network learns from the filter that gets activated. Convolutional layer consists of the complete set of filters and each of these filters produces a separate 2-dimensional activation map. These activation maps will be stacked along the depth dimension and result in the output volume.

In CNN network model, at the convolutional layer, each neuron gets connected only to a local region of the input volume. The spatial extent of this neuronal connectivity is represented by a hyper-parameter called the *receptive field* of the neuron. This receptive field of the neuron is the filter size. This spatial extent's connectivity along the depth axis will be equal to the depth of input volume. These connections tend to be local in space and get full towards the entire depth of the input volume.

With respect to the number of neurons in the output volume, three hyper-parameters are noted to control the size of the output volume - depth, stride and zero-padding. The depth of the output volume refers to the number of filters to be used, wherein each learning searches the existence of difference in the input. The stride is to be specified for sliding the filter.

one pixel at a time, stride = 1

f_{move}

2 pixel at a time, stride = ∠

subsequently for other strides

The movement of the filter is specified by the above equation. This representation of the strides results in smaller output volumes spatially. At times it is required to pad the input volume with zeros around the border, hence, the other hyper-parameter is the size of this zero-padding. Zero-padding allows controlling the spatial size of the output volumes. It should be noted that if all neurons presented in the single depth slice employ the same weight vector, then in every depth slice, the forward pass of the convolutional layer can be computed as the convolution of the neuronal weights with that of the input volume. Thus, the sets of weights are referred in CNN as filter that gets convolved with the input. The limitation of this approach is that it uses lots of memory, as certain values in the input volume arc generated repeatedly for multiple times.

It is to be noted that the backward pass for a convolution operation is also a convolution process. The backward pass also moves to a back propagation neural network. In few works carried out earlier, it is observed that they use 1 x 1 convolution, but for a two-dimensional case it is similar to a point-wise scaling operation. As with CNN model, it is operated more on three-dimensional volumes and also the filters get extended over the full depth of the input volume. It is to be noted that employing 1 x 1 convolution will perform the three-dimensional dot product. Another method of convolution is the dilated convolution, wherein an added hyper-parameter called dilation is included to the convolutional layer. In case of dilated convolution, there is possibility to have filters with spaces between each cell. Implementation will be done in a manner of dilation 0, dilation 1 (gap 1 will be adopted between the filters) and so on. Employing dilated convolutions drastically increases the receptive field.

2-Design *of Pooling Layer*

Between the successive convolutional layers, pooling layers are placed. The presence of pooling layer between the convolutelayers is to gradually decrease the spatial size of the parameters and to reduce the computation in the network. This placement of pooling layer also controls the occurrence of over fitting. The pooling layer works independently on depth slice of the input as well as resizes them

spatially. Commonly employed pooling layer is the one with the filter size of 2 x 2 applied with a stride of 2 down samples. The down sampling occurs for every depth slice in the input by 2 along the height and width. The dimension of the depth parameter remains unaltered in this case. Pooling sizes with higher receptive fields are noted to be damaging. Generally used pooling mechanism is the “max pooling”.

Apart from this operation, the pooling layer can also perform functions like mean pooling or even L2-norm pooling. In the backward pass of a pooling layer, the process is only to route the gradient to the input that possessed the highest value in the forward pass. Hence, at the time of forward pass of the pooling layer, it is important to track the index of the activation function (probably “max”) so that the gradient routing is carried out effectively by a back-propagation network algorithm.

6.12.9. Layer Modelling in CNN and Common CNN Nets

The other layers of importance in convolutional neural network are the normalization layer and the fully connected layer. Numerous normalization layers are developed to be used in CNN model and they are designed in a manner to implement the inhibition procedure of the human brain. Various types of normalization procedures like mean scaling, max scaling, summation process, etc. can be employed if required for operation in the CNN model. Fully connected layers possess full interconnections for all the activations in the previous layer. As regular, their activations are based on computing the net input to the neurons of a layer along with the bias input also.

6.12.10. Conversion of Fully Connected Layer to Convolutional Layer

The main difference between the fully connected and the convolutional layer is that the neurons present in the convolutional layer get connected only to a local region in the input and the neurons in the convolutional voluminous structure share their parameters. The neurons in both fully connected and convolutional layers calculate the dot products and hence their functional form remains the same. Therefore it is possible to perform conversion between the fully connected and the convolutional layers.

Considering any convolutional layer, there exists a fully connected layer which implements one and the same forward pass function. The weight matrix will be a large one and possesses zero entities except at specific blocks (no self-connection and existence of local connectivity') and the weights in numerous blocks tend to be equal (parameter sharing). Also, fully connected layer can be converted into convolutional layer; here the filter size will be set equal to the size

of the input volume and the output will be a single depth column fit across the input volumes. This gives the same result as that of the initial fully connected layer. In both these conversions, the process of converting a fully connected layer to a convolutional layer is generally in practice.

6.13. CNN Layer Sizing

As known, CNN model commonly comprises convolutional layer, pooling layer, and fully connected layer. The rules for sizing the architecture of the CNN model are as follows:

1. The input layer should be designed in such a way that it should be divisible by the convolutional layer should employ small size filters, specifying the stride. The convolutional layer should not alter the spatial dimensions of the input.

2. The pooling layer down samples the spatial dimensions of the input. Commonly used pooling is the max-pooling with a 2 x 2 receptive fields and a stride of 2. Receptive field size is accepted until 3x3 and if it exceeds above 3, the pooling becomes more aggressive and tends to lose information. This results in poor performance of the network.

From all the above, it is clearly understood that the convolutional layers preserve the spatial size of their input. On the other hand, the pooling layers are responsible for down sampling the volumes spatially. Alternatively, if strides greater than 1 or zero-padding are not done to the input in convolutional layers, then it is very important to track the input volumes through the entire CNN architecture and ensure that all the strides and filters work in a proper manner. Smaller strides are generally better in practice. Padding actually improves the performance of the network. When the convolutional layer does not zero-pad the inputs and only performs authenticate convolutions, then the volume size will reduce by a smaller amount after each convolution process.

6.13.1. Common CNN Nets

In the past few years, there were numerous CNN models developed and implemented for various applications. Few of them include

1. *LeNet:* The first convolutional neural network model named after the developer LeCun. It is applied to read zip codes, digits and so on.

2. AlexNet: CNN model in this case was applied to computer vision application.

It was developed in the year 2012 by Alex Krizhevsky and team.

3. ZFNetf: It was developed in the year 2013 by Zeiler and Fergus and hence named as ZFNet. In this network model, the convolutional layers in the middle are expanded and the stride and filter size are made smalt in the first layer.

4. *VGGNet*: It was modelled in the year 2014 by Karen and Andrew. It has phenomenal impact on the depth of the network and it was noted that depth of network parameter plays a major role for better performance.

5. *GoogLeNet* It was developed in the year 2014 from Google by Szegedy and team. This net contributed an Inception module wherein the numbers of parameters in the model are reduced. This network employs mean pooling instead of fully connected layers at the top of the convolutional network. As a result, more number of parameters arc eliminated in this case.

6. *ResNet:* It was modelled in the year 2015 by Kaiming and team, and hence called as Residual Network. This network is the default convolutional neural network. It employs batch normalization and the architecture also docs not consider fully connected layers at the end of the network.

6.13.2. Limitations of CNN Model

The computational considerations are the major limitations of the convolutional neural network model. Memory requirement is one of the problems for CNN models. In the current processor unit, the memory limits from 3/4/6 GB to the latest best version of 12 GB memory. The memory can be handled by

1. Convolutional network implementations should maintain varied memory requirements, like the image data modules

2. Intermediate volume sizes specify the number of activations at each layer of the convolutional network as well is their gradients. Running convolutional network at the time of testing alone reduces the memory by large amount, by storing only the current activations at any layer and eliminating the activations of the previous layer.

3. Network parameters and their size, gradient descent values of the parameters during backward pass in back propagation process and also a step cache when momentum factor is used. The memory required to store a parameter alone should be multiplied by a factor of at least 3 or so.

On calculating the total number of parametric values, the number must be converted to a specified size in GB for memory requirement. For each of the parameters, consider the number of parametric values. Then multiply the number of parametric values by 4 to get the raw number of bytes and then divide it by multiples of 1024 to get the amount of memory in KB, MB and then in GB. In this way, the memory requirement of CNN model can be computed and the limitations can be overcome.

6.14. Deep learning Neural networks:

Machine learning approaches are undergoing a tremendous revolution, which has led to the development of third generation neural networks. The limitations observed in the second-generation neural networks like delayed converged undue local and global minimal problems and so on are handled in the developed third-generation neural networks. One of the prominent third generation neural networks is the deep learning neural networks (DLNNs) and this neural model provides a deep understanding of the input information.

The prominent researcher behind the concept of deep learning neural networks is Professor Hinton from University of Toronto who managed to develop a special program module for constituting the formulation of molecules to produce an effective medicine. Minton's group employed deep learning artificial intelligence methodology to locate the combination of molecules required for the composition of medicine with very limited information on source data. Apple and Google have transformed themselves with deep learning concepts and this can be noted through Apple Siri and Google Street view, respectively.

The learning process in deep learning neural network takes place in two steps. In the first step, the information about the input data’s internal structure is obtained from the existing large array of unformatted data. This extraction of the internal structure is carried out by an auto-associator unit via unsupervised training layer-by-layer, then the formatted data obtained from the unsupervised multi-layer neural network gets processed through a supervised network module employing the already available neural network training methods. It is to be noted that the amount of unformatted data should be as large as possible and the amount of formatted data can be smaller in size (but this need not be an essential criteria).

6.14.1. Network Model and Process Flow of Deep Learning Neural Network

The growth of deep learning neural networks is its deep architecture that contains multiple hidden layers and each hidden layer carries out a non-linear transformation between the layers. DLNNs get trained based on two features:

1. Pre-training of the deep neural networks employing unsupervised learning techniques like auto-encoders layer-by-layer,

2. Fine tuning of the DLNNs employing back propagation neural network.

Basically, auto-encoders are employed with respect to the unsupervised learning technique and the input data is the output target of the auto-encoder. An auto-encoder consists of two parts - encoder and decoder network. The operation of an encoder network is to transform the input data that is present in the form of a high-dimensional space into codes pertaining to low-dimensional space. The operation of the decoder network is to reconstruct the inputs from the corresponding codes. In encoder neural network, the encoding function is given by *"f_Θ"*. The encode vector (E^v) is given by

$$E^v = f_\Theta (x^v)$$

where "x" is the data set of the measured signal.

The reconstruction operation is carried out at the decoder neural network and its function is given by *"g_Θ"*. This reconstruction function maps the data set "x^v" from the low-dimensional space into the high-dimensional space. The reconstructed form is given by

$$\hat{x}^v = g_\theta(E^v)$$

The ultimate goal of these encoder and decoder neural networks is to minimize the reconstruction error $E(x, \hat{x})$ for that many numbers of training samples. $E(x, \hat{x})$ is specified as a loss function that is used to measure the discrepancy between the encoded and decoded data samples. The key objective of the unsupervised auto-encoder is to determine the parameter sets that minimize the reconstruction error "E"

$$\delta_{ae}(\theta,\theta') = \frac{1}{N} \sum_{v=1}^{N} E\ (x^v, g'_\theta(f_\Theta (x^v)))$$

The encoding and decoding functions of the DLNN will be present along with a non-linearity and are given by

$$f_\Theta (x) = f_{af\_e} (b + W_x)$$

$$g_\Theta (x) = f_{af\_d} (b + W_x^T)$$

Where f_{af_e} *and* f_{af_d} refer to the encoder activation function and the decoder activation function, respectively, "b" indicates the bias of the network, and W and

W^T specify the weight matrices of the DLNN model.

The reconstruction error is given by

$E(x, \hat{x}) = \| x - \hat{x} \|$

In order to carry out the pre-training of a DLNN model, the “N" auto-encoders developed in previous module should be stacked. For the given input signal x^v input layer along with the first hidden layer of DLNN arc considered as the encoder neural network of the first auto-encoding process. When the first auto-encoder is noted to be trained by minimizing the reconstruction error, the first trained parameter set θ_1, of the encoder neural network is employed to initialize the first hidden layer of the DLNN and the first encode vector is obtained by

$$E_1^v = f_\Theta (x^v)$$

Now, the input data becomes the encode vector E_1^v. The first and second hidden layers of the DLNN are considered as the encoder neural network for the second auto-encoder. Subsequently, the second hidden layer of the DLNN gets initialized by that of the second trained auto-encoder. This process gets continued upto the N-th auto-encoder that gets trained for initializing the final hidden layer of the DI.NN model. The final or the N-th encode vector in generalized form for the vector x^v is obtained by

$$E_N^v = f_\Theta (E^v_{N-1})$$

where “θ_N” denotes the Nth trained parameter set of the encoder neural network. Thus, in this way, all the DLNN s hidden layers get pre trained by means of the N stacked auto encoders. It is well noted that the process of pre-training avoids local minima and improves generalization aspect of the problem under consideration. Figure 7-5 shows the fundamental architecture of the deep learning neural network.

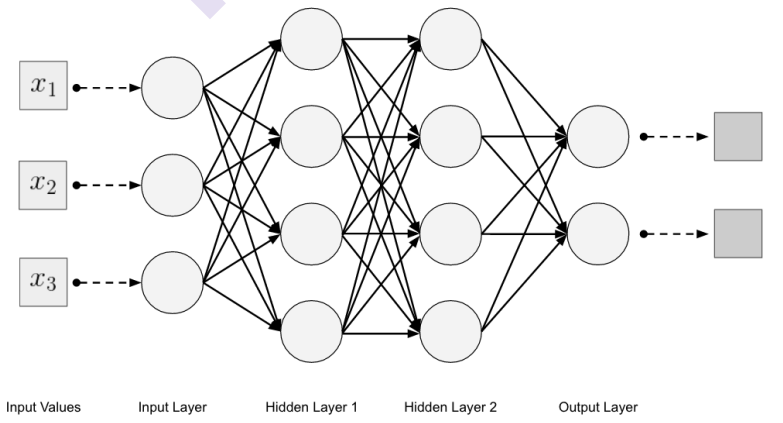


Figure 6.18 Architecture model of deep learning neural network

The above completes the pre training process of DLNN and the nest process is the tine-tuning process in the DLNN model DLNN models output is calculated from the input signal X^y as

$$y^y = f_{\Theta N+1}(E^{y_N})$$

where Θ_{N+1} denotes the trained parameter set of the output layer. Here, back propagation network (BPN)is employed for minimizing the error of the output by carrying out the parameter adjustments in DLNN backwards in case the output the target of x^x is t^y , then the error criterion is given by

$$MSE(\Psi) = 1/N \sum_{y=1}^{n} E(y^y, t^y)$$

Where $\Psi = \{\Theta_1, \Theta_2, \Theta_3, \ldots\ldots \Theta_{N+1}\}$

6.14.2. Training Algorithm of Deep Learning Neural Network:

Step 1: Start the algorithmic process.

Step 2: Obtain the training data sets to feed into the DLNN model and initialize the necessary parameters.

Step 3: Construct DLNN with "N" hidden layers.

Step 4: Perform the training of r-th auto-encoder.

Step 5: Initialize i-th hidden layer parameter of DLNN employing the parameters of the auto encoder.

Step 6: Check whether “i” is greater than “N". If no carry out step 4; if yes go to the next step.

Step 7: Calculate the dimensions of the output layer.

Step 8: Fine tune the parameters of DLNN through the BBN algorithm.

Step 9: With the final fine-tuned DLNN model go to the next step.

Step 10: Return the trained DLNN.

Step 11: Output the solutions achieved.

Step 12: Stop the process on meeting termination condition. The termination condition is the number of iterations or reaching the minimal mean square error.

6.14.3. Encoder Configurations

Encoders are built so as to receive the possible exact configuration of the input at the output end. These encoders belong to the category of auto associator neural units, Auto associator modules, are designed to perform the generating part as well as the synthesizing part. Encoders discussed in this section belong to the synthesized module of auto associator and tor the generation part, a variation of Boltzmann machine as presented in special networks.

An auto encoder is configured to be all open layer neural network Auto encoder for its operation sets its target value equal to that of the Input vector. A model of an auto encoder is as shown in figure 7.6. The encoder model attempts to find approximation of a defined function authenticating that the feedback of a neural network tends to be approximately equal to the values of the given input parameters. The encoder is also capable of compressing the data once the given input signal gets passed to that of the output of the network. The compression is possible in an auto encoder if there exists hidden interconnections or a sort of characteristics correlation. In this manner, auto encoder behaves m a similar manner as the principal component analysis and achieves data reduction (possible compression) in the input side.

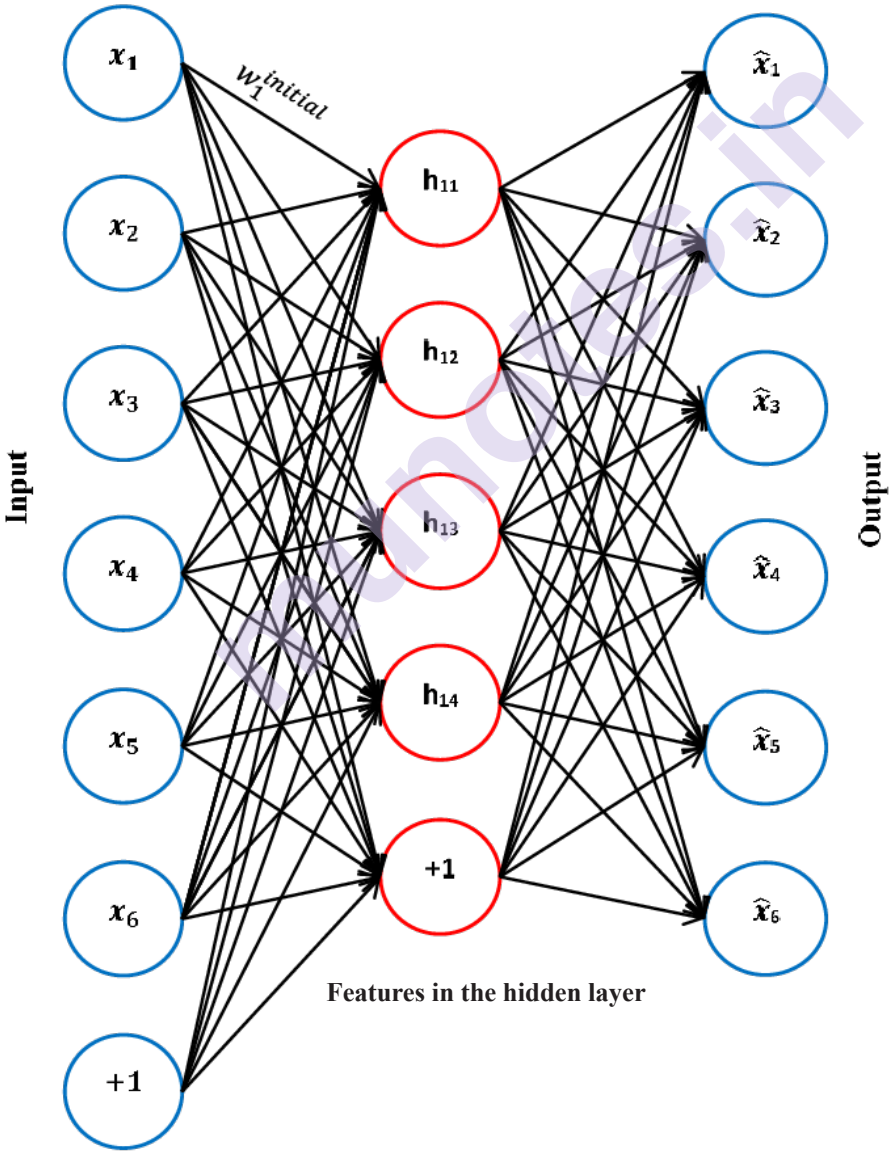


Figure 6.19 Model configuration of an auto encoder

On the other hand, when the auto encoder is trained with the stochastic gradient descent algorithm and the where the number of hidden neurons becomes greater than the number of inputs, it results in the possible decrease in the error values. So, it is applied for various function analysis and compression applications

Another variation in the encoder configuration is the denoting auto encoder. Here, the variation exists in the training process. On training the deep learning neural network for demolishing encoder, corrupted or demonised data (substituted with “0" values) can be given as input. further to this, during the same time, the coned data can be compared with that of the output data. The advantage of this mechanism is that it paves way to restore the damaged data.

6.15. EXTREME LEARNING MACHINE MODEL (ELMM)

Over the years, it has been observed that the k nearest neighbourhood and other few architectures like support machine (SVM) classifiers employed for classification requite more computations due to the repetition of classification and registration, hence they are relatively slow. SVM approach, even though it has the advantages of generalization and can handle high dimensional feature space, assumes that the data are independently and identically distributed. This is not applicable for all sets of data, as they are likely to have noise and related distribution. Storage is also an added disadvantage of SVM classifier.

Other multilayer neural networks which are trained with back propagation algorithm based on gradient descent learning rule. Posses certain limitations like slow conversions, setting the learning rate parameters, local and global minimum occurrences and repeated training process without attaining conversions point.

ELMM is a single hidden layer feed forward neural network where input weights and hidden neuron are randomly selected without training. The output weights are analytically computed employing the least square norm solution and Moore – Penrose inverse of a generalized linear system. This method of determining output weights results in significant reduction of training time. For hidden layer neurons are the activation functions like Gaussian, sigmoidal and so on can be employee for output layer neurons layer linear activation function. This single layer feed forward, network ELM model employee additive neural design instead of kernel based and hence there is random parameter selection.

6.15.2. ELM TRAINING PROGRAM:

For a given training vector pair $N=\{x_t, t_t)\}$, *with* $x_i \in R^n$ $t_i \in R^m$, $i=1,\ldots,N$
activation function f(x) and hidden neuron N, the algorithm is as follows:

Step 1: Start Initialize the necessary parameters, choose suitable activation function and the number of hidden neurons in the hidden layer for the considered problem.

Step 2: Assign arbitrary input weights w_i and bi as b_i

Step 3: Compute the output matrix H at the hidden layer

$$H= ƒ(xΘw+b)$$

Step 4: Compute the output weight ŏ based on the equation

$$ŏ=H*T$$

6.15.3. Other ELM Models

Huang initially proposed ELM in the year 2004 and subsequently numerous researchers worked on ELM and developed certain improved ELM algorithms. ELM was enhanced over the years to improve the network training speed, to avoid local and global minima, to reduce iteration time, to overcome the difficulty in defining learning role parameters and setting the stopping criteria.

Since ELM works on empirical minimization principle, the random selection of input layer weights and hidden layer biases result in non-optimal convergence. In comparison with that of the gradient descent learning rule, ELM may require more number of hidden layer neurons and this reduces ELM's training effect. Henceforth, to speed the convergence and response of ELM training, numerous improvements were made in existing ELM algorithm and modified versions of ELM algorithm were introduced. The following sub-sections present few improvements made by researchers in the existing ELM algorithm.

6.15.4. Online Extreme Learning Machine

ELM is well noted for solving regression and classification problems; it results in better generalization performance and training speed. When considering ELM for real applications which involve minimal data set, it may result in over-fitting occurrences.

Online ELM is also referred to as online sequential extreme learning machine (OSELM) and this works on sequential adaptation with recursive least square algorithm. This was also introduced by Huang in the year 2005. Further to this, online sequential fuzzy ELM (OS-Fuzzy-ELM) has also been developed for implementing different orders of TSK models. In fuzzy-based FLM, randomly all the antecedent parameters of membership functions are assigned first and subsequently the consequent parameters are computed. Zhang, in the year 2011,

developed selective forgetting ELM (SFELM) to overcome the online training issues and applied it to time-series prediction. SFELM's output weight is calculated in a recursive manner at the time of online training based on its generalization performance. SFELM is noted to possess better prediction accuracy.

6.15.5. Pruned Extreme Learning Mac/i/ne

ELM is well known for its short training time and here the number of hidden layer nodes are randomly selected and are analysed for determination of their respective weights. This minimizes the calculation time with fast learning. Rong in the year 2008 modified the architectural design of ELM as the existence of smaller or higher hidden layer neurons will result in Under-fitting and over-fitting problems for classification problems. Pruned ELM (PELM) algorithm was developed as an automated technique to design an ELM. The significance of hidden neurons was measured in PELM by employing statistical approaches. Starting with higher number of hidden neurons, the insignificant ones are then pruned with class labels based on their importance. Henceforth the architectural design of ELM network gets automated. PELM is inferred to have better prediction accuracy for unseen data when compared with basic ELM. there also exists a pruning algorithm that is based on regularized regression method, to determine the required number of hidden neurons in the network architecture. This regression approach starts with higher number of hidden neurons and in due course the unimportant neurons get pruned employing methods like ridge regression, elastic network and so on. In this manner, the architectural design of FILM network gets automated.

6.15.6. Improved Extreme Learning Machine Models

ELM requires more number of hidden neurons due to its random computation of input layer weights and hidden biases. Owing on this, certain hybrid ELM algorithms were developed by researchers to improve the generalization capability. One of the method proposed by Zhu (2005) employs differential evolution (DE) algorithm for obtaining the input weights and Moore-Penrose (MP) inverse to obtain the output weights of an ELM model. Several researchers also attempted to combine ELM with other data processing methods resulting in new ELM learning models and applying the newly developed algorithm for related applications.

ELM at times results in non-optimal performance and possess over-fitting occurrence. This was addressed by Silva in the year 2011 by hybridizing group search optimizer to compute the input weights and ELM algorithm for computing

the hidden layer biases. Here it is required to evaluate the influence of various types of members that tend to fly over the search space bounds. The effectiveness of ELM model gets lowered because at times, the hidden layer output matrix obtained through the algorithm docs not form a full rank matrix due to random generation of input weights and biases. This was overcome by the development of effective extreme learning machine (EELM) neural network model which properly selects the input weights and biases prior to the calculation of output weights ensuring a full column rank of the output matrix.

Thus, considering the existing limitations of ELM models, researchers have involved themselves in developing new variants of ELM models both in the algorithmic side and in the architectural design side. This section has presented few of the variants of ELM models as developed by the researchers and applied for various prediction and classification problems.

6.15.7. Applications of ELM

Neural networks are widely employed in mining, classification, prediction, recognition and other applications. ELM has been developed with an idea to improve the learning ability and provide better generalization performance. Considering the advantages of ELM models, few of its application include

1. Signal processing
2. Image processing
3. Medical diagnosis
4. Automatic control
5. Aviation and aerospace
6. Business and market analysis

Summary:

In this chapter we learn about Simulated Annealing Network, Boltzmann Machine, Gaussian Machine, Cauchy Machine, Probabilistic Neural Net ,Cascade Correlation Network, Cognitron Network ,Neocognitron Network, Cellular Neural Network , Optical Neural Networks, Spiking Neural , Networks (SNN) ,Encoding of Neurons in SNN, CNN Layer Sizing, Deep learning Neural networks, Extreme Learning Machine Model (ELMM) in detail.

Review Questions:

1. Write a short note on Simulated Annealing Networks?
2. Explain Architecture of Boltzmann Machine.
3. Explain Probabilistic Neural Net.
4. Write a short note on Cellular Neural Network.
5. What are the Third-Generation Neural Networks?
6. Explain Architecture of a Convolutional Neural Network
7. What are the Limitations of CNN Model.
8. Write a short note on Deep learning Neural networks.
9. Write a short note on ELM Architecture and Training Algorithm

Reference:

1. "Principles of Soft Computing", by S.N. Sivanandam and S.N. Deepa, 2019, Wiley Publication, Chapter 2 and 3
2. http://www.sci.brooklyn.cuny.edu/ (Artificial Neural Networks, Stephen Lucci PhD)
3. Related documents, diagrams from blogs, e-resources from RC Chakraborty lecture notes and tutorialspoint.com.

❖❖❖

Unit IV

7

INTRODUCTION TO FUZZY LOGIC AND FUZZY

Unit Structure

7.0 Objectives

We begin this chapter with introducing fuzzy logic, classical sets and fuzzy sets followed by the comparison of classical sets and fuzzy sets.

7.1 Introduction to Fuzzy Logic

Fuzzy logic is a form of multi-valued logic to deal with reasoning that is approximate rather than precise. Fuzzy logic variables may have a truth value that ranges between 0 and 1 and is not constrained to the two truth values of classical propositional logic.

"As the complexity of a system increases, it becomes more difficult and eventually impossible to make a precise statement about its behavior, eventually arriving at a point of complexity where the fuzzy logic method born in humans is the only way to get at the problem" – Originally identified & set forth by Lotfi A. Zadeh, Ph.D., University of California, Berkeley.

Fuzzy logic offers soft computing:

- provides a technique to deal with imprecision & information granularity.
- provides a mechanism for representing linguistics construct.

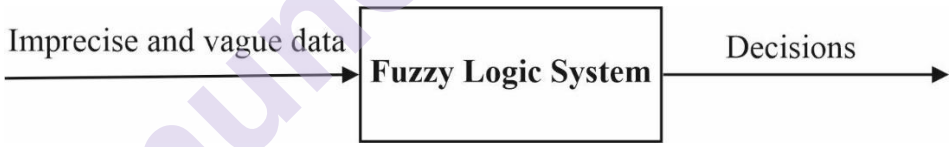


Figure 7.1: A fuzzy logic system accepting imprecise data and providing a decision

The theory of fuzzy logic is based upon the notion of relative graded membership and so are the functions of cognitive processes. It models uncertain or ambiguous data & provides suitable decision. Fuzzy sets that represents fuzzy logic provides means to model the uncertainty associated with vagueness, imprecision & lack of information regarding a problem or a plant or system.

Fuzzy logic operates on the concept of membership. The basis of the theory lies in making the membership function lie over a range of real numbers from 0.0 to 1.0. The fuzzy set is characterized by (0.0,0,1.0). The membership value is "1" if it belongs to the set & "0" if it not member of the set. The membership in the set is found to be binary, that is, either the element is a member of a set or not. It is indicated as

$$\chi_{\mathrm{A}}(x) = \begin{cases} 1 & x \in A \\ 0, & x \notin A \end{cases}$$

E.g. The statement "Elizabeth is Old" can be translated as Elizabeth is a member of the set of old people and can be written symbolically as ➔

$\mu(OLD)$ ➔ where μ is the membership function that can return a value between 0.0 to 0.1 depending upon the degree of the membership.

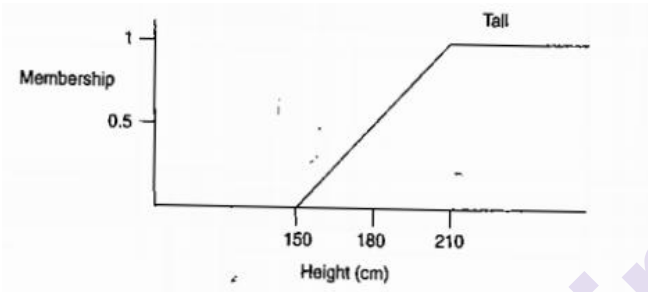


Figure 7.2: Graph showing membership functions for fuzzy set "tall".

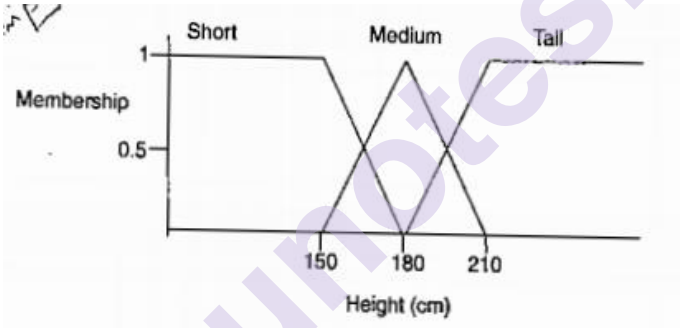


Figure 7.3: Graph showing membership functions for fuzzy set "short", "medium" and "tall".

The membership was extended to possess various "degree of membership" on the real continuous interval [0,1]. Zadeh generalized the idea of a crisp set by extending a valuation set {0,1} (definitely in, definitely out) to the interval of real values (degree of membership) between 1 & 0, denoted by [0,1]. The degree of the membership of any element of fuzzy set expresses the degree of computability of the element with a concept represented by fuzzy set.

Membership Function: A fuzzy set A contains an object x to degree a(x), that is, a(x) = Degree ($x \in A$) and the map $a: X \rightarrow \{Membership\ Degrees\}$

Possibility Distribution: The fuzzy set A can be expressed as $A = \{(x, a(x))\},\ x \in X;$ it imposes an elastic constrain of the possible values

of elements $x \in X$

Fuzzy sets tend to capture vagueness exclusively via membership functions that are mappings from a given universe of discourse X to a unit internal containing membership value. The membership function for a set maps each element of the set to membership value between 0 & 1 and uniquely describes that set. The values 0 and 1 describes "not belonging to" & "belonging to" a conventional set, respectively; values in between represent "fuzziness". Determining the membership function is subjective to varying degree depending on the situation. It depends on an individual's perception of the data in question and does not depend on randomness.

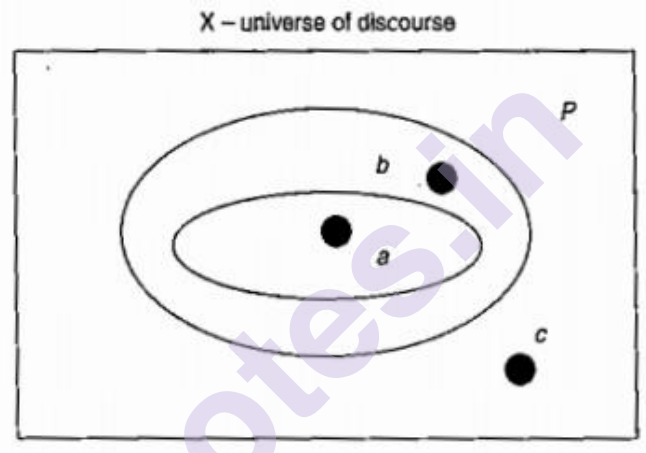


Figure 7.4: Boundary region of a Fuzzy Set

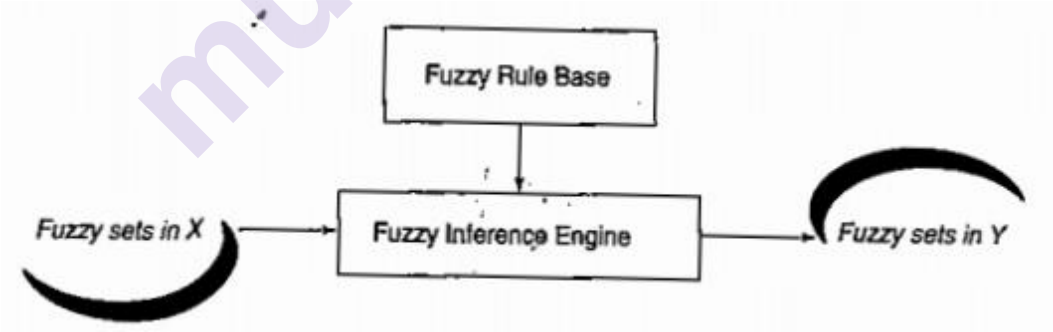


Figure 7.5: Configuration of a pure fuzzy system

Fuzzy logic also consists of fuzzy inference engine or fuzzy rule base to perform approximate reasoning somewhat similar to human brain. The fuzzy approach uses a premise that human don't represent classes of objects as fully disjoint sets but rather as sets in which there may be graded of membership intermediate between full membership and non-membership. A fuzzy set works as a concept that makes it possible to treat fuzziness in a quantitative manner. Fuzzy sets form the building

blocks for fuzzy IF-THEN rules which have general form “IF X is A THEN Y is B” where A and B are fuzzy sets.

The term “fuzzy systems” refers mostly to systems that are governed by fuzzy IF-THEN rules. The IF part of an implication is called antecedent whereas the THEN part is called consequent. The fuzzy system is a set of fuzzy rules that converts inputs to outputs.

The fuzzy inference engine (algorithm) combines fuzzy IF-THEN rules into a mapping from fuzzy sets in the input space X to the fuzzy sets in the output space Y based fuzzy logic principles. From a knowledge representation viewpoint, a fuzzy IF-THEN rule is a scheme for capturing knowledge that involves imprecision. The main features of the reasoning using these rules is its partial matching capability, which enables an inference to be made from a fuzzy rule even when the rule’s condition is partially satisfied. Fuzzy systems, on one hand is rule based system that are constructed from a collection of linguistic rules, on other hand, fuzzy systems are non-linear mappings of inputs to the outputs. The inputs and the outputs can be numbers or vectors of numbers. These rule-based systems can in theory model any system with arbitrary accuracy, i.e. they work as universal approximation.

The Achilles’ heel of a fuzzy system is it rules; smart rules gives smart systems and other rules give less smart or dumb systems. The number of rules increases exponentially with the dimension of the input space. This rule explosion is called the curse of dimensionality & is general problem for mathematical models.

7.2 Classical Sets (Crisp Sets)

Collection of objects with certain characteristics is called set. A classical set/ crisp set is defined as the collection of distinct objects. An individual entity of the set is called as element/ member of the set. The classical set is defined in such a way that the universe of discourse is splitted into two groups: members and non-members. Partial membership does not exist in the case of crisp set.

Whole set: The collection of elements in the universe

Cardinal number: Number of the elements in the set.

Set: The collections of elements within the universe

Subset: The collections of elements within the set.

7.3 Fuzzy Sets

A fuzzy set is a set having degree of membership between 0 & 1. A member of one fuzzy set can also be the member of other fuzzy set in same universe. A fuzzy set A in the universe of disclosure U can be defined as a set of ordered pairs and it is given by

$$A = \{(x, \mu_A(x) | x \in U\}$$

where

$\mu_A(x)$is the degree of membership of x in A and it indicates the degree that x belongs to A. The membership is set to unit interval [0,1] or $\mu_{A(x)} \in [0,1]$.

When the universe of disclosure is discrete & finite, fuzzy set A is given as

$$A = \left\{ \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \frac{\mu_A(x_3)}{x_3} + \cdots \right\} = \left\{ \sum_{i=1}^{n} \frac{\mu_A(x_i)}{x_i} \right\}$$

When the universe of disclosure is continuous & infinite, fuzzy set A is given as

$$A = \left\{ \int \frac{\mu_A(x)}{x} \right\}$$

Universal Fuzzy Set/ Whole Fuzzy Set: If and only if the value of the membership function is 1 for all the members under consideration. Any fuzzy set A is defined on universe U is the subset of that universe.

Empty Fuzzy Set: If and only if the value of the membership function is 0 for all the members under consideration.

Equal Fuzzy Set: two fuzzy set A & B are said to be equal fuzzy sets if $\mu_A(x) = \mu_B(x)$ for all $x \in U$

Fuzzy Power set P(U): The collection of all fuzzy sets and fuzzy subsets on universe U.

7.4 Classical Sets v/s Fuzzy Sets

7.4.1 Operations

	Classical Sets	Fuzzy Sets
Definition	The classical set is defined in such a way in that the universe of the discourse is divided into two groups: members and nonmembers. Consider Set A in Universe U: An object x is a member of a given set $a(x \in A)$ i.e. x belongs to A. An object x is a member of a given set $a(x \notin A)$ i.e. x does not belong to A.	A fuzzy set is a set having degree of membership between 0 & 1. A fuzzy set A in the universe of disclosure U can be defined as a set of ordered pairs and it is given by: $A = \{(x, \mu_A(x) \mid x \in U\}$
Union	The union between two sets gives all those elements in the universe that belong to either set A or set B or both the sets. The union is termed as logical OR operation. $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$	The union of fuzzy sets A & B is defined as: $\mu_{A \cup B}(x) = \mu_A(x) \vee \mu_{\mathbf{B}}(x) = \max\{\mu_A(x), \mu_{\mathbf{B}}(x)\}$ $for\ all\ x \in U$ V indicates max operation
Intersection	The intersection between two sets gives all those elements in the universe that belong to both set A and set B. The union is termed as logical AND operation. $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$	The intersection of fuzzy sets A & B is defined as: $\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_{\mathbf{B}}(x) = \min\{\mu_A(x), \mu_{\mathbf{B}}(x)\}$ $for\ all\ x \in U$ $\wedge$ indicates min operation
Complement	The complement of set A is defined as the collection of all elements in the universe X that do not belong to set A. $\bar{A} = \{x \mid x \notin A,\ x \in X\}$	The union of fuzzy sets A & B is defined as: $\mu_{\bar{A}}(x) = 1 - \mu_A(X) for\ all\ x \in U$

	Classical Sets	Fuzzy Sets
Difference	The difference of set A with respect to set B is the collection of all the elements in the universe that belong to A but does not belong to B. It is denoted by A|B or A-B $A\|B = \{x\|x \in A \; and \; x \notin B\}$ $A - (A \cap B)$	

7.4.2 Properties

	Classical Sets	Fuzzy Sets
Commutativity	$A \cup B = B \cup A$ $A \cap B = B \cap A$	$A \cup B = B \cup A$ $A \cap B = B \cap A$
Associativity	$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$	$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$
Distributivity	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Idempotency	$A \cup A = A$ $A \cap A = A$	$A \cup A = A$ $A \cap A = A$
Transitivity	$if \; A \subseteq B \subseteq C \; then \; A \subseteq C$	$if \; A \subseteq B \subseteq C \; then \; A \subseteq C$
Identity	$A \cup \phi = A; \; A \cap \phi = A$ $A \cup X = X; \; A \cap X = A$	$A \cup \phi = A; \; A \cap \phi = A$ $A \cup X = X; \; A \cap X = A$
Involution (double negation)	$\bar{\bar{A}} = A$	$\bar{\bar{A}} = A$
DeMorgan's Law	$\|A \cup B\| = A \cup B$ $\|A \cap B\| = A \cap B$	$\|A \cup B\| = A \cup B$ $\|A \cap B\| = A \cap B$
Law of Contradiction	$A \cap \bar{A} = \phi$	Not Followed
Law of Excluded Middle	$A \cup \bar{A} = X$	Not Followed

7.5 More Operations on Fuzzy Sets

Algebraic Sum: The algebraic sum (A+B) of two fuzzy sets A & B is defined as

$$\mu_{A\,+\,B}(x) = \mu_A(x) + \mu_{\mathbf{B}}(x) - \mu_A(x)\cdot\mu_{\mathbf{B}}(x)$$

Algebraic Product: The algebraic product (A.B) of two fuzzy sets A & B is defined as

$$\mu_{A\,.\,B}(x) = \mu_A(x)\cdot\mu_{\mathbf{B}}(x)$$

Bounded Sum: The bounded sum $(A \oplus B)$ of two fuzzy sets A & B is defined as

$$\mu_{A\,\oplus\,B}(x) = \min\{1,\ \mu_A(x) + \mu_{\mathbf{B}}(x)\}$$

Bounded Difference: The bounded difference $(A \oplus B)$ of two fuzzy sets A & B is defined as

$$\mu_{A\,\odot\,B}(x) = max\{0,\ \mu_A(x) - \mu_{\mathbf{B}}(x)\}$$

7.6 Functional Mapping of Classical Sets

Mapping is a rule of correspondence between set-theoretic forms and function theoretic forms.

X and Y are two different universe of disclosure. If an element x contained in X corresponds to an element y contained Y, it is called as mapping from X to Y; i.e. $f : X \to Y$

Let A & B be two sets on universe. The function theoretic forms of operation performed between these two sets are given as follows:

Union: $\chi_{A\,\cup\,B}(x) = \chi_A(X) \vee \chi_{\mathbf{B}}(X) = \max\{\chi_A(X), \chi_{\mathbf{B}}(X)\}$ Here $\vee$ is maximum operator.

Intersection: $\chi_{A\,\cap\,B}(x) = \chi_A(X) \wedge \chi_{\mathbf{B}}(X) = \min\{\chi_A(X), \chi_{\mathbf{B}}(X)\}$ Here $\wedge$ is minimum operator.

Complement: $\chi_{\bar{A}}(x) = 1 - \chi_A(X)$

Containment: if $A \subseteq B$, *then* $\chi_A(X) \leq \chi_{\mathbf{B}}(X)$

7.7 Introduction to Classical Relations & Fuzzy Relations

Relationship between the object are the basic concepts involved in decision making & other dynamic system application. Relations represent mapping between sets & connective logic. A classical binary relation represents the presence or absences of connection or interaction or association between the elements of two sets. Fuzzy binary relations impart degrees of strength to connections or association. In fuzzy binary relation, the degree of association is represented by membership grades in the same way as the degree of set membership is represented in fuzzy set.

When r = 2, the relation is a subset of the Cartesian product A1*A2. This relation is called a binary relation from A1 to A2. X & Y are two universe; their Cartesian product X* Y is given by $X * Y = \{(x, y) | x \in X, y \in Y\}$

Every element in X is completely related to every element in Y. The characteristic function, denoted by χ, gives the strength of the relationship between ordered pair of elements in each universe. The characteristic function, denoted by χ, gives the strength of the relationship between ordered pair of elements in each universe.

$$\chi_{X * Y}(x, y) = \begin{cases} 1, & (x, y) \in X * Y \\ 0, & (x, y) \notin X * Y \end{cases}$$

A binary relation in which each element from the first set X is not mapped to more than one element in second set Y is called a **function** and is expressed as $R: X \rightarrow Y$

A **fuzzy relation** is a fuzzy set defined on the Cartesian product of classical set $\{X_1, X_2, X_3, \ldots X_n\}$ where tuples $(x_1, x_2, \ldots, x_n)$ may have varying degree of membership $\mu R(x_1, x_2, \ldots, xn)$ within the relation

$$R(X_1, X_2, \ldots., Xn) = \int_{X_1 * X_2 * \ldots XN} \mu R(x_1, x_2, \ldots, xn) | \ (x_1, x_2, \ldots, xn), \text{xi} \in Xi$$

A fuzzy relation between two sets X & Y is called **binary fuzzy relation** & is denoted by R(X,Y). A binary relation R(X,Y) is referred to as **bipartite graph** when X≠Y.A binary relation on a single set X is called digraph or directed graph.This relation occur when X=Y and is denoted as R(X,X)or R(X2).The matrix representing a fuzzy relation is called fuzzy matrix.A fuzzy relation R is a mapping from Cartesian product space X *Y to interval [0,1]where the mapping strength is expressed by the membership function of the relation for ordered pairs from the two universe [μR(x,y)]

Let

$$\underset{\sim}{X} = \{x_1, x_2, \ldots, x_n\} \quad \text{and} \quad \underset{\sim}{Y} = \{y_1, y_2, \ldots, y_m\}$$

Fuzzy relation $\underset{\sim}{R}(\underset{\sim}{X}, \underset{\sim}{Y})$ can be expressed by an $n \times m$ matrix as follows:

$$\underset{\sim}{R}(\underset{\sim}{X}, \underset{\sim}{Y}) = \begin{bmatrix} \mu_R(x_1, y_1) & \mu_R(x_1, y_2) & . & . & \mu_R(x_1, y_m) \\ \mu_R(x_2, y_1) & \mu_R(x_2, y_2) & . & . & \mu_R(x_2, y_m) \\ . & . & . & . & . \\ . & . & . & . & . \\ \mu_R(x_n, y_1) & \mu_R(x_n, y_2) & . & . & \mu_R(x_n, y_m) \end{bmatrix}$$

A **fuzzy graph** is a graphical representation of a binary fuzzy relation. Each element in X & Y corresponds to a node in the fuzzy graph. The connection links are established between the nodes by the elements of X*Y with nonzero membership grades in R(X,Y). The links may also be present in the forms of arcs. This links are labelled with membership value as $[\mu R(x, y)]$. When $X \neq Y$, the link connecting the two nodes is an undirected binary graph called as **bi** Y, a node is connected to itself and directed links are used; in such case, the fuzzy graph is called **directed graph**. Here, only one set off nodes corresponding to set X is used.

The domain of binary fuzzy relation R(X,Y) is the fuzzy set, dom R(X,Y) having the membership function as:

$$\mu_{\text{domain } R}(x) = \max_{y \in Y} \mu_R(x, y) \quad \forall x \in X$$

The range of binary fuzzy relation R(X,Y) is the fuzzy set, ran R(X,Y) having the membership function as:

$$\mu_{\text{range } R}(y) = \max_{x \in X} \mu_R(x, y) \quad \forall y \in Y$$

7.8 Cartesian Product of the Relation

An **ordered r-tuple** is and ordered sequence of r-elements expressed in the form (a1, a2, a3 … ar).

An **unordered r-tuple** is a collection of r-elements without any restriction in order.

For r = 2, the r-tuple is called an **ordered pair**.

For crisp sets $A_1, A_2, A_3, \ldots A_r$, the set of all r-tuples $(a_1, a_2, a_3, \ldots a_r)$ where $a_1 \in A_1, a_2 \in A_2, \ldots, a_r \in A_r$ is called **Cartesian product** of $A_1, A_2, A_3, \ldots A_r$ and is denoted by $A_1 * A_2 * A_3 * \ldots * A_r$.

If all the a_r's are identical and equal to A, then the Cartesian product $A_1 * A_2 * A_3 * \ldots * A_r$ is denoted as A^r

7.9 Classical Relation v/s Fuzzy Relations

7.9.1 Cardinality

Classical Relations	Fuzzy Relations
Cardinality:	
Consider n elements of universe X being related to the m elements of universe Y. When the cardinality of X= ^{n}X & the cardinality of Y = ^{n}Y, then the cardinality of relation R between the two universe is $n_{X*Y} = n_X * n_Y$ The cardinality of the power set P(X *Y) describing the relation is given by $^{n}P(X * Y) = 2^{(n_X n_Y)}$	The cardinality of fuzzy sets on any universe is infinity; hence the cardinality of a fuzzy relation between two or more universe is also infinity.

7.9.2 Operations

Let R & S be two separate relations on the Cartesian universe X * Y. The null relation and the complete relation are defined by the relation matrices ϕR *and* E_R.

$$\phi_R = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad E_R = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Operations	Classical Relations	Fuzzy Relations
Union	$R \cup S \rightarrow \chi R \cup S(x,y)$ $= \max [\chi R(x,y), \chi S(x,y)]$	$\mu_{R \cup S}(x,y)$ $= \max [\mu_R(x,y), \mu_S(x,y)]$
Intersection	$R \cap S \rightarrow \chi R \cap S(x,y) =$ $min[\chi_R(x,y), \chi_S(x,y)]$	$\mu_{R \cap S}(x,y)$ $= min [\mu_R(x,y), \mu_S(x,y)]$
Complement	$R \rightarrow \chi_R(x,y)$: $\chi_R(x,y)$ $= 1 - \chi_R(x,y)$	$\mu_R(x,y) = 1 - \mu_R(x,y)$
Containment	$R \subset S$ $\rightarrow \chi_R(x,y) : \chi_R(x,y)$ $\leq \chi_S(x,y)$	$R \subset S \Longrightarrow \mu_R(x,y) \leq \mu_S(x,y)$
Identity	$\phi \rightarrow \phi R \quad \& \quad X \rightarrow E_R$	
Inverse		The inverse of fuzzy relation R on X*Y is denoted by R^{-1}. It is relation on Y*X defined by $R^{-1}(y,x) = R(x,y)$ for all pairs $(y,x) \in Y * X$
Projection		For fuzzy relation R(X,Y), let $[R \downarrow Y]$ denote the projection of R onto Y. $\overline{\mu_{[R \downarrow Y]}}(x,y) = \max_{y} \mu_R(x,y)$

7.9.3 Properties

Classical Relations	Fuzzy Relations
Properties	
• Commutativity • Associativity • Distributivity • Involution • Idempotency • DeMorgan's Law • Excluded middle law	• Commutativity • Associativity • Distributivity • Involution • Idempotency • DeMorgan's Law

7.10 Classical Composition and Fuzzy Composition

The operation executed on two binary relations to get a single binary relation is called **composition**.

Let R be a relation that maps elements from universe X to universe Y and S be a relation that maps elements from universe Y to universe Z. The two binary elements R & S are compatible if R ⊆X*Y & S⊆Y*Z.The composition between the two relations is denoted by R∘S.

Consider the universal sets given by:
$X = \{a1, a2, a3\}; Y = \{b1, b2, b3\};\ Z = \{c1, c2, c3\}$

Let the relation R & S be formed as:
$R = X * Y = \{(a1, b1), (a1, b2), (a2, b2), (a3, b3)\}$
$S = Y * Z = \{(b1, c1), (b2, c3),\ (b3, c2)\}$

It can be inferred that:

$$T = R \circ S = \{(a1, c1), (a2, c3), (a3, c2), (a1, c3)\}$$

The composition operations are of two types

1. Max-Min Composition: $T = R \circ S$

$$\chi_T(x, z) = \bigvee_{y \in Y} [\chi_R(x, y) \wedge \chi_S(y, z)]$$

2. Max-product Composition: $T = R \circ S$

$$\chi_T(x, z) = \bigvee_{y \in Y} [\chi_R(x, y) \cdot \chi_S(y, z)]$$

Let A be fuzzy set on universe X & B be fuzzy set on universe Y. The Cartesian product over A and B results in fuzzy relation B and is contained within the entire (complete) Cartesian space $A * B = R\ where\ R \subset X * Y$

The membership function of fuzzy relation is given by $\mu_R(x, y) = \mu_{A\ *\ B}(x, y) =$ min $[\mu_A(x),\ \mu_B(y)]$

For e.g., for a fuzzy set A that has three elements and a fuzzy set B has four elements, the resulting fuzzy relation R will be represented by a matrix size 3 * 4

There are two types of fuzzy composition techniques:

1. Fuzzy Max-min composition
2. Fuzzy Max-product composition

Let R be fuzzy relation on Cartesian space X*Y and S be fuzzy relation on Cartesian Space Y*Z.

Fuzzy Max-min composition:

The max-min composition of R(X,Y) and S(Y,Z) is denoted by $R(X,Y)°S(Y,Z)$ is defined by T(X,Z) as

$$\mu_T(x,z) = \mu_{R \circ S}(x,z) = \min_{y \in Y} \{\max[\mu_R(x,y), \mu_S(y,z)]\} = \bigwedge_{y \in Y} [\mu_R(x,y) \vee \mu_S(y,z)] \quad \forall x \in X, z \in Z$$

Fuzzy Max-product composition:

$$\mu_T(x,z) = \mu_{R \cdot S}(x,z) = \max_{y \in Y} [\mu_R(x,y) \cdot \mu_S(y,z)]$$

$$= \bigvee_{y \in Y} [\mu_R(x,y) \cdot \mu_S(y,z)]$$

7.10.1 Properties

	Classical Composition	Fuzzy Composition
Associative	$(R°S)°M = R°(S°M)$	$(R°S)°M = R°(S°M)$
Commutative	$R°S \neq S°R$	$R°S \neq S°R$
Inverse	$(R°S)^{-1} = S^{-1}°R^{-1}$	$(R°S)^{-1} = S^{-1}°R^{-1}$

7.10.2 Equivalence

	Classical Composition	Fuzzy Composition
Reflexivity	$\chi R(xi,xi) = 1 \; or \; (xi,xi) \in R$	$\mu R(xi,xi) = 1 \; \forall x \in X$
Symmetry	$\chi R(xi,xj) = \chi R(xj,xi)(xi,xj) \in R \Longrightarrow (xj,xi) \in R$	$\mu R(xi,xj) = \mu R(xj,xi) \; \forall xi,xj \in X$
Transitivity	$\chi R(xi,xj) \; and \; \chi R(xj,xk) = 1, \; so \; \chi R(xi,xk) = 1(xi,xj) \in R(xj,xk) \in R, \; so \; (xi,xk) \in R$	$\mu R(xi,xj) = \lambda_1 \; and \mu R(xj,xk) = \lambda_2 \Longrightarrow \mu R(xi,xk) = \lambda \; where \; \lambda = \min(\lambda_1, \lambda_2)$

Fuzzy Max-product transitive can be defined. It is given by

$$\mu_R(x_i, x_k) \geq \max_{x_j \in X} \cdot [\mu_R(x_i, x_j) * \mu_R(x_j, x_k)] \quad \forall (x_i, x_k) \in X^2$$

7.10.3 Tolerance

Classical Composition	Fuzzy Composition
A tolerance relation R1 on universe X is one where the only the properties of reflexivity & symmetry are satisfied.	A binary fuzzy relation that possesses the properties of reflexivity and symmetry is called fuzzy tolerance relation or resemblance relation.
The tolerance relation can also be called proximity relation.	The equivalence relations are a special case of the tolerance relation.
An equivalence relation can be formed from tolerance relation R1 by (n-1) compositions with itself, where n is the cardinality of the set that defines R1, here it is X	The fuzzy tolerance relation can be reformed into fuzzy equivalence relation in the same way as a crisp tolerance relation is reformed into crisp equivalence relation
$\underbrace{R_1^{n-1}}_{\text{Tolerance relation}} = R_1 \circ R_1 \circ \cdots \circ R_1 = \underbrace{R}_{\text{Equivalence relation}}$	$\underbrace{\tilde{R}_1^{n-1}}_{\text{Fuzzy tolerance relation}} = \tilde{R}_1 \circ \tilde{R}_1 \circ \cdots \circ \tilde{R}_1 = \underbrace{\tilde{R}}_{\text{Fuzzy equivalence relation}}$

7.11 Non-Interactive Fuzzy Set

The independent events in probability theory are analogous to noninteractive fuzzy sets in fuzzy theory. We are defining fuzzy set A on the Cartesian space X= X_1 x X_2. Set A is separable into two noninteractive fuzzy sets called orthogonal projections if and only if

$$\tilde{A} = \text{OPr}_{X_1}(\tilde{A}) \times \text{OPr}_{X_2}(\tilde{A})$$

where

$$\mu_{\text{OPr}_{X_1}(\tilde{A})}(x_1) = \max_{x_2 \in X_2} \mu_{\tilde{A}}(x_1, x_2) \quad \forall x_1 \in X_1$$

$$\mu_{\text{OPr}_{X_2}(\tilde{A})}(x_2) = \max_{x_1 \in X_1} \mu_{\tilde{A}}(x_1, x_2) \quad \forall x_2 \in X_2$$

The equations represent membership functions for the orthographic projections of A on universes X_1 and X_2. respectively.

Summary

In this chapter, we have discussed the basic definitions, properties and operations on classical sets and fuzzy sets. Fuzzy sets are tools that convert the concept of fuzzy logic into algorithms. Since fuzzy sets allow partial membership, they provide computer with such algorithms that extend binary logic and enable it to take human-like decisions. In other words, fuzzy sets can be thought of as a media through which the human thinking is transferred to a computer. One difference between fuzzy sets and classical sets is that the former does not follow the law of excluded middle and law of contradiction.

The relation concept is used for nonlinear simulation, classification, and control. The description on composition of relations gives a view of extending fuzziness into functions. Tolerance and equivalence relations are helpful for solving similar classification problems. The noninteractivity between fuzzy sets is analogous to the assumption of independence in probability modelling.

Review Questions

1. Explain fuzzy logic in detail.
2. Compare Classical set and fuzzy set.
3. Enlist and explain any three classicals set operations.
4. Enlist and explain any three fuzzy sets operations.
5. Enlist and explain any three classical set properties.
6. Enlist and explain any three fuzzy sets properties.
7. Write a short note on fuzzy relation.
8. Compare classical relations and fuzzy relations.
9. Write a short note classical composition and fuzzy composition.

Bibliography, References and Further Reading

- Artificial Intelligence and Soft Computing, by Anandita Das Battacharya, SPD 3rd, 2018
- Principles of Soft Computing, S.N. Sivanandam, S.N.Deepa, Wiley, 3rd , 2019
- Neuro-fuzzy and soft computing, J.S.R. Jang, C.T.Sun and E.Mizutani, Prentice Hall of India, 2004

❖❖❖

Unit IV

8

MEMBERSHIP FUNCTIONS, DEFUZZIFICATION, FUZZY ARITHMETIC AND FUZZY MEASURES

Unit Structure

8.0 Objectives

This chapter begins with explaining the membership function and later introduces the concept of fuzzification, defuzzification and fuzzy arithmetic.

8.1 Introduction to Membership Function

Membership function defines fuzziness in a fuzzy set irrespective of the elements in the discrete or continuous. The membership functions are generally represented in graphical form. There exist certain limitations for the shapes used in graphical form of membership function. The rules that describes fuzziness graphically are also fuzzy. Membership can be thought of as a technique to solve empirical problems on the basis of experience rather than knowledge.

8.2 Features of the Membership Function

The membership function defines all the information contained in a fuzzy set. A fuzzy set A in the universe of discourse X can be defined as a set of ordered pairs: $A=\{(x,\mu A(x)) \mid x \in X\}$ where $\mu A(.)$ is called membership function of A. The membership function $\mu A(.)$ maps X to the membership space M,i.e.$\mu A : X \rightarrow M$.

The membership value ranges in the interval [0,1]. Main features involved in characterizing membership function are:

- **Core**: The core of a membership function for some fuzzy set A is defined as that region of universe that is characterized by complete membership in the set A. The core has elements x of the universe such that $\mu A(x) = 1$. The core of a fuzzy set may be an empty set.

- **Support**: The support of a membership function for a fuzzy set A is defined as that region of universe that is characterized by a nonzero membership. The support comprises elements x of the universe such that $\mu A(x) > 0$. A fuzzy set whose support is a single element in X with $\mu A(x) = 1$ is referred to as a fuzzy singleton.

- **Boundary**: The support of a membership function for a fuzzy set A is defined as that region of universe containing elements that have nonzero but not complete membership. The boundary comprises of those elements of x of the universe such that $0 < \mu A(x) < 1$. The boundary elements are those which possess partial membership in fuzzy set A.

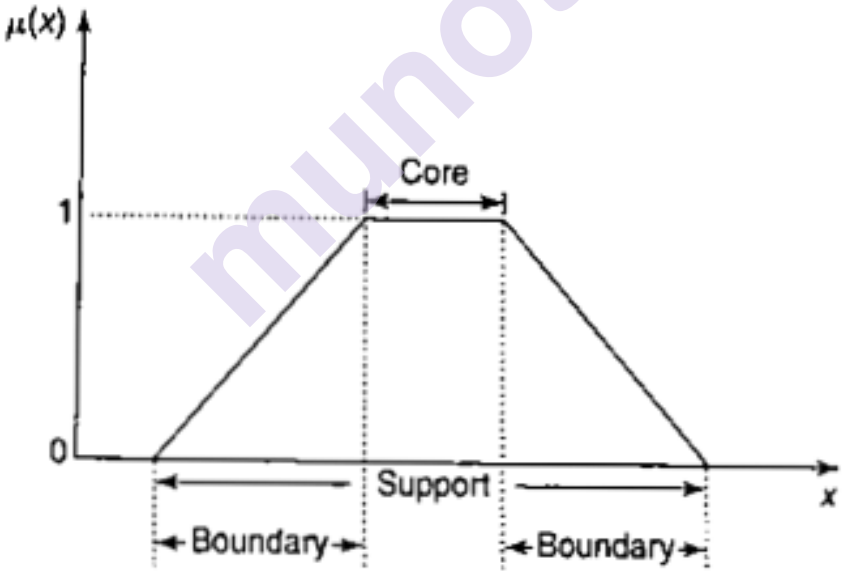


Figure 8.1: Properties of Membership Functions

Other types of Fuzzy Sets

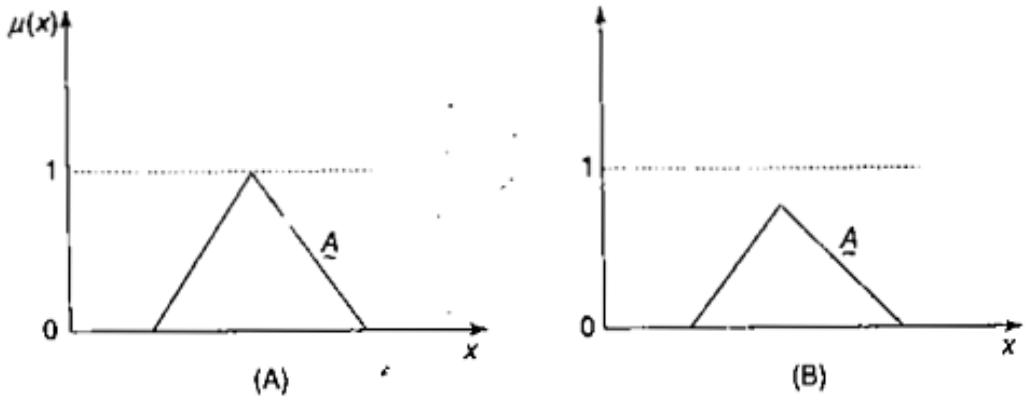


Figure 8.2: (A) Normal Fuzzy Set and (B) Subnormal Fuzzy Set

- **Normal fuzzy set**: A fuzzy set whose membership function has at least one element x in the universe whose membership value is unity.
    - **Prototypical element**: The element for which the membership is equal to 1.
- **Subnormal fuzzy set**: A fuzzy set wherein no membership function has it equal to 1.
- **Convex fuzzy set:** A convex fuzzy set has membership function whose membership values are strictly monotonically increasing or strictly monotonically decreasing or strictly monotonically increasing than strictly monotonically decreasing with increasing values for the elements in the universe.
- **Nonconvex fuzzy set:** the membership value of the membership function is not strictly monotonically increasing or decreasing or strictly monotonically increasing than decreasing.

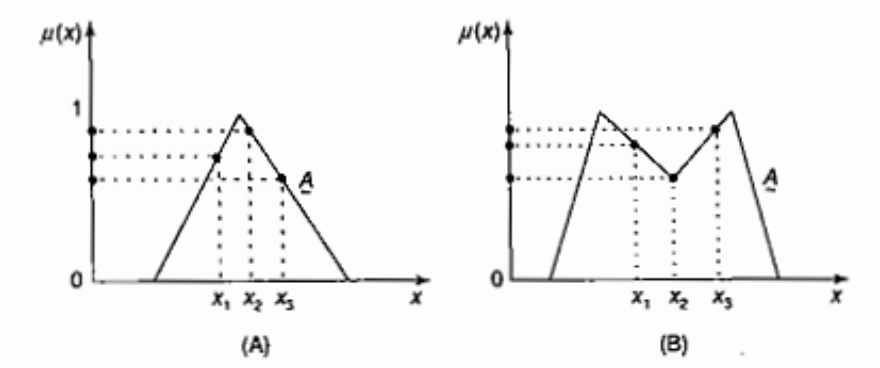


Figure 8.3: (A) Convex Normal Fuzzy Set and (B) Nonconvex Normal Fuzzy Set

The intersection of two convex fuzzy set is also a convex fuzzy set. The element in the universe for which a particular fuzzy set A has its value equal to 0.5 is called **crossover point** of membership function. There can be more than one crossover point in fuzzy set. The maximum value of the membership function of the fuzzy set A is called **height of the fuzzy set**. If the height of the fuzzy set is less than 1, then the fuzzy set is called **subnormal fuzzy set**. When the fuzzy set A is a convex single –point normal fuzzy set defined on the real time, then A is termed as a **fuzzy number**.

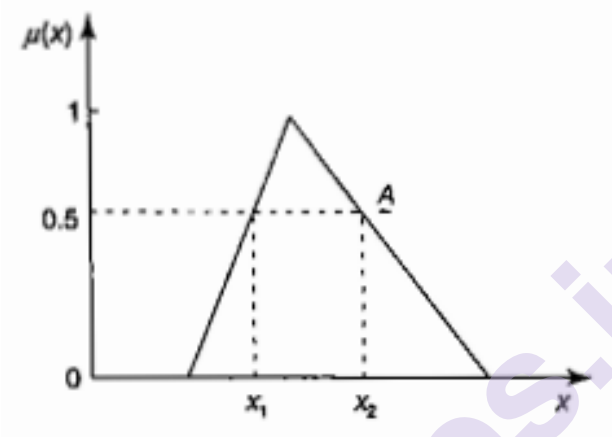


Figure 8.4: Crossover Point of a Fuzzy Set

8.3 Overview of Fuzzification

Fuzzification is the process of transforming a crisp set to a fuzzy set or a fuzzy set into a fuzzier set. This operation translates accurate crisp input value into linguistic variables. Quantities that we consider to be accurate, crisp & deterministic, possess uncertainty within themselves. The uncertainty arises due to vagueness, imprecision or uncertainty.

For a fuzzy set A={μi/xi|xi∈X},a common fuzzification algorithm is performed by keeping μi constant and xi being transformed to a fuzzy set Q(xi) depicting the expression about xi. The fuzzy set Q(xi) is referred to as the kernel of fuzzification.

The fuzzified set A can be expressed as:

$$\tilde{A} = \mu_1 Q(x_1) + \mu_2 Q(x_2) + \cdots + \mu_n Q(x_n)$$

where the symbol ~ means fuzzified. This process of fuzzification is called **support fuzzification (s-fuzzification)**.

Grade fuzzification (g-fuzzification) is another method where xi is kept constant and μi is expressed as a fuzzy set.

8.4 Methods of Membership Value Assignment

Following are the methods for assigning membership value:

- Intuition
- Inference
- Rank ordering
- Angular fuzzy sets
- Neural Network
- Genetic Algorithm
- Inductive Reasoning

8.4.1 Intuition

Intuition method is the base upon the common intelligence of human. It is capacity of the human to develop membership functions on the basis of their own intelligence and understanding capability. There should be an in-depth knowledge of the application to which membership value assignment has to be made.

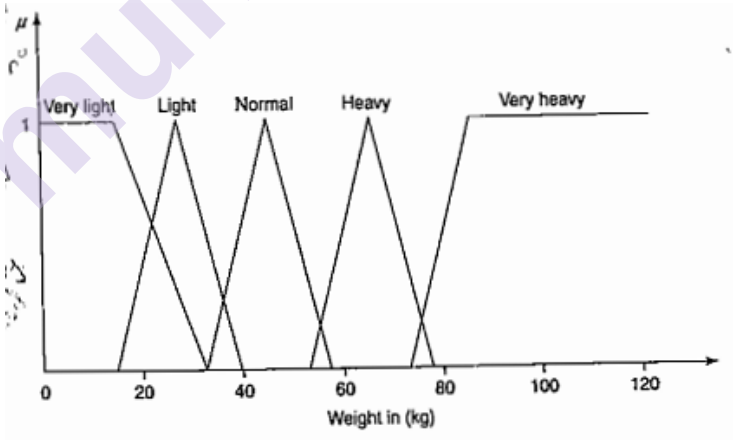


Figure 8.5: Membership functions for the Fuzzy variable "weight"

8.4.2 Inference & Rank Ordering

The inference method uses knowledge to perform deductive reasoning. Deduction achieves conclusion by means of forward inference.

Rank ordering is carried on the basis of the preferences. Pairwise comparisons enable us to determine preferences & resulting in determining the order of membership.

8.4.3 Angular Fuzzy Sets

Angular fuzzy set 's' is defined on a universe of angles, thus repeating the shapes every 2π cycles. The truth value of the linguistic variable is represented by angular fuzzy sets. The logical proposition is equated to the membership value "1" is said to be "true" and that preposition with membership value 0 is said to be "false". The intermediate values between 0 & 1 correspond to proposition being partially true or partially false.

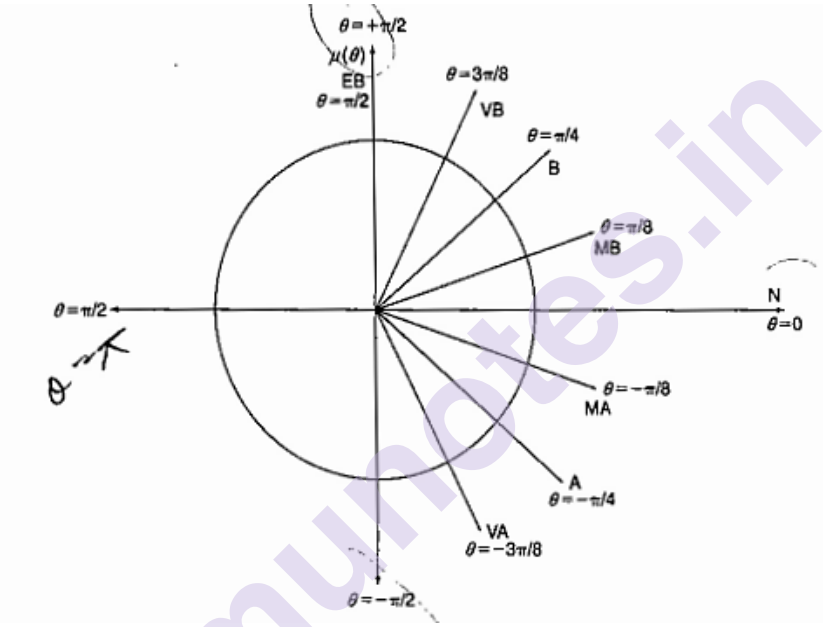


Figure 8.6: Model of Angular Fuzzy Set

The values of the linguistic variable vary with "θ" & their membership values are on the μ(θ)axis.The membership value corresponding to the linguistic term can be obtained from equation μt(θ)=t.tan (θ) where t is the horizontal projection of radial vector

8.4.4 Neural Network

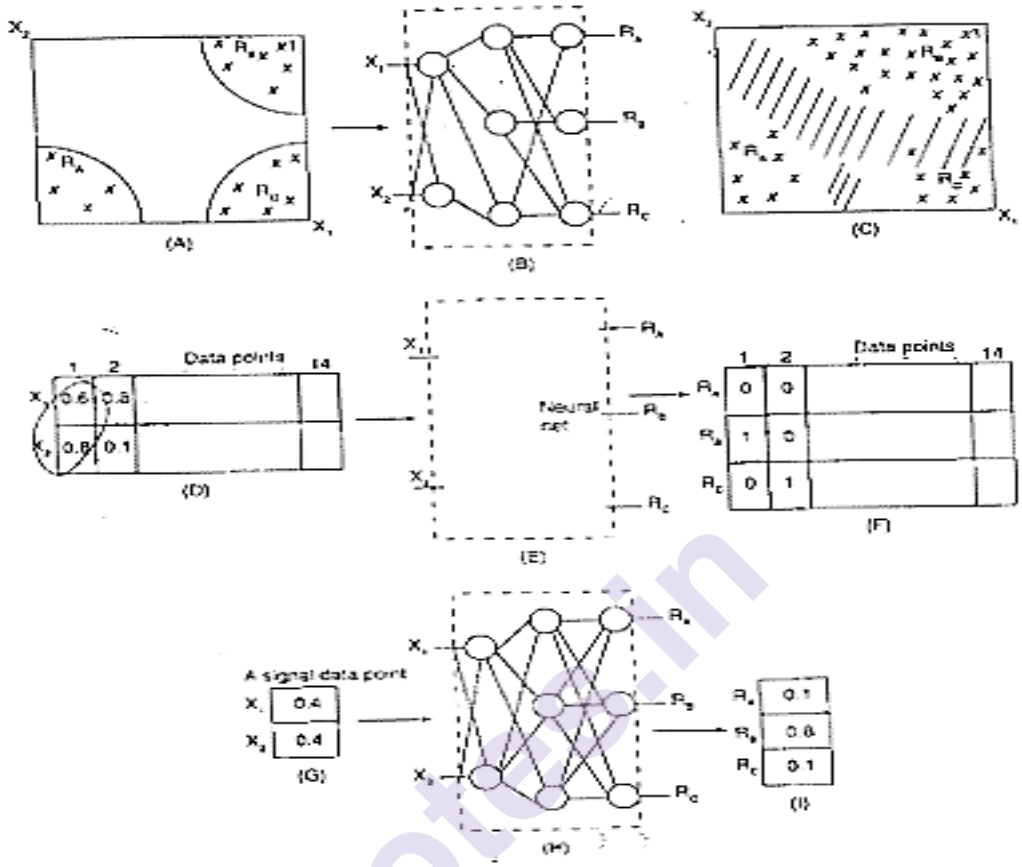


Figure 8.7: Fuzzy Membership function evaluated from Neural Networks

8.4.5 Genetic Algorithm

Genetic algorithm is based on the Darwin's theory of evolution, the basic rule is "survival of the fittest". Genetic algorithms use the following steps to determine the fuzzy membership function:

- For a particular functional mapping system, the same membership functions & shapes are assumed for various fuzzy variables to be defined.
- These chosen membership functions are then coded into bit strings.
- Then these bit strings are concatenated together
- The fitness function to be used here is noted. In genetic algorithm, fitness function plays a major role similar to that played by activation function in neural network.
- The fitness function is used to evaluate the fitness of each set of membership function.
- These membership functions define the functional mapping of the system

8.4.6 Inductive Reasoning

Induction is used to deduce causes by means of backward inference. The characteristics of inductive reasoning can be used to generate membership functions. Induction employs entropy minimization principles, which clusters the parameters corresponding to the output classes. To perform inductive reasoning method, a well-defined database for the input-output relationship exist. Induction reasoning can be applied for complex systems where database is abundant & static.

Laws of Induction:

- Given a set of irreducible outcomes of experiment, the induced probabilities are probability consistent with all the available information that maximize the entropy of the set.
- The induced probability of a set of independent observation is proportional to the probability density of the induced probability of single observation.
- The induced rule is that rule consistent with all available information of that minimizes the entropy

The third law stated above is widely used for development of membership function.

The membership functions using inductive reasoning are generated as follow:

- A fuzzy threshold is to be established between classes of data.
- Using entropy minimization screening method, first determine the threshold line
- Then start the segmentation process
- The segmentation process results into two classes.
- Again, partitioning the first two classes one more time, we obtain three different classes.
- The partitioning is repeated with threshold value calculation, which lead us to partition the data set into a number of classes and fuzzy set.
- Then on the basis of shape, membership function is determined.

8.5 Overview of Defuzzification

Defuzzification is mapping process from a space of fuzzy control actions defined over an output universe of discourse into space of crisp control action. A defuzzification process produces a nonfuzzy control action that represents the

possibility distribution of an inferred fuzzy control action. Defuzzification process has the capability to reduce a fuzzy set into a crisp single-valued quantity or into a crisp set; to convert a fuzzy matrix into a crisp matrix; or to convert a fuzzy number into a crisp number. Mathematically, the defuzzification process may also termed as "rounding off". Fuzzy set with a collection of membership values or a vector of values on the unit interval may be reduced to a single scalar quantity using defuzzification process.

8.6 Concept of Lamba-Cuts for Fuzzy Sets (Alpha-Cuts)

Consider a fuzzy set A. The set A_λ $(0 < \lambda < 1)$, called the lamba $(\lambda) -$ cut (or alpha $[\alpha]$-cut) set, is a crisp

set of the fuzzy set & defined as:

$$A_\lambda = \{x | \mu_A(x) \geq \lambda\};\ \lambda \in [0,1]$$

The set A_λ is called a weak lambda-cut set if it consists of all the elements of fuzzy set whose

membership functions have values greater than or equal to specified value.

The set A_λ is called a strong lambda-cut set if it consists of all the elements of fuzzy set whose

membership functions have values strictly greater than specified value.

$$A_\lambda = \{x | \mu_A(x) > \lambda\};\ \lambda \in [0,1]$$

The properties of λ-cut sets are as follows:

1. $(\underset{\sim}{A} \cup \underset{\sim}{B})_\lambda = A_\lambda \cup B_\lambda$
2. $(\underset{\sim}{A} \cap \underset{\sim}{B})_\lambda = A_\lambda \cap B_\lambda$
3. $(\bar{\underset{\sim}{A}})_\lambda \neq (\bar{A}_\lambda)$ except when $\lambda = 0.5$
4. For any $\lambda \leq \beta$, where $0 \leq \beta \leq 1$, it is true that $A_\beta \subseteq A_\lambda$, where $A_0 = X$.

8.7 Concept of Lamba-Cuts for Fuzzy Relations

Let $\underset{\sim}{R}$ be a fuzzy relation where each row of the relational matrix is considered a fuzzy set. The jth row in a fuzzy relation matrix $\underset{\sim}{R}$ denotes a discrete membership function for a fuzzy set $\underset{\sim}{R}_j$. A fuzzy relation can be converted into a crisp relation in the following manner:

$$R_\lambda = \{(x, y) | \mu_{\underset{\sim}{R}}(x, y) \geq \lambda\}$$

where R_λ is a λ-cut relation of the fuzzy relation $\underset{\sim}{R}$.

For two fuzzy relations $\underset{\sim}{R}$ and $\underset{\sim}{S}$ the following properties should hold:

1. $(\underset{\sim}{R} \cup \underset{\sim}{S})_\lambda = R_\lambda \cup S_\lambda$
2. $(\underset{\sim}{R} \cap \underset{\sim}{S})_\lambda = R_\lambda \cap S_\lambda$
3. $(\bar{\underset{\sim}{R}})_\lambda \neq (\bar{R}_\lambda)$ except when $\lambda = 0.5$
4. For any $\lambda \leq \beta$, where $0 \leq \beta \leq 1$, it is true that $R_\beta \subseteq R_\lambda$.

8.8 Methods of Defuzzification

Defuzzification is the process of conversion of a fuzzy quantity into a precise quantity. The output of a fuzzy process may be union of two or more fuzzy membership functions defined on the universe of discourse of the output variable.

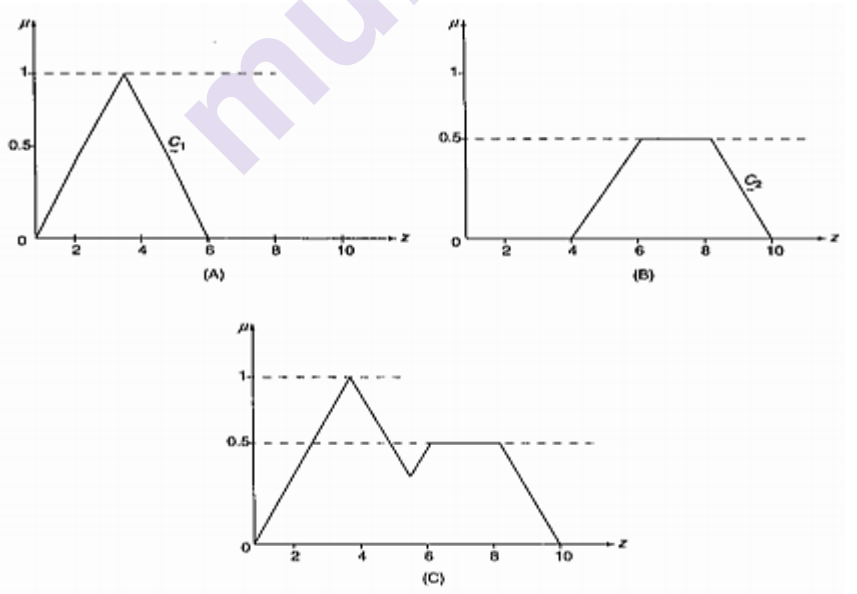


Figure 8.8 (A): First part of fuzzy output, (B) second part of fuzzy output, (C) union of parts (A) and (B)

Defuzzification Methods

- Max-membership principle
- Centroid method
- Weighted average method
- Mean-Max membership
- Centers of Sums
- Center of largest area
- First of maxima, last of maxima

8.8.1 Max-membership Principle

This method is also known as height method and is limited to peak output functions. This method is given by the algebraic expression:

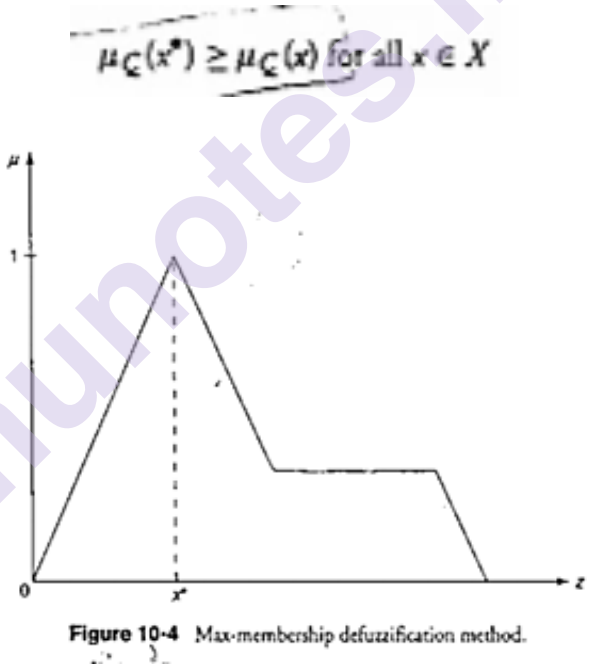


Figure 10-4 Max-membership defuzzification method.

Figure 8.9: Max-membership Defuzzification Method

8.8.2 Centroid Method

This method is also known as center of mass, center of area, center of gravity,

$$x^* = \frac{\int \mu_{\tilde{C}}(x) \cdot x dx}{\int \mu_{\tilde{C}}(x) dx}$$

$\int$ is denotes an algebraic integration.

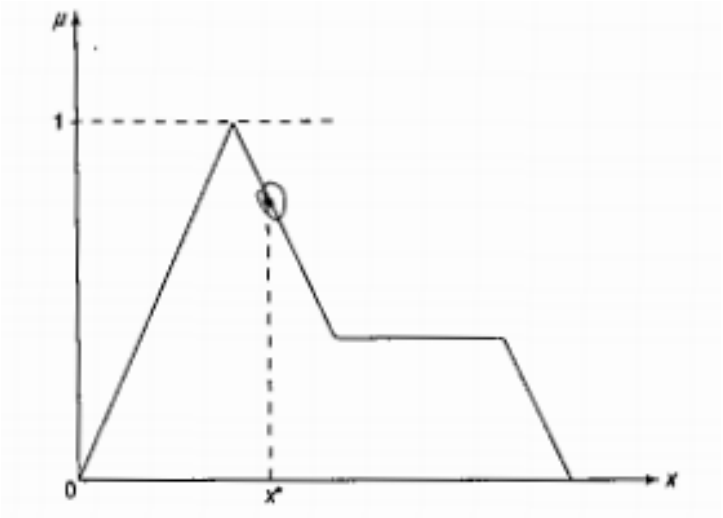


Figure 8.10: Centroid Defuzzification Method

8.8.3 Weighted Average Method

This method is valid for symmetrical output membership functions only. Each membership function is weighted by its maximum membership value.

$$x^* = \frac{\sum \mu_C(\bar{x}_i) \cdot \bar{x}_i}{\sum \mu_C(\bar{x}_i)}$$

$\sum$ denotes algebraic sum and xi is the maximum of the i^{th} membership function.

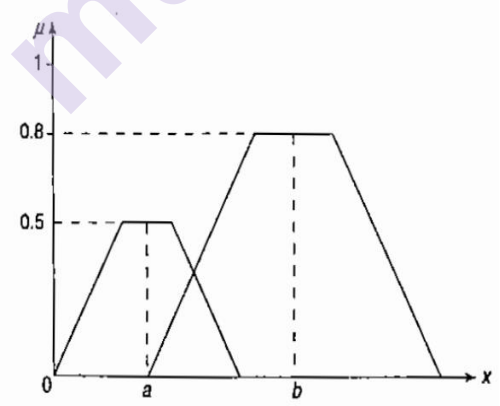


Figure 8.11: Weighted average defuzzification method (two symmetrical membership functions)

8.8.4 Mean-Max Membership

This method is also known as the middle of maxima. The locations of the maxima membership can be nonunique.

$$x^* = \frac{a+b}{2}$$

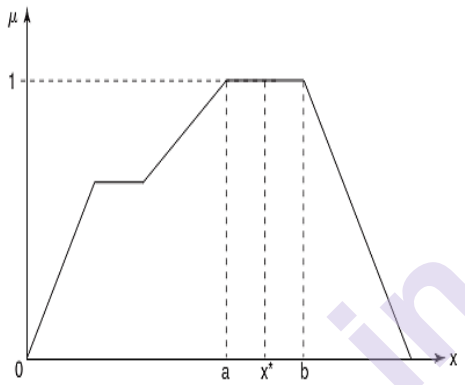


Figure 8.12: Mean-max membership defuzzification method

8.8.5 Centers of Sums

This method employs the algebraic sum of the individual fuzzy subsets. Advantage: Fast calculation. Drawback: intersecting areas are added twice. The defuzzified value x* is given by:

$$x^* = \frac{\int_x x \sum_{i=1}^{n} \mu_{C_i}(x)dx}{\int_x \sum_{i=1}^{n} \mu_{C_i}(x)dx}$$

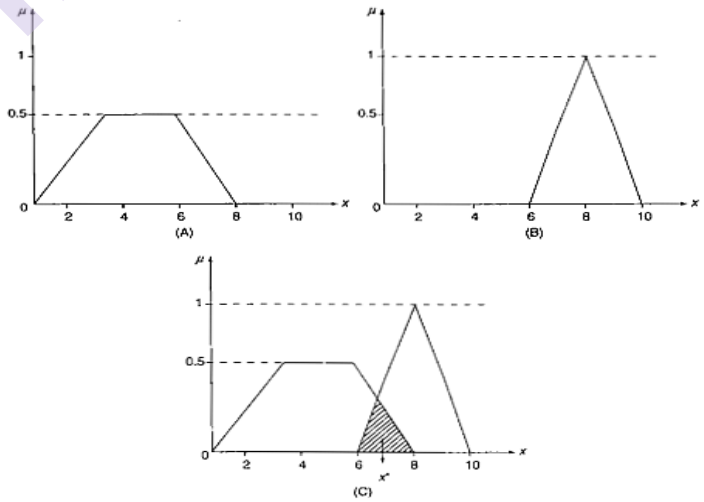


Figure 10-8 (A) First and (B) second membership functions, (C) defuzzification.

Figure 8.13: (A) First and (B) Second Membership functions, (C) Defuzzification

8.8.6 Centers of Largest Area

This method can be adopted when the output consists of at least two convex fuzzy subsets which are not overlapping. The output in this case is biased towards a side of one membership function. When output fuzzy set has at least two convex regions then the center-of-gravity of the convex fuzzy sub region having the largest area is used to obtain the defuzzified value x*. This value is given by:

$$x_* = \frac{\int \mu^{cr}(x) dx}{\int \mu^{cr}(x) \cdot x dx}$$

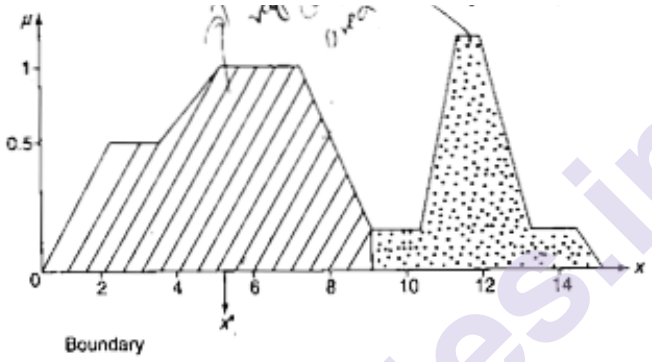


Figure 8.14: Center of Largest Area Method

8.8.7 First of Maxima, Last of Maxima

This method uses the overall output or union of all individual output fuzzy sets c_j for determining the smallest value of the domain with the maximized membership in c_j.

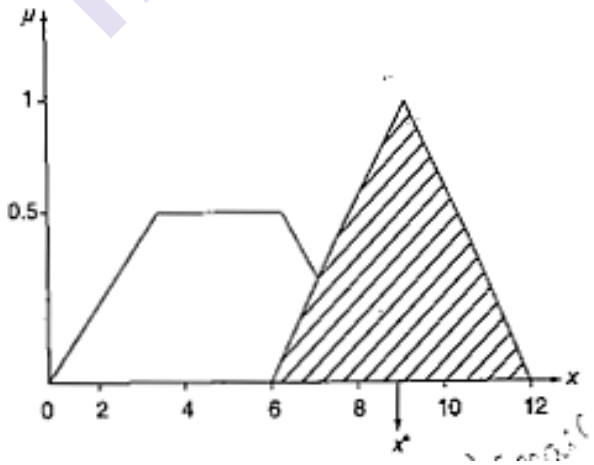


Figure 8.15: First of maxima (last of maxima) method

The steps used for obtaining x* are:

- Initially, the maximum height in the union is found

$$\text{hgt}(\mathcal{C}_i) = \sup_{x \in X} \mu_{\mathcal{C}_i}(x)$$

where sup is supremum, i.e., the least upper bound

- Then the first of maxima is found:

$$x^* = \inf_{x \in X} \left\{ x \in X \middle| \mu_{\mathcal{C}_i}(x) = \text{hgt}(\mathcal{C}_i) \right\}$$

where inf is the infimum, i.e. the greatest lower bound.

- After this the last of maxima is found:

$$x^* = \sup_{x \in X} \left\{ x \in X \middle| \mu_{\mathcal{C}_i}(x) = \text{hgt}(\mathcal{C}_i) \right\}$$

8.9 Overview of Fuzzy Arithmetic

Fuzzy arithmetic is based on the operations and computations of fuzzy numbers. Fuzzy numbers help in expressing fuzzy cardinalities and fuzzy quantifiers. Fuzzy arithmetic is applied in various engineering applications when only imprecise or uncertain sensory data are available for computation. The imprecise data from the measuring instruments are generally expressed in the form of intervals, and suitable mathematical operations are performed over these intervals to obtain a reliable data of the measurements (which are also in the form of intervals). This type of computation is called interval arithmetic or interval analysis.

8.10 Interval Analysis of Uncertain Values

Fuzzy numbers are an extension of the concept of intervals. Intervals are considered at only one unique level. Fuzzy numbers consider them at several levels varying from 0 to 1. In interval analysis, the uncertainty of the data is limited between the intervals specified by the lower bound & upper bound. The following are the various types of intervals:

- $[a1, a2] = \{x | a1 \leq x \leq a2\}$ is closed interval

- $[a1, a2) = \{x | a1 \leq x < a2\}$ is an interval closed at left end side & open at right end.
- $(a1, a2] = \{x | a1 < x \leq a2\}$ is an interval open at left end side & closed at right end.
- $(a1, a2) = \{x | a1 < x < a2\}$ is an open interval, open at both left end and right end.

8.11 Mathematical operations on Intervals

Let $A = [a1, a2]$ & $B = [b1, b2]$ be the intervals defined. If $x \in [a1, a2]$ & $y \in [b1, b2]$

Addition (+): $A + B = [a1, a2] + [b1, b2] = [a1 + b1, a2 + b2]$

Subtraction (-): $A - B = [a1, a2] - [b1, b2] = [a1 - b2, a2 - b1]$

We subtract the larger value out of b1 & b2 from a1. The smaller value out of b1 & b2 from a2 is subtracted.

Multiplication (.): Let the two intervals of confidence be A=[a1,a2] & B=[b1,b2] defined on non-negative real line.

$A.B = [a1, a2].[b1, b2] = [a1.b1,\ a2.b2]$

If we multiply an interval with a non-negative real number $\propto$

$$\propto .A = [\propto, \propto].[a1, a2] = [\propto .a1, \propto .a2]$$

$$\propto .B = [\propto, \propto].[b1, b2] = [\propto .b1, \propto .b2]$$

Division (÷): The division two intervals of confidence defined on non-negative real line is given by.

$A \div B = [a1, a2] \div [b1, b2] = [a1/b1,\ a2/b2]$

If b1 = 0 then the upper bound increases to

$+\infty$. If $b1 = b2 = 0$, then interval of confidence is extended to $+\infty$

Image

($\bar{A}$): If $x \in [-a2, -a1]$. Also if $A = [a1, a2]$ then its image $\bar{A} = [-a2, -a1]$.
Note that $A + \bar{A} = [a1, a2] + [-a2, -a1] = [a1 - a2, a2 - a1] \neq 0$

The subtraction becomes addition of an image.

Inverse (A^{-1}): If

$x \in [a1, a2]$is a subset of a positive real line, then its inverse is given by $\left(\frac{1}{x}\right) = \left[\frac{1}{a2}, \frac{1}{a1}\right]$. Similarly, the inverse of A is given by $A^{-1} = [a1, a2]^{-1}$ $= \left[\frac{1}{a2}, \frac{1}{a1}\right]$. The division becomes multiplication of an inverse. For division by a non − negative number $\propto > 0$ i. e. $\left(\frac{1}{\propto}\right)$.

A, we obtain $A \div \propto = A.\left[\frac{1}{\propto}, \frac{1}{\propto}\right] = [\frac{a1}{\propto}, \frac{a2}{\propto}]$

Max and Min Operations: $A = [a1, a2]$ & $B = [b1, b2]$

Max: $A \vee B = [a1, a2] \vee [b1, b2] = [\, a1 \vee b1,\ a2 \vee b2]$

Min: $A \wedge B = [a1, a2] \wedge [b1, b2] = [\, a1 \wedge b1,\ a2 \wedge b2]$

Table 8.1: Set Operations on Intervals

Conditions	Union, ∪	Intersection, ∩
$a_1 > b_2$	$[b_1, b_2] \cup [a_1, a_2]$	ϕ
$b_1 > a_2$	$[a_1, a_2] \cup [b_1, b_2]$	ϕ
$a_1 > b_1, a_2 < b_2$	$[b_1, b_2]$	$[a_1, a_2]$
$b_1 > a_1, b_2 < a_2$	$[a_1, a_2]$	$[b_1, b_2]$
$a_1 < b_1 < a_2 < b_2$	$[a_1, b_2]$	$[b_1, a_2]$
$b_1 < a_1 < b_2 < a_2$	$[b_1, a_2]$	$[a_1, b_2]$

Table 8.2: Algebraic Properties of Intervals

Property	Addition (+)	Multiplication (·)
Commutativity	$\underset{\sim}{A} + \underset{\sim}{B} = \underset{\sim}{B} + \underset{\sim}{A}$	$\underset{\sim}{A} \cdot \underset{\sim}{B} = \underset{\sim}{B} \cdot \underset{\sim}{A}$
Associativity	$(\underset{\sim}{A} + \underset{\sim}{B}) + \underset{\sim}{C} = \underset{\sim}{A} + (\underset{\sim}{B} + \underset{\sim}{C})$	$(\underset{\sim}{A} \cdot \underset{\sim}{B}) \cdot \underset{\sim}{C} = \underset{\sim}{A} \cdot (\underset{\sim}{B} \cdot \underset{\sim}{C})$
Neutral number	$\underset{\sim}{A} + 0 = 0 + \underset{\sim}{A} = \underset{\sim}{A}$	$\underset{\sim}{A} \cdot 1 = 1 \cdot \underset{\sim}{A} = \underset{\sim}{A}$
Image and inverse	$\underset{\sim}{A} + \overline{\underset{\sim}{A}} = \overline{\underset{\sim}{A}} + \underset{\sim}{A} \neq 0$	$\underset{\sim}{A} \cdot \underset{\sim}{A}^{-1} = \underset{\sim}{A}^{-1} \cdot \underset{\sim}{A} \neq 1$

8.12 Fuzzy Number

A fuzzy number is a normal, convex membership function on the real line R. Its membership function is piecewise continuous. That is, every λ-cut set Aλ, λ∈[0,1],of a fuzzy number A is a closed interval of R & the highest value of

membership of A is unity.For two given numbers A & B in R,for specific $\lambda 1 \in [0, 1]$, we obtain two dosed intervals:

$$A_{\lambda 1} = [a1^{(\lambda 1)}, a2^{(\lambda 2)}] \; from \; fuzzy \; number \; A$$

$$B_{\lambda 1} = [b1^{(\lambda 1)}, b2^{(\lambda 2)}] \; from \; fuzzy \; number \; B$$

Fuzzy number is an extension of the concept of intervals. Fuzzy numbers consider them at several levels with each of these levels corresponding to each λ-cut of the fuzzy numbers.The notation $A_{\lambda 1}=[a1(^{\lambda 1}),a2(^{\lambda 2})]$ can be used to represent a closed interval of a fuzzy number A at a λ h -level.

Let us discuss the interval arithmetic for closed intervals of fuzzy numbers. Let (*) denote an arithmetic operation, such as addition, subtraction, multiplication or division, on fuzzy numbers. The result $\underline{A} * \underline{B}$, where $\underline{A}$ and $\underline{B}$ are two fuzzy numbers is given by

$$\mu_{\underline{A}*\underline{B}}(z) = \bigvee_{z=x*y} [\mu_{\underline{A}}(x), \mu_{\underline{B}}(y)]$$

Using extension principle (see Section 11.3), where $x, y \in R$, for min ($\wedge$) and max ($\vee$) operation, we have

$$\mu_{\underline{A}*\underline{B}}(z) = \sup_{z=x*y} [\mu_{\underline{A}}(x) * \mu_{\underline{B}}(y)]$$

Using λ-cut, the above two equations become

$$(\underline{A} * \underline{B})_\lambda = \underline{A}_\lambda * \underline{B}_\lambda \text{ for all } \lambda \in [0, 1]$$

where $\underline{A}_\lambda = [a_1^{(\lambda)}, a_2^{(\lambda)}]$ and $\underline{B}_\lambda = [b_1^{(\lambda)}, b_2^{(\lambda)}]$. Note that for $a_1, a_2 \in [0, 1]$, if $a_1 > a_2$, then $\underline{A}_{a_1} \subset \underline{A}_{a_2}$.

On extending the *addition and subtraction* operations on intervals to two fuzzy numbers $\underline{A}$ and $\underline{B}$ in R, we get

$$\underline{A}_\lambda + \underline{B}_\lambda = [a_1^\lambda + b_1^\lambda, a_2^\lambda + b_2^\lambda]$$

$$\underline{A}_\lambda - \underline{B}_\lambda = [a_1^\lambda - b_2^\lambda, a_2^\lambda - b_1^\lambda]$$

Similarly, on extending the *multiplication and division* operations on two fuzzy numbers $\underline{A}$ and $\underline{B}$ in R^+ (non-negative real line) = $[0, \infty]$, we get

$$\underline{A}_\lambda \cdot \underline{B}_\lambda = [a_1^\lambda \cdot b_1^\lambda, a_2^\lambda \cdot b_2^\lambda]$$

$$\underline{A}_\lambda \div \underline{B}_\lambda = \left[\frac{a_1^\lambda}{b_2^\lambda}, \frac{a_2^\lambda}{b_1^\lambda}\right], \quad b_2^\lambda > 0$$

The *multiplication* of a fuzzy number $\underset{\sim}{A} \subset R$ by an ordinary number $\beta \in R^+$ can be defined as

$$(\beta \cdot \underset{\sim}{A})_\lambda = [\beta a_1^\lambda, \beta a_2^\lambda]$$

The *support* for a fuzzy number, say $\underset{\sim}{A}$, is given by

$$\text{supp } \underset{\sim}{A} = \{x | \mu_{\underset{\sim}{A}}(x) > 0\}$$

which is an interval on the real line, denoted symbolically as A. The support of the fuzzy number resulting from the arithmetic operation $\underset{\sim}{A} * \underset{\sim}{B}$, i.e.,

$$\underset{\underset{\sim}{A}*\underset{\sim}{B}}{\text{supp}}(z) = A * B$$

is the arithmetic operation on the two individual supports, A and B, for fuzzy numbers $\underset{\sim}{A}$ and $\underset{\sim}{B}$, respectively.

In general, arithmetic operations on fuzzy numbers based on λ-cut are given by (as mentioned earlier)

$$(\underset{\sim}{A} * \underset{\sim}{B})_\lambda = A_\lambda * B_\lambda$$

The algebraic properties of fuzzy numbers are listed in Table 11-3. The operations on fuzzy numbers possess the following properties as well.

1. If A and B are fuzzy numbers in R, then $(A + B)$ and $(A - B)$ are also fuzzy numbers. Similarly if A and B are fuzzy numbers in R^+, then $(A \cdot B)$ and $(A \div B)$ are also fuzzy numbers.
2. There exist no image and inverse fuzzy numbers, $\overline{A}$ and A^{-1}, respectively.
3. The inequalities given below stand true:

$$(A - B) + B \neq A \quad \text{and} \quad (A + B) \cdot B \neq A$$

Table 8.3 Algebraic Properties of Addition and Multiplication on Fuzzy Numbers

Property	Addition	Multiplication
Fuzzy numbers	$A, B, C \subset R$.	$A, B, C \subset R^+$
Commutativity	$A + B = B + A$	$A \cdot B = B \cdot A$
Associativity	$(A + B) + C = A + (B + C)$	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$
Neutral number	$A + 0 = 0 + A = A$	$A \cdot 1 = 1 \cdot A = A$
Image and inverse	$A + \overline{A} = \overline{A} + A \neq 0$	$A \cdot A^{-1} = A^{-1} \cdot A \neq 1$

8.13 Fuzzy Ordering

The technique for fuzzy ordering is based on the concept of possibility measure. For a fuzzy number A, two fuzzy sets, A_1 & A_2 are defined. For this number, the set of numbers that are possibly greater than or equal to A is denoted as A_1 and is defined as

$$\mu_{A_1}(w) = \prod_A(-\infty, w) = \sup_{u \le w} \mu_A(u)$$

In a similar manner, the set of numbers that are necessarily greater than A is denoted as A_2 and is defined as

$$\mu_{A_2}(w) = N_A(-\infty, w) = \inf_{u \ge w} [1 - \mu_A(u)]$$

where $\prod_A$ and N_A are possibility and necessity measures.

We can compare A with B_1 & B_2 by index of comparison such as the possibility or necessity measure of a fuzzy set. That is, we can calculate the possibility and necessity measures, in the set μ_A of fuzzy sets B_1 & B_2. On the basis of this, we obtain four fundamental indices of comparison.

1. $\prod_A(B_1) = \sup_u \min(\mu_A(u), \sup_{v \le u} \mu_B(v)) = \sup_{u \ge v} \min(\mu_A(u), \mu_B(v))$
   This shows the possibility that the largest value X can take is at least equal to smallest value that Y can take.
2. $\prod_A(B_2) = \sup_u \min(\mu_A(u), \inf_{v \ge u}[1 - \mu_B(v)]) = \sup_u \inf_{v \ge u} \min(\mu_A(u), [1 - \mu_B(v)])$
   This shows the possibility that the largest value X can take is greater than the largest value that Y can take.
3. $N_A(B_1) = \inf_u \max(1 - \mu_A(v), \sup_{v \le u} \mu_B(v)) = \inf_u \sup_{v \le u} \max(1 - \mu_A(u), \mu_B(v))$
   This shows the possibility that the smallest value X can take is at least equal to smallest value that Y can take.
4. $N_A(B_2) = \inf_u \max(1 - \mu_A(u), \inf_{v \ge u}[1 - \mu_B(v)]) = 1 - \sup_{u \le v} \min[\mu_A(u), \mu_B(v)]$
   This shows the possibility that the smallest value X can take is greater than the largest value that Y can take.

8.14 Fuzzy Vectors

A vector P = (P_1, P_2, ... , P_n) is called a fuzzy vector if for any element we have 0 $\le$ P $\le$ 1 for i = 1 to n. Similarly, the transpose of the fuzzy vector e denoted by P^T, is a column vector if P is a row vector, i.e.,

$$P^T = \begin{bmatrix} P_1 \\ P_2 \\ \cdot \\ \cdot \\ \cdot \\ P_n \end{bmatrix}$$

Let P & Q as fuzzy vector on length n.

Fuzzy inner product: $\underset{\sim}{P} \cdot \underset{\sim}{Q}^T = \bigvee_{i=1}^{n} (P_i \wedge Q_i)$

Fuzzy outer product: $\underset{\sim}{P} \oplus \underset{\sim}{Q}^T = \bigwedge_{i=1}^{n} (P_i \vee Q_i)$

The complement of fuzzy vector ~P has constraint $0 \leq \sim P \leq 1 \; for \; i = 1 \; to \; n$

$$\sim P = (1 - P_1, 1 - P_2, \ldots 1 - Pn) = (\sim P_1, \sim P_2, \ldots, \sim Pn)$$

Largest component is defined as its upper bound: $\hat{P} = \max_i(P_i)$

Smallest component is defined as its lower bound: $\underset{\wedge}{P} = \min_i(P_i)$

Properties of Fuzzy Vector

1. $\overline{\underset{\sim}{P} \cdot \underset{\sim}{Q}^T} = \overline{\underset{\sim}{P}} \oplus \overline{\underset{\sim}{Q}}^T$
2. $\overline{\underset{\sim}{P} \oplus \underset{\sim}{Q}^T} = \overline{\underset{\sim}{P}} \cdot \overline{\underset{\sim}{Q}}^T$
3. $\underset{\sim}{P} \cdot \underset{\sim}{Q}^T \leq (\hat{P} \wedge \hat{Q})$
4. $\underset{\sim}{P} \oplus \underset{\sim}{Q}^T = (\underset{\wedge}{P} \vee \underset{\wedge}{Q})$
5. $\underset{\sim}{P} \cdot \underset{\sim}{P}^T = \hat{P}$
6. $\underset{\sim}{P} \oplus \underset{\sim}{P}^T \geq \underset{\wedge}{P}$
7. If $\underset{\sim}{P} \subseteq \underset{\sim}{Q}$ then $\underset{\sim}{P} \cdot \underset{\sim}{Q}^T = \hat{P}$ and if $\underset{\sim}{Q} \subseteq \underset{\sim}{P}$ then $\underset{\sim}{P} \oplus \underset{\sim}{Q}^T = \underset{\wedge}{P}$
8. $\underset{\sim}{P} \cdot \overline{\underset{\sim}{P}} \leq \frac{1}{2}$
9. $\underset{\sim}{P} \oplus \overline{\underset{\sim}{P}} \leq \frac{1}{2}$

8.15 Extension Principles

The extension principle allows generalization of crisp sets into fuzzy sets framework & extends point-to-point mappings for fuzzy sets.

Given a function $f: M \rightarrow N$ and a fuzzy set in M, where

$$\underset{\sim}{A} = \frac{\mu_1}{x_1} + \frac{\mu_2}{x_2} + \cdots + \frac{\mu_n}{x_n}$$

the extension principle states that

$$f(\underset{\sim}{A}) = f\left(\frac{\mu_1}{x_1} + \frac{\mu_2}{x_2} + \cdots + \frac{\mu_n}{x_n}\right) = \frac{\mu_1}{f(x_1)} + \frac{\mu_2}{f(x_2)} + \cdots + \frac{\mu_n}{f(x_n)}$$

If f maps several elements of M to the same element y in N (i.e., many-to-one mapping), then the maximum among their membership grades is taken. That is,

$$\mu_{f(\underset{\sim}{A})}(y) = \max_{\substack{x_i \in M \\ f(x_i) = y}} [\mu_{\underset{\sim}{A}}(x_i)]$$

where x_i's are the elements mapped to same element y. The function f maps n-tuples in M to a point in N.

Let M be the Cartesian product of universes $M = M_1 \times M_2 \times \cdots \times M_n$ and $\underset{\sim}{A}_1, \underset{\sim}{A}_2, \ldots, \underset{\sim}{A}_n$ be n fuzzy sets in $M_1, M_2, \ldots, M_n$, respectively. The function f maps an n-tuple $(x_1, x_2, \ldots, x_n)$ in the crisp set M to a point y in the crisp set V, i.e., $y = f(x_1, x_2, \ldots, x_n)$. The function $f(x_1, x_2, \ldots, x_n)$ to be extended to act on the n fuzzy subsets of $M, \underset{\sim}{A}_1, \underset{\sim}{A}_2, \ldots, \underset{\sim}{A}_n$ is permitted by the extension principle such that

$$\underset{\sim}{I} = f(\underset{\sim}{A})$$

where $\underset{\sim}{I}$ is the fuzzy image of $\underset{\sim}{A}_1, \underset{\sim}{A}_2, \ldots, \underset{\sim}{A}_n$ through $f(\cdot)$. The fuzzy set $\underset{\sim}{B}$ is defined by

$$\underset{\sim}{B} = \{(y, \mu_{\underset{\sim}{B}}(y)) \mid y = f(x_1, x_2, \ldots, x_n), (x_1, x_2, \ldots, x_n) \in M\}$$

where

$$\mu_{\underset{\sim}{B}}(y) = \sup_{\substack{(x_1, x_2, \ldots x_n) \in M \\ y = f(x_1, x_2, \ldots x_n)}} \min[\mu_{\underset{\sim}{A}_1}(x_1), \mu_{\underset{\sim}{A}_2}(x_2), \ldots, \mu_{\underset{\sim}{A}_n}(x_n)]$$

with a condition that $\mu_{\underset{\sim}{B}}(y) = 0$ if there exists no $(x_1, x_2, \ldots, x_n) \in M$ such that $y = f(x_1, x_2, \ldots, x_n)$.

8.16 Overview of Fuzzy Measures

A fuzzy measure explains the imprecision or ambiguity in the assignment of an element $\propto$ to two or more crisp sets. For representing uncertainty condition, known as ambiguity, we assign a value in the unit interval [0, 1] to each possible crisp set to which the element in the problem might belong. The value assigned represents the degree of evidence or certainty or belief of the element's membership in the set. The representation of uncertainty of this manner is called fuzzy measure. The difference between a fuzzy measure and a fuzzy set on a universe of elements is that, in fuzzy measure, the imprecision is in the assignment of an element to one of two or more crisp sets, and in fuzzy sets, the imprecision is in the prescription of the boundaries of a set.

A fuzzy measure is defined by a function **g: P(X)** → [0,1] which assigns to each crisp subset of a universe of discourse X a number in the unit interval [0,1],where P(X) is power set of X.A fuzzy measure is a set function. To qualify a fuzzy measure,the function g should possess certain properties.A fuzzy measure is also described as follows: g: B → [0,1] where B ⊂P(X) is a family of crisp subsets of X Here B is a Borel field or a σ field. Also, g satisfies rhe following three axioms of fuzzy measures:

- Boundary condition (g1): $g(\emptyset) = 0; g(X) = 1$
- Monotonicity (g2): for every classical set $A, B \in P(X)$, $if\ A \subseteq B,\ then\ g(A) \leq g(B)$
- Continuity (g3): for sequence $A_i \in P(X) | i \in N)$ of subsets X, if either $A_1 \subseteq A_2 \ldots or\ A_1 \supseteq A_2 \ldots then\ \lim_{i\to\infty} g(A_i) = g\left(\lim_{i\to\infty} A_i\right)$

where N is the set of all positive integers

A σ field or Borel field satisfies the following properties:

- $X \in B\ \&\ \emptyset \in B$
- $if\ A \in B,\ then \sim A \in B$
- B is closed under set union operation, i.e. if $A \in B\ \&\ B \in B\ (\sigma\ field),\ then\ A \cup B \in B\ (\sigma\ field)$

The fuzzy measure excludes the additive property of standard measures, h. The additive property states that when two sees A and B are disjoint, then $h(A \cup B) = h(A) + h(B)$. Since $A \subseteq A \cup B\ \&\ B \subseteq A\ \cup B$, and because fuzzy measure g possesses monotonic property, we have $g(A \cup B) \geq \max\ [g(A),\ g(B)]$. Since $A \cap B \subseteq A\ \&\ A \cap B \subseteq B$, and because fuzzy measure g possesses monotonic property, we have $g(A \cup B) \leq min\ [g(A),\ g(B)]$.

8.17 Belief & Plausibility Measures

The belief measure is a fuzzy measure that satisfies three axioms g1, g2 and g3 and an additional axiom of subadditivity. A belief measure is a function $bel: B \rightarrow [0,1]$ satisfying axioms g1, g2 and g3 of fuzzy measures and subadditivity axiom. It is defined as follows:

$$\text{bel}(A_1 \cup A_2 \cup \cdots \cup A_n) \geq \sum_i \text{bel}(A_i) - \sum_{i<j} \text{bel}(A_i \cap A_j) + \cdots + (-1)^{n-1} \text{bel}(A_1 \cap A_2 \cap \cdots \cap A_n)$$

for every $n \in N$ and every collection of subsets of X. N is set of all positive integer. This is called axiom 4 (g4).

Plausibility is defined as Pl(A)=1 -bel($\bar{A}$) for all A∈B(CP(X)).Belief measure can be defined as bel(A)=1-Pl($\bar{A}$).Plausibility measure can also be defined independent of belief measure.A plausibility measure is a function Pl:B→[0,1] satisfying axioms g1, g2, g3 of fuzzy measures and the following subadditivity axiom (axiom g5):

$$\text{Pl}(A_1 \cap A_2 \cap \cdots \cap A_n) \leq \sum_i \text{Pl}(A_i) - \sum_{i<j} \text{Pl}(A_i \cup A_j) + \cdots + (-1)^{n-1} \text{Pl}(A_1 \cup A_2 \cup \cdots \cup A_n)$$

for every $n \in N$ and all collection of subsets of X

The belief measure and the plausibility measure are mutually dual, so it will be beneficial to express both of them in terms of a set function m, called a basic probability assignment. The basic probability assignment m is a set function, : $B \rightarrow [0,1] such\ that\ m(\emptyset = 0) and\ \sum_{A \in B} m(A) =$

1. The basic probability assignments are not fuzzy measures. The quantity $m(A) \in [0,1]$, $A \in B(CP(X))$, is called A's basic probability number.

Given a basic assignment m, a belief measure and a plausibility measure and a plausibility measure can be uniquely determined by:

$$\text{bel}(A) = \sum_{B \subseteq A} m(B)$$

$$\text{Pl}(A) = \sum_{B \cap A \neq 0} m(B)$$

The relations among $m(A)$, bel(A) and Pl(A) are as follows:

1. $m(A)$ measures the belief that the element $(x \in X)$ belongs to set A alone, not the total belief that the element commits in A.
2. bel (A) indicates total evidence that the element $(x \in X)$ belongs to set A and to any other special subsets of A
3. Pl (A) includes the total evidence that the element $(x \in X)$ belongs to set A or to other special subsets of A plus the additional evidence or belief associated with sets that overlap with A.

Based on these relations, we have

$$\text{Pl}(A) \geq \text{bel}(A) \geq m(A) \quad \forall A \in B\ (\sigma \text{ field})$$

Belief and plausibility measure are dual to each other. The corresponding basic assignment m can be obtained from a given plausibility measure Pl:

$$m(A) = \sum_{B \subseteq A} (-1)^{(A-B)} [1 - \text{Pl}(\bar{B})] \quad \forall A \in B\ (\sigma \text{ field})$$

Every set $A \in B(\text{CP}(X))$ for which $m(A) > 0$ is called a focal element of m. Focal elements are subsets of X on which the available evidence focuses.

8.18 Probability Measures

A probability measure is the function $P: B \rightarrow [0,1]$ satisfying the three axioms g1, g2 & g3 of fuzzy measures and the additivity axioms (axiom g6) as follows $P(A \cup B) = P(A) + P(B)$ *whenever* $A \cap B = \emptyset,\ A, B \in B$.

Theorem : *"A belief measure bel on a finite σ-field B, which is a subset of P(X), is a probability measure if and only if its basic probability assignment m is given by m({x}) = bel({x}) and m(A) = 0 for all subsets of X that are not singletons."*

The theorem indicates fiat a probability measure on finite sets can be represented uniquely by a function defined on the elements of the universal set X rather than its subsets. The probability measures on finite sets can be fully represented by a function, P: X → [0, 1] such that P(x) = m({x}). This function P(X) is called probability distribution function.

Within probability measure, the total ignorance is expressed by the uniform probability distribution function:

$$P(x) = m(\{x\}) = \frac{1}{|X|} \; for\ all\ x \in X$$

The plausibility and belief measures can be viewed as upper & lower probabilities that characterize a set of probability measures.

8.19 Possibility & Necessity Measures

A group of subsets of a universal set is nested if these subsets can be ordered in a way that each is contained in the next; i.e. $A1 \subset A2 \subset A3 \ldots \subset An, Ai \in P(X)$ are nested sets. When the focal elements of a body of evidence (E, m) are nested, the linked belief and plausibility measures are called consonants, because here the degrees of evidence allocated to them do not conflict with each other.

Theorem: *"Consider a consonant body of evidence (E, m), the associated consonant belief and plausibility measures possess the following properties:*

$$bel\,(A \cap B) = \min\big(bel(A), bel(B)\big)$$

$$Pl(A \cup B) = \max\,(Pl(A),\ Pl(B))$$

for all $A, B\ \in B(CP(X))$.

Consonant belief and plausibility measures are referred to as necessity & possibility measures & are denoted by N and $\prod$, respectively.

The possibility measure $\prod$ & necessity measure N are function:

$\prod: B \rightarrow [0,1]$ & N: B $\rightarrow [0,1]$ such that $\prod$ & N both satisfy the axioms g1,g2 & g3 of fuzzy measures and following axiom g7:

$$\prod(A \cup B) = \max\big(\prod(\mathrm{A}), \prod(B)\big) \qquad \forall A, B\ \in B$$

$$N(A \cap B) = \min\big(N(\mathrm{A}), N(B)\big) \qquad \forall A, B\ \in B$$

Necessity and possibility are special subclasses of belief and plausibility measures, they are related to each other by

$$\prod(A) = 1 - N(\bar{\mathrm{A}}) \quad \& \quad N(A) = 1 - \prod(\bar{\mathrm{A}}) \qquad \forall\, A \in \sigma\, field$$

The properties given below are based on the axiom *g7* and above set of equations.

1. $\min[N(A), N(\bar{A})] = N(A \cap \bar{A}) = 0$. This implies that A or $\bar{A}$ is not necessary at all.
2. $\max[\prod(A), \prod(\bar{A})] = \prod(A \cup \bar{A}) = \prod(X) = 1$. This implies that either A or $\bar{A}$ is completely possible.
3. $\prod(A) \geq N(A)\ \forall A \subseteq \sigma$ field.
4. If $N(A) > 0$ then $\prod(A) = 1$ and if $\prod(A) < 1$ then $N(A) = 0$.

The two equations indicate that if an event is necessary then it is completely possible. If it is not completely possible then it is not necessary. Every possibility measure $\prod$ on $B \subset P(x)$ can be uniquely determined by a possibility distribution function

$$\prod : x \rightarrow [0, 1]$$

using the formula

$$\prod(A) = \max_{x \in A} \prod(x) \quad \forall x \in \sigma \text{ field}$$

The necessity and possibility measure are mutually dual with each other. As a result we can obtain the necessity measure from the possibility distribution function. This is given as

$$N(A) = 1 - \prod(\bar{A}) = 1 - \max_{x \notin A} \prod(x)$$

The total ignorance can be expressed in terms of the possibility distribution by $\prod(x_n) = 1$ and $\prod(x_i) = 0$ for $i = 1$ to $n - 1$, corresponding to $\prod(A_n) = \prod(X) = 1$ and $\prod(A) = 0$.

8.20 Measure of Fuzziness

The fuzzy measures concept provides a general mathematical framework to deal with ambiguous variables. Measures of uncertainty related to vagueness are referred to as measures of fuzziness. A measure of fuzziness is a function $f: P(X) \rightarrow R$ where R is the real line and P(X) is the set of all fuzzy subsets of X. The function f satisfies the following axioms:

- Axiom 1 (f1): f(A) = 0 if and only if A is a crisp set.
- Axiom 2 (f2): If A (shp) B, then f(A) ≤ f(B), where A (shp) B denotes that A is sharper than B.
- Axiom 3 (f3): f(A) takes the maximum value if and only if A is maximally fuzzy.

Axiom f1 shows that a crisp set has zero degree of fuzziness in it. Axioms f2 and f3 are based on concept of "sharper" and "maximal fuzzy," respectively.

1. The first fuzzy measure can be defined by the function:

$$f(A) = -\sum_{x \in A} \{\mu_A(x) \log_2 [\mu_A(x)] \; [1-\mu_A(x)] \log_2 [1-\mu_A(x)]\}$$

It can be normalized as

$$f'(A) = \frac{f(A)}{|x|}$$

where $|x|$ is cardinality of universal set X. This measure of fuzziness can be considered as the entropy of a fuzzy set

2. A (shp) B, A is sharper than B, is defined as

$$\mu_A(x) \leq \mu_B(x) \quad \text{for } \mu_B \leq 0.5$$
$$\mu_A(x) \geq \mu_B(x) \quad \text{for } \mu_B(x) \geq 0.5 \quad \forall x \in X$$

3. A is maximally fuzzy if

$$\mu_A(x) = 0.5 \quad \text{for all } x \in X$$

8.21 Fuzzy Integrals

Let K be a mapping from X to [0,1]. The fuzzy integral, in the sense of fuzzy measure g, of K over a subset A of X is defined as

$$\int_A K(x) \cdot g = \sup_{\alpha \in [0,1]} \min[\beta, g(A \cap H_\beta)]$$

where $H_\beta = \{x \in x | K(x) \geq \beta\}$. Here, A is called the domain of integration. If $k = a \in [0, 1]$ is a constant, then its fuzzy integral over X is "a" itself, because $g(X \cap H_\beta) = 1$ for $\beta \leq a$ and $g(X \cap H_\beta) = 0$ for $\beta > a$, i.e.,

$$\int_x a \bullet g = a, \quad a \in [0, 1]$$

Consider X to be a finite set such that $X = \{x_1, x_2, \ldots, x_n\}$. Without loss of generality, assuming the function to be integrated, k can be obtained such that $k(x_1) \geq k(x_2) \geq \cdots \geq k(x_n)$. This is obtained after proper ordering. The basic fuzzy integral then becomes

$$\int_X k(x) \cdot g = \max_{i=1 \text{ to } n} \min[k(x_i), g(H_i)]$$

where $H_i = \{x_1, x_2, \ldots, x_i\}$. The calculation of the fuzzy measure "g" is a fundamental point in performing a fuzzy integration.

Summary

This chapter starts with the discussion about membership functions and their features. The formation of the membership function is the core for the entire fuzzy system operation. The capability of human reasoning is important for membership functions. The inference method is based on the geometrical shapes and geometry, whereas the angular fuzzy set is based on the angular features. Using neural networks and reasoning methods the memberships are tuned in a cyclic fashion and are based on rule structure. The improvements are carried out to achieve an optimum solution using generic algorithms. Thus, the membership function can be formed using any one of the methods.

Later we have discussed the methods of converting fuzzy variables into crisp variables by a process called as defuzzification. Defuzzification process is essential because some engineering applications need exact values for performing the operation. Defuzzification is a natural and essential technique. Lambda-cuts for fuzzy sets and fuzzy relations were discussed. Apart from the Lambda-cut method, seven defuzzification methods were presented. The method of defuzzification should be assessed on the basis of the output in the context of data available.

Finally, we discussed fuzzy arithmetic, which is considered as an extension of interval arithmetic. One of the important tools of fuzzy set theory introduced by Zadeh is the extension principle, which allows any mathematical relationship between nonfuzzy elements to be extended to fuzzy entities. This principle can be

applied to algebraic operations to define set-theoretic operations for higher order fuzzy sets. The belief and plausibility measures can be expressed by the basic probability assignment m, which assigns degree of evidence or belief indicating that a particular element of X belongs to set A and not to any subset of A. The main characteristic of probability measures is that each of them can be distinctly represented by a probability distribution function defined on the elements of a universal set apart from its subsets. Fuzzy integrals defined define by Sugeno (1977) are also discussed. Fuzzy integrals are used to perform integration of fuzzy functions.

Review Questions

1. What is membership function? Enlist and explain its features.
2. Write a short note on fuzzification.
3. Explain any three methods of membership value assignments in detail.
4. Write a short note on defuzzification.
5. What is Lambda-cuts for fuzzy set and Fuzzy relations?
6. Explain any three methods of defuzzification in detail.
7. Write a short note on fuzzy arithmetic.
8. What are the mathematical operations on intervals of fuzzy.
9. Write a short note on fuzzy number and fuzzy ordering.
10. Write a short note on fuzzy vectors.
11. Write a short note on belief and plausibility measures.
12. Write a short note on possibility and necessity measures.

Bibliography, References and Further Reading

- Artificial Intelligence and Soft Computing, by Anandita Das Battacharya, SPD 3rd, 2018
- Principles of Soft Computing, S.N. Sivanandam, S.N.Deepa, Wiley, 3rd , 2019
- Neuro-fuzzy and soft computing, J.S.R. Jang, C.T.Sun and E.Mizutani, Prentice Hall of India, 2004

❖❖❖

UNIT 5

9

GENETIC ALGORITHM

Unit Structure

Learning Objectives

- Gives an introduction to natural evolution.
- Lists the basic operators (selection, crossover, mutation) and other terminologies used in Genetic Algorithms (GAs).
- Discusses the need for schemata approach.
- Details the comparison of traditional algorithm with GA.
- Explains the operational flow of simple GA.
- Description is given of the various classifications of GA- Messy GA, adaptive GA, hybrid GA, parallel GA and independent sampling GA.
- The variants of parallel GA (fine-grained parallel GA and coarse-grained parallel GA) are included.
- Enhances the basic concepts involved in Holland classifier system.
- The various features and operational properties of genetic programming are provided.
- The application areas of GA are also discussed.

Thales R. Darwin says that "Although the belief that an organ so perfect as the eye could have been formed by natural selection is enough to stagger any one; yet in the case of any organ, if we know of a long series of gradations in complexity, each good for its possessor, then, under changing conditions of life, there is no logical impossibility in the acquirement of any conceivable *degree of perfection through natural selection."*

9.0 Introduction

Thales Darwin has formulated the fundamental principle of natural selection as the main evolutionary tool. He put forward his ideas without the knowledge of basic hereditary principles. In 1865, Gregory Mendel discovered these hereditary principles by the experiments he carried out on peas. After Mendel's work genetics was developed. Morgan experimentally found that chromosomes were the carriers of hereditary information and that genes representing the hereditary factors were lined up on chromosomes. Darwin's natural selection theory and natural genetics remained unlinked until 1920s when it was proved that genetics and selection were in no way contrasting each other. Combination of Darwin’s and Mendel’s ideas leads to the modern evolutionary theory.

In The Origin of Species, Thales Darwin stated the theory of natural evolution. Over many generations, biological organisms evolve according to the principles of natural selection like "survival of the fittest" to reach some remarkable forms of accomplishment. The perfect shape of the albatross wing, the efficiency and the similarity between sharks and dolphins and so on are good examples of what random evolution with absence of intelligence can achieve. So, if it works so well in nature, it should be interesting to simulate natural evolution and try to obtain a method which may solve concrete search and optimization problems.

For a better understanding of this theory, it is important first to understand the biological terminology used in evolutionary computation. It is discussed in Section 1.2

In 1975, Holland developed this idea in *Adaptation in Natural and Artificial Systems.* By describing how to apply the principles of natural evolution to optimization problems, he laid down the first GA. Holland's theory has been further developed and now GAs stand up as powerful adaptive methods to solve search and optimization problems. Today, GAs are used to resolve complicated optimization problems, such as, organizing the time table, scheduling job shop, playing games.

What are Genetic Algorithms?

GAs is adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random; instead they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, especially those that follow the principles first laid down by Thales Darwin, "survival of the fittest," because in nature, competition among individuals for seamy resources results in the fittest individuals dominating over the weaker ones.

Why Genetic Algorithms?

They are better than conventional algorithms in that they are more robust. Unlike older AI systems, they do not break easily even if the inputs are changed slightly or in the presence of reasonable noise. Also, in searching a large state-space, multimodal state-spare or *n*-dimensional source, a GA may offer significant benefits over more typical optimization techniques (linear programming, heuristic, depth-first and praxis.)

9.1 Biological Background

The science that deals with the mechanisms responsible for similarities and differences in a species is called Genetics. The word "genetics" is derived from the Greek word "genesis" meaning "to grow" or "to become. “The science of genetics helps us to differentiate between heredity and variations and accounts for the resemblances and differences during the process of evolution. The concepts of GAs are directly derived from natural evolution and heredity. The terminologies involved in the biological background of species are discussed in the following subsections.

9.2 The Cell

Every animal/human cell is a complex of many "small" factories that work together. The centre of all this is the cell nucleus. The genetic information is contained in the cell nucleus. Figure 9-1 shows anatomy of the animal cell and cell nucleus.

Chromosomes

All the genetic information gets stored in the chromosomes. Each chromosome is build of deoxyribonucleic acid (DNA). In humans, chromosomes exist in pairs (23 pairs found). The chromosomes are divided into several parts called genes. Genes code the properties of species, i.e., the characteristics of an individual. The possibilities of combination of the genes for one property are called alleles, and a gene can take different alleles. For example, there is a gene for eye colour, and all the different possible alleles are black, brown, blue and green (since no one has red or violet eyes!). The set of all possible alleles present in a particular population forms a gene pool. This gene pool can determine all the different possible variations for the future generations. The size of the gene pool helps in determining the diversity of the individuals in the population. The set of all the genes of a specific species is called *genome.* Each and every gene has a unique position on the genome called

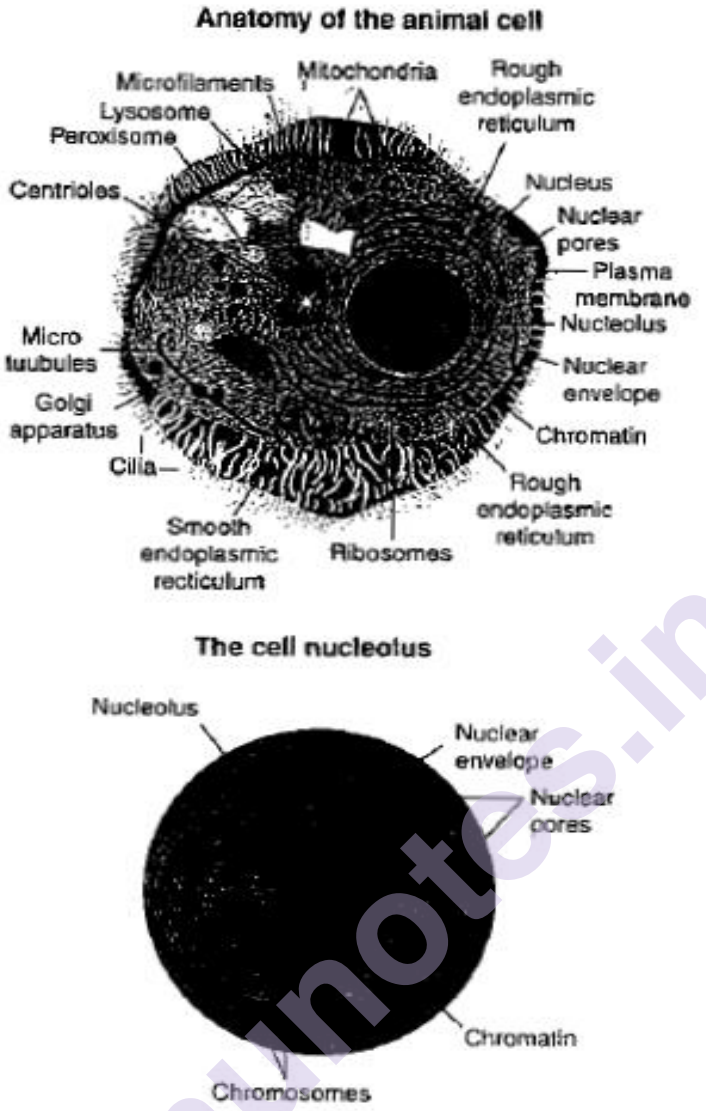


Fig9-1 anatomy of the animal cell and cell nucleus

Locus. In fact, most living organisms store their genome on several chromosomes, but in the GAs, all the genes are usually stored on the same chromosomes. Thus, chromosomes and genomes are synonyms with one other in GAs. Figure 9-2 shows a model of chromosome.

9.2.3 Genetics

For a particular individual, the entire combination of genes is called *genotype.* The *phenotype* describes the physical aspect of decoding a genotype to produce the phenotype. One interesting point of evolution is that selection is always done on the phenotype whereas the reproduction recombines genotype. Thus, morphogenesis plays a key role between section and reproduction. In higher life forms, chromosomes contain two sets of genes. These are known as *diploids.* In the

case of conflicts between two values of the same pair of genes, the dominant one will determine the phenotype whereas the other one, called *recessive,* will still be present and

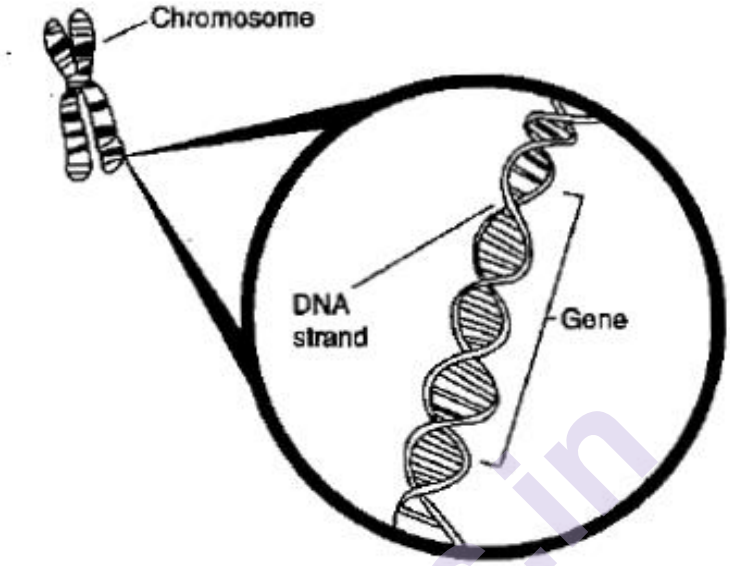


Figure 9-2 Model of chromoson

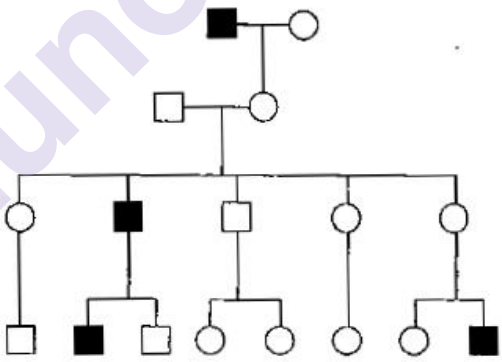


Figure 9-3 Development of genotype to Phonotype

Can be passed onto the offspring. *Diploid* allows a wider diversity of alleles. This provides a useful memory mechanism in changing or noisy environment. However, most GAs concentrates on haploid chromosomes because they are much simple to construct. In haploid representation, only one set of each gene is stored, thus the process of determining which allele should be dominant and which one should be recessive is avoided. Figure9-3 shows the development of genotype to phenotype.

9.2.4 Reproduction

Reproduction of species via genetic information is carried out by the following;

1. *Mitosis:* In mitosis the same genetic information is copied to new offspring. There is no exchange of information. This is a normal way of growing of multicell structures, such as organs. Figure 9-4 shows mitosis form of reproduction.
2. *Meiosis:* Meiosis forms the basis of sexual reproduction. When meiotic division takes place, two gametes appear in the process. When reproduction occurs, these two gametes conjugate to a zygote which becomes the new individual. Thus in this case, the genetic information is shared between the parents in order to create new offspring. Figure 9-5 shows meiosis form of reproduction.

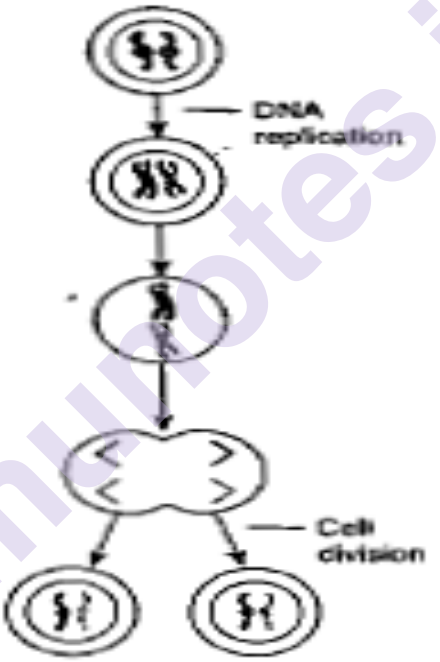


Figure 15-4 Mitosis form of reproduction.

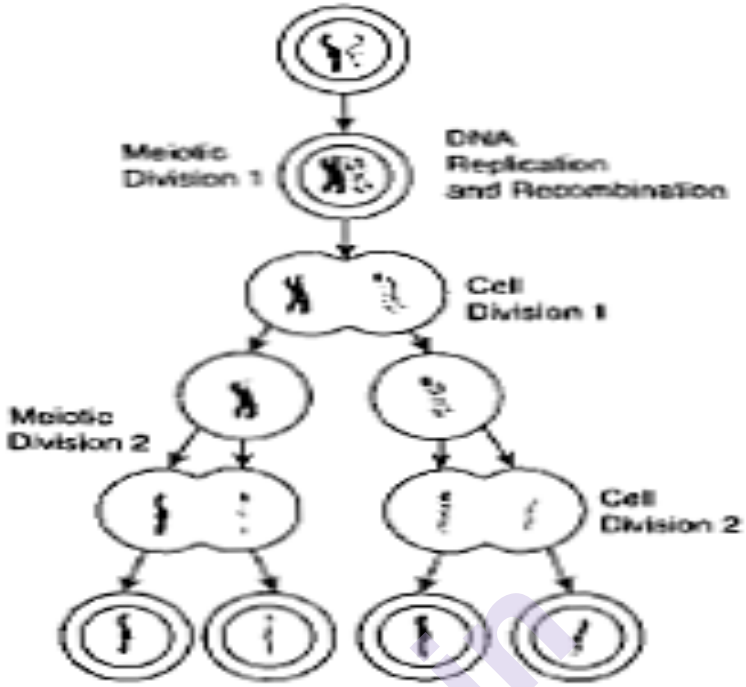


Figure 9-5 Meiosis form of reproduction

Table 9·1 Comparison of natural evolution and genetic algorithm terminology

Natural evolution	**Genetic algorithm**
Chromosome	String
Gene	Feature or character
Allele	Feature value
Locus	String position
Genotype	Structure or coded string
Phenotype	Parameter set, a decoded structure

9.2.5 Natural Selection

The origin of species is based on "Preservation of favourable variations and rejection of unfavourable variations." The variation refers to the differences shown by the individual of a species and also by offspring's of the same parents. There are more individuals born than can survive, so there is a continuous struggle for life. Individuals with an advantage have a greater chance of survival, i.e., the survival of the fittest. For example, Giraffe with long necks can have food from tall trees as well from the ground; on the other hand, goat and deer having smaller neck can have food only from the ground. As a result, natural selection plays a major role in this survival process.

Table 9.1 gives a list of different expressions, which are common in natural evolution and genetic algorithm.

9.3 Traditional Optimization and Search Techniques

The basic principle of optimization is the efficient allocation of scarce resources. Optimization can be applied to any scientific or engineering discipline. The aim of optimization is to find an algorithm which solves a given class of problems. There exists no specific method which solves all optimization problems. Consider a function,

$$f(x) : [x^1, x''] \rightarrow [0, 1] \qquad \text{.....(1)}$$

Where

$$f(x) = \begin{cases} 1 \text{ if } |x - a| < \epsilon, & \epsilon > 0, \\ -1 \text{ elsewhere} \end{cases} \text{.....(2)}$$

For the above function, f can be maintained by decreasing ϵ or by making the interval of $[x^1, x'']$ large. Thus, a difficult task can be made easier. Therefore, one can solve optimization problems by combining human creativity and the raw processing power of the computers.

The various conventional optimization and search techniques available are discussed in the following subsections.

9.3.1 Gradient Based Local Optimization Method

When the objective function is smooth and one needs efficient local optimization, it is better to use gradient-based or Hessian-based optimization methods. The performance and reliability of the different gradient methods vary considerably. To discuss gradient-based local optimization, let us assume a smooth objective function (i.e., continuous first and second derivatives). The object function is denoted by

$$f(x) : K^n \rightarrow R \qquad \text{.....(3)}$$

The first derivatives are contained in the gradient vector $\Delta f(x)$

$$\Delta f(x) = : \begin{bmatrix} \partial f(x) i \partial x_1 \\ \\ \partial f(x) i \partial x_n \end{bmatrix} \qquad \text{.....(4)}$$

The second derivatives of the object function are contained in the Hessian matrix *H(x):*

$$H(x) = \nabla^T \nabla f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial^2 x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f(x)}{\partial^2 x_n} \end{pmatrix} \quad \text{……..(5)}$$

Few methods need only the gradient vector, but in the Newton's method we need the Hessian matrix. The general pseudo code used in gradient methods is as follows:

Select an initial guess value $x^1$and set n = I.

Repeat

Solve the search direction P^n from Eq. (5) below.

Determine the next iteration point using Eq. (5) below:

$x^{n+I} = X^n + \lambda_n P^n$

Set$n$=n+l.

Until $|| X^n - X^{n-1} || < \in$ ……(6)

These gradient methods search for minimum and not maximum. Several different methods are obtained based on the details of the algorithm.

The search direction P^n in conjugate gradient method is found as follows:

$$P^n = -\Delta f(X^n) + \beta_n P^{n-1} \quad \text{.....(7)}$$

In second method,

$$\beta_n P^n = -\Delta f(x^n) \quad \text{……..(8)}$$

is used for finding search direction. The matrix β_n in Eq. (6) estimates the Hessian and is updated in each iteration. When β_n is defined as the identity matrix, the steepest descent method occurs. When the matrix B_n is the Hessian H (x^n), we get the Newton's method.

The length λ_n of the search step is computed using:

$$\lambda_n = \text{argmin}\, f(a^n + \lambda P_{an}) \qquad \text{.....(9)}$$
$$\lambda_n > 0$$

The discussed is a one-dimensional optimization problem. The steepest descent method provides poor performance. As a result, conjugate gradient method can be used. If the second derivatives are easy to compute, then Newton's method may provide best results. The secant methods are faster than conjugate gradient methods, but there occurs memory problems. Thus, these local optimization methods can be combined with other methods to get a good link between performance and reliability.

9.3.2 Random Search

Random search is an extremely basic method. It only explodes the search space by randomly selecting solutions and evaluates their fitness. This is quite an unintelligent strategy, and is rarely used. Nevertheless, this method is sometimes worth testing. It doesn't take much effort to implement it, and an important number of evaluations can be done fairly quickly. For new unresolved problems, it can be useful to compare the results of a more advanced algorithm to those obtained just with a random search for the same number of evaluations. Nasty surprises might well appear when comparing, for example, GAs to random search. It's good to remember that the efficiency of GA is extremely dependent on consistent coding and relevant reproduction operators. Building a GA which performs no more than a random search happens more often than we can expect. If the reproduction operators are just producing new random solutions without any concrete links to the ones selected from the last generation, the GA is just doing nothing else than a random search.

Random search does have a few interesting qualities. However good the obtained solution may be, if it's not optimal one, it can be always improved by continuing the run of the random search algorithm for long enough. A random search never gets stuck at any point such as a local optimum. Furthermore, theoretically, if the search space is finite, random search is guaranteed to reach the optimal solution. Unfortunately, this result is completely useless. For most of problems we are interested in, exploiting the whole search space takes lot of time.

9.3.3 Stochastic Hill Climbing

Efficient methods exist for problems with well-behaved continuous fitness functions. These methods use a kind of gradient to guide the direction of search. *Stochastic hill climbing* is the simplest method of these kinds. Each iteration

consists in choosing randomly a solution in the neighbourhood of the current solution and retains this new solution only if it improves the fitness function. Stochastic hill climbing converges towards the optimal solution if the fitness function of the problem is continuous and has only one peak (unimodal function).

On functions with many peaks (multimodal functions), the algorithm is likely to stop on the first peak it finds even if it is not the highest one. Once a peak is reached, hill climbing cannot progress anymore, and that is problematic when this point is a local optimum. Stochastic hill climbing usually starts from a random select point. A simple idea to avoid getting stuck on the first local optimal consists in repeating several hill climbs each time starting from a different randomly chosen point. This method is sometimes known as *iterated hill climbing.* By discovering different local optimal points, chances to reach the global optimum increase. It works well if there are not too many local optima in the search space. However, if the fitness function is very "noisy" with many small peaks, stochastic hill climbing is definitely nor a good method to use. Nevertheless, such methods have the advantage of being easy to implement and giving fairly good solutions very quickly.

9.3.4 Simulated Annealing

Simulated annealing (SA) was originally inspired by formation of crystal in solids during cooling. As discovered a long time ago by Iron Age blacksmiths, the slower the cooling, the more perfect is the crystal formed. By cooling, complex physical systems naturally converge rewards a stare of minimal energy. The system moves randomly, but the probability to stay in a particular configuration depends directly on the energy of the system and on its temperature. This probability is formally given *by* Gibbs law:

$$i^n = e^{E/kT} \qquad \ldots\ldots(10)$$

where *E* stands for the energy, *k* is the Boltzmann constant and *T is* the temperature. In the mid0l970s, Kirkpatrick by analogy of these physical phenomena; laid out the first description of SA.

As in the stochastic hill climbing, the iteration of the SA consists of randomly choosing a new solution in the neighbourhood of the actual solution. If the fitness function of the new solution is better than the fitness function of the current one, the new solution is accepted as the new current solution. If the fitness function is not improved, the new solution is retained with a probability:

$$P = e^{-|f(y)-f(x)|/kT} \qquad \ldots\ldots(11)$$

Where *f*(y) - *f(x)* is the difference of the fitness function between the new and the old solution.

The SA behaves like a hill climbing method but with the possibility of going downhill to avoid being trapped at local optima. When the temperature is high, the probability of deteriorate the solution is quite important, and then a lot of large moves are possible to explode the search space. The more the temperature decreases, the more difficult it is to go downhill. The algorithm thus tries to climb up from the current solution to reach a maximum. When temperature is lower, there is an exploitation of the current solution. If the temperature is too low, number deterioration is accepted, and the algorithm behaves just like a stochastic hill climbing method. Usually, the SA stars from a high temperature which decreases exponentially. The slower the cooling, the better it is for finding good solutions. It even has been demonstrated that with an infinitely slow cooling, the algorithm is almost certain to find the global optimum. The only point is that infinitely slow cooling consists in finding the appropriate temperature decrease rate to obtain a good behaviour of the algorithm.

SA by mixing exploitation features such as the random search and exploitation features like hill climbing usually gives quite good results. SA is a serious competitor of GAs. It is worth trying to compare the results obtained by each. Both are derived from analogy with natural system evolution and both deal with the same kind of optimization problem. GAs differ from SA in two main features which makes them more efficient. First, GAs use a population-based selection whereas SA only deals with one individual at each iteration. Hence Gas are expected to cover a much larger landscape of the search space at each iteration; however, SA iterations are much more simple, and so, often much faster. The grocer advantage of GA is its exceptional ability to be parallelized, whereas SA does not gain much of this. It is mainly due to the population scheme use by GA. Second, Gas use recombination operators, and are able to mix good characteristics from different solutions. The exploitation made by recombination operators are supposedly considered helpful to find optimal salmons of the problem. On the other hand, SA is still very simple to implement and gives good this. SAs have proved their efficiency over a large spectrum of difficult problems, like the optimal layout or primed circuit board or the famous travelling salesman problem.

9.3.5 Symbolic Artificial Intelligence

Most symbolic artificial intelligence (AI) systems are very static. Most of them can usually only solve one given specific problem, since their architecture was designed for whatever that specific problem was in the first place. Thus, if the given problem were somehow to be changed, these systems could have a hard time adapting to them; since the algorithm that would originally arrive co the solution may be either incorrect or less efficient. GAs were created to combat these problems. They are

basically algorithms based on natural biological evolution. The architecture of systems that implement GAs is more able to adapt to a wide range of problems. A GA functions by generating a large set of possible solutions to a given problem. It then evaluates each of chose solutions, and decides on a "fitness level" (you may recall the phrase: "survival of the fittest") for each solution set. These solutions then breed new Solutions. The parent solutions that were more "fit" are more likely m reproduce, while those that were less "fit" are more unlikely to do so. In essence, solutions are evolved over time. This way we evolve our search space scope to a point where you can find the solution. GAs can be incredibly efficient if programmed correctly.

9.4 Genetic Algorithm and Search Space

Evolutionary computing was introduced in the 1960s by I. Rothenberg in the work "Evolution Strategies. “This idea was then developed by other researches. GAs were invented by John Holland and developed this idea in his book "Adaptation in Natural and Artificial Systems" in the year 1975. Holland proposed GA as a heuristic method based on "survival of the finest." GA was discovered as a useful tool for search and optimization problems.

9.4.1 Search Space

Most often one is looking for the best solution in a specific set of solutions. The space of all feasible solutions (the set of solutions among which the desired solution resides) is called search space (also state space). Each and every point in the search space represents one possible solution. Therefore, each possible solution can be “marked" by its fitness value, depending on the problem definition. With GA one looks for the best solution among a number of possible solutions- represented by one point in the search space; GAs are used to search the search space for the best solution, e.g., minimum. The difficulties in this case are the local minima and the starting point of the search. Figure 9.6 gives an example of search space.

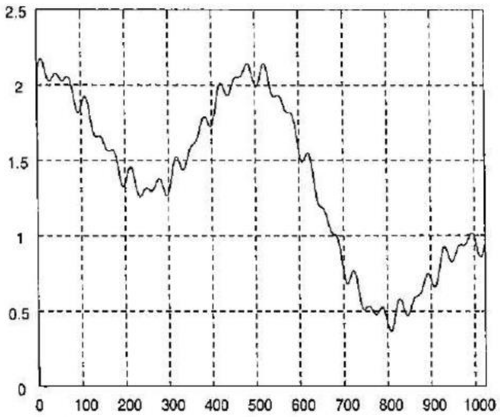


Figure 9.6 : An example of search space.

9.4.2 Genetic Algorithms World

GA raises again a couple of important features. First, it is a stochastic algorithm; randomness has an essential role in GAs. Both selection and reproduction need random procedures. A second very important point is that GAs always considers a population of solutions. Keeping in memory more than a single solution at each iteration offers a lot of advantages. The algorithm can recombine different solutions to the better ones and so it can use the benefits of assortment. A population-based algorithm is also very amenable for parallelization. The robustness of the algorithm should also be mentioned as something essential for the algorithm's success. To business refers to the ability to perform consistently well on a broad range of problem types. There is no particular requirement on the problem before using GAs, so it can be applied to resolve any problem. All these features make GA a really powerful optimization tool.

With the success of GAs, other algorithms making use of the same principle of natural evolution have also emerged. Evolution strategy, genetic programming are some algorithms similar to these algorithms. The classification is not always clear between the different algorithms, thus to avoid any confusion, they areal gathered in what is called Evo1ationary Algorithms.

The analogy with nature gives these algorithms something exciting and enjoyable. Their ability to deal successfully with a wide range of problem area, including those which are difficult for other methods to solve makes them quite powerful. However today, GAs is suffering from too much readiness. GA is a new field, and parts of the theory still have to be properly established. We can find almost as many opinions on GAs as there are researchers in this field. In this document, we will generally find the most current point of view. But things evolve quickly in GAs too, and some comments might not be very accurate in few years.

It is also important to mention GA limits in this introduction. Like most stochastic methods, GAs is not guaranteed to find the global optimum salmon to a problem; they are satisfied with finding "acceptably good" solutions to the problem. GAs are extremely general too, and so specific techniques for solving particular problems are likely to out-perform GAs in both speed and accuracy of the final result. GAs are something worth trying when everything else fails or when we know absolutely nothing of the search space. Nevertheless, even when such specialized techniques exist, it is often interesting to hybridize them with a GA in order to possibly gain some improvements. It is important always to keep an objective point of view; do not consider that GAs is a panacea for resolving all optimization problems. This warning is for those who might have the temptation to resolve anything with GA.

The proverb says "If we have a hammer, all the problems look like a nails." GAs do work and give excellent results if they are applied properly on appropriate problems.

9.4.3 Evolution and Optimization

To depict the importance of evolution and optimization process, consider a species Basilosaurus that originated 45 million years ago. The Basilosaurus was a prototype of a whale (Figure 9-7). It was about 9 m long and

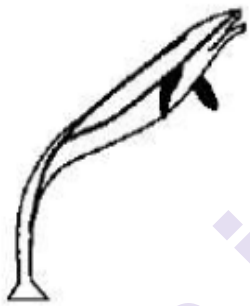


Figure 9-7 Basilosaurus.

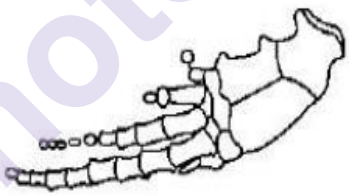


Figure 9·8 Tutsiops flipper.

weighed approximately 5 tons. It still had a quasi-independent head and posterior paws, and moved using undulatory movements and hunted small preys. Its anterior members were reduced to small flippers with an elbow inoculation; Movements in such a viscous element (water) are very hard and require big efforts. The anterior members of basilosaurus were not really adapted to swimming. To adapt them, a double phenomenon must occur the shortening of the "arm" with the locking of the elbow articulation and the extension of the fingers constitute the base structure of the flipper (refer Figure 9-8).

The image shows that two fingers of the common dolphin are hypertrophied to the detriment of the rest of the member. The basilosaurus was a hunter; it had to be fast and precise. Through time, subjects appeared with longer fingers and short arms.

They could move faster and more precisely than before, and therefore, live longer and have many descendants.

Meanwhile, other improvements occurred concerning the general aerodynamic like the integration of the head to the body, improvement of the profile, strengthening of the caudal fin, and so on, finally producing a subject perfectly adapted to the constraints of an aqueous environment. This process of adaptation and this morphological optimization is so perfect that nowadays the similarity between a shark, a dolphin or submarine is striking. The first is a cartilaginous fish (Chondrichryen) that originated in the Devonian period (-400 million years), long before the apparition of the first mammal. Darwinian mechanism hence generated an optimization process-hydrodynamic optimization- for fishes and others marine animals –auto dynamic optimization for pterodactyls, birds and bars. This observation is the basis of GAs.

9.4.4 Evolution and Genetic Algorithms

The basic idea is as follows: the genetic pool of a given population polemically contains the solution, or a better solution, to a given adaptive problem. This solution is not “active” because the genetic combination on which it relies split among several subjects. Only the association of different genomes can lead to the solution. Simplistically speaking, we could by example consider that the shortening of the paw and the extension of the fingers of our basilosaurus are controlled by two "genes." No subject has such a genome, but during reproduction and crossover, new genetic combination occur and, finally, a subject can inherit a "good gene “from both parents his paw is now a flipper.

Holland method is especially effective because he not only considered the role of mutation (mutations improve very seldom the algorithms), but also utilized genetic recombination (crossover): these recombination, the crossover of partial solutions, greatly improve the capability of the algorithm to approach, and eventually find, the optimum.

Recombination of sexual reproduction is a key operator for natural evolution. Technically, it takes two genotypes and it produces a new genotype by mixing the gene found in the originals. In biology, the most common form of recombination is crossover: two chromosomes are cur at one point and the halves are spliced to create new chromosomes. The effect of recombination is very important because it allows characteristics from two different parents to be assorted. If the father and the mother possess different good qualities, we would expect that all the good qualities will be passed to the child. Thus the offspring, just by combining all the

good features from its parents, may surpass its ancestors. Many people believe that this mixing of genetic material via sexual reproduction is one of the most powerful features of GAs. As a quick parenthesis about sexual reproduction, GA representation usually does not differentiate male and female individuals (without any perversity). As in many livings species (e.g., snails) any individual can be either a male or a female. Infact, for almost all recombination operators, mother and father are interchangeable.

Mutation is the other way to get new genomes. Mutation consists in changing the value of genes. In natural evolution, mutation mostly engenders non-viable genomes. Actually mutation is not a very frequent operator in natural evolution. Nevertheless, in optimization, a few random changes can be a good way of exploiting the search space quickly.

Through those low-level notions of genetic, we have seen how living beings store their characteristic information and how this information can be passed into their offspring. It very basic but it is more than enough to understand the GA theory.

Darwin was totally unaware of the biochemical basics of genetics. Now we know how the genetic inheritable information is coded in DNA, RNA, and proteins and that the coding principles are actually digital, much resembling the information storage in computers. Information processing is in many ways totally different, however. The magnificent phenomenon called the evolution of species can also give some insight into information processing methods and optimization, in particular. According to Darwinism, inherited variation is characterized by the following properties:

1. Variation must be copying because selection does not create directly anything, but presupposes a large population to work on.
2. Variation must be small-scaled in practice. Species do not appear suddenly.
3. Variation is undirected. This is also known as the blind watch maker paradigm.

While the natural sciences approach to evolution has for over a century been to analyse and study different aspects of evolution to find the underlying principles, the engineering sciences are happy to apply evolutionary principles, that have been heavily tested over billions of years, to arrack the most complex technical problems, including protein folding.

9.5 Genetic Algorithm vs. Traditional Algorithms

The principle of Gas is simple: emirate genetics and natural selection by a computer program: The parameters of the problem are coded most naturally as a DNA- like linear data structure, a vector or a suing. Sometimes, when the problem is naturally two or three dimensional, corresponding array structures are used.

A set, called *population,* of these problem-dependent parameter value vectors is processed by GA. To start, there is usually a totally random population, the values of different parameters generated by a random number generator. Typical population size is from few dozens to thousands. To do optimization we need a cost function or fitness function as it is usually called when Gas are used. By a fitness function we can select the best solution candidates from the population and delete the not so good specimens.

The nice thing when comparing GAs to other optimization methods is that the fitness function can be nearly anything that can be evaluated by a computer or even something that cannot In the latter case it might be a human judgment that cannot be seated as a crisp program, like in the case of eye witness, where a human being selects from the alternatives generated by GA. So, there are not any definite mathematical restrictions on the properties of the fitness fraction. It may be discrete, multimodal, etc.

The main criteria used to classify optimization algorithms are as follows: continuous/discrete, constrained/unconstrained and sequential/parallel. There is a clear difference between discrete and continuous problems. Therefore, it is instructive to notice that continuous methods are sometimes used to solve inherently discrete problems and vice versa. Parallel algorithms are usually used to speed up processing. There are, however, some *cases* in which it is more efficient to run several processors in parallel rather than sequentially. These cases include among others those in which there is high probability of each individual search run to get stuck into a local extreme.

Irrespective of the above classification, optimization methods can be further classified into deterministic and non-deterministic methods. In addition, optimization algorithms can be classified as local or global. Interns of energy and entropy local search correspond to entropy while global optimization depends essentially on the fitness, i.e., energy landscape.

GA differs from conventional optimization techniques in following ways:

1. GAs operate with coded versions of the problem parameters rather than parameters themselves, i.e., GA works with the coding of solution sec and nor with the solution itself.
2. Almost all conventional optimization techniques search from a single point, but GAs always operate on a whole population of points (strings), i.e., GA uses population of solutions rather than a single solution for searching. This plays a major role to the robustness of GAs. It improves the chance of reaching the global optimum and also helps in avoiding local stationary point.
3. GA uses fitness fiction for evaluation rather than derivatives. As a result, they can be applied to any kind of continuous or discrete optimization problem. The key point to be performed here is to identify and specify a meaningful decoding function.
4. GAs use probabilistic transition operates while conventional methods for continuous optimization apply deterministic transition operates, i.e., Gas does not use deterministic rules.

These are the major differences that exist between GA and conventional optimization techniques.

9.6 Basic Terminologies in Genetic Algorithm

The two distinct elements in the GA are individuals and populations. An individual is a single solution while the population *is* the set of individuals currently involved in the search process.

9.6.1 Individuals

An individual is a single solution. Individual groups together two forms of solutions as given below:

I. The chromosome which *is* the raw "genetic" information (genotype) that the GA deals.
2. The phenotype which is the expressive of the chromosome in the terms of the model.

A chromosome is subdivided into genes. A gene is the GA's representation of a single factor for a control factor. Each factor in the solution set corresponds to a gene in the chromosome. Figure 9-9 shows the representation of a genotype.

A chromosome should in some way contain information about the solution that it represents. The morphogenesis function associates each genotype with its phenotype. It simply means that each chromosome must define one unique solution, but it does not mean that each solution is encoded by exactly one chromosome. Indeed, the morphogenesis function *is* not necessarily objective, and it is even sometimes impossible (especially with binary representation). Nevertheless, the morphogenesis function should at least be subjective. Indeed;

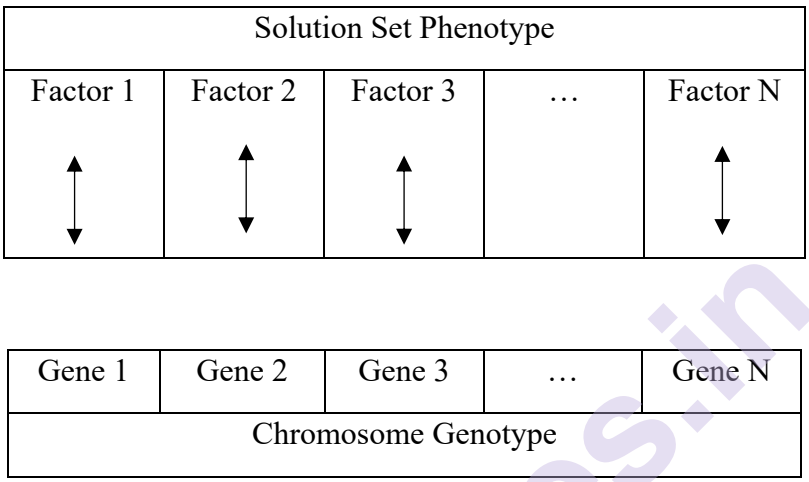


Figure 9·9 Representation of genotype and phenotype.

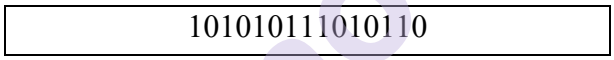


Figure 9·10 Representation of a chromosome.

all the candidate solutions of the problem must correspond to at least one possible chromosome, to be sure that the whole search space can be exploited. When the morphogenesis function that associates each chromosome to one solution is not injective. i.e., different chromosomes can encode the same solution, the representation is said to be degenerated. A slight degeneracy is not so worrying, even if the space where the algorithm is looking for the optimal solution is inevitably enlarged. Bur a too important degeneracy could be a more serious problem. It can badly affect the behaviour of the GA, mostly because if several chromosomes can represent the same phenotype, the meaning of each gene will obviously not correspond to a specif1c characteristic of the solution. It may add some kind of confusion in the search. Chromosomes encoded by bit strings are given in Figure 9-10.

9.6.2 Genes

Genes are the basic "instructions" for building a GA. A chromosome *is* a sequence of genes. Genes may describe possible solution to a problem, without actually being the solution. A gene is a bit string of arbitrary lengths. The bit string is a binary representation of number of intervals from a lower bound. A gene is the GNs representation of a single factor value for a control factor, where control factor must have an upper bound and a lower bound. This range can be divided into the number of intervals that can be expressed by the gene's bit string. A bit string of length *"n"* can represent (2^n1 - 1) intervals. The size of the interval would be (range)/ (2^n- 1).

The structure of each gene *is* defined in a record of phenotyping parameters. The phenotype parameters are instructions for mapping between genotype and phenotype. It can also be said as encoding a solution set into a chromosome and decoding a chromosome to a solution set. The mapping between genotype and phenotype is necessary to convert solution sets from the model into a form that the GA can work with, and for converting new individuals from the GA into a form that the model can evaluate. In a chromosome, the genes are represented as shown in Figure 9-11.

9.6.3 Fitness

The fitness of an individual in a GA is the value of an objective function for its phenotype. For calculating fitness, the chromosome has to *be* first decoded and the objective function has to be evaluated. The fitness

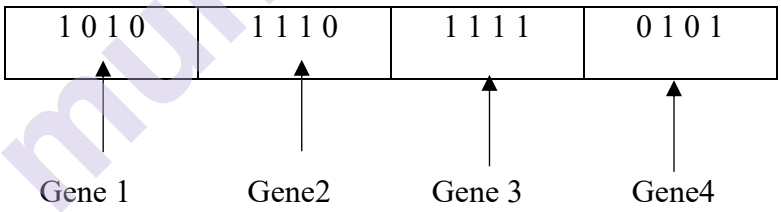


Figure 9·11 Representation of a gene.

not only indicates how good the solution is, but also corresponds to how does the chromosome is to the optimal one.

In the case of multicriterion optimization, the fitness function is definitely more difficult to determine. In multicriterion optimization problems, there is often a dilemma as how to determine if one solution is better than another. What should be done if a solution is better for one criterion but worse for another? But here, the trouble comes more from the definition of a "better" salmon rather than from how to implement a GA to resolve it. If sometimes a fitness function obtained by a simple combination of the different criteria can give good result, it supposes that

criterions can be combined in a consistent way. But, for more advanced problems, it may be useful to consider something like Pareto optimally or other ideas from multicriterian optimization theory.

9.6.4 Populations

A population is a collection of individuals. A population consists of a number of individuals being reseed, the phenotype parameters defining the individuals and some information about the search space. The two important aspects of population used in GAs are:

1. The initial population generation.
2. The population size.

For each and every problem, the population size will depend on the complexity of the problem. It is often a random initialization of population. In the case of a binary coded chromosome this means chat each bit is initialized to a random 0 or 1. However, there may be instances where the initialization of population is carried out with some known good solutions.

Ideally, the first population should have a gene pool as large as possible in order to be able to explode the whole search space. All the different possible alleles of each should be present in the population. To achieve this, the initial population is, in most of the cases, chosen randomly. Nevertheless, sometimes a kind of heuristic can be used to seed ·the initial population. Thus, the mean fitness of the population is already high and it may help the GA to find good solutions faster. Bur for doing this one should be sure that the gene pool is spillage enough. Otherwise, if the population badly lacks diversity, the algorithm will just explode a small part of the search space and never find global optimal solutions.

The size of the population raises few problems too. The larger the population is, the easier it is m explode the search space. However, it has been established that the time required by a GAm converge is $O(n \log n)$ function evaluations where n is the population size. We say that the population has converged when all the individuals are very much alike and further improvement may only be possible by mutation. Goldberg has also shown that GA efficiency to reach global optimum instead of local ones is largely determined by the size of the population. To sum up, a large population is quite useful. However, it requires much more computational cost memory and time. Practically, a population size of around 100 individuals is quite frequent, but anyway this size can be changed according to the time and the memory disposed on the machine compared to the quality of the result to be reached.

Population	Chromosome 1	1 1 1 0 0 0 1 0
	Chromosome 2	2 0 1 1 1 1 0 1 1
	Chromosome 3	1 0 1 0 1 0 1 0
	Chromosome 4	1 1 0 0 1 1 0 0

Figure 9-12 Population.

Population being combination of various chromosomes is represented as in Figure 9-12. Thus the population in Figure 9-12 consists of four chromosomes.

9.7 Simple GA

GA handles a population of possible solutions. Each solution is represented through a chromosome, which is just an abstract representation. Coding all the possible solutions into a chromosome is the first part, but certainly not the most straightforward one of a GA. A set of reproduction operators has to be determined, coo. Reproduction operators are applied directly on the chromosomes, and are used to perform mutations and recombination over solutions of the problem. Appropriate representation and reproduction operators are the determining factors, as the behaviour of the GA is extremely dependent on it. Frequency, it can be extremely difficult to find a representation that respects the structure of the search space and reproduction operators that are coherent and relevant according to the properties of the problems.

The simple form of GA is given by the following.

I. Scan with a randomly generated population.

2. Calculate the fitness of each chromosome in the population.

3. Repeat the following steps until n offspring's have been created:
   * Select a pair of parent chromosomes from the current population.
   * With probability *Pc* crossover the pair at a randomly chosen point co forms two offspring's.
   * Mutate le two offspring's at each locus with probability *Pm.*

4. Replace the current population with the new population.

5. Go to seep 2.

Now we discuss each iteration of this process.

Generation: Selection: is supposed to be able to compare each individual in the population. Selection is done by using a fitness function. Each chromosome has an associated value corresponding to the fitness of the solution it represents. The fitness should correspond to an evaluation of how good the candidate solution is.

The optimal solution is the one which maximizes the fitness function. GAs deal with the problems that maximize the fitness function. Bur, if the problem consists of minimizing a cost function, the adaptation is quite easy. Either the cost function can be transformed into a fitness function, for example by inverting it; or the selection can be adapted in such way that they consider individuals with low evaluation functions as better. Once the reproduction and the fitness function have been properly defined, a GA is evolved according to the same basic structure. It starts by generating an initial population of chromosomes. This first population must offer a wide diversity of genetic materials. The gene pool should be as large as possible so that any solution of the search space can be engendered. Generally, the initial population is generated randomly. Then, the GA loops over an iteration process to make the population evolve. Each iteration consists of the following steps:

1. *Selection:* The first step consists in selecting individuals for reproduction. This selection is done randomly with a probability depending on the relative fitness of the individuals so that best ones are often chosen for reproduction rather than the poor ones.
2. *Reproduction:* In the second step, offspring are bred by selected individuals. For generating new Chromosomes, the algorithm can use both recombination and mutation.
3. *Evaluation:* Then the fitness of the new chromosomes is evaluated.
4. *Replacement:* During the last step, individuals from the old population are killed and replaced by the new ones.

   The algorithm is stopped when the population converges toward the optimal solution.

   BEGIN/* genetic algorithm"/
   Generate initial population;
   Compare fitness of each individual;
   WHILE NOT finished DO LOOP
   BEGIN
   Select individuals from old generations
   For mating;
   Create offspring by applying
   Recombination and/or mutation
   The selected individuals;
   Compute fitness of the new individuals;
   Kill old individuals *w* make room for

New chromosomes and insert
Offspring in the new generalization;
IF Population has converged
THEN finishes: =TRUE;
END
END

Genetic algorithms are not too hard to program or understand because they are biological based. An example of a flowchart of a GA is shown in Figure 9-13.

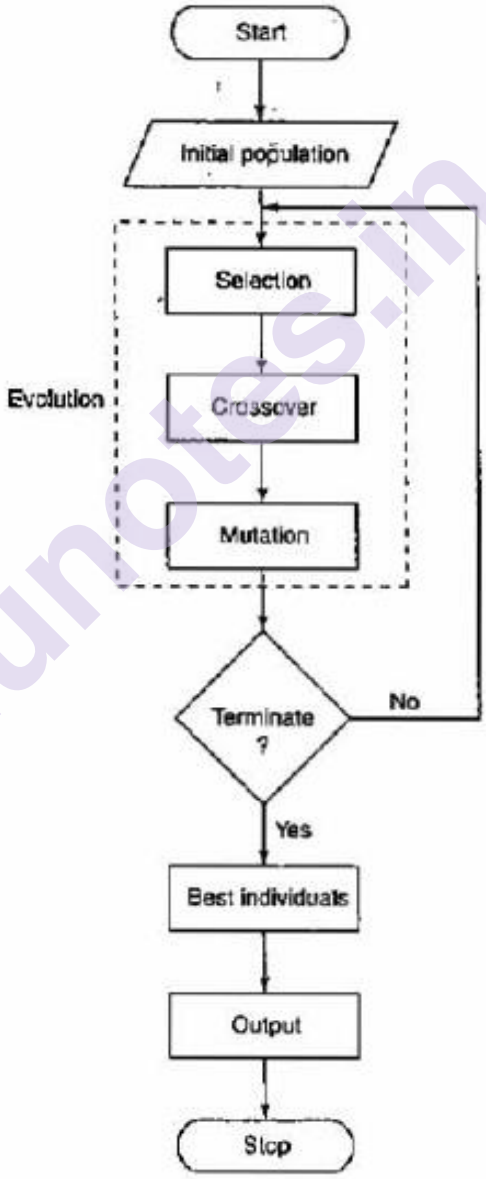


Figure 9·13 Flowchart for genetic algorithm.

9.8 General Genetic Algorithm

The general GA is as follows:

Step 1: *Create a random initial state:* An initial population is created from a random selection of solutions J (which are analogous to chromosomes). This is unlike the situation for symbolic AI systems, where the initial State in a problem is already given.

Step 2: *Evaluate fitness:* A value for fitness is assigned to each solution (chromosome) depending on how close it actually is *w* solving the problem (thus arriving to the answer of the desired problem).

(These "solutions" are not to be confused with "answers" to the problem; think of them as possible

Characteristics that the system would employ in order to reach the answer.)

Step 3 *Reproduce (and children mutate):* Those chromosomes with a higher fitness value are more likely to reproduce offspring (which can mutate after reproduction). The offspring is a product of the father and mother, whose composition consists of a combination of genes from the row (this process is known as "crossing over").

Step 4: *Nat generation:* If the new generation contains a solution that produces an output that is dose enough or equal to the desired answer then the problem has been solved. If this is not the case, then the new generation will go through the same process as their parents did. This will continue L until a solution is reached.

Table 9·2 : Fitness value for corresponding

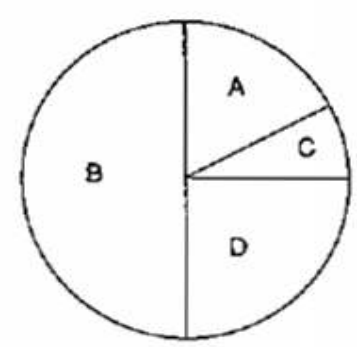
Chromosomes (Example 9.1)

Chromosome	Fitness
A : 00000110	2
B : 11101110	6
C : 00100000	1
D : 00110100	3

Table 9·3: Fitness value for corresponding

Chromosomes

Chromosome	Fitness
A : 01101110	5
B : 00100000	1
C : 10110000	3
D : 01101110	5



Fitness-proportionate selection

(Roulette wheel sampling)

Figure 9·14 Roulette wheel sampling for proportionate selection

Example 9.1: Consider 8-bitchromosomes with the following properties:

1. Fitness function *f(x)* = number of 1 bits in chromosome;
2. Population size $N = 4$;
3. Crossover probability $Pc = 0.7$;
4. Mutation probability $Pm = 0.001$;

Average fitness of population= 12/4 = 3.0.

1. If B and C are selected, crossover is not performed.
2. If B is mutated, then

$$B : 11101110 \rightarrow B' : 01101110$$

3. If B and D are selected, crossover is performed.

$$B : 11101110\ E : 10110100 \rightarrow D : 00110100\ F : 01101110$$

4. If E is mutated, then

$$E : 10110100 \rightarrow E' : 10110000$$

Best-fit string from previous population is lost, but the average fitness of population is as given below:

Average fitness of population 14/4 = 3.5

Tables 9-2 and 9-3 show the fitness value for the corresponding chromosomes and Figure 9-14 shows the Roulette wheel selection for the fitness proportionate selection.

9.9 Operators in Genetic Algorithm

The basic operators that are to be discussed in this section include: encoding, selection, recombination and mutation operators. The operators with their various types are explained with necessary examples.

9.9.1 Encoding

Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problem. For example, one can encode directly real or integer numbers.

9.9.1.1 Binary Encoding

The most common way of encoding is a binary string, which would be represented as in **Figure 9-9.**

Each chromosome encodes a binary (bit) suing. Each bit in the suing can represent some characteristics of the solution. Every bit string therefore is a solution but not necessarily the best solution. Another possibility is that the whole string can represent a number. The way bit strings can code differs from problem to problem.

Binary encoding gives many possible chromosomes with a smaller number of alleles. On the other hand, this encoding is not natural for many problems and sometimes corrections must be made after genetic operation is completed. Binary coded strings with Is and Os are mostly used. The length of the string depends on the accuracy. In such coding

1. Integers are represented exactly.
2. Finite number of real numbers can be represented.
3. Number of real numbers represented increases with string length.

9.9.1.2 Octal Encoding

This encoding uses string made up of octal numbers (0-7) (see Figure 9-16).

Chromosome 1	**1 1 0 1 0 0 0 1 1 0 1 0**
Chromosome 2	**I 0 1 1 1 1 1 1 1 1 1 0 0**

Figure 9·9 Binary encoding.

Chromosome 1	**03467216**
Chromosome 2	**9723314**

Figure 9·16 Octal encoding

Chromosome1	9CE7
Chromosome 2	3DBA

Figure 9·17 Hexadecimal encoding.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosomes	8 5 6 7 2 3 1 4 9

Figure 9·18 Permutation encoding.

9.9.1.3 Hexadecimal Encoding

This encoding uses string made up of hexadecimal numbers (0-9, A-F) (see Figure 9-17).

9.9.1.4 Permutation Encoding (Real Number Coding)

Every chromosome is a string of numbers, represented in a sequence. Sometimes corrections have to be done after genetic operation is complete. In permutation encoding, every chromosome is a suing of integer/real values, which represents number in a sequence.

Permutation encoding (Figure 9-18) is only useful for ordering problems. Even for this problem, some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e., have real sequence in it).

9.9.1.5 Value Encoding

Every chromosome is a string of values and the values can be anything connected *w* the problem. This encoding produces best results for some special problems. On the other hand, it is often necessary to develop new genetic operator's specific to the problem. Direct value encoding can be used in problems, where some complicated values, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult.

In value encoding (Figure 9-19), every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or characters to some complicated objects. Value encoding is very good for some special problems. On the other hand, for this encoding it is often necessary to develop some new crossover and mutation specific for the problem.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Figure 9-19 Value encoding.

9.9.1.6 Tree Encoding

This encoding is mainly used for evolving program expressions for genetic programming. Every chromosome is a tree of some objects such as functions and commands of a programming language.

9.9.2 Selection

Selection is the process of choosing two parents from the population for crossing. After deciding on an encoding, the next step is to decide how to perform selection, i.e., how *to* choose individuals in the population that will create offspring for the next generation and how many offspring each will create. The purpose of selection is in emphasize fitter individuals in the-population in hopes that their offspring have higher fitness. Chromosomes are selected from the initial population to be parents for reproduction. The problem is how to select these chromosomes. According to Darwin's theory of evolution the best ones survive to create new offspring. Figure **9-20** shows the basic selection process.

Selection is a method that randomly picks chromosomes out of the population according *to* their evaluation function. The higher the fitness function, the better chance that an individual will be selected. The selection pressure is defined as the degree to which the better individuals are favoured. The higher selection pressured, the more the better individuals are favoured. This selection pressure drives the GA to improve the population fitness over successive generations.

The convergence rate of GA is largely determined by the magnitude of the selection pressure, with higher selection pressures resulting in higher convergence rates. *GAs* should be able to identify optimal or nearly optimal solutions under a wide range of selection scheme pressure. However, if the selection pressure is too low, the convergence rate will be slow, and the GA will take unnecessarily longer to find the optimal solution. If the selection pressure is too high, there is an increased change of the GA prematurely converging to an incorrect (sub-optimal) solution. In addition to providing selection pressure, selection schemes should also preserve population diversity, as this helps to avoid premature convergence.

Typically we can distinguish two types of selection scheme, proportionate-based selection and ordinal based selection. Proportionate-based selection picks out individuals based upon their fitness values relative to the fitness of the other individuals in the population. Ordinal-based selection schemes select individuals not upon their raw fitness, bur upon their rank within the population. This requires that the selection pressure is independent of the fitness distribution of the population, and is solely based upon the relative ordering (ranking) of the population.

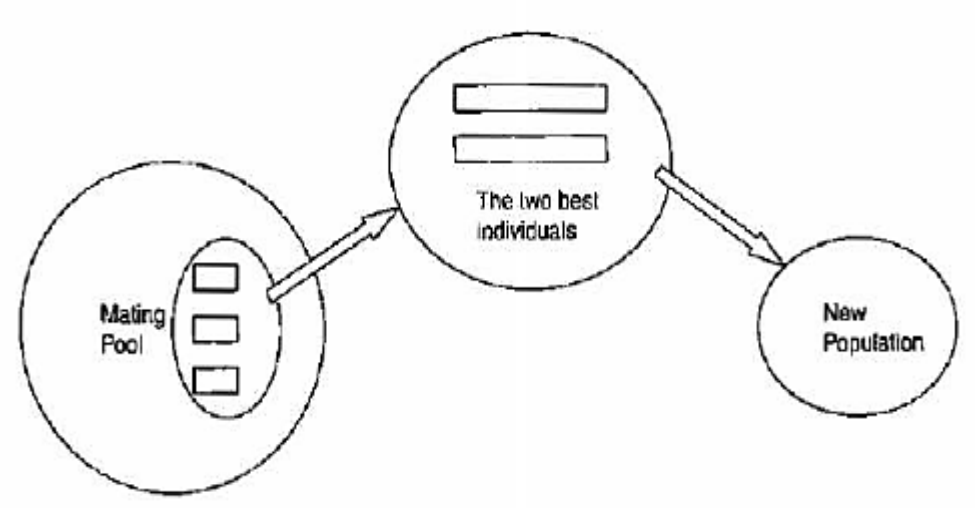


Figure 9-20 Selection.

It is also possible to use a scaling function to redistribute the fitness range of the population in order to adapt the selection pressure. For example, if all the solutions have their finesses in the range [999, 1000], the probability of selecting a better individual than any other using a proportionate based method will note important. If the fitness every individual is bringing to the range [0, 1] equitable, the probability of selecting good individual instead of bad one will be important. Selection has to be balanced with variation from crossover and mutation. Too strong selection means sub-optimal highly fit individuals will take over the population, reducing the diversity needed for change and progress; too weak selection will result in too slow evolution. The various selection methods are discussed in the following subsections.

9.9.2.1 Roulette Wheel Selection

Roulette selection is one of the traditional GA selection techniques. 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. The principle of Roulette selection is a linear search through a Roulette wheel with the store in the wheel weighted in proportion to the individual's fitness values. A target value is set, which is a random proportion of the sum of the finesses in the population. The population is stepped through until the target value is reached. This is only a moderately strong selection technique, since fir individuals are not guaranteed to be selected for, bur somewhat have a greater chance. A fit individual will contribute more to the target value, but if it does not exceed it, the next chromosome in line has a chance, and it may be weak. It is essential that the population not be sorted by fitness, since this would dramatically bias the selection.

The Roulette process can also be explained as follows: The expected value of an individual is individual's fitness divided by the actual fitness of the population. Each individual is assigned a slice of the Roulette wheel, the size of the slice being proportional to the individual's fitness. The wheel is spun *N* times, where *N* is the number of individuals in the population. On each spin, the individual under the wheel's marker is selected to be in the pool of parents for the next generation. This method is implemented as follows:

1. Sum the total expected value of the individuals in the population. Let it be T.
2. Repeat *N* times:
    i. Choose a random integer *"r"* between 0 and *T.*
    ii. Loop through the individuals in the population, summing the expected values, until the sum is greater than or equal to *"r."* The individual whose expected value puts the sum over this limit is the one selected.

Roulette wheel selection is easier to implement bur is noisy. The rate of evolution depends on the variance of fitness's in the population.

9.9.2.2 Random Selection

This technique randomly selects a parent from the population. In terms of disruption of genetic codes, random selection is a little more disruptive, on average, than Roulette wheel selection.

9.9.2.3 Rank Selection

The Roulette wheel will have a problem when the fitness values differ very much. If the best chromosome fitness is 90%, its circumference occupies 90% of Roulette wheel, and then other chromosomes have too few chances to be selected. Rank Selection ranks the population and every chromosome receives fitness from the ranking. The worst has fitness 1 and the best has fitness *N.* It results in slow convergence but prevents too quick convergence. It also keeps up selection pressure when the fitness variance is low. It preserves diversity and hence leads to a successful search. In effect, potential parents are selected and a tournament is held to decide which of the individuals will be the parent. There are many ways this can be achieved and two suggestions are:

1. Select a pair of individuals at random. Generate a random number *R* between 0 and 1. If *R* <ruse the first individual *as* a parent. If the *R* $\geq r$ then use the second individual as the parent. This is repeated to select the second parent. The value of *r* is a parameter to this method.
2. Select two individuals at random. The individual with the highest evaluation becomes the parent. Repeat to find a second parent.

9.9.2.4 Tournament Selection

An ideal selection strategy should be such that it is able to adjust its selective pressure and population diversity so as to fine-rune GA search performance. Unlike, the Roulette wheel selection, the tournament selection strategy provides selective pressure by holding a tournament competition among *Nu* individuals.

The best individual from the tournament is the one with the highest fitness, who is the winner of *Nu.* Tournament competitions 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 the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding genes. This method is more efficient and leads to an optimal solution.

9.9.2.5 Boltzmann Selection

SA is a method of function minimization or maximization. This method simulates the process of slow cooling of molten metal to achieve the minimum function value in a minimization problem. Controlling a temperature-like parameter introduced with the concept of Boltzmann probability distribution simulates the cooling phenomenon.

In Boltzmann selection, a continuously varying temperature controls the rate of selection according to a preset schedule. The temperature starts out high, which means that the selection pressure is low. The temperature is gradually lowered, which gradually increases the selection pressure, thereby allowing the GA to narrow in more closely to the best part of the search space while maintaining the appropriate degree of diversity.

A logarithmically decreasing temperature is found useful for convergence without getting stuck to a local minima state. However, it takes time to cool down the system to the equilibrium state.

Let f_{ax} be the fitness of the currently available best string. If the next string has fitness *f (X:)* such that $f(X;)>f_{max}$• then the new string is selected. Otherwise it is selected with Bole/Mann

$$P= \exp[-\{f_{max}- f(Xi)\} /T] \ldots\ldots\ldots\ldots(17)$$

probability where $T = To\ (1-\alpha)^k$ and $k = (1 + 100\ *g/G)$; *g* is the current generation number; *G* the maximum value of *g.* The value of CI:' can be chosen from the range [0, 1] and that of *T0* from the range [5, 100]. The final stare is reached when

computation approaches zero value *of T,* i.e., the global solution is achieved at this point.

The probability that the best string is selected and introduced into the mating pool is very high. However, Elitism can be used to eliminate the chance of any undesired loss of information during the mutation stage. Moreover, the execution time is *less.*

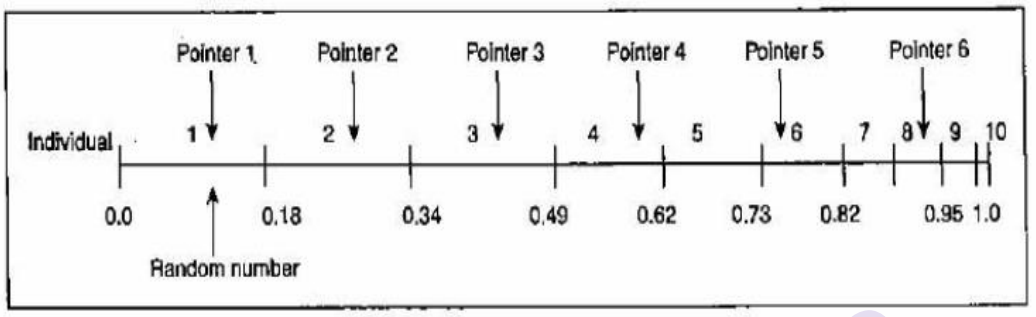


Figure 9·21 Stochastic universal sampling.

Elitism

The first best chromosome or the few best chromosomes are copied to the new population. The rest is done in a classical way. Such individuals can be lost if they are not selected to reproduce or if crossover or mutation destroys them. This significantly improves the GA's performance.

9.9.2.6 Stochastic Universal Sampling

Stochastic universal sampling provides zero bias and minimum spread. The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness exactly as in Roulette wheel selection. Here equally spaced pointers are placed over the line, as many as there are individuals to be selected. Consider *N Pointer* the number of individuals to be selected, then the distance between the pointers are *1/N Pointer* and the position of the first pointer is given by a randomly generated number in the range [0, *1/N Pointer].* For 6 individuals to be selected, the distance between the pointers is 1/6 = 0.167.

Figure 9-21 shows the selection for the above example.

Sample of 1 random number in the range [0, 0.167]: 0.1.
After selection the mating population consists of the individuals,

1,2,3,4,6,8

Stochastic universal sampling ensures selection of offspring that is closer to what is deserved as compared to Roulette wheel selection.

9.9.3 Crossover (Recombination)

Crossover is the process of taking two parent solutions and producing from them a child. After the selection (reproduction) process, the population is enriched with better individuals. Reproduction makes clones of good strings but does not create new ones. Crossover operator is applied to the mating pool with the hope that it creates a better offspring.

Crossover is a recombination operator that proceeds in three steps:

1. The reproduction operator selects at random a pair of two individual strings for the mating.
2. A cross site is selected at random along the string length.
3. Finally, the position values are swapped between the two strings following the cross site.

That is the simplest way how to do that is to choose randomly some crossover point and copy everything before this point &on the first parent and then copy everything after the crossover point from the other parent. The various crossover techniques are discussed in the following subsections.

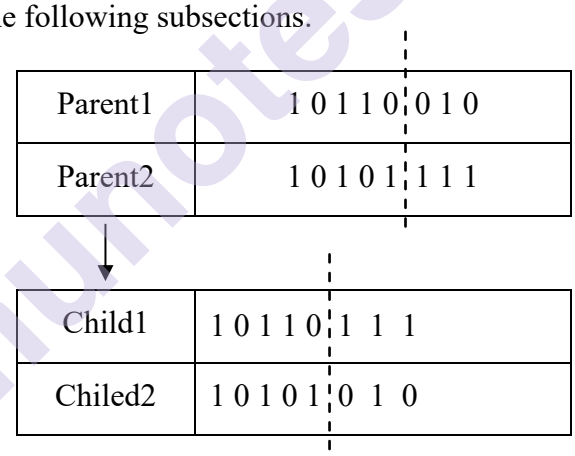


Figure 9·22: Single-point crossover

9.9.3.1 Single-Point Crossover

The traditional genetic algorithm uses single-point crossover, where the two mating chromosomes are cut once at corresponding points and the sections after the cuts exchanged. Here, a cross site or crossover point is selected randomly along the length of the mated strings and bits next to the cross sites are exchanged. Inappropriate site is chosen, bender children can be obtained by combining good parents, else it severely hampers string quality.

Figure 9-22 illustrates single point crossover and it can be observed that the bits next to the crossover point are exchanged to produce children. The crossover point can be chosen randomly.

9.9.3.2 Two Point *Crossover*

Apart from single point crossover, many different crossover algorithms have been devised, often involving more than one cut point. It should be noted that adding further crossover points reduces the performance of the GA. The problem with adding additional crossover points is that building blocks are more likely to be disrupted. However, an advantage of having more crossover points is that the problem space may be searched more thoroughly.

In two-point crossover, two crossover points are chosen and the contents between these points are exchanged between two mated parents.

In Figure 9-23 the dotted lines indicate the crossover points. Thus the comments between these points are

exchanged between the parents to produce new children for mating in the next generation.

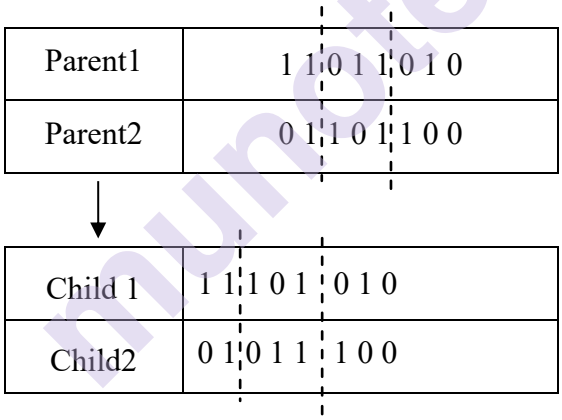


Figure 9-23 Two-point crossover

Originally, GAs were using one point crossover which cuts two chromosomes in one point and splices the two halves to create new ones. But with this one-point crossover, the head and the rail of one chromosome cannot be passed together to the offspring. If both the head and the rail of a chromosome contain good genetic information, none of the offspring obtained directly with one-point crossover will share the two good features. Using a two-point crossover one can avoid this drawback, and so it is generally considered better than one-point crossover. In fact, this problem can be generalized to each gene position in a chromosome. Genes that are close on a chromosome have more chance to be passed together to the offspring

obtained through N-points crossover. It leads to an unwanted correlation between genes next to each other. Consequently, the efficiency of an N-point crossover will depend on the position of the genes within the chromosome. In a genetic representation, genes that encode dependent characteristics of the solution should be close together. To avoid all the problem of genes locus, a good thing is to use a uniform crossover as recombination operator.

9.9.3.3 Multipoint Crossover (N·Point Crossover)

There are two ways in this crossover. One is even number of cross sires and the other odd number of cross sites. In the case of even number of cross sires, the cross sites are selected randomly around a circle and information's exchanged. In the case of odd number of cross sites, a different cross point is always assumed at the string beginning.

9.9.3.4 Uniform Crossover

Uniform crossover is quite different from the N-point crossover. Each gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a random generated binary crossover mask of the same length as the chromosomes. Where there is a 1 in the crossover mask, the gene miscopied from the first parent, and where there is a 0 in the mask the gene is copied from the second parent. Anew crossover mask is randomly generated for each pair of parents. Offspring, therefore, contain a mixture of genes from each parent. The number of effective crossing point is not fixed, but will average L/2 (where *L* is the chromosome length).

In Figure 9-24, new children are produced using uniform crossover approach. It can be noticed that while producing child 1, when there is a 1 in the mask, the gene is copied from parent 1 else it is copied from parent 2. On producing child 2, when there is a 1 in the mask, the gene is copied from parent 2, and when there is a 0 in the mask, the gene is copied from the parent 1.

9.9.3.5 Three Parent Crossover

In this crossover technique, three parents are randomly chosen. Each bit of the first parent is compared with the bit of the second parent. If both are the same, the bit is taken for the offspring; otherwise the bit from the third parent is taken for the offspring. This concept is illustrated in **Figure 9-25.**

Parent 1	**1 0 1 1 0 0 1 1**
Parent 2	**0 0 0 1 1 0 1 0**
Mask 1	**1 1 0 1 0 1 1 0**
Child 1	**1 0 0 1 1 0 1 0**
Child 2	**0 0 1 1 0 0 1 1**

Figure 9·24 Uniform crossover

Parent 1	**11010001**
Parent 2	**01101001**
Parent 3	**01101100**
Child	**01101001**

Figure 9·25 Three parent crossover

9.9.3.6 Crossover with Reduced Surrogate

The reduced surrogate operator constraints crossover to always produce new individuals wherever possible. This is implemented by restricting the location of crossover points such that crossover points only occur where gene values differ.

9.9.3.7 Shuffle Crossover

Shuffle crossover is related to uniform crossover. A single crossover position (as in single point crossover) is decreed. But before the variables are exchanged, they are randomly shuffled in both parents. After recombination, the variables in the offspring are unstuffed. This removes positional bias as the variables are randomly reassigned each time crossover is performed.

9.9.3.8 Precedence Preservative Crossover

Precedence preservative crossover (PPX) was independently developed for vehicle touting problems by Blanton and Wainwright (1993) and for scheduling problems by Bierwirth et al. (1996). The operator passes on precedence relations of operations given in two parental permutations to one offspring at the same race, while no new precedence relations are introduced. PPX is illustrated below for a. problem consisting of six operations A-F. The operator works as follows:

l. A vector of length Sigma, sub *i* == 1 to*mi,* representing the number of operations involved in the problem, is randomly filled with elements of the set {1, 2).

2. This vector defines the order in which the operations are successively drawn from parent I and parent 2.

3. We can also consider the parent and offspring permutations as lists, for which the operations "append “and "delete" are defined.

4. First we scan by initializing an empty offspring.

5. The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector.

6. After an operation is selected, it is deleted in both parents.

7. Finally the selected operation is appended to the offspring.

8. Step 7 is repeated until both parents are empty and the offspring domains all operations involved.

Note that PPX does not work in a uniform crossover manner due tithe "deletion-append" scheme used. Example is shown in Figure 9-26.

9.9.3.9 Ordered Crossover

Ordered two-point crossover is used when the problem is order based, for example in U shaped assembly line balancing, etc. Given two parent chromosomes, two random crossover points are selected partitioning

Parent permutation 1	A B C D E F
Parent permutation 2	C A B F D E
Select parent no. (1/2)	1 2 1 1 2 2
Offspring permutation	A C B D F E

Figure 9·26 Precedence preservative crossover (PPX).

Parent 1:4 2 | 1 3 | 65 Child 1:4 2 | 31 | 65
Parent 2:2 3 | 1 4 | 56 Child 2:2 3 | 41 | 56

Figure 9·27 Ordered crossover

them into a left, middle and right portions. The ordered two point crossover behaves in the following way: child 1 inherits its left and right section from· parent l, and its middle section is determined by the genes in the middle section of parent 1 in the order in which the values appear in parent 2. A similar process is applied to determine child 2. This is shown in Figure 927.

9.9.3. 10 Partially Matched Crossover

Finally matched crossover (PMX) can be applied usefully in the TSP. Indeed, TSP chromosomes are simply sequences of integers, where each integer represents a different city and the order represents the time at which acidy is visited. Under this representation, known *as* permutation encoding, we are only interested in labels and not alleles. It may be viewed as a crossover of permutations that guarantees that all positions arc found exactly once in each offspring, i.e., both offspring receive a full complement of genes, followed by the corresponding filling in of alleles from their parents. PMX proceeds as follows:

1. The two chromosomes are aligned.
2. Two crossing sires are selected uniformly at random along the strings, defining a marching section.
3. The matching section is used to effect a cross through position-by-position exchange operation.
4. Alleles are moved to their new positions in the offspring.

The following illustrates how PMX works.

Name 9 8 4 . 5 6 7 . 1 8 2 1 0 Allele 1 0 1 . 0 0 1 . 1 1 0 0
Name 8 7 1 . 2 3 1 0 . 9 5 4 6 Allele 1 1 1 . 0 1 1 . 1 1 0 1

Figure 9·28 Given strings

Consider the two strings shown in Figure 9-28, where the dots mark the selected cross points. The marching section defines the position-wise exchanges that must take place in both parents to produce the offspring. The exchanges are read from the marching section of one chromosome to that of the other. In the example illustrate in Figure 9-28, the numbers that exchange places are 5 and 2, 6 and 3, and 7 and 10. The resulting offspring are as shown in Figure 9-29. PMX is dealt in derail in the next chapter.

Name 9 8 4 . 2 3 1 0 . 1 6 5 7 Allele 1 0 1 . 0 1 0 . 1 0 0 1
Name 8 1 0 1 . 5 6 7 . 9 2 4 3 Allele 1 1 1 . 1 1 1 . 1 0 0 1

Figure 9·29 partially matched crossover.

9.9.3.11 Crossover Probability

The basic parameter in crossover technique is the crossover probability (P_t).Crossover probability is a parameter to describe how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is

crossover, offspring are made from parts of both parents' chromosome. If crossover probability is 100%, then all offspring are made by crossover. If it is O%, whole new- generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survive to next generation.

9.9.4 Mutation

After crossover, the strings are subjected to mutation. Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the tale of recovering the lost genetic materials as well as for randomly distributing genetic information. It is an insurance policy against the irreversible loss of genetic material. Mutation has been traditionally considered as a simple search operator. If crossover is supposed to exploit the current solution to find better ones, mutation is supposed to help for the exploitation of the whole search space. Mutation isvie¥1ed as a background operator to maintain genetic diversity in the population. It introduces new genetic structures in the population by randomly modifying some of its building blocks. Mutation helps escape from local minima's trap and maintains diversity in the population. It also keeps the gene pool well stocked, thus ensuring periodicity. A search space is said to be argotic if there is a non-zero probability of generating any solution from any population state.

There are many different forms of mutation for the different kinds of representation. For binary representation, a simple mutation can consist in inverting the value of each gene with a small probability. The probability is usually taken about 1/ *L,* where *L* is the length of the chromosome. It is also possible to implement kind of hill climbing mutation operators that do mutation only if it improves the quality of the solution. Such anoperawr can accelerate the search; however, care should be taken, because it might also reduce the diversity in the population and make the algorithm converge toward some local optima. Mutation of a bit involves flipping a bit, changing 0 to1 and vice-versa.

9.9.4 1 Flipping

Flipping of a bit involves changing 0 to 1 and 1 to 0 based on a mutation chromosome generated. Figure 9-30explains mutation flipping concept. A parent is considered and a mutation chromosome is randomly generated. For a 1 in mutation chromosome, the corresponding bit in parent chromosome is flipped (0 to 1 and1 to 0) and child chromosome is produced. In the case illustrated in Figure 9-

30, 1 occurs at 3 places of mutation chromosome, the corresponding bits in parent chromosome are flipped and the child is generated.

9.9.4.2 Interchanging

Two random positions of the string are chosen and the bits corresponding to those positions are interchanged (Figure 9.31).

Parent	**1 0 1 1 0 1 0 1**
Mutation chromosome	**1 0 0 0 1 0 0 1**
Child	**0 0 1 1 1 1 0 0**

Figure 9·30 Mutation flipping.

Parent	**1 0 1 1 0 1 0 1**
Child	**1 1 1 1 0 0 0 1**

Figure 9·31 Interchanging

Parent	**1 0 1 1 0 1 0 1**
Child	**1 0 1 1 0 1 1 1**

Figure 9·32 Reversing.

9.9.4.3 Reversing

A random position is chosen and the bits next to that position is reversed and child chromosome is produced (Figure 9-32).

9.9.4.4 Mutation Probability

An important parameter in the mutation technique is the mutation probability (P,). It decides how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) within any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed; if it is 0%, nothing is changed. Marion generally prevents the GA from falling into local extremes.

Mutation should not occur very often, because then GA will in fact change to ralidom search.

9.10 Stopping Condition for Genetic Algorithm Flow

In short, the various stopping condition are listed as follows:

1. *Maxim 11m generations:*. The GA stops when the specified number of generations has evolved.
2. *Elapsed time:* The genetic process will end when a specified time has elapsed.

   Note: If the maximum number of generation has been reached before the specified time has elapsed, the process will end.
3. *No change in fitness:* The genetic process will end if there is no change tithe population's best fitness for a specified number of generations.

   Note: If the maximum number of generation has been reached before the specified number of generation with too changes has been reached, the process will end.
4. *Stall generations: The* algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length "Stall generations."
5. *Stall time limit.* The algorithm stops if there is no improvement in the objective function during animerval of time in seconds equal to "Stall time limit."·

The termination or convergence criterion finally brings the search to a halt. The following are the few methods of termination techniques.

9.10.1 Best Individual

A best individual convergence criterion stops the search once the minimum fitness in the population drops below the convergence value. This brings the search *w* a faster conclusion, guaranteeing at least one good solmion.

9.10.2 Worst Individual

Worst individual terminates the search when the least fir individuals in the population have fitness less than me convergence criteria. This guarantees the entire population *w* be of minimum standard, although the best individual may not be significantly better than the worst. In this case, a stringent convergence value

may never be met, in which case the search will terminate after the maximum has been exceeded.

9.10.3 Sum of Fitness

In this termination scheme, the search is considered to have satisfaction converged when the sum of the fitness in the entire population is less than or equal to the convergence value in the population record. This guarantees that virtually all individuals in the population will be within a particular fitness range, although it is bener to pair this convergence criteria with weakest gene replacement, otherwise a few unfit individuals in the population will blow out the fitness sum. The population size has to be considered while setting the convergence value.

9.10.4 Median Fitness

Here at least half of the individuals will be better than or equal to the convergence value, which should give a good range of solutions to choose from.

9.11 Constraints in Genetic Algorithm

If the GA considered consists of only objective function and no information about the specifications of variable, then it is called *unconstrained optimization problem.* Consider, an unconstrained optimization problem of the form

$$\text{Minimize } f(x) = x^2 \quad \ldots\ldots\ldots\ldots(18)$$

and there is no information about *"x"* range. GA minimizes this function using its operators in random specifications.

In the case of constrained optimization problems, the information is provided for the variables under consideration. Constraints are classified as:

1. Equality relations.
2. Inequality relations.

GA geneses a sequence of parameters to be rested using the system under consideration, objective function (to be maximized or minimized) and the constraints. On running. the system, the objective function is evaluated and constraints are checked to see if there are any violations. If there are no violations, the parameter set is assigned the fitness value corresponding to the objective function evaluation. When the constraints are violated, the solution is infeasible and thus has no fitness. Many practical problems are constrained and it is very difficult to find a feasible point that is best. As a result, one should get some

information out of infeasible solutions, irrespective of their fitness ranking in relation tithe degree of constraint violation. Thesis performed in penalty method.

Penalty method is one where a constrained optimization problem is transformed to an unconstrained optimization problem by associating a penalty or cost with all constraint violations. This penalty *is* included in the objective function evaluation.

Consider the original constrained problem in maximization form:

$$\text{Maximize } f(x)$$

$$\text{Subject to } gi(x) \geq 0, \quad i = 1, 2, 3, \ldots, n$$

where x is a *k*-vector. Transforming this to unconstrained form:

$$\text{Maximize } f(x) + P \sum_{i=1}^{N} \varphi[g,(x)] \quad \ldots\ldots\ldots(19)$$

where Φ is the penalty function and *P* is the penalty coefficient. There exist several alternatives for this penalty function. The penalty function can be squared for all violated constraints. In certain situations, the unconstrained solution converges to the constrained solution as the penalty coefficient *p* rends to infinity.

9.12 Problem Solving Using Genetic Algorithm

9.12.1 Maximizing a Function

Consider the problem of maximizing the function,

$$f(x) = x^2 \quad \ldots\ldots(20)$$

where x is permitted to vary between 0 and 31. The steps involved in solving this problem are as follows:

Step I: For using GA approach, one must first code the decision variable *"x"* into a finite length string. I Using a five bit (binary integer) unsigned integer, numbers between 0(00000) and 31(11111) can be obtained.

The objective function here *is* $f(x) = x^2$ which is to be maximized. A single generation of a GA is performed here with encoding, selection, crossover and mutation. To start with, select initial population at random. Here initial population of size 4 is chosen, but any number of populations can be selected based on the requirement and application. Table 9-4 shows an initial population randomly selected.

Table 9·4 Selection

String No.	Initial population (randomly selected)	x value	Fitness $f(x) = x^2$	$Prob_i$	Percentage Probability (%)	Expected count	Actual count
1	0 1 1 0 0	12	144	0.1247	11.47	0.4987	1

String No.	Initial population (randomly selected)	x value	Fitness $f(x) = x^2$	$Prob_i$	Percentage Probability (%)	Expected count	Actual count
2	1 1 0 0 1	25	625	0.5411	54.11	2.1645	2
3	0 0 1 0 1	5	25	0.0216	2.16	0.0866	0
4	1 0 0 1 1	19	361	0.3126	31.26	1.2502	1
Sum			195	1.0000	100	4.0000	4
Average			288.75	0.2500	25	1.0000	1
Maximum			625	0.5411	54.11	2.1645	2

Step 2: Obtain the decoded *x* values for the initial population generated. Consider string 1.

$$01100 = 0 * 2^4 + 1 * 2^3 + I * 2^2 + 0 * 2^1 + 0 * 2^0$$
$$= 0+ 8 + 4 + 0 + 0$$
$$= 12$$

Thus for all the four strings the decoded values are obtained.

Step 3: Calculate the fitness or objective function. This is obtained by simply squaring the *"x"*

value, since the given function is $f(x) = x^2$ When $x = 12$, the fitness value is

$$f(x) = x^2 = (12)^2 = 144$$

For $x = 25$, $f(x) = x^2 = (25)^2 = 625$

and so on, until the entire population is computed.

Step 4: Compute the probability of selection,

$$Probi = \frac{f(x)_i}{\sum_{i=1}^{n} f(x)_i} \ldots.(21)$$

where *n* is the number of populations; *f(x)* is the fitness value corresponding to a particular

Individual in the population;

$\sum f(x)$ is the summation of all the fitness value of the entire population.

Considering string l,

$$\text{Fitness } f(x) = 144$$
$$\sum f(x) = 195$$

The probability that string 1 occurs is given by

$$P_1 = 144/195 = 0.1247$$

The percentage probability is obtained as

$$0.1247 * 100 = 12.47\%$$

The same operation is done for all the strings. It should be noted that summation of probability select is l.

Step 5: The next step is to calculate the expected count, which is calculated as

$$\text{Expected count} = \frac{f(x)_i}{[Avg\ f(x)]_i} \quad \ldots\ldots\ldots\ldots(22)$$

Where

$$(Avg\ f(x))_i = \left[\frac{\Sigma_{i=1}^{n} f(x)_i}{\bar{n}}\right] \quad \ldots\ldots\ldots\ldots\ldots..(23)$$

For string 1,

Expected count = Fitness/Average = 144/288.75 = 0.4987

We then compute the expected count for the entire population. The expected count gives an idea of which population can be selected for further processing in the mating pool.

Step 6: Now the actual count is to be obtained to select the individuals who would participate in the crossover cycle using Roulette wheel selection. The Roulette wheel is formed as shown Figure 9-33.

The entire Raul we wheel covers 100% and the probabilities of selection as calculated in step 4 for the entire populations are used as indicators to fit into the Roulette wheel. Now the wheel may be spun and the number of occurrences of population is noted to get actual count.

1. String I occupies 12.47%, so there is a chance for it to occur at least once. Hence its actual count may be I.
2. With string 2 occupying 54.11% of the Roulette wheel, it has a fair chance of being selected twice. Thus its actual count can be considered as 2.
3. On the other hand, string 3 has the least probability percentage of 2.16%, so their occurrence for next cycle is very poor. As a result, ire actual count is 0.

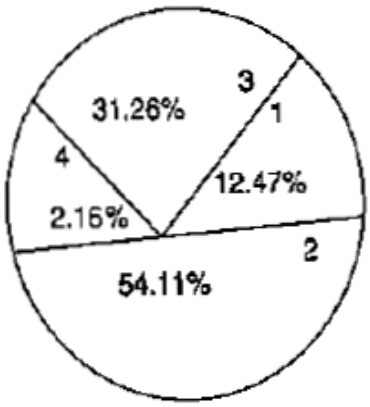


Figure 9·33 Selection using Roulette wheel.

Table 9·5 Crossover

String no.	Mating Pool	Crossover point	Offspring after crossover	*x* value	Fitness value $f(x) = x^2$
1	0 1 1 0 0	4	0 1 1 0 1	13	169
2	1 1 0 0 1	4	1 1 0 0 0	24	576
3	1 1 0 0 1	2	1 1 0 1 1	27	729
4	1 0 0 1 1	2	1 0 0 0 1	17	289
Sum					1763
Average					440.75
Maximum					729

4. String 4 with 31.26% has at least one chance for occurring while Roulette wheel is spun, thus its actual count is 1.

The above values of actual count are tabulated as shown is Table 9-5.

Step 7: Now, write the mating pool based upon the actual count as shown in Table 9-5.

The actual count of string no. 1 is I; hence it occurs once in the mating pool. The actual count of string no. 2 is 2, hence it occurs twice in the mating pool. Since the actual count of string no. 3 is 0, it does not occur in the mating pool. Similarly, the actual count of string no. 4 being I, it occurs once in the mating pool. Based on this, the mating pool is formed.

Step 8: Crossover operation is performed *w* produce new offspring (children). The crossover point is specified and based on the crossover point, single-point crossover is performed and new offspring is produced. The parents are

Parent 1	0 1 1 0 0
Parent 2	1 1 0 0 1

The offspring is produced as

Offspring 1	0 1 1 0 1
Offspring 2	1 1 0 0 0

In a similar manner. crossover is performed for the next strings.

Step 9: After crossover operations. new offspring are produced and *"x* .. value.\ are decoded and I mess is calculated.

Step 10: In this step, mutation operation is performed to produce new offspring. After crossover operation. As discussed in Section 9.9.4.1 mutation-Aipping operation is performed and new offspring are produced. Table 9-6 shows the new offspring after mutation. Once the offspring are obtained L after mutation, they are decoded ta*x* value and the fitness values are computed.

This completes one generation. The mutation is performed on a bit-bit by basis. The crossover probability and mutation probability were assumed to be 1.0 and 0.001, respectively. Once selection, crossover and mutation are performed, the new popular ion is now ready to be rested. This is performed by decoding the new strings created by the simple GA after mutation and calculates the fitness function values from the *x* values thus decoded. The results for successive cycles of simulation are shown in Tables 9-4 and 96.

Table 9-6 Mutation

String no.	Offspring after crossover	Mutation chromosomes for Ripping	Offspring after crossover	*x* value	Fitness $f(x) = x^2$
1	0 1 1 0 1	1 0 0 0 0	1 1 1 0 1	29	841
2	1 1 0 0 0	0 0 0 0 0	1 1 0 0 0	24	576
3	1 1 0 1 1	0 0 0 0 0	1 1 0 1 1	27	729
4	1 0 0 0 1	0 0 1 0 0	1 0 1 0 0	20	400
Sum					2546
Average					636.5
Maximum					841

From the rabies, it can be observed how GAs combine high-performance notions to achieve bercer performance. In the rabies, it can be noted how maximal and average performances have improved in the new population. The population average fitness has improved from 288.75 to 636.5 in one generation. The maximum fitness has increased from 625 to 841 during the same period. Though random processes make this best solution, its improvement can also be seen successively. The best string of the initial population (1 1 0 0 1) receives no chances for its existence because of its high, above-average performance. When this combines at random with the next highest string (1 0 0 1 1) and is crossed at crossover point 2 (as shown in Table 9-5), one of the resulting strings (1 1 0 1 1) proves to be a very best solution indeed. Thus after mutation at random, a new offspring (1 1 1 0 1) is produced which is an excellent choice.

This example has shown one gene ion of a simple GA.

9.13 The Schema Theorem

In this section. we will formulate and prove the fundamental research on the behaviour of GAs- the so-called Schema Theorem. Although being completely incomparable with convergence research's for conventional optimization methods, it still provides valuable insight two the intrinsic principles of GAs. Assume a GA with proportional selection and an arbitrary bur fixed fitess function *f* Let us make the following notations:

1. The number of individuals which fulfil *H* at time step tare denoted as

$$rH,r = \backslash Br \cap H\backslash$$

2. The expression *f* (t) refers to the observed average fitness at time t:

$$\bar{f}(t) = \frac{1}{m}\sum_{i=1}^{m} f(b_{i.t})$$

3. The term *f (H,* t) stands for the observed average fitness of schema *H* in time step t:

$$\bar{f}(H,t) = \frac{1}{rH.s}\sum_{i\in\{Ab_{ix}\in H\}} f(b_{i.t})$$

Theorem (Schema Theorem - Holland 1975). Assuming we consider a simple GA. the following inequality holds for eveq schema *H:*

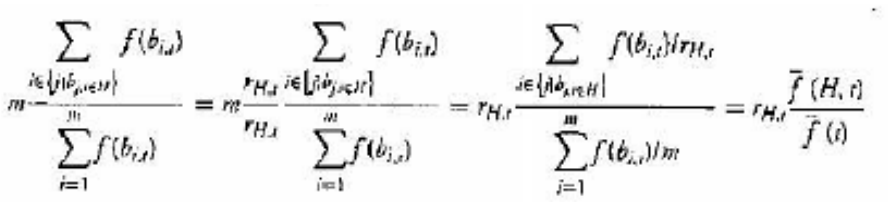
$$E[r_{H,t+1}] \geq r_{H,t}\frac{\bar{f}(H,t)}{\bar{f}(t)}\left(1 - p_c\frac{\delta(H)}{n-1}\right)(1-p_M)^{O(H)}$$

Proof. The probability that we select an individual fulfilling H is

$$\frac{\sum_{i\in\{Ab_{i,t}\}} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})}$$

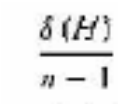
This probability does not change throughout the execution of the selection loop. Moreover, each of the*m* individuals is select::d independent of the others. Hence the number of selected individuals. which fulfil *H,* is binomially distributed with

sample amount *m* and the probability. We obtain, therefore, that the expected number of selected individuals fulfilling H is



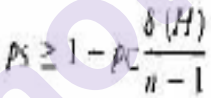
……(24)

If two individuals at crossed, which bmh fulfil *H,* the two offspring's again fulfil *H.* The number of strings fulfilling *H* can only decrease if one string. which fulfils *H,* is crossed with a string which does not fulfil *H.* but, obviously, only if the cross sire is chosen somewhere in between the specifications of *H.* The probability that the cross sire is chosen within the detaining length of H is



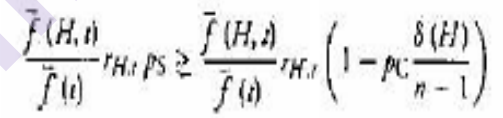
……..(25)

Hence the survival probability *ps* of *H,* i.e., the probability that a string fulfilling *H* produces an offspring also fulfilling *H.* can be estimated as follows (crossover is only done with probability*):*



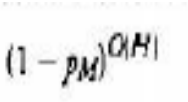
……..(26)

Selection and crossover are carried our independently, so we may compute the expected number of strings fulfilling *H* after crossover simply as



……..(27)

After crossover, the number of strings fulfilling H can only decrease if a suing fulfilling *His* ahered by mutation at a specification of *H.* The probability that all specifications of *H* remain untouthed by mutation is obviously



……..(28)

The arguments in the proof of the Sthema Theorem can be applied analogously too many other crossover and mutation operations.

9.13.1 The Optimal Allocation of Trials

The Sthema Theorem has provided the insight that building blocks receive exponentially increasing trials in future generations. The question remains, however, why this could be a good strategy. This leads to an important and well analyzed problem from statistical decision theory- the two-armed bandit problem and its generalization, the k-armed bandit problem. Although this seems like a detour from our main concern, we shall soon understand the connection to GAs.

Suppose we have a gambling machine with two slots for coins and two arms. The gambler can deposit the coin either two the left or the right slot. After pulling the corresponding arm, either a reward is given or the coin is lost. For mathematical simplicity, we just work with outcomes, i.e., the difference between the reward (which can be zero) and the value of the coin. Let us assume that the left arm produces an outcome with mean value $\mu 2$ and a variance $\sigma 2^2$ while the right arm produces an outcome with mean value $\mu 2$ and variance $\sigma 1^2$. Without loss of generality, although the gambler does not know this, assume that $\mu 1 \geq \mu 2$·

Now the question arises which arm should be played. Since we do not know beforehand which arm is associated with the higher outcome, we are faced with an interesting dilemma. Not only must we make a sequence of decisions about which arm to play, we have to collect, at the same time, information about which is the bener arm. This trade-off between exploitation of knowledge and its exploitation is the key issue in this problem and, as rums out later, in GAs, too.

A simple approach to this problem is to separate exploitation from exploitation. More specifically, we could perform a single experiment at the beginning and thereafter make an irreversible decision that depends on the results of the experiment. Suppose we have N coins. If we allocate an equal number *n* {where *2n* N) of trials to both arms, we could allocate the remaining *N- 2n* uials to the observed bener arm. Assuming we know all involved parameters, the expected loss is given as

$$L(N. n) = (\mu 1 - \mu 2)\{(N - n)q(n) + n[1 - q(n)l\}$$

where *q(n)* is the probability that the worst arm is the observed best arm after *2n* expetimental trials. The underlying idea is obvious: In case that *we* observe that the worse arm is the best, which happens with probability *q(n),* the total number of trials allothed to the right arm is *N - 11.* The loss is, therefore, *(J1 1 -Jl2)(N- n).* In the reverse case where we actually observe that the best arm is the best, which happens with probability I - *q(n),* the loss is only whir we get less because we

played the worse arm *11* times, i.e., (Ill -*112)11.* Taking the central limit theorem into account, we can approximate *q (n)* with the rail of a normal distribution:

$$q(n) \approx \frac{1}{\sqrt{2\pi}} \frac{e^{-c^2/2}}{c}$$

where ……..(29)

$$c = \frac{\mu_1 - \mu_2}{\sqrt{\sigma_1^2 + \sigma_2^2}} \sqrt{n}$$

Now we have m specify a reasonable experiment size n. obviously, if we choose *n* = 1, the obtained information is potentially unreliable. If we choose, however, *n* = *N/2* there are no trials left to make use of the information gained though the experimental phase. What we see is again the trade-off between exploitation with almost no exploitation *(n* = 1) and exploitation without exploitation {n = N/2). It does not take a Nobel prize winner to see that the optimal way is somewhere in the middle. Holland has studied this problem in detail. He came to the conclusion that the optimal strategy is given by the following equation:

where

$$n^4 \approx b^2 \ln\left(\frac{N^2}{8\pi b^4 \ln N^2}\right)$$

……..(30)

$$b = \frac{\sigma_1}{\mu_1 - \mu_2}$$

Making a few transformations, we obtain that

$$N - n^4 \approx \sqrt{8\pi b^4 \ln N^2}\, e^{n^4/2b^2}$$ ……..(31)

That is, the optimal strategy is m allocate slightly more than an exponentially increasing number of trials to the observed best arm. Although no gambler is able to apply this strategy in practice, because it requires knowledge of the mean values Jll and *JLz,* we still have found an important bound of performance a decision strategy should try to approach.

A GA, although the direct connection is not yet fully clear, actually comes close to this ideal, giving at least an exponentially increasing number of trials to the observed best building blocks. However, one may still wonder how the two-armed bandit problem and GAs are related. Let us consider an arbitrary string position. Then there are two sthemata of order one which have their only specification in this position. According to the Sthema Theorem, the GA implicitly decides between

these two sthemata, where only incomplete data are available (observed average fitness values). In this sense, a GA solves a lot of two-armed problems in parallel.

The Sthema Theorem, however, is not restricted to sthemata of order one. Looking at competing sthemata (different sthemata which are specified in the same positions). We observe that a GA is solving an enormous number of k-armed bandit problems in parallel. The k-armed bandit problem, although much more complicated, is solved in an analogous way - the observed better alternatives should receive an exponentially increasing number of trials. This is exactly what a GA does.

9.13.2 Implicit Parallelism

So far we have discovered two distinct, seemingly conflicting views of genetic algorithms:

1. The algorithmic view that GAs operate on strings;
2. The sthema-based interpretation.

So, we may ask what a GA really processes, strings or sthemata? The answer is surprising: Both. Now a day, the common interpretation is chat a GA processes an enormous amount of sthemata implicitly. This is accomplished by exploiting the currently available, incomplete information about these sthemata continuously, while trying to explore more information about them and other, possibly better sthemata.

This remarkable property is commonly called the implicit parallelism of GAs. A simple GA has only *m* structures in one time step, without any memory or bookkeeping about the previous generations. We will now ny to get a feeling how many sthemata a GA actually processes.

Obviously, there are 3^n sthemata of length *n.* A single binary string fulfils *n* sthema of order 1, (2^n) sthemata of order 2, in general, (k^n) sthemata of order *k.* Hence, a string fulfils

$$\sum_{k=1}^{n} \binom{n}{k} = 2^n \qquad \text{……..(32)}$$

Theorem. Consider a randomly generated start population of a simple GA and let *e* E (0, 1) be a fixed error bound. Then sthemata of length

$$1, \; < E\,(n - l) + 1$$

have a probability of at least (1-∈) to survive one-point crossover (compare with the proof of the Sthema Theorem). If the population size is chosen as $m = 2^l/2$, the number of sthemata, which survive for the next generation, is of order $O(m^3)$.

9.14 Classification of Genetic Algorithm

There exist wide variety of GAs including simple and general GAs discussed in Sections 9.4 and 9.5, respectively. Some or her variants of GA are discussed below.

9.14.1 Messy Genetic Algorithms

In a "classical" GA, the genes are encoded in a fixed order. The meaning of a single gene is determined by its position inside the string. We have seen in the previous chapter that a GA is likely to converge well if the optimization risk can be divided two several short building blocks. What, however, happens if the coding is chosen such that couplings occur between distant genes? Of course, one-point crossover rends to disadvantage long sthemata {even if they have low order) over short ones.

Messy GAs try *w* overcome this difficulty by using a variable-length, position-independent coding. The key idea is to append an index to each gene which allows identifying its position. A gene, therefore, is no longer represented as a single allele value and a fixed position, but as a pair of an index and an allele. Figure 9-34(A) shows how this "messy" coding works for a string of length 6.

Since with the help of the index we can identify the genes uniquely, genes may be swapped arbitrarily without changing the meaning of the string. With appropriate genetic operations, which also change the order of the paits, the GA could possibly group coupled genes to get her automatically.

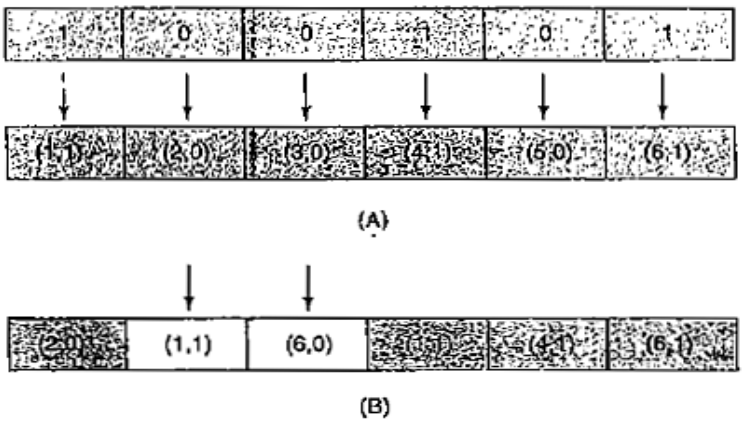


Figure 9-34 (A) Messy coding and (B) positional preference; Genes with indices 1 and 6 occur twice, the firm occurrences are used.

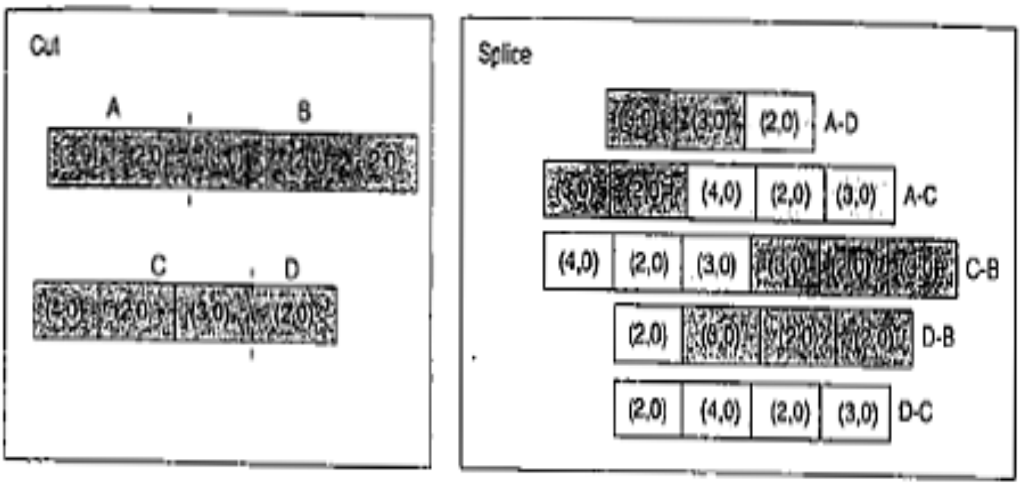


Figure 9·35 the cut and splice operation.

Owing to the free arrangement of genes and the variable length of the encoding, we can, however, run into. Problems, which do not occur, in a simple GA. First of all, it can happen that there are two entries in a string, which correspond to the same index but have conflicting alleles. The most obvious way to overcome this "over-specification" is positional preference- the first entry, which refers to a gene, is taken. Figure 9-34(B) shows an example. The reader may have observed that the genes with indices 3 and 5 do not occur at all in the example in Figure 9-34(B). This problem of “under specification" is more complicated and its solution is not as obvious as for over=-specification. Of course, a lot of variants are reasonable.

One approach could be to theck all possible combinations and to rake the best one (fork missing genes, there are *2k* combinations). With the objective to reduce this effort, Goldberg ct al. have suggested using so-called competitive templates for finding specifications for missing genes. It is nothing else than applying a local hill climbing method with random initial value to the *k* missing genes.

While messy GAs usually work with the same mutation operator as simple GAs (every allele is altered with a low probability pM), the crossover operator is replaced by a more general cut and splice operator which also allows to mate parents with different lengths. The basic idea is to choose cut sites for both parents independently and to splice the four fragments. Figure 9-35 shows an example.

9.14.2 Adaptive Genetic Algorithms

Adaptive GAs are those whose parameters, such as the population size, the crossing over probability, or the mutation probability, are varied while the GA is running. A simple variant could be the following: The mutation rate is changed according to changes in the population- the longer the population does not improve, the higher

the mutation rare is chosen. Vice versa, it is decreased again as soon as an improvement of the population occurs.

9.14.2.1 Adaptive Probabilities of Crossover and Mutation

It is essential to have two characteristics in GAs for optimizing multimodal functions. The first characteristic is the capacity to converge wan optimum (local or global) after locating the region containing the optimum. The second characteristic is the capacity to explore new regions of the solution space in search of the global optimum. The balance between these characteristics of the GA is dictated by the values of *Pw* and *Pn* and the type of crossover employed. Increasing values *of Pw* and *Pr* promote exploitation at the expense of exploitation. Moderately large values of *Pc* (in the range 0.5-1.0) and small values of *Pw* (in the range 0.001-0.05) are commonly employed in GA practice. In this approach, we aim at achieving this trade-off between exploitation and exploitation in a different manner, by varying, and *Pm* adaptively in response to the fitness values of the solutions; *Pr* and *Pm* are increased when the population tends to get stuck at a local optimum and are decreased when the population is scattered in the solution space.

9.14.2.2 Design of Adaptive pc and Pm

To vary *Pr* and *Pm* adaptively for preventing premature convergence of the GA to a local optimum, it is essential to identify were the GA is converging to an optimum. One possible way of detecting is to observe average fitness value *f* of the population in relation to the maximum fitness value *fmax* of the population. The value *f*max - *f* is likely to be less for a population that has converged to an optimum solution than that for a population scattered in the solution space. We have observed the above property in all our experiments with GAs, and Figure 9-36 illustrates the property for a typical case. In Figure 9-36 we notice that *f*max – *f* decreases when the GA converges to a local optimum with a fitness value of 0.5. (The globally optimal solution has a fitness value of 1.0.) We use the difference in the average and maximum fitness value, *f*max - *f, as* a yardstick for detecting the convergence of the GA. The values of *Pc* and *Pm* are varied depending on the value of *fmax. - f.* Since *Pc* and *Pm* have to be increased when the GA converges to a local optimum, i.e., when *f*max - *f* decreases, *Pc* and *Pm* will have to be varied inversely with *f*max - *f.* The expressions that we have chosen for *Pc* and *Pm* are of the form

$$Pc = k_1 / (fmax - f)$$

$$Pm = k_2 / (fmax - f)$$

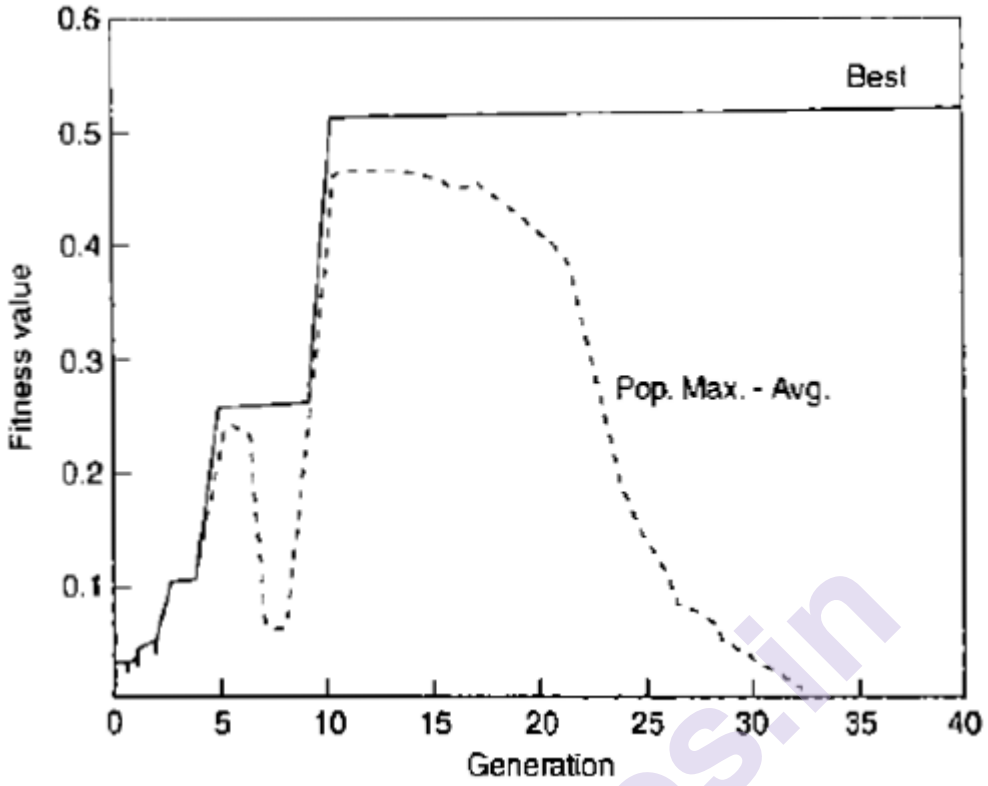


Figure 9·36 Variation of *f*max – *f* and *f* best (best fitness).

It has to be observed in the above expressions that *Pc* and *Pm* do not depend on the fitness value of any particular solution, and have the same values for all the solution of the population. Consequently, solutions with high fitness values as well as solutions with low fitness values are subjected to the same levels of mutation and crossover. When a population converges to a globally optimal solution (or even a locally optimal solution), *Pc* and *Pm* increase and may cause the disruption of the neat-optimal solutions. The population may never converge to the global optimum. Though we may prevent the GA from getting stuck at a local optimum, the performance of the GA (in terms of the generations required for convergence) will certainly deteriorate.

To overcome the above-stated problem, we need to preserve "good" solutions of the population. This can be achieved by having lower values of *Pc* and *Pm* for high fitness solutions and higher values of *Pc* and *Pm* for low fitness solutions. While the high fitness solutions aid in the convergence of the GA, the low fitness solutions prevent the GA from getting stuck at a local optimum. The value of *Pm* should depend not only on *f*max – *f* but also on the fitness value *f* of the solution. Similarly, *Pc* should depend on the fitness values of both the parent solutions. The closer *f* is to *f*max the smaller *Pm* should be, i.e., *Pm* should vary directly *as fmax – f.*

Similarly, *Pc* should vary directly as *f*max – *f'*, where *f'* is the larger of the fitness value of the solutions to be crossed. The expressions for *Pc* and *Pm* now take the forms

$$p_c = k_1[(f_{max} - f')/(f_{max} - \bar{f})],\ k_1 \leq 1.0$$
$$p_m = k_2[(f_{max} - f)/(f_{max} - \bar{f})],\ k_2 \leq 1.0 \quad \text{……..(33)}$$

(Here k_1 and k_2 have to be less than 1.0 to constrain *Pc* and *Pm* to the range 0.0-1.0.)

Note that *Pc* and *Pm* are zero for the solution with the maximum fitness. *Alsop, = k1* for a solution with *f = f,* and Pm = k_2 for a solution with *f = f.* For solution with subaverage fitness values, i.e., *f < f, Pc* and *Pm* might assume values larger than 1.0. To prevent the overshooting of *Pc* and *Pm* beyond 1.0, we also have the following constraints:

$$p_c = k_3,\ f' \leq \bar{f}$$
$$p_m = k_4,\ f \leq \bar{f} \quad \text{……..(34)}$$

where $k_3, k_4 \leq 1.0$.

9.14.2.3 Practical Considerations and Choice of Values for k_1, k_2, k_3 and k_4

In the previous subsection, we saw that for a solution with the maximum fitness value *Pc* and *Pm* are both zero. The best solution in a population is transferred undisrupted into the next generation. Together with the selection mechanism, this may lead to an exponential growth of the solution in the population and may cause premature convergence. To overcome the above-mued problem, we introduce a default mutation rate (of 0.005) for every solution in the Adaptive Genetic Algorithm (AGA).

We now discuss the choice of values for k1, *kz, k3* and *k4.* For convenience, the expressions for *Pc* and *Pm* are given as

$$p_c = k_1(f_{max} - f')/(f_{max} - \bar{f}),\ f \geq \bar{f}$$
$$p_c = k_3,\ f < \bar{f}$$
$$p_m = k_2(f_{max} - f')/(f_{max} - \bar{f}),\ f \geq \bar{f}$$
$$p_m = k_4,\ f < \bar{f} \quad \text{……..(35)}$$

where $k_1, k_2, k_3, k_4 \leq 1.0$.

It has been well established in GA literature that moderately large values of *Pc* (0.5 < *Pc* < 1.0) and small values of *Pm* (0.001 < *Pm* < 0.05) are essential for the successful working of GAs. The moderately large values of *Pc* promote the extensive recombination of sthemata, while small values of *Pm* are necessary to prevent the disruption of the solutions. These guidelines, however, are useful and relevant when the values of *Pc* and *Pm* do not vary.

One of the goals of the approach is to prevent the GA from getting stuck at a local optimum. To achieve this goal, we employ solutions with subaverage fitnesses to search the search space for the region containing the global optimum. Such solutions need to be completely disrupted, and for this purpose we use a value of 0.5 for k_4. Since solutions with a fitness value of *f* should also be disrupted completely, we assign a value of 0.5 to k_2 as well.

Based on similar reasoning, we assign $k_1$and k_3 a value of 1.0. This ensures that all solutions with a fitness value less than or equal to *f* compulsorily undergo crossover. The probability of crossover decreases as the fitness value (maximum of the fitness values of the parent solutions) tends to *f*max and is 0.0 for solutions with a fitness value equal to *f*max.

9.14.3 Hybrid Genetic Algorithms

As they use the fitness function only in the selection step, GAs are blind oprimizers which do not use any auxiliary information such as derivatives or other specific knowledge about the special strucrure of theobjective function. If there is such knowledge, however, ir is unwise and inefficient not to make use of ir.Several investigations have shown that a lot of synergism lies in the combination of genetic alj!orirhms andconventional methods.

The basic idea is co divide the optimization task into two complementary parts. The GA does the coarse, global optimization while local refinement is done by the conventional method (e.g. gradient-based, hill climbing, greedy algorithm, simulated annealing, ere.). A number of variants are reasonable:

1. The GA performs coarse search first. Afrer the GA is completed, local refinement is done.
2. The local method is integrated in the GA. For instance, every *K* generations, the population is doped witha locally optimal individual.
3. Both methods run in parallel: All individuals are continuously used as initial values for the local method. The locally optimized individuals are re-implanred into the current generation.

In this section a novel optimization approach is used that switthes between global and local search methods based on the local topography of the design space. The global and local optimizers work in concert to efficiently locate quality design points better than either could alone. To determine when it is apptopriate to execute a local search, some characteristics about the local area of the design space need to be determined. One good source of information is contained in the population of designs in the GA. By calculating the relative homogeneity of the population we can get a good idea of whether there are multiple local optima located within this local region of the design space.

To quantify the relative homogeneity of the population in each subspace, the coefficient of variance of the objective function and design variables is calculated. The coefficient of variance is a normalized measure of variation, and unlike the actual variance, is independent of the magnitude of the mean of the population. A high coefficient of variance could be an indication that there are multiple local optima present. Very low values could indicate that the GA has converged to a small area in the design space, warranting the use of alocal search algorithm to find the best design within this region.

By calculating the coefficient of variance of the both the design variables and the objective function as the optimization progresses, it can also be used as a criterion to switch from me global to the local optimizer. As the variance of the objective values and design variables of the population increases, it may indicate that the optimizer is exploting new areas of the design space or hill climbing. If the variance is decreasing, the optimizer may be converging toward local minima and the optimization process could be made more efficient by switching to a local search algorithm.

The second method, regression analysis, used in this section helps us determine when to switch between the global and local optimizer. The design data present in the current population of the GA can be used toprovide information as to the local topography of the design space by attempting to fit models of various order to it.

The use of regression analysis to augment optimization algorithms is not new. In problems in which the objective function or consrrainrs are computationally expensive, approximations to the design space are created by sampling the design space and then using regression or other methods to create a simple mathematical model that closely approximates the actual design space, which may be highly nonlinear. The design space can then be exploted to find regions of good designs or optimized to improve the performance of the system using the predictive surrogate approximation models instead of the computarionally expensive analysis code, resulting in large computational savings. The most common regression

models are linear and quadratic polynomials created by performing ordinary least squares regrssion on a set of analysis data.

To make dear the use of regression analysis in this way, consider Figure 9-37, which represents a complex design space. Our goal is to minimize this function, and as a first step the GA is run. Suppose that afrer acertain number of generarions the population consists of the sampled points shown in the figure. Since the population of the GA is spread throughout the design space, having yet to converge into one of the local minima, it seems logical to continue the GA for additional generations. Ideally, before the local optimizer is run it would be beneficial to have some confidence that its starting point is somewhere within the mode that contains the optimum. Fitting a second-order response surface to the data and noting the large error (the R2 value is 0.13), ther is a dear indication that the GA is currently exploting multiple modes in the design space.

In Figure 9-38, the same design space is shown but afrer the GA has begun to converge into the part of the design space containing the optimal design. Once again a second-order approximation is fir to GA's population. The dotted line connects the points predicted by the response surface. Note how much smaller the error is in the approximation (the R2 is 0.96), which is a good indication that the GA is currently exploting a single mode within the design space. At this point, the local optimizer can be made to quickly converge to the best solution within this area of the design space, thereby avoiding the slow convergence propenies of the GA.

After each generarion of the global optimizer the values of the coefficient of determination and the coefficient of variance of the enrire population are compared with the designer specified threshold levels.

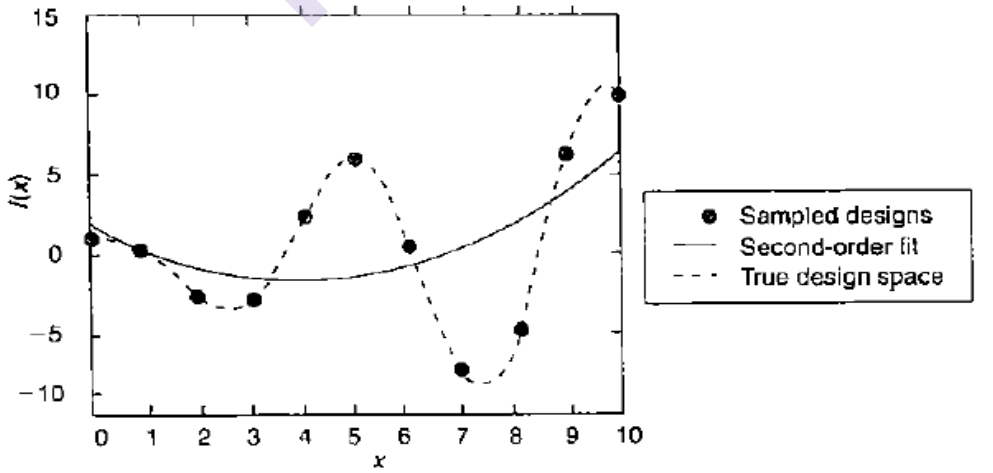


Figure 9·37 Apptoximating multiple modes with a second-order model.

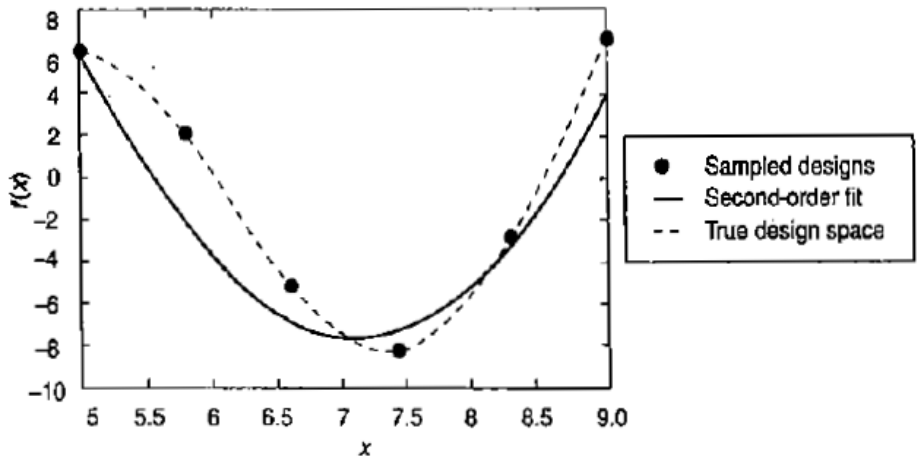


Figure 9·38 : Apptoximating a single mode with a second-order model.

The first threshold simply states that if coefficient of determination of the population exceeds a designer set value when a second-order regression analysis is performed on the design data in the current GA population, then a local search is started from the current 'best design' in the population. The second threshold is based on the value of the coefficient of variance of the entire population. This threshold is also set by the designer and can range upwards from O%. If it increases at a rate greater than the threshold level then a local sarch is execuced from the best point in the population.

The flowchart in Figure 9-39 illustrates the stages in the algorithm. The algorithm can switch repeatedly between the global search (Stage 1) and the local search (Stage 2) during execution. In Stage I, the global search is initialized and then monitored. This is also where the regression and statistical analysis occurs.

In Stage 2 the local search is executed when the threshold levels are exceeded, and then this solution is passed back and integrated two the global search. The algorithm scops when convergence is achieved for the global optimization algorithm.

9.14.4 Parallel Genetic Algorithm

GAs are powerful search techniques that are used successfully to solve problems in many different disciplines. Parallel GAs (PGAs) are particularly easy to implement and promise substantial gains in performance. As such, there has been extensive research in this field. The section describes some of the most significant problems in modeling and designing multi-population PGAs and presents some recent advancemenrs.

One of the major aspects of GA is their ability to be parallelized. Indeed, because natural evolution deals with an entire population and not only with particular individuals, it is a remarkably highly parallel process. Except in the selection phase, during which there is competition between individuals, the only interactions between remembers of the population occur during the reproduction phase, and usually, no more than two individuals are necessary to engender a new child. Otherwise, any other operations of the evolution, in particular the evaluation of each member of the population, can be done separately. So, neatly all the operations in a genetic algorithm are implicitly parallel.

PGAs simply consist in distributing the task of a basic GA on different processors. As those tasks are implicitly parallel, little time will be spent on communication; and rhus, the algorithm is expected to run much faster or to find more accurate this.

It has been established chat GA's efficiency co find optimal solution is largely determined by the population size. With a larger population size, the genetic diversity increases, and so the algorithm is more likdy to find a global optimum! A large population requires more memory to be scored; it has also been ptoved that it takes a longer time to converge. If *n* is the population size, the convergence is expected aft:er *n* log(n) function evaluations.

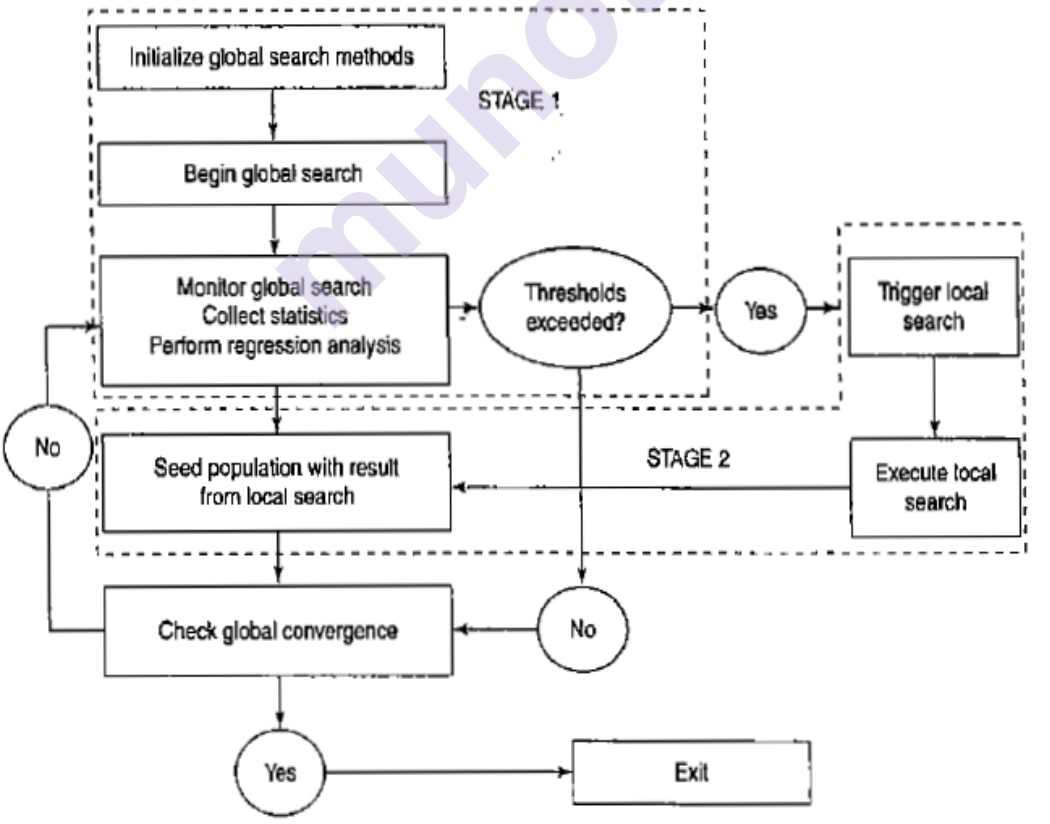


Figure 9·39: Steps in two·stage hybrid optimization approach.

The use of mday's new parallel computers not only provides more storage space but also allows the use of several processors to produce and evaluate more solutions in a smaller amount of time. By parallelizing the algorithm, it is possible D increase population size, reduce the computational cost, and so improve the performance of the GA.

Probably the first attempt to map GAs to existing parallel computer architectures was made in 1981 by John Grefensrerre. But obviously today, with the emergence of new high-performance computing (HPC), PGA is really a flourishing area. Researthers try to improve performance of GAs. The stake is to show that GAs are one of the besr optimization methods to be used with HPC.

9.14.4.1 Global Parallelization

The first attempt to parallelize GAs simply consists of global parallelization. This approach nics to explicitly parallelize the implicit parallel tasks of the "sequential" GA. The nature of the problems remains unchanged. The algorithm still manipulates a single population where each individual can mare with any other, but the breeding of new children and/or their evaluation are now made in parallel. The basic idea is that different processors can create new individuals and compme their fir ness in parallel almost without any communication among each other.

To start with, doing the evaluation of the population in parallel is something really simple co implement. Each processor is assigned a subset of individuals to be evaluated. For example, on a shared memory computer, individuals could be stored in shared memory, so that each processor can read the chtomosori:tes assigned and c:an write back the resnlr of the fitness computation. This method only supposes iliat the GA works with a generational update of the population. Of course, some synchtonization is needed between generations.

Generally, most of the computational time in a GA is spent calling the evaluation function. The time spent in manipulating the chromosomes during the selection or recombination phase is usually negligible. By assigning to each processor a subset of individuals m evaluate, a speedup proportional to the number of processors can be expeaed if there is a good load balancing between them. However, load balancing should not be a problem as generally the time spent for the evolution of an individual does not really depend on dle individual. A simple dynamic stheduling algorithm is usually enough to share the population between each processor equally.

On a distribmed memory compUter, we can smre the population in one "master" processor responsible for sending the individuals to the other processors, i.e.,

"slaves." The master processor is also responsible for collecting the result of the evaluation. A drawback of this distributed memory implementation *is* that a bottleneck may occur when slaves are idle while only the master is working. But a simple and good use of the master processor can improve the load balancing by distributing individuals dynamically tothe slave processors when they finish their jobs.

A further seep could consist in applying thegenetic operators in parallel. In fact, the interaction inside the population only occurs during selection. The breeding, involving only two individuals to generate he offspring, could easily be done simultaneously over *n/2* paits of individuals. But it is not chat clear if it worth doing so. Crossover is usually very simple and not *so* time-consuming; the point is nor that too much time will be lost during the communication, but that the time gain in the algorithm will be almost nothing compared to the effort produced to change the code.

This kind of global parallelization simply shows how easy it can be to transpose any GA onto a parallel machine and how a speed-up sublinear to the number of processors may be expected.

9.14.4.2 Classification of Parallel GAs

The basic idea behind most parallel programs is to divide a cask into chunks and co solve the chunkssimulraneously using multiple processors. This divide-and-conquer approach can be applied to*GAs* in many different ways, and the literature contains many examples of successful parallel implementations. Some parallelizacion methods use a single population, while others divide the population into several relatively isolated subpopulacions. Some methods can exploit massively parallel computer architectures, while others are better suited to multicomputers with fewer and more powerful processing elements.

There are three main cypes of PGAs:

1. global single-population master-slave GAs,
2. single-population fine-grained,
3. multiple-population coarse-grained GAs.

In a master-slave GA there is a single panmicric population (just as in a simple GA), but the evaluation of fitness is distributed among several processors (see Figure 9-40). Since in this type of PGA, selection and crossover consider the entire population it is also known as global PGA. Fine-grained PGAs are suited for massively parallel computers and consist of one spatially structured population.

Selection and mating are resrricred to a small neighbothood, but neighbothoods overlap permitting some interaction among all the individuals (see Figure 9-41 for a sthematic of this class of GAs). The ideal case is co have only one individual for every processing element available.

Multiple-popuJarion (or multiple-deme) GAs are more sophisticated, as they consist in several subpopulacions which exchange individuals occasionally (Figure 9-42 has a sthematic). *This* exchange of individuals Master Workers

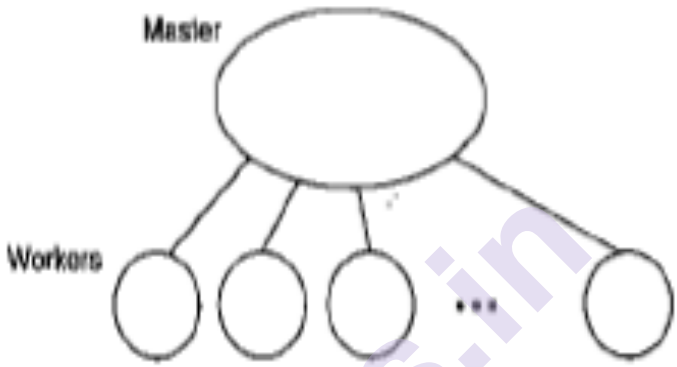


Figure 9•40 A sthematic of a master-slave PGA. The master stores the population, executes GA operations and distributes individuals to the slaves. The slaves only evaluate the fitness of the individuals.

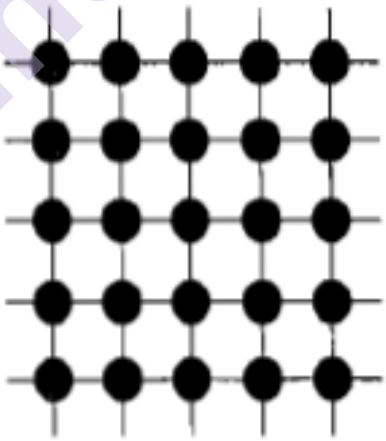


Figure 9·41 A sthematic of a fine-grained PGA. This class ofPGAs has one spadally distributed popularion, and ir can be implemented very efficiently on massively parallel compmers.

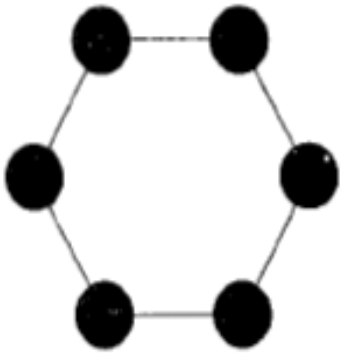


Figure 9·42 A sthematic of a mulciple-populaTion PGA. Each process is a simple GA, and there is (infrequent) communicadon between the populations.

is called migration and, as we shall see in later sections, it is conttolled by several parameters. Multiple-deme GAs are very popular, but also are the class ofPGAs which is most difficult to understand, because the effects of migration are not fully understood. Multiple-deme PGAs introduce fundamental changes in the operation of the GA and have a different behavior than simple *GAs.*

Multiple-deme PGAs are known with different names. Sometimes they are known as "distributed" GAs, because they are usually implemented on distributed memory MIMD computers. Since the computation to communication ratio is usually high, they are occasionally called coarse-grained GAs. Finally, multipledeme GAs resemble the "island model" in Population Genetics which considers relatively isolated demes, so the PGAs are also known ·as "island" PGAs. Since the size of the demes is smaller than the population used by a serial GA, we would expect that lhe PGA converges faster. However, when we compare the performance of the serial and the parallel algorithms, we must also consider the qualicy of the solutions found in each case. Therefore, while it is true that smaller demes converge faster, it is also true iliar the qualicy of the solution might be poorer.

It is important to emphasize that while the master-slave parallelization method does not affect the behaviour of the algorithm, the last two methods change the way the GA works. For example, in master-slave *PGAs,* selection takes into account all the population, but in the other two *PGAs,* seleccion only considers a subset of individuals. Also, in the mascerslave any two individuals in the population can mare (i.e., there is random mating), but in the other methods mating is restricted to a subset of individuals.

The final merhod to parallelize *GAs* combines multiple demes with masrerslave or finegrained GAs. We call this class of algorithms *hierarchical PGAs,* because at a higher level they are multipledeme algorithms with single-population PGAs (either

master-slave or finegrained) at the lower level. A hierarchical PGA combines the benefits of its components, and it ptomises bener performance than any of them alone.

Master-slave parallelization*:* This section reviews the masterslave (or global) parallelization method. The algorithm uses a single population and the evaluation of the individuals and/or the application of genetic operators are done in parallel. As in the serial GA, each individual may compete and mate with any other (thus selection and mating are global). Global PGAs are usually implemented as masrer-slave programs, where the master stores the population and the slaves evaluate the fitness.

The most common operation iliac is parallelized is the evaluation of the individuals, because the fitness of an individual is independent from the rest of the population, and there is no need to communicme during this phase. The evaluation of individuals is parallelizcd by assigning a fraction of the population to each of the processors available. Communication occurs only as each slave receives its subset of individuals to evaluate and when the slaves return the fitness values. If the algorithm *stops* and waits to receive the fitness values for all the population before ptoceeding intothe next generation, then the algorithm is synchtonous. A synchtonous mastcrslave GA has exacdy the same ptoperties as a simple GA, with speed being the only difference. However, ir is also possible to implement an asynchtonous master-slave GA where the algorithm does nor stop to wait for any slow processors, but it does nor work exactly like a simple GA. Most global PGA implementations are synchtonous and the rest of the paper assumes that global PGA carry our exactly the same search of simple GAs.

The global paralleliz.acion model does nor assume anything about the underlying computer architecture, and it can be implemented efficiently on shared memory and distributed-memory computers. On a sharedmemory multiprocessor, the popul:.tion could be sloted in shared memory and each processor can read the individuals assigned co it and write the evaluation results back without any conflicts.

On adisrributed-memorycomputer, the population can be scored in one processor. This "master" processor would be responsible for explicidy sending the individuals tothe other processors {the "slaves") for evaluation, collecting the results and applying thegenetic operators toproduce the next generation. The number of individuals assigned to any processor may be constant, but in some cases (like in a multiuser envitonment where the uciliz.acion of processors is variable) it may be

necessary to balance the computational load among the processors by using a dynamic stheduling algorithm (e.g., guided self stheduling).

Multiple-deme parallel GAs*:* The important characteristics of multiple-deme PGAs are the use of a few relarively large subpopulations and migration. Mulciple-deme GAs are the most popular parallel method, and many papers have been written describing innumerable aspects and derails of their implementation.

Probably the first systematic srudy of PGA<i with multiple populations was Grosso's dissertation. His objective was to simulate the interaction of several parallel subcomponents of an evolving population. Grosso simulated diploid individuals (so there were two subcomponentS for each "gene"), and the population was divided into five demes. Each deme exchanged individuals with all the others with a fixed migration rate.

With conttolled expetiments, Gtosso found cha the improvement of the average population fitness was fasrer in the smaller demes rhan in a single large panmictic population. This confirms a longheld principle in Population Genetics: favorable traits spread faster when the demes are small chan when the demes are large. However, he also observed that when the demes were isolated, the rapid rise in fitness stopped at a lower fitness value than with the large population. In other words, the quality of the solution found after convergence was worse in the isolated case chan in the single population.

With a low migration rate, the demes still behaved independently and exploted different regions of the search space. The migrants did not have a significant effect on the receiving deme and the quality of the solutions was similar to the case where the demes were isolated. However, at intermediate migration rates the divided population found solutions similar to those found in the panmictic population. These observations indicate that there is a critical migration rate below which the performance of the algorithm is obstructed by the isolation of the demes, and above which the partitioned population finds solutions of the same quality as the panmictic population.

It is interesting that such important observations were made so long ago, at the same time that other systematic studies ofPGAs were underway. For example, Tanese ptoposed a PGA with the demes connected on a fourdimensional hypercube topology. In Tanese's algorithm, migration occurred at fixed intervals between processors aJong one dimension of the hypercube. The migrants were chosen ptobabilistically from the best individuals in the subpopulation, and they replaced the worst individuals in the receiving deme. Tanese carried out three sees of

expetiments. In the first, the interval between migrations was ser to five generations, and the number of processors varied. In tests with two migration rates and varying the number of processors, the PGA found results of the same quality as the serial GA. However, it is difficulc to see from the xpetimental results if the PGA found the solutions sooner than the serial GA, because the range of the cimes is too large. In the second set of expetiments, Tanese varied the mutation and crossover rates in each deme, attempting to find parameter values to balance explotation and exploitation. The third set of expetiments studied the effect of the exchange frequency on the search, and the results showed thar migrating too frequendy or too infrequently degraded the performance of the algorithm.

The multideme PGAs are popular due to the following several reasons:

l. Multiple-deme *GAs* seem like a simple extension of the serial GA. The recipe is simple: take a few conventional (serial) *GAs,* run each of them on a node of a parallel computer, and at some predetermined times exchange a few individuals.

2. There is relatively little extra effort needed to convert a serial GA into a multiple-deme GA. Most of the program of the serial GA remains the same and only a few subtoutines need to be added co implement migration.

3. Coarse-grain parallel computers are easily available, and even when they are not, it is easy co simulate one with a network of workstations or even on a single processor using free software (like MPI or PVM).

There are a few important issues noted fromthe above sections. For example, PGAs are very ptomising in termsofthegains in performance. Also, PGAsare more complex than their serial counterpartS. In particular, the migration of individuals from one deme to another is conttolled by several p:uameters like (a) the topology that defines the connections between the subpopulations, (b) a migr;uion r;Ht:: rh.lt conttols how m;my individuals migrate and (c) a migration inrerval that affecrs thefreqU<'lK·of mir.1inn. In rht.' btl' 1 1lS(h .ullll·arl· 1990 the research on PGA:; began to explote alternatives to make PGAs faster and to understand better how they worked.

Around this time the first theorecical srudies on *PGAs* began to appear and the empirical research attempted to identify favorable parameters. This section reviews some of that early theoretical work and expetimental srudies on migration and topologies. Also in this period, more researthers began to use mulciplepopulation *GAs* co solve application problems, and this section ends with a brief review of their work.

One of the directions in which the field matured is that PGAs began to be tested with very large and difficult test functions.

***Fine-grained PGAs**:* The devdopment of massively paraHel compmers triggers a new approach of PGAs. To take advantage of new architectures with even a greater number of processors and less communication coslS, fine-grained *PGAs* have been devdoped. The population is now partitioned into a la..tge number of very smaJl subpopulations. The limit (and may be ideal) case is to have just one individual for every processing element available.

"Basically, the population is mapped onto a connected processor graph, usually, one individual on each processor. (But it works also more than one individual on each processor. In this case, it is preferable to choose a multiple of the number of processors for the population size.) Mating is only possible between neighboring individual, i.e, individuals stored on neighboring processors. The selection *is* also done in a neighbothood of each individual and so depends only on local information. A motivation behind local selection is biological. In nature there is no global selection, instead natural selection is a local phenomenon, raking place in an individual's local envitonment.

If we want to compare this model to the island model, each neighbothood can be considered as a different deme. But here, the demes overlap ptoviding a way *w* disseminate good solutions actoss the entire population. Thus, the topology does not need *w* explicitly define migration toads and migration rare.

It is common to place the population on a two-dimensional or three-dimensional torus grid because in many massively parallel computers the processing elements are connected using this topology. Consequently each individual has four neighbors. Expetimentally, it seems that good results can be obtained using a topology with a medium diameter and neighbothoods nor too large. Like the coarse-grained models, it worth trying to simulate this model even on a single processor to improve the results. Indeed, when the population is stored in a grid like this, after few generations, different optima could appear in different places on the grid.

To sum up, with parallelization of GA, all the different models ptoposed and all the new models we can imagine by mixing those ones, can demonstrate how well GA are adapted to parallel compmarion. In fact, the too many implementations reponed in the literature may even be confusing. We really need to understand what truly affects the performance of PGAs.

Fine-grained PGAs have only one population, but have a spatial structure that limits the interactions between individuals. An individual can only compere and mate

with its neighors; but since the neighbothoods overlap good solutions may disseminate actoss the entire population.

Robertson parallelized the GA of a classifier system on a Connection Machine 1. He parallelized the selection of parents, the selection of classifiers to replace, mating, and cl-ossover. The execution time of his implementation was independent of the number of classifiers (up to 16K, the number of processing elements in the CM-1).

***Hierarchical parallel algorithms**:* A few researthers have cried to combine two of the methods to parallelize GAs, producing hierarchical PGAs. Some of these new hybrid algorithms add a new degree of complexity to .the already complicated scene of PGAs, but other hybrids manage to keep the same complexity as one of their components. When two methods of pacallelizing *GAs* are combined they form a hierarchy. At the upper level most of the hybrid PGAs ace multiple-population algorithms.

Some hybrids have a fine-grained GA at the lower level (see Figure 9-43). For example Gruau invented a "mixed" PGA. In his algorithm, the population of each deme was placed on a two-dimensional grid, and the demes themselves were connected as a two-dimensior:tal toM. Migration between demes occurred at regulae intervals, and good results were reported for a novel neucal network design and uaining application.

Another type of hierarchical PGA uses a master-slave on each of the demes of a multi-population GA (see Figure 9-44). Migration occurs between demes, and the evaluation of the individuals is handled in parallel. This approach does not inttoduce new analytic problems, and it can be useful when worlt:ing with complex applications with objective functions that need a considerable amount of compurarion time. Bianchini and

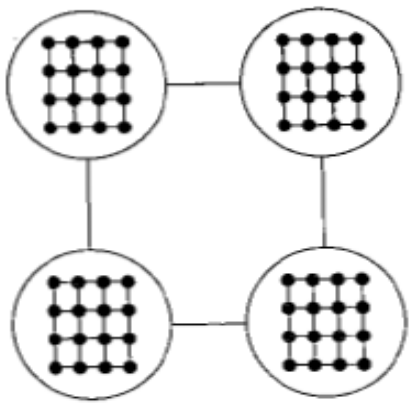


Figure 9-43 Hierarchical GA combines a multiple-deme GA (ar the upper level) and a fine-grained GA {at the lower level).

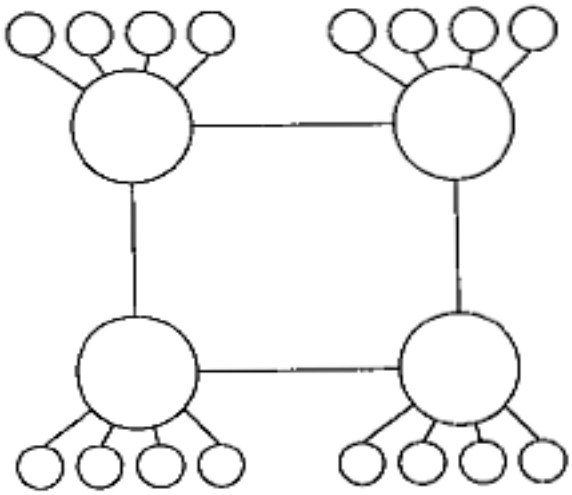


Figure 9-44 A sthematic of a hierarchical PGA. At the upper level this hybrid is a mulci-deme PGA where each node is a master-slave GA.

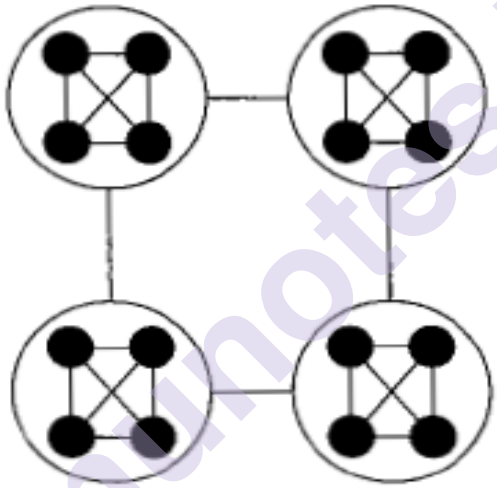


Figure 9·45 This hybrid uses mulciple-deme GAs ar both the upper and the lower levels. At the lower level the migration rate is faster and the communicarions topology is much denser than at the upper level.

Btown presented an example of this method of hybridizing PGAs, and showed that it can find a solution of the same qualiry as of a masrerslave PGA or a multipledeme GAin less time.

Interestingly, a very similar concept was invented by Goldberg in the context of an objecr·oriented implementation of a "community model" PGA. In each "community" there are multiple houses where parents reproduce and the offsprings are evaluated. *Alsо,* there are multiple communities and ir is possible that individuals migrate to other places.

A third method of hybridizing PGAs is to use multiple-deme *GAs* at both the upper and the lower levels (see Figure 9-45). The idea is to force panmiaic mixing ar the lower level by using a high migration rate and a dense topology, while a low migration rate is used at the high level. The complexity of this hybrid would be equivalent to a multiplepopularion GA if we consider the gtoups of panmicric subpopularions as a single deme. This method has nor been implemented yet. Hierarchical implementations can reduce the execution time more than any of their components alone.

9.14.4.3 Coarse· Grained PGAs - The Island Model

The second class of PGA is once again inspired by nature. The population is now divided into a few subpopulations or demes, and each of these relatively large demes evolves separately on different processors. Exchange between subpopularions is possible via a migration operator. The term *island model* is easily understandable; the GA behave as if the world was constituted of islands where populations evolve isolated from each other. On each island the population is free to converge award different optima. The migration operator allows "merissage" of the different sub populations and is supposed to mix good features that emerge locally in the different demes.

We can notice chat this time the nature of the algorithm changes. An individual can no longer breed with any other from the entire population, but only with individuals of the same island. Amazingly, even if this algorithm has been developed to be used on several processors, it is wonh simulating it sequentially on one processor. It has been shown on a few problems that better results can be achieved using this model. This algorithm is able to give different suboptimal solutions, and in many problems, it is an advantage if *we* need to determine a kind of landscape in the search space to know where the good solutions are located. Another great advantage of the island model is iliat cite population in each island can evolve wiili different rules. That can be used for multicriterion optimization. On each island, selection can be made according to different fitness functions, representing different criterions. For example it can be useful to have as many islands as criteria, plus another central island where 'selection is done with a multicriterion fitness function.

The migration operator allows individuals to move betwen islands, and therefore, m mix criteria.

In lirerarure this model is sometimes also referred as the coarsegrained PGA. (In parallelism, grain size refers m the ratio of time spent in computation and time spent in communication; when the ratio is high the processing is called coarsegrained). Sometimes, we can also find the term "distributed" GA, since they are usually implemented on distributed memory machines (MIMD Computers).

Technically there are three important features in the coarsegrained PGA: the topology that defines connections between sub populations, migration rare that conttols how many individuals migrate, migration intervals chat affect how often the migration occurs. Even if a lot of work has been done to find optimal mpology and migration parameters, here, intuition is still used more often than analysis with quite good results.

Many topologies can be defined m connect the demes, but the most common models are the island model and the steppingstones model. In the basic island model, migration can occur between any subpopulations, whereas in the Stepping stone demes are disposed on a ring and migration is restricted to neighbouring demes. Works have shown that cite topology of the space is nor so important as long as ir has high connectivity and small diameter to ensure adequate mixing as time proceeds.

Choosing the right time for migration and which individuals should migrate appears to be more complicated. Quite a lot of work is done on this subject, and problems come from the following dilemmas. We can observe that species are converging quickly in small isolated populations. Nevertheless, migrations should occur after a time long enough for allowing the development of goods characteristics in each subpopulation. It also appears that, immigration *is* a trigger for evolutionary changes. If mjgrarion occurs after each new generation, the algorithm is more or le equivalent to a sequencia\ GA with a larger population. In praaice, migration occurs either after a fixed number of iterations in each deme or at uniform periods of time. Migrants are usually selected randomly from the best individuals in the population and they replace the worst in the receiving deme. In fact, intuition is still mainly used to fix migration rare and migration intervals; there is absolurely nothing rigid, each personal cooking recipe may give good results.

9.14.5 Independent Sampling Genetic Algorithm (ISGA)

In the independent sampling phase, we design a core stheme, named the "Building Block Detecting Strategy" (BBDS), to extract relevam building block information of a fitness landscape. In this way, an individual is able to sequentially construct more highly fir partial solutions. For Toyal Toad Rl, the global optimum can be attained easily. For other more complicared fitness landscapes, we allow a number of individuals to adopt the BBDS and independently evolve in parallel so that each sthema region can be given samples indepcndently. During this phase, the population is expected to be seeded with ptomising genetic material. Then follows the breeding phase, in which individuals are paired for breeding based on two mate-selection sthemes (Huang, 2001): individuals being assigned mates by natural

selection only and individuals being allowed to actively choose their mares. In the Iauer case, individuals are able to distinguish candidate mates that have the same fitness yet have different string structures, which may lead to quite different performance after crossover. This is nor achievable by natural selection alone since it assigns individuals of the same fitness the same probability for being mares, without explicitly raking into account string suucrures. In short, in the breeding phase individuals manage to construct even more ptomising sthemata thtough the recombination of highly fir building blocks found in the first phase. Owing to the thatacteristic of independent sampling of building blocks that distinguishes the ptoposed GAs from tonventional GAs, we name this type of GA independent sampling genetic algorithms (ISGAs).

9.14.9 Tomparison of ISGA with PGA

The independent sampling phase of ISGAs is similar m the fine-grained PGAs in the sense that each individual evolves autonomously, although ISG.As do not adopt the population scrucrure. An initial population *is* randomly generated. Then in every cycle each individual does local hill climbing, and creates the next population by mating with a parmer in its neighbothood and replacing parents if offsprings are better. By tontrast, IS Gas partition the genetic processing into two phases: the independent sampling phase and the breeding phase as described in the preceding section. Third, the approach employed by each individual for improvement in IS *GAs* is different from that of the PGAs. During the independent sampling phase of ISGAs, in each cycle, through the BBDS, each individual attempts to extract relevant informacion of potential building blocks whenever its fitness increases. Then, based on the sthema information accumulated, individuals tontinue to tonstruct more tomplicated building blocks. However, the individuals of fine-grained PGAs adopt a local hill climbing algorithm that does not manage to extract relevant information of potential sthemata.

The motivation of the two phased ISGAs was partially from the messy genetic algorithms (mGAs). The two stages employed in the mGA.s are "prtwordial phase" and "juxtaPositional phase," in which the mGAs first emphasize candidate building blocks based on the guess at the order *k* of small sthemata, then just aposing them to build up global optima in the second phase by "cut" and "splice" operators. However, in the first phase, the *mGAs* still adopt centralized selection to emphasize some candidate sthemata; this in rum results in the loss of samples of other potentially ptomising sthemata. By tontrast, IS GAs manage to postpone the emphasis of candidate building blocks to the latter stage, and highlight the fearure of independent sampling of building blocks to suppress hitchhiking in the first

phase. *As* a result, population is more diverse and implicit parallelism can be fulfiUed to a larger degree. Thereafter, during the second phase, ISGA.s implement population breeding thtough two mateselecrion sthemes as discussed in the preceding section. In the following subsections, we present the key tomponenrs of ISGAs in detail and show the tomparisons between the expetimental results of the ISGAs and those of several other *GAs* on two benchmark test functions.

9.14.5.2 Tomponents of ISGAs

ISGAs are divided into two phases: the independent sampling phase and the breeding phase. We describe them as follows.

Independent sampling phase*:* To implement independent sampling of various building blocks, a number of strings are allowed *w* evolve in parallel and each individual searthes for a possible evolutionary path entirely independent of others.

In this section, we develop a new searching strategy, BBDS, for each individual to evolve based on the accumulated knowledge for potentially useful building blocks. The idea is to allow each individual to probe valuable information toncerning beneficial sthemata thtough resting its fitness increase since each time a fitness increase of a string tould tome from the presence of useful building blocks on it. In short, by systematically resting each bit to examine whether this bit is associated with the fitness increase during each cycle, a cluster of bits tonstituting potentially beneficial sthemata will be untovered. Iterating this process guarantees the formation oflonger and longer candidate building blocks.

The operation of BBDS on a string can be described as follows:

1. Generate an empty set for tollecting genes of candidate sthemata and create an initial string with uniform probability for each bit until its fitness exceeds 0. (Retord the current fitness as *Fit.)*
2. Except the genes of candidate schemata collected, from lefr to right, successively all the other bits, one at a time, evaluate the resuhing string. If the resulting fitness is less than *Fit,* retord this bit's position and original value as a gene of candidate sthemata.
3. Except the genes retorded. Randomly generate all the other bits of the string until the resulting string's fitness exceeds *Fit.* Replace *Fit* by the new fitness.
4. Go to steps 2 and 3 until some end criterion. The idea of this strategy is that the tooperation of certain genes (bits) makes for good fitness.

Once these genes tome in sight simultaneously, [hey tontribute a fitness increase *w* the string tontaining them; thus any .loss of one of these genes leads to the fitness decrease of the string. This is essentially what step 2 does and after this step *we* should be able to tollect a set of genes of candidate sthemata. Then at step 3, we keep the tollected genes of candidate sthel) lata fixed and randomly generate other bits, awaiting other building blocks to appear and bring forth another fitness in crease.

However, step 2 in this strategy only emphasizes the f1mess dtop due to a particular bit. It ignores the possibility that the same bit leads to a new fitness rise because many loci tould interact in an extremely non linear fashion. To rake this into actount, the second version ofBBDS is inttoduced thtough the change in seep 2 as follows.

Step 2: Except the genes of candidate sthemata tollected, from left to right, successively all the other bits, one at a time, evaluate the resulting string. If the resulting fitness is less than *Fit,* retord this bit's position and original value as a gene of candidate sthemata. If the resulting fitness exceeds *Fit,* substitute this bit's 'new' value for the old value, replace *Fit* by this new fitness, retord this bit's posicion and 'new' value as a gene of candidate sthemata, andre-execute this step.

Because this version of BBDS cakes into consideration the fitness increase resulted from that particular bit, iris expected to cake less *time* for detecting. Other versions of RBDS are of tourse possible. For example, in step 2, if *the* same bit resuhs in a fitness increase, ir can be retorded as a gene of candidate sthemata, and the ptocedure tontinues to test the residual bits yetwithour tompletely traveling back to the first bit to reexamine each bit. However, the empirical results obtained rhus far indicate that the performance of this alternative is quire similar to that of the second version. More expetimental results are needed to distinguish the difference between them.

The overall implementation of the independent sampling phase of ISGAs is thtough the ptoposed BBDS to get autonomous evolution of each string until all individuals in the population have reathed some end criterion.

Breeding phase: After the independent sampling phase, individuals independendy build up their own evolutionary avenues by various building blocks. Hence the population is expected to tontain diverse beneficial sthemata and premature tonvergence is alleviated to some degree. However, factors such as deception and intompatible sthemata (i.e., two sthemata have different bit values ar common defining positions) still could lead individuals to arrive at suboptimal regions of a

fitness landscape. Since building blocks for some strings to leave suboptimal regions may be embedded in other srrings, the search for ptoper maring partners and then exploiting the building blocks on them are critical for overwhelming the difficulty of strings being trapped in undesired regions. In Huang (2001) the importance of mate selection has been investigated and the results showed that the GAs is able to improve their performance when the individuals are allowed to select maces to a larger degree.

In this section, we adopt two mate-selection sthemes analyzed in Huang (2001) *w* breed the population: individuals being assigned mates by natural selection only and individuals being allowed to actively choose their mares. Since natural selection assigns strings of the same fitness the same probability for being parents, individuals of identical fitness yet distinct string structures are treated equally. This may result in significant loss of performance improvement after crossover.

We adopt the tournament selection stheme (Mitthell, 1996) as the tole of natural selection and the mechanism for choosing mates in the breeding phase is as follows:

During each mating evem, a binary tournament selection with ptobabilicy 1.0 is performed to select the first individual out of the two fittest randomly sampled individuals according to the following sthemes:

1. Run the binary tournament selection again to choose the partner.
2. Run another two times of the binary tournament selection to choose two highly fit candidate partners; then the one more dissimilar to the first individual *is* selected for mating.

The implementation of the breeding phase is thtough iterating each breeding cycle which consists of (a) two parents obtained on the basis of the mateseleccion sthemes above. (b) Two-point crossover operator (crossover rate 1.0) is applied to these parents. (c) Both parents are replaced with both offsprings if any of the two offsprings is better than them. Then steps (a), (b) and (c) are repeated until the population size is reathed and this is a breeding cycle.

9.14.6 Real-Coded Genetic Algorithms

The variant of GAs for rea.lvalued optimization that is closest to the original GA are socalled realcoded GAs. Let us assume that we are dealing with a free Ndimensional realvalued optimization problem, which means $X = R^N$ without tonstraints. In a real-coded GA, an individual is then represented as an N-dimensional vector of real numbers:

$$b = (Xi, \ldots, XN)$$

As selection does not involve the particular toding, no adaptation needs to be made- all selection sthemes discussed so far are applicable withour any restriction. What has to be adapted to £his special structure are the genetic oper.uions crossover and mutation.

9.14.6.1 Crossover Operators for Real-Coded GAs

So far, the following crossover sthemes are most common for real-coded GAs:

Flat crossover: Given two parents $b^1 = (x^1/_2, \ldots, x^1/_N)$ and $b^2 = (x^2/_1, \ldots, x^2/_N)$, a vector of random values from the unit interval (AJ , ... , AN) is chosen and the offspring $b = (x\{, \ldots, xfv)$ is tomputed as a vector of linear tombinations in the following way (for all $i = 1, \ldots, N$):

$$x^1_i = \lambda_i - x^1_i + (1 - \lambda_i) - x^2_i$$

BLX-α crossover is an extension of flat crossover, which allows an offspring allele to be also located outside the interval

$$[\min(x^1_i, x^2_j), \max(x^1_i, x^2_i)]$$

In BLX- α crossover, each offspring allele is chosen as a uniformly disuibuted random value from the imerval

$$[\min (x^1_i, x^2_j), \max(x, {}^1_i, x^2_i) + 1\text{-}\alpha]$$

where $l = \max(x^1_i,x^2_i) - \min (x^1_i,x^2_i)$. The parameter a has to be chosen in advance. For $a = 0$, BLX-a crossover becomes identical to flat crossover.

Simple crossover is nothing else but classical one-point crossover for real vectors, i.e., a crossover site k ε 2{ 1, ... , N- 1} is chosen and cwo offspring are created in the following way:

$$b^1 = (x^1_i, \ldots\ldots, x^1_{k,} x^1_{k+1} \ldots., x^2_N)$$

$$b^N = (x^2_1, \ldots\ldots, x^2_{k,} x^1_{k+1} \ldots., x^1_N)$$

Discrete crossover is analogous to classical uniform crossover for real vectors. An offspring b of the two parents b^1 and b^2 is composed from alleles, which are randomly chosen either as x^1_i or x^2_i.

9.14.6.2 Mutation Operators for Real-Coded GAs

The following mutation operators are most common for real-coded GAs:

1. *Random mutation:* For a randomly chosen gene i of an individual $b = (x_l, \ldots, X_N)$, the allele $x;$ is replaced by a randomly chosen value from a predefined interval Ia, b,].

2. *Nonuniform mutation :* In nonuniform mutation, the possible impact of mutation decreases with the number of generations. Assume that *fmax* is the predefined maximum number of generations. Then, with the same setup as in random mumion, the allele x_i is replaced by one of the two values

$$= x1+A\ (t,b;-\ x1)$$

$$:if= x;-A\ (r,x;-\ a;)$$

The choice as to which of the two is taken is determined by a random expetiment with two outtomes that have equal probabilities 1/2 and I /2. The random variable A *(t,* x) determines a mutation step from the range 10, xl in the following way:

$$D.\ (t,x) = \text{x(J-),IHd•m•IJ')}$$

In this formula, A is a uniformly distributed random value from the unit interval. The parameter *r* determines the influence of the generation index ton the disrribution of mutation step sizes over the imerval IO,xl.

9.15 Holland Classifier Systems

A Holland classifier system is a classifier system of the Michigan type which processes binary messages of a fixed length thtough a rule base whose rules are adapted actording to response of the envitonment.

9.15.1 The Production System

First of all, the tommunication of theproduction system with the envitonment is done via an arbitrarily long list of messages. The derectors translate responses from the environment two binary messages and place them on the message list which is then scanned and changed by the rule base. Finally, the effectors translate output messages two actions on the envitonment, such as forces or movements.

Messages are binaty strings of the same length *k.* More formally, a message belongs *w* {0, l}k. The rule base consists of a fixed number (m) of rules (classifiers) which tonsist of a fixed number (r) of conditions and an acrion, where both conditions and actions are strings oflength *k* over the alphabet {0, 1, *}.The asterisk plays the tole of a wildcard, a 'don't care' symbol.

A condition is matthed if and only if there is a message in the list which matthes the tondition in all nonwildcard positions. Moreover, conditions, except the first one, may be negated by adding a'-' prefix. Such a prefixed tondition is satisfied if and only if there is no message in the list which marthes the string associated with the tondition. Finally, a rule fires if and only if all the conditions are satisfied, i.e.,

the conditions are tonnected with AND. Such 'firing' rules tompere to put their action messages on the message list.

In the action pans, the wildcard symbols have a different meaning. They take thetole of 'pass through' element. The outpm message of a firing rule, whose action parr tontains a wildcard, is composed from the actually the reason why Ilegations of the first conditions are not allowed. More formally; the outgoing message *m* is defined as

$$m = \begin{cases} a[i] & \text{if } a[i] \neq * \\ m[i] & \text{if } a[i] = * \end{cases} \quad i = 1, \ldots, k$$

where *a* is the action part of the classifier and *m* is the.(Ilessage which matthes the first tondition. Formally, a classifier is a suing of the form

$$\text{Cond}_1, |'\text{-}'|| \text{ Cond}_2, \ldots\ldots, |'\text{-}' \text{ Cond}, / \text{Action}$$

where the brackets shouJd express the optionalicy of the "-" prefixes. Depending on the toncrete nee¢; of the task to be solved, it may be desirable to allow messages to be preserved for the next step. More specifically, if a message is not interpreted and removed by the effectors interface, it can make another classifier fire in the next step. In practical applications, this is usually actomplished by reserving a few bits of the messages for identifying the origin of the messages (a kind of variable index called tag).

Tagging offers new opportunities to transfer information about the current step intothe next step simply by placing ragged messages on the list, which are not interpreted, by the output interface. These messages, which obviously tontain information about the previous step, can support the decisions in the next step. Hence, apptopriate use of rags permits rules to be toupled to act sequenrially. In some sense, such messages are the memory of the system.

A single execmion cycle of the production system consists of the following steps:

1. Messages from the environment are appended tothe message list.
2. All the conditions of all classifiers are thecked against the message list *w* obtain the set of firing rules.
3. The message list *is* erased.
4. The firing classifiers participate in a tompetition to place their messages on the list.
5. The winning classifiers place their actions on the list.

6. The messages directed to the effectors are executed.

This ptocedure is repeated iteratively. How step 6 is done, if these messages are deleted or nor, and so on, depends on the toncrete implementation. It is, on the one hand, possible to choose a representation such that the effectors can interpret each output message. On the other hand, it is possible to direct messages explicitly to the effectors with a special tag. If no messages are directed to the effectors, the system is in a iliinking phase.

A classifier Rl is called tonsumer of a classifier R2 if and only if there is a message mO which fulfills at least one ofRl's conditions and has been placed on the list by R2. Tonversely, R2 is called a supplier of Rl.

9.15.2 The Bucket Brigade Algorithm

As already mentioned, in each time step *t,* we assign a strength value $u_{i,t}$ to each classifier *Ri.* This strength value represents the torrectness and importance of a classifier. On the one hand, the strengrh value influences the chance of a classifier to place its action on the output list. On the other hand, the suength values are used by the rule distovery system, which we will soon discuss.

In Holland classifier systems, the adaptation of the strength values depending on the feedback (payoff) from the envitonment is done by the so.called bucket brigade algorithm. It can be regarded as a simulared economic system in which various agents, here the classifiers, participate in an auction, where the chance to buy the right to post the action depends on the strength of the agents.

The bid of classifier *Ri* at timet is defined as

$$B;,, = CLrJ;,,S;$$

where *CL* E [0, 1] is a learning parameter, similar to learning rates in anificial neural nets, and *s,-* is the specificity, the number of nonwildcard symbols in the tondition pan of the classifier. If *CL* is chosen small, the system adapts slowly. If it is chosen too high, the strengths rend to oscillate chaotically. Then the rules have to tompete for the right for placing their"output messages on the list. In the simplest case, this can be done by a random expetiment like the selection in a genetic algorithm. For i:h bidding classifier it is decided randomly if it wins or not, where the probability that it wins is proportional to its bid:

$$P[R_i \text{ wins}] = \frac{B_{i,t}}{\sum_{j \in Sa_t} B_{j,t}}$$

In rhis equation, Sar1 is the set of indices of all classifiers which are satisfied at time*t.* Classifiers which get the right to post their output messages are called winning classifiers.

Obviously, in this approach more than one winning classifier is allowed. C f tourse, or her selection sthemes are reasonable, for instance, the highest bidding agent wins alone. This is necessary to avoid the conflict between two winning classifiers. Now let us discuss how payoff from the envitonment is disrtibuted and how the strengths are adapted. For this purpose, let us denme the set of classifiers, which have supplied a winning gent *R;* in step *t* with 5;,1• Then the new strength of a winning agent is reduced by its bid and increased by its portion of the payoff *P1* received &om the environment:

$$u_{i,t+1} = u_{i,t} + \frac{P_t}{w_t} - B_{i,t}$$

where *w1* is the number of winning agents in the actual time step. A winning agent pays its bid to its suppliers which share the bid among each other equally in the simplest case:

$$u_{i,t+1} = u_{i,t} + \frac{B_{i,t}}{|S_{i,t}|} \quad \text{for all } R_i \in S_{i,t}$$

If a winning agent has also been active in the previous step and supplies another winning agent, the value above is additionally increased by one portion of the bid the tonsumer offers. In the case that two winning agents have supplied each other mutually, the portions of the bids are exchanged in the above manner. The SHengrhs of all other classifiers *Rm* which are neither winning agents nor suppliers of winning agents, are reduced by a certain factor (they pay a rax):

$$u_{n,1+1} = N_{n,1} (1 - T)$$

T is a small value lying in the interval [0, 1]. The intention of taxation is to punish classifiers which never contribute anything to the outputof thesystem. With this concept, redundant classifiers, which never become active, can be filtered out.

The idea behind credit assignment in general and bucket brigade in particular is *w* increase the strengths of rules, which have ser the stage for later successful actions. The problem of determining such classifiers, which were responsible for conditions under which it was later on possible to receive a high payoff, can be very difficult. Consider, for instance, the game of thess again, in which very early moves can be significant for a late success or failure. In fact, the bucker brigade algorithm can solve this problem, although strength is only transferred to the suppliers, which

were active in the previous step. Each time the same sequence is activated, however, a little bir of the payoff is transferred one step back in the sequence. It is easy to see that repeated successful execution of a sequence increases the mengrhs of all involved classifiers.

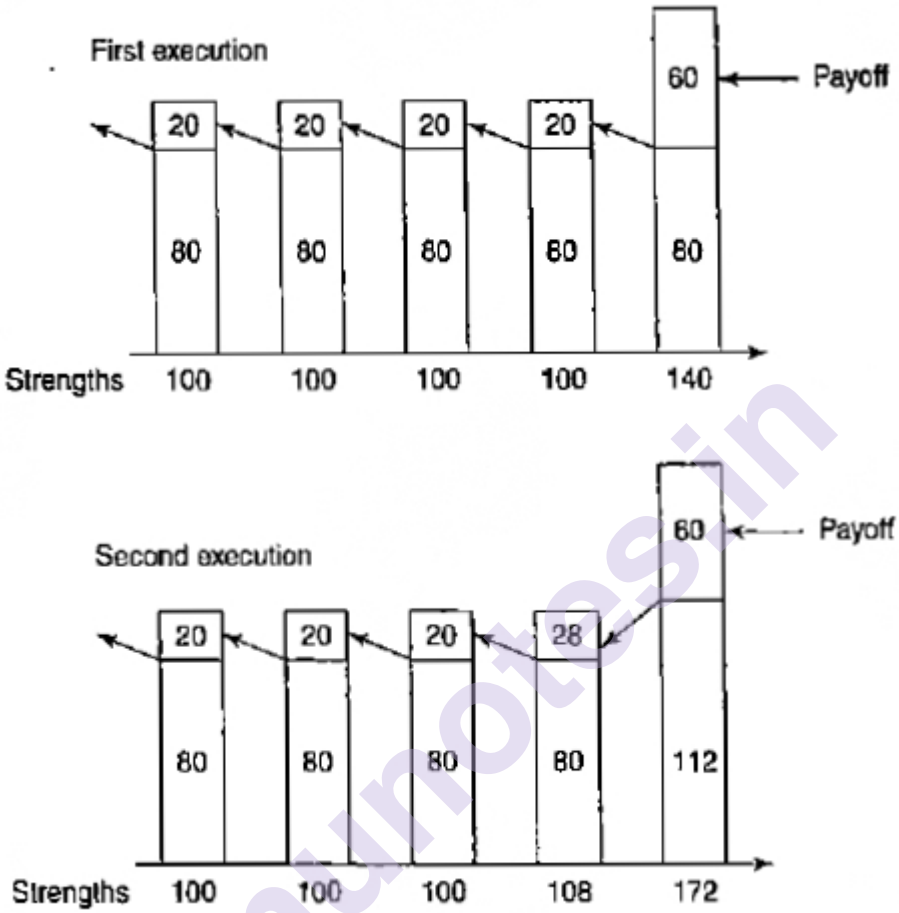


Figure 9·46 The bucker brigade principle.

Figure 9-46 shows a simple example of how the bucker brigade algorithm works. For simplicity, we consider a sequence of five classifiers which always bid 20% of their strength. Only after the fifth step, after the activation of the fifth classifier, a payoff of 60 is received. The further development of the strengths in this example is shown in the Table lS-7. It is easy to see from this example that the reinforcement of the strengths is slow at the beginning, but it accelerates later. Exactly this property tontributes much to the robustness of classifier systems - they tend to be cautious at the beginning, trying not to rush conclusions, but, after a certain number of similar situations, the system adopts the rules more and more.

It might be clear that a Holland classifier system only works if successful sequences of classifier activations are observed sufficiently often. Otherwise the bucket

brigade algorithm does not have a chance to reinforce the strengths of the successful sequence ptoperly.

9.15.3 Rule Generation

The purpose of the rule distovery system is to eliminate low-firred rules and to replace them by hopefully better ones. *The* fitness of a rule is simply its strength. Since the classifiers of a Holland classifier system themselves are strings, the application of a GA to the problem of rule induction is straightforward, though many variants are reasonable. Almost all variants have one thing in common: the GA is nor invoked in each time step, but only every nth step, where *11* has to be set such that enough information about the performance of new classifiers can be obtained in the meantime. A. Geyer-Schuh., for instance, suggests the following ptocedure, where the strength of new classifiers is initialized with the average strength of the current rule base:

1. Select a subpopulation of a certain size at random.
2. Compute a new set of rules by applying the genetic operations- selection, crossingover and muration - to this subpopularion.
3. Merge the new sub population with the rule base omitting duplicates and replace the worst classifiers.

Table 9·7 An example for repeated propogation of payoffs

Strength after the					
3rd	100.00	100.00	101.60	120.80	172.00
4th	100.00	100:32	103.44	136.16	197.60
5th	100.06	101.34	111.58	92.54	234.46
6th	100.32	103.39	119.78	168.93	247.57
.					
.					
.					
10^{th}	106.56	124.17	164 .44	224.84	278.52
.					
.					
.					
25th	29.86	253.20	280.36	294.52	299.24
.					
.					
.					
execution of the sequence					

This process of acquiring new rules has an interesting sideffect. Iris more rhan just the exchange of parts of conditions and actions. Since we have nor stared restrictions for manipulating rags, the GA can retombine parts of already existing rags m invent new tags. In the following. rags spawn related rags establishing new touplings. These new tags survive if they tonrribute to useful interactions. In this sense, the GA additionally creates experience-based internal structures autonomously.

9.16 Genetic Programming

Genetic programming (GP) is also part of the gtowing set of evolutionary algorithms that apply the search principles of natural evolution in a variety of differem problem domains, notably parameter optimization. Evolutionary algorithms and GP in particular, follow Darvin's principle of differential natural selection. This principle states that the follow"ing preconditions must be fulfilled for evolution to occur via (natural) selection:

1. There are entities called individuals which form a population. These entities can reproduce or can be reproduced.
2. There is herediry in reproduction, rhat is to say that individuals produce similar offspring.
3. In the tourse of reproduction, there is variery which affects the likelihood of survival and therefore of reproducibility of individuals.
4. There are finite resources which cause the individuals to tompete. Owing to over reproduction of individuals nor all can survive the struggle for existance. Differential natural selections will exert a tontinuous pressure towards improved individuals.

In the long run, GP and other evolutionary computing technologies will revolutionize program devel opmem. Present methods are not mamre enough for deploymem as automatic programming systems. Nevertheless, GP has already made intoads two automatic programming and will tontinue to do so in the foreseeable fmure. Likewise, the application of evolution in machine-learning problems is one of the potentials we will exploit over the coming decade.

GP is part of a more general Held known as evolutionary tomputation. Evolutionary tomputation is based on the idea that basic concepts of biological reproduction and evolution can serve as a metaphor on which computer-based, goal-directed problem solving can be based. The general idea is that a computer program can maintain a

population of artifacts represented using some suitable computer-based data structures. Elements of that population can then mare, mutate, or otherwise reproduce and evolve, directed by a fitness measure that assesses the quality of the population with respect to the goal of the task at hand.

GP is an automated method for creating a working computer program from a high-level problem statement of a problem. GP starts from a high-level statement of 'what needs to be done' and automarically creates a computer program to solve the problem.

One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it. GP addresses this challenge by ptoviding a method for automatically creating a working tompmer program from a high-level problem statement of the problem. GP achieves this goal of *automatic programming* (also sometimes called *program synthesis* or *program induction)* by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations. The operations include reproduction, crossover, mutation and architecture-altering operations patterned after gene duplication and gene deletion in nature.

GP is a domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, GP iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. The genetic operations include crossover, mutation, reproduction, gene duplication and gene deletion. GP is an excellent problem solver, a superb function apptoximator and an effective tool for writing functions to solve specific tasks. However, despite all these areas in which it excels, it still does not replace programmers; rather, it helps them. A human still must specify the fitness function and identify the problemto which GP should be applied.

9.16.1 Working of Genetic Programming

GP typically starts with a population of randomly generated tom purer programs composed of the available programmatic ingredients. GP iteratively transforms a population of computer programs into a new generation of the population by applying analogs of naturally occurring genetic operations. These operations are applied to individual(s) selected from the population. The individuals are ptobabilisrically selected to participate in the genetic operations based on their fitness (as measured by the f1tness measure provided by the human user in the third

preparatory step). The iterative transformation of the population is executed inside the main generational loop of the run of G P.

The executional steps of GP (i.e., the flowchart of GP) are as follows;

1. Randomly create an initial population (generation 0) of individual computer programs composed of the available functions and terminals.
2. Iteratively perform the following subsreps (called a *genemtion)* on the population until the termination criterion is satisfied:
    * Execute each program in the population and ascertain its fitness (explicitly or implicitly) using the problem's fitness measure.
    * Select one or two individual program(s) from the population with a probability based on fitness (with reselecrion allowed) to participate in the genetic operations in the next subsrep.
    * Create new individual program(s) for the populaiion by applying the following genetic operations with specified probabilities:
        (a) *Reproduction:* Topy the selected individual program to the new population.
        (b) *Crossover:* Create new offspring program(s) for the new population by recombining randomly chosen parts from two selected programs.
        (c) *Mutation:* Create one new offspring program for the new population by randomly mutating a randomly chosen part of one selected program.
        (d) *Archirecrure-altring operation* - Choose an architecture altering operation from the available repertoire of such operations and create one new offspring program for the new population *by* applying the chosen architecture-altering operation to one selected program.
3. After the termination criterion is satisfied, the single best program in the population produced during the run (the besr-so-far individual) is harvested and designated as the result of the run. If the run is successful, the result may be a solution (or approximate solution) to the problem.

GP is problem-independent in the sense that the flowchart specifying the basic sequence of executional steps is not modified for each new run or each new problem. There is usually no discretionary human intervention or interaction during

a run of genetic programming (although a human user may exercise judgment as to whether to terminate a run).

Figure 9-47 below is a flowchart showing the executional steps of a run ofGP. The flowchart shows the genetic operations of crossover, reproduction and mutation as well as the architecrurealrering operations. This flowchart shows a two-offspring version of the crossover operation.

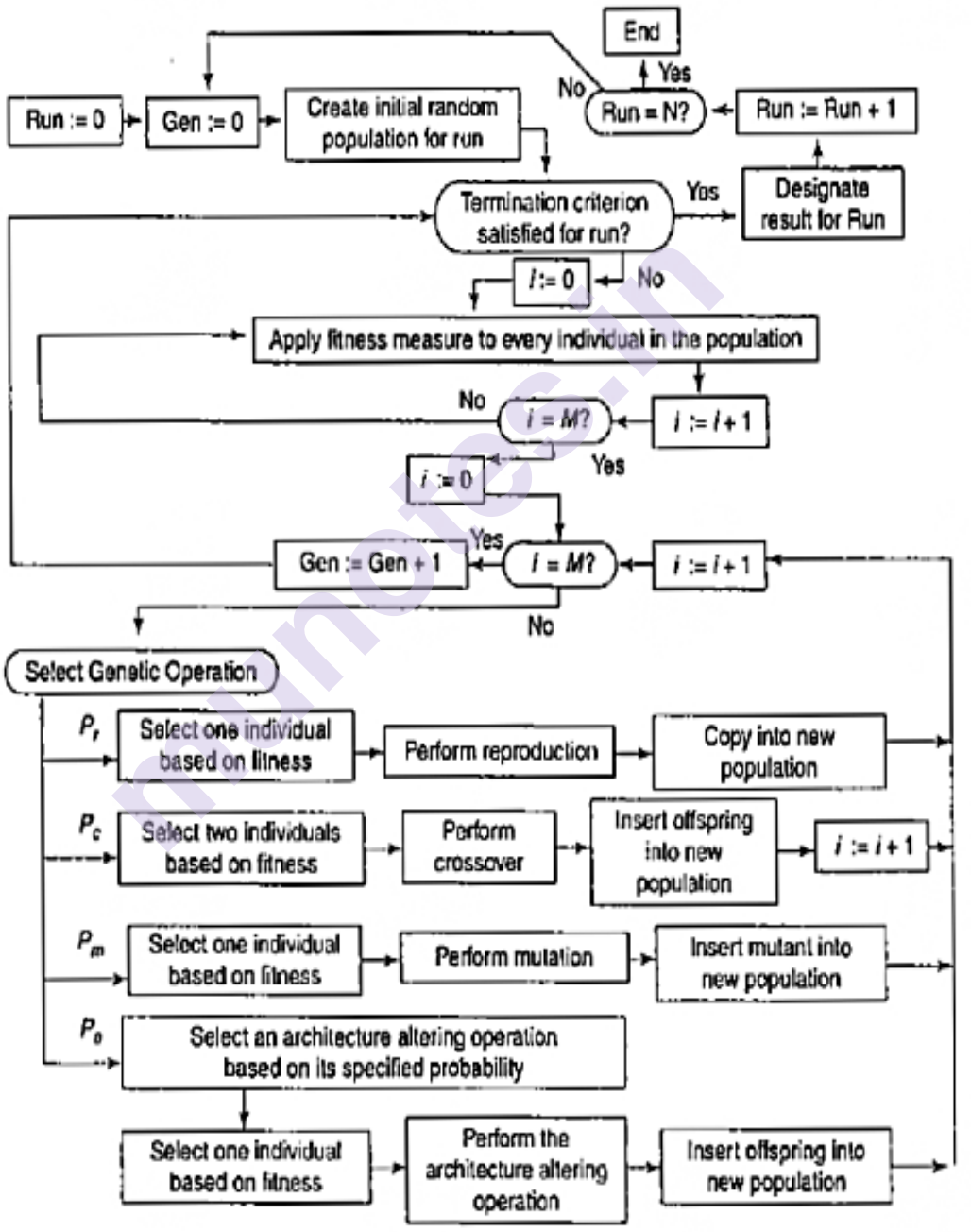


Figure 9-47 Flowchart of genetic programming.

The flowchart of GP is explained as follows: GP starts with an initial population of computer programs composed of functions and terminals apptopriate to the problem. The individual programs in the initial population are typically generated by recursively generating a rooted point-labeled program tree composed of random choices of the primitive functions and terminals (provided by the human user as part of the first and setond preparatory steps, a run ofGP). The initial individuals are usually generated subject to a pre-established maximum size (specified by the user as a minor parameter as pan of the founh preparatory step}. In general, the programs in the population are of different sizes (number of functions and terminals) and of different shapes (the particular graphical arrangement of functions and terminals in the program tree).

Each individual program in the population is executed. Then, each individual program in the population is either measured or tompared in rerms of how well it performs the task at hand (using the fitness measure provided in the third preparatory step). For many problems, this measurement yields a single explicit numerical value called *fitness.* The fitness of a program may be measured in many different ways, including, for example, in terms of the amount of error between its output and the desired output, the amount of time (fuel, money, etc.) required to bring a system to a desired target stare, the accuracy of the program in retognizing patterns or classifying objects into classes, the payoff that a game-playing program produces, or the tompliance of a tomplex structure (such as an antenna, circuit, or tonttoller) with user-specifted design criteria. The execution of the program sometimes returns one or more explicit vaJues. Alternatively, the execution of a program may tonsist only of side effecrs on the stare of a world (e.g., a robot's actions). Alternatively, the execution of a program may produce both return values and side effects.

The fitness measure is, for many practical problems, mulriobjecrive in the sense that it tombines two or more differem elements. The different elements of the fitness measure are often in tompetition with one another to some degree.

For many problems, each program in the population is executed over a representative sample of different *fituess cases.* These fitness *cases* may represent different values of the program's inpur(s), differem initial conditions of a system, or different envitonments. Sometimes the fitness *cases* are tonstructed probabilistically.

The creation of the initial random population is, in effect, a blind random search of the search space of the problem. It provides a baseline for judging future search effons. Typically, the individual programs in generation 0 all have exceedingly

poor fitness. Nevertheless, some individuals in the population are {usually) more fir than odters. The difference. in fitness are dten exploited by GP. GP applies Darwinian selection and the genetic operations to create a new population of offspring programs from the current population.

The genetic operations include crossover, mutation, reproduction and the architecture-altering operations. These genetic operations are applied to individual(s) that are ptobabilistically selected from the population based on fitness. In this ptobabilistic selection process, better individuals are favored over inferior individuals. However, the best individual in the population is not necessarily selected and the worst individual in the population is not necessarily passed over.

After the genetic operations arc performed on the current population, the population of offspring (i.e. the new generation) replaces the current population {i.e., the now-old generation). This iterative process of measuring fitness and performing the genetic operations· is reeated over many generations.

The run of GP terminates when the termination criterion (as provided by the fifth preparatory step) is satisfied. The outcome of the run is specified by the method of result designation. The best individual ever encountered during the run (i.e., the best-so-far individual) is typically designated as the result of the run.

All programs in the initial random population {generation 0) of a run of GP are symmetrically valid, executable programs. The genetic operations that are performed during the run (i.e., crossover. mutation, reproduction and the architecture-altering operations) are designed to produce offspring that art: syntactically valid, executable programs. Thus, ever individual created during a run of genetic programming (including, in pmicular, the best-of-run individual) is" syntactically valid, executable program.

9.16.2 Characteristics of Genetic Programming

GP now toutinely delivers high-return human-competitive machine intelligence, the next four subsections explain what we mean by the terms human-competitive, high-return, routine and machine intelligence.

9.16.2.1 Human-Competitive

In attempting to evaluate an automated problem-solving method, the question arises as to whether there is any real substance tothe demonstrative problems that are published in connection with the method. Demonstrative problems in the fields of artificial intelligence and machine learning are often connived to problems that

circulate exclusively inside academic groups that study a particular methodology. These problems typically have little relevance to any issues pursued by any scientist or engineer outside the fields of artificial intelligence and machine learning.

ln his 1983 talk entitled *"Al: Where It Has Been and Where It Is Going,"* machine learning pioneer Arthur Samuel said:

The aim is …… to get machines to exhibit behaviour, which of done by human, would be assumed to involve the use of intelligence.

Samuel's statement reflects the common goat articulated by the pioneers of the 1950s in the fields of artificial intelligence and machine learning. Indeed, getting machines to produce human like results is the reason for the existence of the fields of artificial intelligence and machine learning. To make this goal more concrete, we say that a result is "human-competitive" if it satisfies one or more of the eight criteria in Table 9-8. These eight criteria have the desirable attribute of being at arms-length from the fields of artificial intelligence, machine learning and GP. That is a result cannot acquire the rating of 'human-competitive' merely because it is endorsed by researchers inside the specialized fields that are attempting to create machine intelligence, machine learning and GP. That is, a result cannot acquire the rating of 'human-competitive' merely because it is endorsed by researchers inside the specialized fields that are attempting to create machine intelligence. Instead a result produced by an automated method must earn the rating of human-competitive dependent of the fact that it was generated by an automated method.

9.16.2.2 High-Return

What is delivered by the accrual automated operation of an artificial method in comparison to the amount of knowledge, information, analysis and intelligence that is pre-supplied by the human employing the method?

We define the *AI ratio* (the 'artificial-to-intelligence' ratio) of a problem-solving method as the ratio of that which is delivered by the automated operation of the *artificial* method to the amount of *intelligence* that is supplied by the human applying the method to a particular problem.

Table 9·8 Eight criteria for saying that an automatically created research is human-competitive

Criterion

A The result was patented as an invention in the past, is an improvement over a parented invention or would quality today as a permeable new invention.

B The result is equal to or beuer than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.

C The result is equal to better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.

D The result is publishable in its own right as a new scientific result-independent of the fact that the result was mechanically created.

E The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.

F The result is equal to or better than a research that was considered an achievement in its field at the time it was first discovered.

G The result solves a problem of indisputable difficulty in its field.

H The result holds its own or wins a regulated tom petition involving human contestants (in the form of either live human players or human-written computer programs).

The AI ratio is especially pertinent to methods for getting computers to automatically solve problems because it measures the value added by the artificial problem-solving method. Manifestly, the aim of the fields of artificial intelligence and machine learning is to generate human-competitive results with a high AI ratio.

Deep Blue: An Arnficin/ lme//igence Milestone (Newborn, 2002) describes the 1997 defeat of the human world thess champion Garry Kaspatov by the Deep Blue computer system. This commanding example of machine indigence is clearly a human-competitive result (by virtue of satisfying criterion H of Table 9-8). Feng-Suing Hsu (the system architect and chip designer for the Deep Blue project) recounts the intensive work on the Deep Blue project at IBM's T. J. Watson Research Centre between 1989 and 1997 {Hsu, 2002). The team of scientists and engineers spent years developing the software and the specialized computer chips to efficiently evaluate large numbers of alternative moves as part of a massive parallel state-space search. In short, the human developers invested an enormous amount of "!" in the project. In spite of the fact that Deep Blue delivered a high

{human-competitive) amount of "A," the project has a low return when measured in terms of the A-to-l ratio.

The aim of the fields of artificial intelligence and machine learning is to get computers to automatically generate human-competitive results with a high AI ratio- not to have humans generate human-competitive results themselves.

9.16.2.3 Routine

Generality is a precondition to what we mean when we say that an automated problem-solving method is "combine" Once the generality of a method is established, "routineness" means that relatively little human effort is required to get the method to successfully handle new problems within a particular domain and to successfully handle new problems from a different domain. The ease of making the transition to new problem lies at the hearr of what we mean by routine. A problem-solving method cannot be considered routine if its executional steps must be substantially augmented, deleted, rearranged, reworked or customized by the human user for each new problem.

9.16.2.4 Machine Intelligence

We use the term machine intelligence to refer to the broad vision articulated in AJan Turing's 1948 paper emided *"Intelligent Machinery"* and his 1950 paper entitled *"Computing Machinery and Intelligence."*

In the 1950s, the terms machine intelligence, artificial intelligence and machine learning all referred to the *goal* of getting "machines to exhibit behaviour, which if done by humans, would be assumed to involve the use of intelligence" {to again quote Arthur Samuel).

However, in the intervening five decades, the terms "artificial intelligence" and "machine learning" progressively diverged from their original goal-oriented meaning. These terms are now primarily associated with particular *methodologies* for attempting to achieve the goal of getting computers to automatically solve problems. Thus, the term "artificial intelligence" is today primarily associated with attempts to get computers to solve problems using methods that rely on knowledge, logic, and various analytical and mathematical methods. The term "machine learning" is today primarily associated with attempts to get computers to solve problems that use a particular small and somewhat arbitrarily chosen set of methodologies (many of which are statistical in nature). The narrowing of these terms is in marked contrast to the broad field envisioned by Samuel at the time when he toned the term "machine learning" in the 1950s, the thatter of the original founders of the field of artificial indigence, and the broad vision encompassed by

Turing's term "machine intelligence." Of course, the shift in focus from broad goals to narrow methodologies is an all too common sociological phenomenon in academic research.

Turing's term "machine intelligence" did not undergo this arteriosclerosis because, by accident of history, it was never appropriated or monopolized by any group of academic researchers whose primary dedication is to a particular methodological approach. Thus, Turing's term remains catholic today. We prefer to use Turing's term because it still communicates the broad *goal* of getting computers to automatically solve problems in a human-like way. ,

In his 1948 paper, Turing identified three broad approaches by which human competitive\'e machine intelligence might be achieved: The first approach was a logic-driven search. Turing's interest in this approach is not surprising in light of Turing's own pioneering work in the 1930s on the logical foundations of computing. The second approach for achieving machine intelligence was what he called a "cultural search" in which previously acquired knowledge is accumulated, stored in libraries and brought to bear in solving a problem - the approach taken by modern knowledge-based expert systems. Turing's first two approaches have been pursued over the past 50 years by the \'past majority of researchers using the methodologies that are today primarily associated with the term "artificial inelegance."

9.16.3 Data Representation

Without any doubt, programs can be considered as strings. There are, however, two important limitations which make it impossible to use the representations and operations from our simple GA:

l. It is mostly inappropriate to assume a fixed length of programs.

2. The probability to obtain syntactically correct programs when applying our simple initialization crossover and mutation procedures is hopelessly low.

Lt is, therefore, indispensable to modify the data representation and the operations such that syntactical correctness is easier to guarantee. The common approach to represent programs in GP is to consider programs as trees. By doing so, initialization can be done recursively, crossover can be done by exchanging sub trees and random replacement of sub trees can serve as mutation operation.

Since their only construct are nested lists programs in LISP-like languages already have a kind of tree-like Structure. Figure 9-48 shows an example how the function 3x + sin(x + I) can be implemented in a LISP like language and how such an LISP-like Function can he split up into a tree. Let can be noted that the tree n: presentation corresponds to the nested lists. The program consists of tonic expressions, like

variables and constants, which act as leave nodes while functions act as no leave nodes

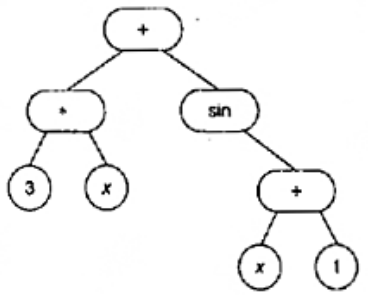


Figure 9-48 The tree representation of 3x+ sin (x + 1).

There is one important disadvantage of the LISP approach-iris difficult to introduce type checking. In case of a purely numeric function like in the above example, there is no problem at all. However, it can be desirable to process numeric data, .mings and logical expressions simultaneously. This is difficult to handle if we use a tree representation like that in Figure 948.

A. Geyer-Schulz bas proposed a very general approach, which overcomes this problem allowing maximum flexibility. He suggested representing programs by their syntactical derivation trees with respect to a recursive 'definition of underlying language in Backus-Naur form (BNF). This works for any ton text-free language. He is far beyond the stop of this lecture to go into much derail about formal languages. We will explain the basics with the help of a simple example. Consider the following language which is suitable for implementing binary logical expressions:

```
S     := <exp>
<exp> := (var) | "("<neg> <exp>")" | "("<exp> <bin> <exp>")";
<var> := "x" | "y";
<neg> := "NOT"
<bin> := "AND" | "OR";
```

The BNF description consists of so-called syntactical rules. Symbols in angular brackets < > are called nomerminal symbols, i.e. symbols which have to be expanded. Symbols between quotation marks are called terminal symbols, i.e., they cannot be expanded any further. The first rule S:=<exp> defines the staining symbol. A BNF rule of the general shape,

< non terminal > := < deriv1 > | < deriv2> | ... | < deriv11 >;

defines how a non-terminal symbol may be expanded, where the different varies are separated by vertical bars.

In order to get a feeling of how to work with the BNF grammar description, we will now show step-by-step how the expression (NOT (*x* OR *y*)) can be derivated from the above language. For simplicity, we omit quotation marks for the terminal symbols:

1. We have to begin with the start symbol: <exp>
2. We replace hexpi with the second possible derivation:
   <exp> → (<neg><exp>)
3. The symbol <neg> may only he expanded with the terminal symbol NOT:
   (<neg> <exp>) → (NOT <exp>i
4. Next. we replace: <exp> with the third possible derivation:
   (NOT <exp>) → (NOT {<exp><bin><exp>))
5. We expand the second possible derivation for <bin>:
   (NOT (<exp> <bin> <exp>)) → (NOT (<exp> OR <exp>))
6. The first occurrence of <exp> is expanded with the first derivation:
   (NOT (<exp> OR <exp>)) → (NOT (<var> OR <exp>))
7. The .second occurrence of <exp> is expanded with the first derivation, too:
   (NOT (<virr> OR <exp>)) → (NOT (<var> OR <var>))
8. Now we replace the first <var> with the corresponding first alternative:
   (NOT (<var> OR <var>)) → (NOT tx OR <var>))
9. Finally, the last non-terminal symbol is expanded with the second alternative:
   (NOT ix OR <var>)) → (NOT tx OR y))

Such a recursive derivation has an inherent tree structure. For the above example, this derivation tree has been visualized in Figure 9.49. The syntax of modern programming languages can be specified in BNF. Hence, our data model would be applicable to all of them. The question is whether this is useful. Koza's hypothesis includes that the programming language has to be chosen such that the given problem is solvable. This does not necessarily imply that we have no choose the language such that virtually any solvable problem can be solved. It is obvious that the size of the search grows with the complexity of the language. We know that the size of the search space influences the performance of a GA – the larger the

language. We know that the size of the search space influences the performance of a GA – the larger the slower.

It is therefore, recommendable to restrict the language to necessary constructs and to avoid superfluous constructs. Assume, for example, that we want to do symbolic regression, but we are only interested in polynomials with integer coefficients. For such an application, it would be an overkill to introduce rational constants or to include exponential functions in the language. A good choice could be the following.

```
S       := <func>;
<func>  := <var> | "(" <const> | "(" <func> <bin> <func>")";
<var>   := "x";
<const> := <int> | <const> <int>;
<int>   := "0" |...| "9";
<bin>   := "+" | "-" | "*";
```

For representing rational functions with integer coefficients, it.is sufficient to add the division symbol "*/*" to the possible derivations of the binary operator <bin>.

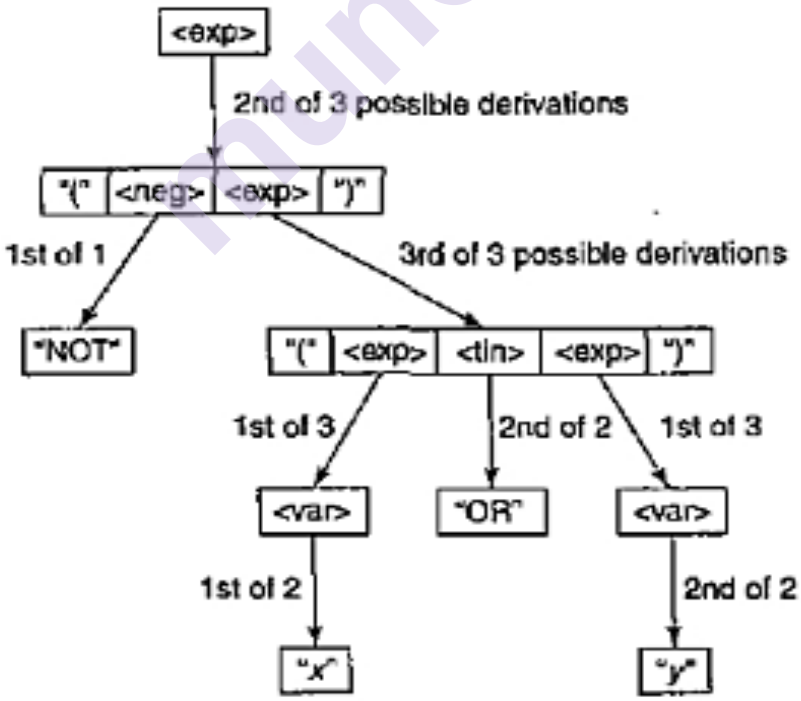


Figure 9·49 The derivation tree of (NOT *(x OR y)*).

Another example: The following language could be appropriate for discovering trigonometric identities:

```
S       := <func>;
<func>  := <var> | <const> | <trig> "("<func>")" | "("<func> <bin> <func>")"
<var>   := "x";
<const> := "0" | "1" | "π";
<trig>  := "sin" | "cos";
<bin>   := "+" | "−" | "÷";
```

There are basically two different variants of how *w* generate random programs with respect to a given BNF grammar:

1. Beginning from the starting symbol, it is possible to expand nonterminal symbols recursively, where we have to choose randomly if we have more than one alternative derivation. This approach is simple and fast, but has some disadvantages: First, it is almost impossible to realize a uniform distribution. Second, one has to implement some constraints with respect to the depth of the derivation trees in order to avoid excessive growth of the programs. Depending on the complexity of the underlying grammar, this can be a tedious task.

2. Geyer-Schulz has suggested to prepare a list of all possible derivation trees up to a certain depth and to select from this list randomly applying a uniform distribution. Obviously, in this approach, the problems in terms of depth and the resulting probability distribution are elegantly solved, but these advantages go along with considerably long computation times.

9. 16.3. 1 Crossing Programs

It is trivial to *see* that primitive string-based crossover of programs almost never yields syntactically correct program. Instead, we should use the perfect syntax information a derivation tree provides. Already in the USP times of Gp, sometime before the BNF-based representation was known, crossover was usually implemented as the exchange of randomly selected subtrees. In case that the subtrees (sub expressions) may have different types of return values (e.g., logical and numerical), it is not guaranteed iliar crossover preserves syntactical correctness.

The derivation tree based representation overcomes this problem in a very elegant way. If we only exchange subtrees which start from the same nonterminal symbol, crossover can never violate syntactical correctness. In this sense, the derivation tree

model provides implicit type checking. In order to demonstrate in more detail how this crossover operation works, let us reconsider the example of binary logical expressions. *k* parents, we take the following expressions:

(NOT (x OR y))

((NOT x) OR (x AND y))

Figure l5-50 shows graphically how the two children (NOT *(x OR (x* AND y))) ((NOT x) OR y) are obtained.

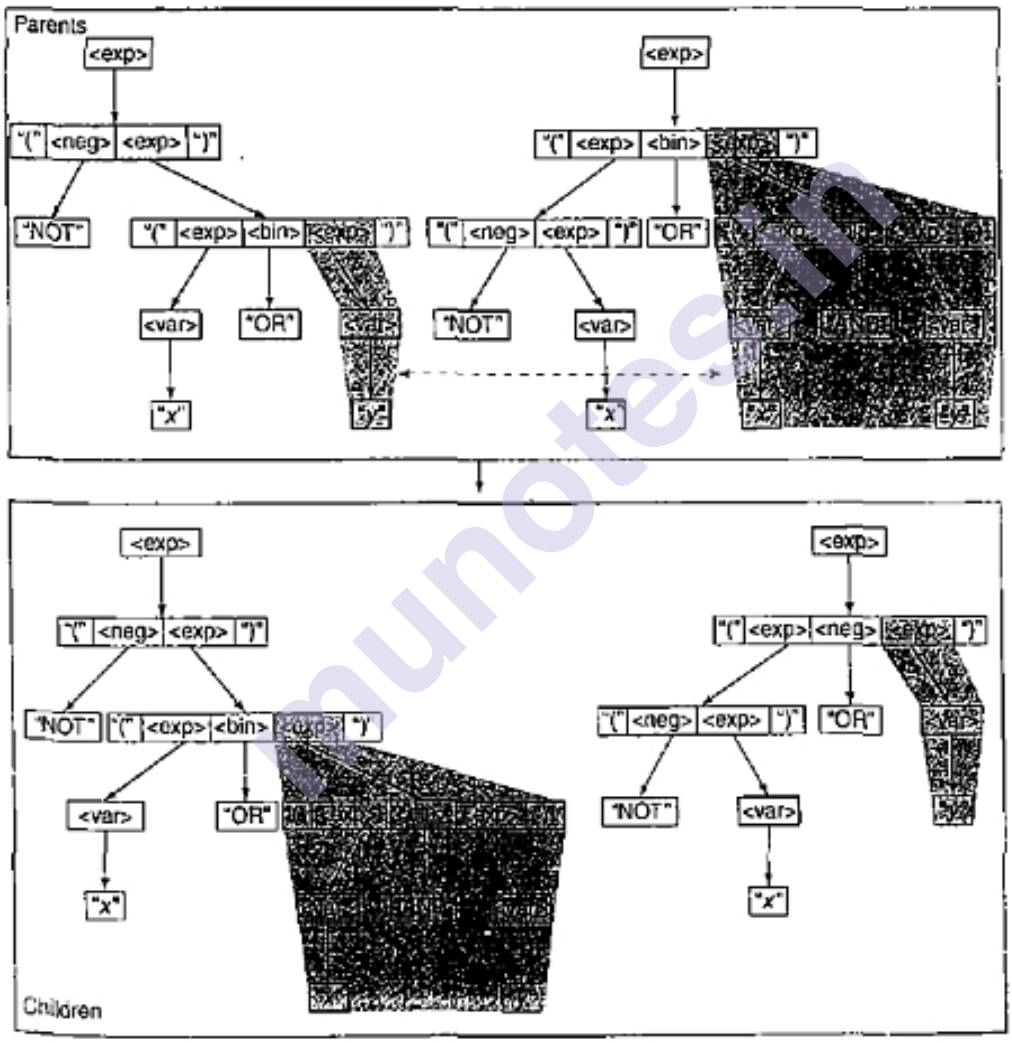


Figure 9-50 An example for crossing two binary logical expressions.

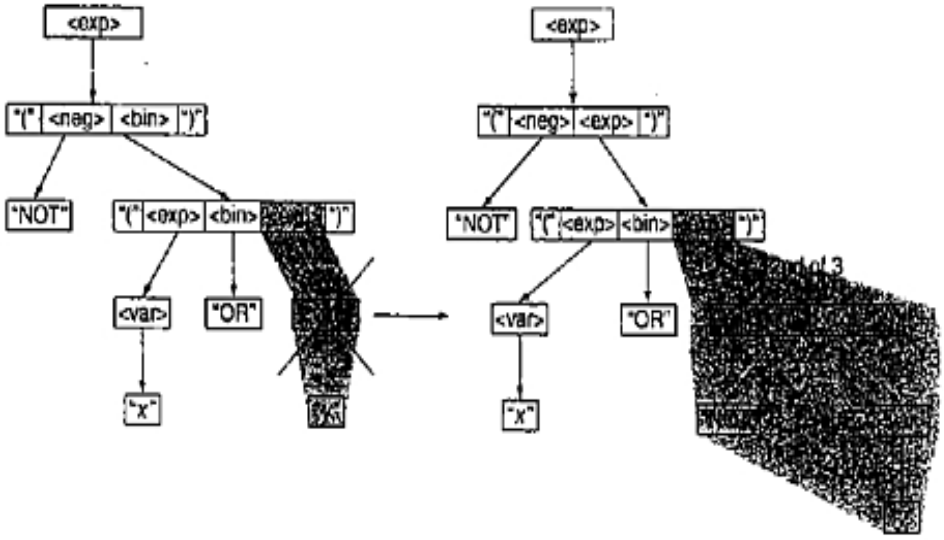


Figure 9·51 An example for making a derivation tree

9.16.3.2 Mutating Programs

We have always considered mutation as the random deformation of a chromosome. It is therefore, not surprising that the most common mutation in genetic programming is the random replacement of a randomly selected subtree. The only modification is that we do not necessarily start from the start symbol but from the nonterminal symbol at the root of the subtree we consider. Figure 9.51 shows as example where in the logical expression (NOT (x OR y)). Te variable y is replaced by (NOT *y*).

9.16.3.3 The Fitness Function

There is no common recipe for specifying an appropriate fitness functions which wrongly depends on the given problem. It is, however, worth emphasizing that it is necessary to provide enough information to guide the GA to the solution. More specifically, it is not sufficient to define a fitness function which assigns 0 to a program which does not solve the problem and 1 to a problem. Such a fitness function would correspond to needle-in-haystack problem. In the sense a proper fitness measure should be a gradual concept for judging the correctness of programs.

In many applications, the fitness function is based on a comparison of desired and actually obtained output. Koza, for instance, uses the simple sum of quadriatic errors for symbolic regression and the discover of trigonometric identities:

$$f(F) = \sum_{i=1}^{N} |y_i - F(x_i)|^2$$

In this definition, F is the mathematical function which corresponds to the program under evaluation. The list (xi, y), $1 \leq 1 \leq N$ consists of reference pairs – a desired output y, is assigned to each input 1. Check the samples have to be chosen such that the considered input space is covered sufficiently well.

Numeric error-based fitness functions usually imply minimization problem. Some other applications may imply maximization tasks. There are basically two well-known transformation which allow to standardize fitness functions such that always minimization or maximization tasks are obtained.

Consider an arbitrary "raw" fitness function f. Assuming that the number of individuals in the population is not fixed (m, at time t), the standardized fitness is computed as

$$f_S(b_{i,t}) = f(b_{i,t}) - \max_{j=1}^{m_t} f(b_{j,t})$$

It *f* has to be maximized and as

$$f_S(b_{i,t}) = f(b_{i,t}) - \min_{j=1}^{m_t} f(b_{j,t})$$

If *f* has to be minimized. One possible variant is to consider the best individual of the last *k* generations instead of only considering the actual generation.

Obviously, standardized fitness transform's any optimization problem into a minimization task. Roulette wheel selection relies on the fact that the objective is maximization of the fitness function. Koza has suggested a simple transformation such that, in any case, a maximization problem is obtained.

With the assumptions of previous definition, the adjusted fitness is computed as

$$f_A(b_{i,t}) = \max_{j=1}^{m_t} f_S(b_{j,t}) - f_S(b_{j,t})$$

Another variant of adjusted fitness is defined as

$$f'_A(b_{i,t}) = \frac{1}{1 + f_S(b_{j,t})}$$

For applying GP *w* a given problem, the following points have to be satisfied.

1. An appropriate fitness function, which provides enough information to guide the GA to the solution (mostly based on examples).

2. A syntractical description of a programming language, which contains as much elements as necessary for solving the problem.
3. An interpreter for the programming language.

The main application areas of GP include: Computer Science, Science, Engineering, and entertainment.

9.17 Advantages and Limitations of Genetic Algorithm

The advantages of GA are as follows:

1. Parallelism.
2. Liability.
3. Solution space is wider.
4. The fitness landscape is complex.
5. Easy to discover global optimum.
6. The problem has multi objective function.

The limitations of GA are as follows:

1. The problem of identifying fitness function.
2. Definition of representation for the problem.
3. Premature convergence occurs.
4. The problem of choosing various parameters such as the size of the population, mutation rare, crossover rare, the selection method and its strength.

9.18 Applications of Genetic Algorithm

An effective GA representation and meaningful fitness evaluation are the keys of the success in GA applications. The appeal of GAs tomes & on their simplicity and elegance as to bust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems. GAs are useful and efficient when

1. the search space is large, complex or poorly understood;
2. domain knowledge is scarce or expert knowledge is difficult to encode to narrow the search space;· .
3. no mathematical analysis is available;

4. traditional search methods fail.

The advantage of the GA approach is the ease with which it can handle arbitrary kinds of constraints and objectives; all such things can be handled as weighted components of the fitness function, making it easy to adapt the GA scheduler to the particular requirements of a very wide range of possible overall objectives.

GAs have been used for problem-solving and for modeling. GA are applied to many scientific, engineering problems, in business and entertainment including:

1. Optimization: GAs have been used in a wide variety of optimization tasks, including numerical optimization and combinatorial optimization problems such as traveling salesman problem (TSP), circuit design (Louis, 1993), job shop scheduling (Goldstein, 1991) and video &sound quality optimization.
2. Automatic programming. GAs have been used to evolve computer programs for specific tasks and to design other commercial structures, for example, cellular automata and sorting networks.
3. Machine and robot learning. GAs have been used for many machine-learning applications, including classifications and prediction, and protein structure prediction. GAs have also been used to design neural networks, to evolve rules for learning classifier systems or symbolic production systems, and to design and control robots.
4. Economic models: GAs have been used to model processes of innovation, the development of bidding strategies and the emergence of economic markets.
5. Immune system models: GAs have been used to model various aspects of the natural immune system, including somatic mutation during an individual's lifetime and the discovery of multi-gene families during evolutionary time.
6. Ecologjcal models: GAs have been used to model ecological phenomena such as biological arms races, host-parasite to evolutions, symbiosis and resource flow in ecologies.
7. Population genetics models: GAs have been used to study questions in population genetics, such as 'under what conditions will a gene for recombination be evolutionarily viable?'
8. *Interactions between evolution and learning. GAs* have been used to study how individual learning and species evolution affect' one another.

9. *Models of social systems: GAs* have been used to study evolutionary aspects of social systems, such as the evolution of cooperation (Chughtai, 1995), the evolution of communication and trail-following behavior in ants.

9.19 Summary

Genetic algorithms are original systems based on the supposed functioning of the living. The method is very different & the classical optimization algorithms as it:

1. Uses the encoding of the parameters, not the parameters themselves.
2. Works on a population of points, not a unique one.
3. Uses the only values of the function to optimize, not their derived function or other auxiliary knowledge.
4. Uses probabilistic transition function and not determinist ones.

lt is important to understand that the functioning of such an algorithm does not guarantee success. The problem is in a stochastic system and a genetic pool may be too far from the solution, or for example, a too fast convergence may hair the process of evolution. These algorithms are, nevertheless, extremely efficient, and are used in fields as diverse as stock exchange, production scheduling or programming of assembly robots in the automotive industry.

GAs can even be faster in finding global maxima that conventional methods, in particular when derivatives provide misleading information. It should be noted that in most cases where conventional methods can be applied, GAs are much slower because they do not take auxiliary information such as derivatives into account. In these optimization problems, there is no need to apply a GA, which gives less accurate solutions after much longer computation time. The enormous potential of GAs lies elsewhere- in optimization of non-differentiable or even discontinuous functions, discrete optimization, and program in junction.

lt has been claimed that via the operations of selection, crossover and mutation, the GA will converge over successive generations towards the global (or near global) optimum. This simple operation should produce a fast, useful and to bust technique largely because of the face that GAs combine direction and chance in the search in an effective and efficient manner. Since population implicity contain much more information than simply the individual fitness stores, GAs combine the good information hidden in a solution with good information from another solution to produce new solutions with good information inherited from both parents, inevitable}' (hopefully) leading towards optimality.

In this chapter we have also discussed the various classifications of GAs. The class of parallel GAs is very complex, and its behavior is affected by many parameters. It seems that the only way to achieve a greater understanding of parallel *GAs* is to study individual facets independent!}', and we have seen that some of the most influential publications in parallel GAs concentrate on only one inspect (migration rates, communication topology or deme size) either ignoring or making simplifying assumptions on the others. Also the hybrid GA, adaptive GA, independent sampling GA and messy GA has been included with the necessary information.

Genetic programming has been used to model and control a multitude of processes and to govern their behavior according to fitness based automatically generated algorithm. Implementation of generic programming will benefit in the coming year from new approaches which include research from developmental biology. Also, it will be necessary to learn to handle the redundancy forming pressures in the evolution of to the. Application of genetic programming will continue to broaden. Many applications focus on controlling behaviour of real or virtual agents. In this role, genetic programming may contribute considerably to the growing field of social and behavioural simulations. A brief discussion on Holland classifier system is also included in this chapter.

9.20 Review Questions

1. State Charles Darwin's theory of evulsions.
2. What is meant by genetic algorithm?
3. Compare and contrast traditional algorithm and genetic algorithm.
4. Stare the importance of genetic algorithm.
5. Explain in detail about the various operators involved in genetic algorithm.
6. What the various types of crossover and mutation techniques?
7. With a neat flowchart, explain the operation of a simple genetic algorithm.
8. State the general genetic algorithm.
9. Discuss in detail about the various types of genetic algorithm in derail.
10. State schema theorem.
11. Write than note on Holland classifier systems.
12. Differentiate between messy GA and parallel GA
13. What is the importance of hybrid GAs?

14. Describe the concepts involved in real-coded genetic algorithm.
15. What is genetic programming?
16. Compare genetic algorithm and genetic programming.
17. List the characteristics of genetic programming.
18. With a neat flowchart, explain the operation of genetic programming.
19. How are data represented in genetic programming?
20. Mention the application of genetic algorithm.

Exercise Problems

1. Determine the maximum of function x x x^5 (0.007x+ 2) using genetic algorithm by wiring a program.
2. Determine the maximum of function exp(-3x) + sin(6 r x) using genetic algorithm. Given range = [0.004 0.7]; bits = 6; population = 12; generations = 36; mutation = 0.005; mutation = 0.3.
3. Optimize the logarithmic function using a genetic algorithm by writing a program. Genetic Algorithm
4. Solve the logical AND function using genetic algorithm by writing a program.
5. Solve the XNOR problem using genetic algorithm by writing a program.
6. Determine the maximum of function exp(5x) + sin (7rr x) using genetic algorithm. Given range = [0.002 0.6]; bits = 3; population == 14; generations = 36; mutation = 0.006; matenum = 0.3.

REFERENCES

https://link.springer.com/article/10.1007/BF00175354

https://www.csd.uwo.ca/~mmorenom/cs2101a_moreno/Class9GATutorial.pdf

https://www.egr.msu.edu/~goodman/GECSummitIntroToGA_Tutorial-goodman.pdf

https://www.researchgate.net/publication/228569652_Genetic_Algorithm_A_Tutorial_Review

S.Rajasekaran, G. A. Vijayalakshami , Neural Networks, Fuzzy Logic and Genetic Algorithms: Synthesis & Applications, Prentice Hall of India, 2004

❖❖❖

UNIT 5

10

HYBRID SOFT COMPUTING TECHNIQUES

Learning Objectives

- Neuro-fuzzy hybrid systems.
- Comparison of fuzzy systems with neural networks.
- Properties of Neuro-fuzzy hybrid systems.
- Characteristics of Neuro-fuzzy hybrids.
- Cooperative neural fuzzy systems.
- General Neuro-fuzzy hybrid systems.
- Adaptive Neuro-fuzzy Inference System (ANFIS) in MATLAB.
- Genetic Neuro hybrid systems.
- Properties of genetic Neuro hybrid systems.
- Genetic algorithm based back-propagation network (BPN).
- Advantages of Neuro-genetic hybrids.
- Genetic fuzzy hybrid and fuzzy genetic hybrid systems.
- Genetic fuzzy rule based systems (GFRBSs).
- Advantages of genetic fuzzy hybrids.
- Simplified fuzzy ARTMAP.
- Supervised ARTMAP system.

10.1 Introduction

In general, neural networks, fuzzy systems and genetic algorithms are distinct soft computing techniques evolved from the biological computational strategies and nature's way to solve problems.

All the above three techniques individually have provided efficient solutions to a wide range of simple and complex problems pertaining to different domains. As

discussed these three techniques can be combined together in whole or in part, and may be applied to find solution to the problems, where the techniques do not work individually. The main aim of the concept of hybridization is to overcome the weakness in one technique. While applying it and bringing out the strength of the other technique to find solution by combining them. Every soft computing technique has particular computational parameters (e.g., ability to learn, decision making) which make them suited for a particular problem and not for others. It has to be noted that neural networks are good at recognizing patterns but they are not good at explaining how they reach their decisions. On the contrary, fuzzy logic is good at explaining the decisions but cannot automatically acquire the rules used for making the decisions. Also, the tuning of membership functions becomes an important issue in fuzzy modelling. Since this tuning can be viewed as an optimization problem, either neural network (Hopfield neural network gives solution to optimization problem) or genetic algorithms offer a possibility to solve this problem. These limitations act as a central driving force for the creation of hybrid soft computing systems where two or more techniques are combined in a suitable manner that overcomes the limitations of individual techniques.

The importance of hybrid system is based on the varied nature of the application domains. Many complex domains have several different component problems each of which may require different types of processing. When there is a complex application which has two distinct sub-problems, say for example, a signal processing and serial shift reasoning, then a neural network and fuzzy logic can be used for solving these individual tasks, respectively. The use of hybrid systems is growing rapidly with successful applications in areas such as engineering design, stock market analysis and prediction, medical diagnosis, process control, credit card analysis, and few other cognitive simulations.

Thus, even though the hybrid soft computing systems have a great potential to solve problems, if not applied appropriately they may result in adverse solutions. It is not necessary that when individual techniques give good solution, hybrid systems would give an even better solution. The key driving force is to build highly automated, intelligent machines for the future generations using all these techniques.

10.2 Neuro-Fuzzy Hybrid Systems

A *neuro-fuzzy hybrid system* (also called fuzzy neural hybrid), proposed by J. S. R. Jang, is a learning mechanism that utilizes the training and learning algorithms from neural networks to find parameters of a fuzzy system (i.e., fuzzy sets, fuzzy

rules, fuzzy numbers, and so on). It can also be defined as a fuzzy system that determines its parameters by processing data samples by using a learning algorithm derived from or inspired by neural network theory. Alternately, it is a hybrid intelligent system that fuses artificial neural networks and fuzzy logic by combining the learning and connectionist structure of neural net\works with human-like reasoning style of fuzzy systems.

Neuro-fuzzy hybridization is widely termed as Fully Neural Network (FNN) or Neuro-Fuzzy System (NFS). The human like reasoning style of fully systems is incorporated by NFS (the more popular term is used henceforth) through the use of fuzzy sets and a linguistic model consisting of a set of lF-THEN fuzzy rules. NFSs are universal approximates with the ability to solicit imerpretable lF-THEN rules; this is their main strength. However, the strength of NFSs involves imerpretability versus accuracy, requirements that are contradictory in fuzzy modelling.

In the field of fuzzy modelling research, the Neuro-fuzzy is divided into two areas:

l. Linguistic fuzzy modelling focused on imerpretability (mainly the Mamdani model).

2. Precise fuzzy modelling focused on accuracy [mainly the Takagi-Sugeno-Kang (TSK) model].

10.2.1 Comparison of Fuzzy Systems with Neural Networks

From the existing literature, it can be noted that neural networks and fuzzy systems have some things in common. If there does not exist any mathematical model of a given problem, then neural networks and fuzzy systems can be used for solving that problem (e.g:, pattern recognition, regression, or density estimation). This is the main reason for the growth of the intelligent computing techniques. Besides having individual advantages, they do have certain disadvantages that are overcome by combining both concepts.

When neural networks are concerned, if one problem is expressed by sufficient number of observed examples then only it can be used. These observations are used to train the *black box.* Though no prior knowledge about the problem is needed extracting comprehensible rules from a neural network's structure is very difficult.

A fuzzy system, on the other hand, does not need learning examples as prior knowledge; rather linguistic rules are required. Moreover, linguistic description of the input and output variables should be given. If the knowledge is incomplete, wrong or contradictory, then the fuzzy system must be runed. This is a time consuming process. Table 10.1 shows how combining both approaches brings out the advantages, leaving out the disadvantages.

Table 10-1 Comparison of neural and fuzzy processing

Neural processing	Fuzzy processing
Mathematical model not necessary	Mathematical model not necessary
Learning can be done from scratch	A prior knowledge is needed
There are severa1 learning algorithms	Learning is not' possible
Black-box behaviour	Simple interpretation and implementation

10.2.2 Characteristics of Neuro-Fuzzy Hybrids

The general architecture of Neuro-fuzzy hybrid system is as shown in Figure 5.2-I. A fuzzy system-based NFS is trained by means of a data-driven learning method derived from neural network theory. This heuristic causes local changes in the fundamental fuzzy system. At any stage of the learning process- before, during, or after- it can be represented as a set of fuzzy rules. For ensuring the semantic properties of the underlying fuzzy system, the learning procedure is constrained.

An NFS approximates ann-dimensional unknown function, partly represented by training examples. Thus fuzzy rules can be interpreted as vague prototypes of the training data. *As* shown in Figure 5.2-1, an NFS is Inputs Outputs given by a three-layer feed forward neural network model. It can also be observed that the first layer corresponds m the input variables, and the second and third layers correspond to the fuzzy rules and output variables, respectively. The fuzzy sets are converted to (fuzzy) connection weights.

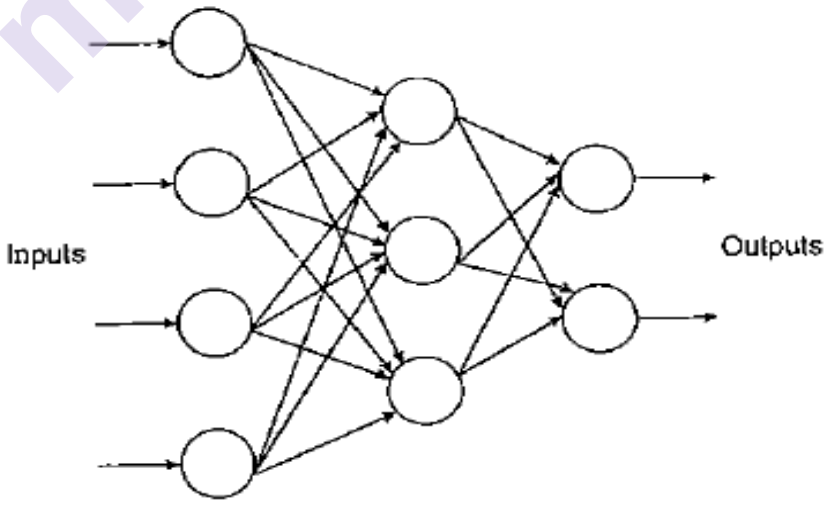


Figure 10.1 Architecture of Neuro-fuzzy hybrid system.

NFS can also be considered as a system of fuzzy rules wherein the system can be initialized in the form of fuzzy rules based on the prior knowledge available. Some researchers use five layers- the fuzzy sets being encoded in the units of the second and the fourth layer, respectively. It is, however, also possible for these models to be transformed into three-layer architecture.

10.2.3 Classifications of Neuro·Fuzzy Hybrid Systems

NFSs can be classified imo the following two systems:

l. Cooperative NFSs.

2. General Neuro-fuzzy hybrid systems.

10.2.3.1 Cooperative Neural Fuzzy Systems

In this type of system, the artificial neural network (ANN) and fuzzy system work independently from each other. The ANN attempts to learn the parameters from the fuzzy system. Four different kinds of coopemive fuzzy neural networks are shown in Figure 10.2

The FNN in Figure 10.2(A) learns fuzzy set from the given gaining data. This is done, usually, by fining membership functions with a neural network; the fuzzy sets then being determined offline. This is followed by their utilization m form the fuzzy system by fuzzy rules that are given, and not learned. The NFS in Figure 10-2(8) determines, by a neural network, the *fuzzy* rules from the training data. Here again, the neural networks learn offline before the fuzzy system is initialized. The rule learning happens usually by clustering on self-organizing feature maps. There is also the possibility of applying fuzzy clustering methods to obtain rules.

For the neuro-fuzzy model shown in Figure l0-2(C), the parameters of membership function are learnt online, while the fuzzy system is applied. This means that, initially, fuzzy rules and membership functions must be defined beforehand. Also, in order to improve and guide the learning step, the error has to be measured. The model shown in Figure 10-2(0) determines the rule weights for all fuzzy rules by a neural network. A rule is determined by its rule weight-interpreted *as* the influence of a rule. They are then multiplied with the rule output.

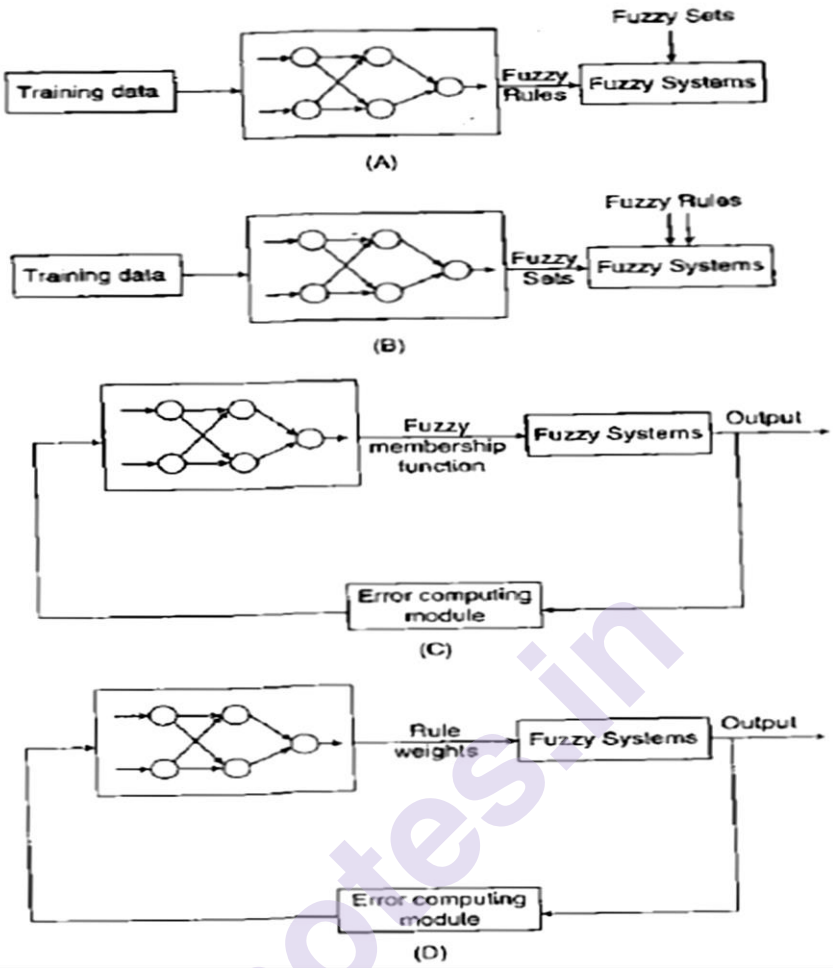


Figure 10.2 Cooperative neural fuzzy systems.

10.2.3.2 General Neuro·Fuzzy Hybrid Systems (General NFHS)

General Neuro-fuzzy hybrid *systems* (NFHS) resemble neural networks where a fuzzy system is interpreted as a neural network of special kind. The architecture of general NFHS gives it an ad van rage because there is no communication between fuzzy system and neural network. Figure 16-3 illustrates an NFHS. In this figure the rule base of a fuzzy system is assumed to be a neural network; the fuzzy sets are regarded as weights and the rules and the input and output variables as Neurons. The choice m include or discard Neurons can be made in the learning step. Also, the fuzzy knowledge base is represented by the Neurons of the neural network; this overcomes the major drawbacks of both underlying systems.

Membership functions expressing the linguistic terms of the inference rules should be formulated for building a fuzzy controller. However, in fuzzy systems, no formal approach exists to define these functions. Any shape, such as Gaussian or triangular or bell shaped or trapezoidal, can be considered as a membership function with an

arbitrary set of parameters. Thus for fuzzy systems, the optimization of these functions in terms of generalizing the data is very important; this problem can be solved by using neural networks.

Using learning rules, the neural network must optimize cite parameters by fixing a distinct shape of the membership functions; for example, triangular. But regardless of the shape of the membership functions, training data should also be available.

The Neuro fuzzy hybrid systems can also be modelled in an another method. In cit's case, the training data is grouped into several clusters and each duster is designed to represent a particular rule. These rules are defined by the crisp data points and are not defined linguistically. Hence a neural network, in this case, might be applied to train the defined dusters. The resting can be carried out by presenting a random resting sample to the trained neural network. Each and every output unit will return a degree which extends to fit to the anr.ecedem of rule.

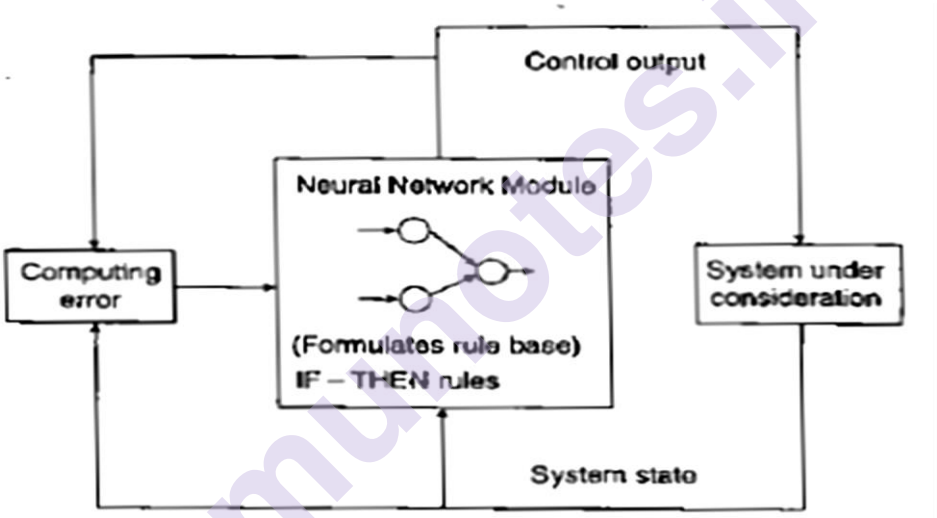


Figure 16·3 A general Neuro-fuzzy hybrid system.

10.2.4 Adaptive Neuro-Fuzzy Inference System (ANFIS) in MATLAB

The basic idea behind this Neuro-adaptive learning technique is very simple. This technique provides a method for the fuzzy modelling procedure m learn information about a data set, in order to compute the membership function parameters that best allow the associated fuzzy inference system to track the given input output data. This learning method works similarly to that of neural networks.

ANFIS Toolbox in MATLAB environment performs the membership function parameter adjustments. The function name used in activate this molbox in anfis. ANFIS toolbox can be opened in MATLAB either at command line ptompt or at Graphical User Interface. Based on the given input-output dam set, ANFIS mol box

builds a Fuzzy Inference System whose membership functions are adjusted either using back Propagation network training algorithm or Adaline network algorithm, which uses least mean square learning rule. This makes the fuzzy syHem to learn from the data they model.

The Fuzzy Logic Toolbox function that accomplishes this membership function parameter adjustment is called anfis. The actonym ANFIS derives its name from adaptive Neuro-fuzzy inference system. The anfis function can be accessed either from the command line or thtough dte ANFIS Editor GUI. Using a given input/output data set, the toolbox function anfis constructs a fuzzy inference system (FIS) whose membership function parameters are adjusted using either a back-Propagation algorithm alone or in combination with a least squares type of method. This enables fuzzy systems w learn from the data they are madeling.

10.2.4. 1 FIS Structure and Parameter Adjustment

A network-type structure similar to that of a neural network can be used to interpret the input/output. This structure maps inputs thtough input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs. During the learning process, the parameters associated with the membership functions will change. A gradient vector facilitates the computation (or adjustment) of these parameters, providing a measure of how well the fuzzy inference system models the input/output data for a given set of parameters. After obtaining the gradient vector, any of several optimization routines could be applied to adjust the parameters for reducing some error measure (defined usually by the sum of the squared difference between the accrual and desired outputs). anfis makes use of either back-propagacion or a combination of adaline and back-propagation, for membership function parameter estimation.

10.2.4.2 Constraints of ANFIS

When compared to the general fuzzy infetence.sysrems anfis is more complex. It is not available for all of the fuzzy inference system options and only supports Sugeno-rype systems. Such systems have the following properties:

1. They should be the first- or zetoth-order Sugeno-type systems.
2. They should have a single ourpur that is obtained using weighted average defuzzificarion. All output membership functions must be the same rype and can be either linear or constant.
3. They do not share rules. The number of output membership functions must be equal to the number of rules.
4. They must have unity weight for each rule.

o, all the customization options that basic fuzzy inference allows cannot be epted by anfis. In simpler words, membership functions and defuzzification ctions cannot be made according to one's choice, rather those ptovided should ısed.

2.4.3 The ANFIS Editor GUI

ger started with the ANFIS Editor GUI, type anfisedit at the MATLAB ımand prompt. The GUI as in Figure 10-4 will appear on your screen.

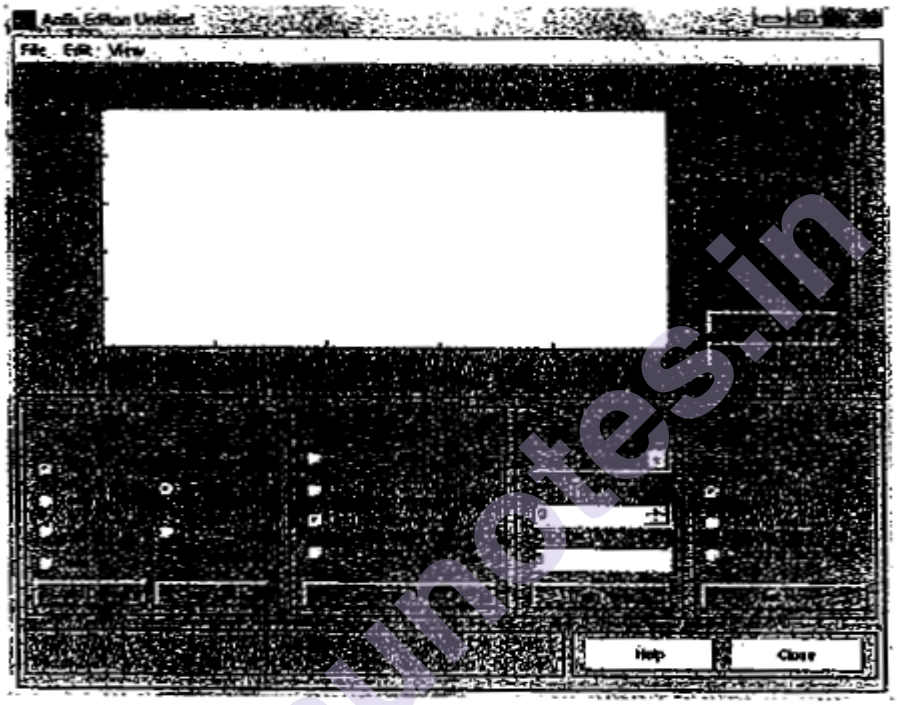


Figure 10-4 ANFIS Editor in MATLAB.

m this GUI one can:

Load data (training, resting and checking) by selecting apptopriate radio buttons in the Load Data portion of the GUI and then cliclcing Load Data. The loaded data is planed on the plot region.

Geneme an initial FIS model or load an initial FlS model using the options in the Generate FIS portion of the GUI.

View the FIS model structure once an initial FIS has been generated or loaded by clicking the Structure button.

Choose the FIS model parameter optimization method: back-Propagation ora mixture of back-propagation and least squares (hybrid method).

Choose the number of training epochs and the training error tolerance.

6. Train th.e FIS model by clicking the Train Now button. This training adjusts the membership function parameters and plots the training (and/or checking data) error plot(s) in the plot region.

7. View the FIS model output versus the training, checking, or testing data output by clicking the Test Now button. This function plots the test data against the PIS output in the plot region.

One can also use the ANFIS Editor GUI menu bar to load an FIS training initialization, save your trained FIS, open a new Sugeno system, or open any of the other GUIs to interpret the trained FIS model.

10.2.4.4 Data Formalities and the ANFIS Editor GUI

To scan training an FIS using either anfis or the ANFIS Editor GUI, one needs co have a training data set char contains desired inpudoutput data paits of the rarger sysrem to be modeled. In certain cases, optional resting data set may be available that can check the generalization capability of the resuhing fuzzy inference system, and/or a checking data sec that helps with model overfirring during the training. One can account for overfiuing by resting the FIS trained on the training data against the checking data and choosing the membership function parameters to be those associated with the minimum checking error, if these errors indicate model overfitting. To determine this, their training error plots have to be examined fairly closely. Usually, these rraining and checking data sets are stored in separate files after being collected based on observations of the target sysrem.

10.2.4.5 More on ANFIS Editor GUI

A minimum of two and maximum six arguments can be taken up by the command anfis whose general format is

[fismat 1, trnError, ss, fismat 2, chkError] = ….....

Anfis (trnData, fismat, trnOpt, dispOpt, chkData, method);

Here trnOpt (training options), dispOpt (display options), chkData (checking data), and method (training method) are optional. All of the output arguments are also optional. In this section we will discuss the arguments and range components of the command line function anfis as well as the analogous functionaliry of the ANFIS Editor GUI. Only the training data set must exist before implementing anfis when the ANFIS Editor GUI is invoked using anfisedit. The step-size will be fixed when the adaptive NFS is trained using this GUI tool.

Training Data

Both anfis and ANFIS Editor GUI require the training data, trnData, as an argument. For the target system to be modeled each tow of trndata is a desired input/output pair; a row starts with an input vector and is followed by an output value. So, the number of rows of trndata is equal to the number of training data pairs. Also, because there is only one output, the number of columns of trndata is one more than the number of inputs.

Input FIS Structure

The input FIS Structure, fismat, can be obtained from any of the following fuzzy editors:

1. The FIS Editor.
2. The Membership Function Editor.
3. The Rule Editor from ilie ANFIS Editor GUI (which allows a FIS structure to be loaded from a file or the MATLAB workspace).
4. The command line function, genfisl (for which one needs to give only numbers and cypES of membership functions).

The FIS structure contains both the model structure (specifying, e.g., number of rules in the FIS, the number of membership functions for each input, etc.) and the parameters (which specify the shapes of the membership functions).

For updating membership function parameters, anfis learning employs two methods:

1. Back-propagation for all parameters (a steepest descent method).
2. A hybrid method involving back-Propagation for the parameters associated with the input membership functions and leastsquares estimation for the parameters associated with the output membership functions.

This means that thtoughout the learning process, at least locally, the training error decreases. So, as the initial membership functions increasingly resemble the optimal ones, it becomES easier for the model parameter rraining to converge. In the setting up of these ·tnitial membership function parameters in the FIS structure, it may be helpful to have human expenise about the target system co be modeled.

Based on a fixed number of membership functions, the genfisl function produces a FIS structure. This structure invokes the so-called curse of dimensionality and causes excessive propagation of the number of rules when the number of inputs is

moderately large (more than four or five). To enable some dimension reduction in the fuzzy inference system, the Fuzzy Logic Toolbox software ptovides a method- a FIS structure can be generated using the clustering algorithm discussed in Subtractive Clustering. To use this clustering algorithm, select the Sub. Clustering option in the Generate FIS portion of the ANFIS Editor GUI, before the FIS is generated. The data is partitioned by the subtractive clustering method into gtoups called dusters and generates a F1S with the minimum number of rules required to distinguish the fuzzy qualities associated with each of the clusters.

Training Options

One can choose a desired error tolerance and number of training epochs in the ANFIS Editor GUI tool. For the command line anfis, training option trnOpt: is a vector specifying the stopping criteria and the stepsize adaptation strategy:

1. trnOpt (1) : number of training epochs; default = 10
2. trnOpt (2) : error. tolerance; default= 0
3. trnOpt (3) : initial step-size; default= 0.01
4. trnOpt (4) : step·size decrease rate; default'" 0.9
5. trnOpt (5) : step--size increase rate; default= 1.1

The default value is taken if any element of trnOpt is missing or is an NaN. The training process stops if the designated epoch number is reached or the error goal is achieved, whichever comes first.

The srep-size profile is usually a curve that increases initiaHy, reaches a maximum, and then decreases for the remainder of the training. This ideal step-size profile can be achieved by adjusting the initial step-size and the increase and decrease rates (trnOpt (3) - trnOpt (5)). The default values are set up to cover a wide range of learning tasks. These step-size options may have to be modified, for any specific application, in order to optimize the training. There are, however, no user.specilied step-size options for uaining the adaptive Neuro-fuzzy inference system generated using the ANFIS Editor GUI.

Display Options

They apply only to the command line function anfis. The display options argument, dispOpt, is a vector of either ls or Os that specifies the information to be displayed (print in ilie MATLAB command window) before, during, and after the training process. To denote print this option, 1 is used and to denote do not print this option, 0 is used.

1. dispOpt (1) : display ANFIS information; default = 1
2. dispOpt (2) : display error (each epoch); default = I
3. dispOpt (3) : display step-size (each epoch); default = 1
4. dispOpt (4) : display final results; default = 1

All available information is displayed in the default mode. If any element of dispDpt is missing or is NaN, the default value is used.

Method

To estimate membership function parameters, both the command line anfis and the ANFIS Editor GUI apply either a back-Propagation form of the steepest descent method, or a combination of back-Propagation and the least-squares method. The choices for this argument are hybrid or backPropagation. In the command line function, anfis, these method choices are designated by 1 and 0, respectively.

Output FIS Structure for Training Data

The output FIS structure corresponding to a minimal training error is fismat l. This is the FIS structure one uses to represent the fuzzy system when there is no checking data used for modd ctoss-validation. Also, when the checking data option is not used, this data represents the FIS structure that is saved by the ANFIS Editor GUI. When one uses the checking data option, the output saved is that associated with the minimum checking error.

Training Error

This is the difference between the training data output value and the output of the fuzzy inference sysrem corresponding to the same uaining data input value (the one associated with that training data output value.)

The root mean squared error (RMSE) of the training data set at each epoch is recorded by the training error trnError; and fismat l is the snapshot of the FIS structure when the training error measure is at its minimum. *As* the system is trained, the ANFIS Editor GUI plots the training error versus epochs curve.

Step-Size

With the ANFIS Editor GUI, one cannot control the step-size options. The step-size array ss records the step-size during the uaining, using the command line anfis. If one plots ss, one gets the step-siz.e profile which serves as a reference for adjusting the initial step-size, and the corresponding decrease and increase rates.

The guidelines followed for updating the step-size (ss) for the command line function anfis are:

l. If the error undergoes four consecutive reductions, increase the step-size by multiplying it by a constant (ssinc) greater than one.

2. If the error undergoes two consecucive combinations of one increase and one reduction, decrease the step-size by multiplying it by a constant (ssdec) less than one.

For the initial step-size, the default value is 0.01; for ssinc and ssdec, they are 1.1 and 0.9, respectively. All the default values can be changed via the training option for the command line anfis.

Checking Data

For testing the generalization capability of the fuzzy inference system at each epoch, the checking data, chkData, is used. The checking data and the training data have the same format and elements of the former are generally distinct from those of the latter.

For learning tasks for which the input number is large and/or the data itself is noisy, the checking data is important. A fuzzy inference system needs to track a given input/output data set welL The model structure used for anfis is fixed, which means that there is a tendency for the model to overfit the data on which it is trained, especially for a large number of training epochs. In case overfitting occurs, the fuzzy inference system may not respond well to other independent data sets, especially if they are corrupted by noise. In these situations, a validation or checking dam set can be useful. To cross-validate the fuzzy inference model, this data set is used; ctoss-validation requires applying the checking data to the model and then seeing how well the model responds to this data.

The checking data is applied to the model at each tmining epoch, when the checking data option is used with anfis either via the command line or using the ANFIS Editor GUI. Once the command line anfis is invoked, the model parameters £hat correspond to the minimum checking error are returned via the output argument fismat2. The FIS membership function parameters computed using the ANFIS Editor GUI when both training and checking data are loaded, are associated with the training epoch that has a minimum checking error.

The assumptions made when using the minimum checking data error epoch to set the membership function parameters are:

checking data error decreases as the training begins.

2. The checking data increases ar some point in the training after the data overfitting occurs.

The resulting FIS may or may not be the one which is required to be used, depending on the behavior of the checking data error.

Output FIS Structure for Checking Data

The output FIS structure with the minimum checking error is the output of the command line anfis,

Fismat 2. If checking data is used for ctoss-validation, this FIS structure is the one rhat should be used for further calculation.

Checking Error

This is the difference becween the checking data ourpuc value and the output of the *fuzzy* inference system corresponding to the same checking dala input value, which is the one associated with that checking data output value. The Toot Mean Square Error (RMSE) is reCorded for clte checking data at each epoch, by the checking error chkError. The snapshot of ilie FIS structure when the checking error has its minimum value is fismat 2. The checking error versus epochs curve is planed by the ANFIS Editor GUI, as the system is trained.

10.3 Genetic Neuro·Hybrid Systems

A Neuro-genetic hybrid or a genetic-Neuro·hybrid system is one in which a neural network employs a genetic algorithm to optimize its structural parameters iliat define its architecture. In general, neural networks and genetic algorithm refers to two distinct methodologies. Neural networks learn and execute different tasks using several examples, classify phenomena, and model nonlinear relationships; that is neural networks solve problems by self-learnig and self-organizing. On the other hand, genetic algorithms present themselves as a potential solution for the optimization of parameters of neural networks.

10.3.1 Properties of Genetic Neuro·Hybrid Systems

Certain properties of genetic Neuro-hybrid systems are as follows:

1. The parameters of neural networks are encoded by genetic algorithms as a string of properties of the network, that is, chromosomes. A large population

of chromosomes is generated, which represent the many possible parameter sets for the given neural network.

2. Genetic Algorithm- Neural Network, or GANN, has the abilicy to locate the neighborhood of the optimal solution quickly, compared to other conventional search mategies.

Figure 10-5 shows the block diagram for the genetic-Neuro-hybrid systems. Their drawbacks are: the large amount of memory required for handling and manipulation of chromosomes for a given network; and also the question of scalabiliry of this problem as the size of the networks become large.

10.3.2 Genetic Algorithm Based Back-Propagation Network (BPN)

BPN is a method of reaching multi-layer neural networks how to perfonn a given task. Here learning occurs during this training phase. The basic algorithm with architecture is discussed in Chapter 3 (Section 3.5) of this book in derail. The limitations of BPN are as follows:

1. BPN do not have the abiliry to recognize new patterns; they can recognize patterns similar to those they have learnt.

2. They must be sufficiently trained so that enough general features applicable to both seen and unseen instances can be extracted; there may be undesirable effects due to over training the network.

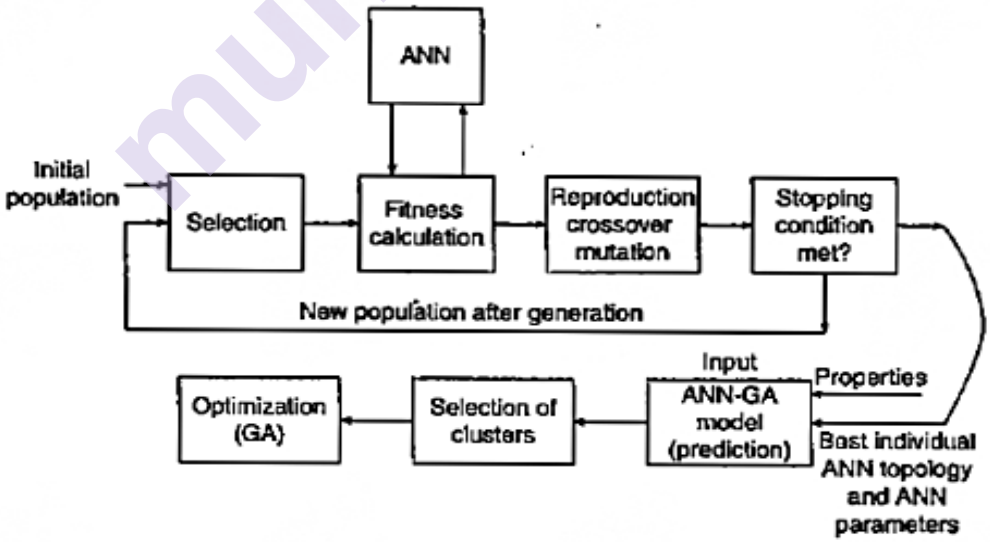


Figure 10-5 Block diagram of genetic-Neuro hybrids.

Also, it may be nored that the BPN determines its weight based on gradient search technique and hence it may encounter a local minima problem. Though genetic algorithms do nor guarantee to find global optimum solution, they are good in quickly finding good acceptable solutions. Thus, hybridization ofBPN with genetic algorithm is expected to ptovide many advantages compared to what they alone can. The basic concepts and working of genetic algorithm are discussed in Chapter 15. However, before a genetic algorithm is executed,

1. A suitable coding for the problem has to be devised.
2. A fitness function has to be formulated.
3. Parents have to be selected for reproduction and then ctossed over to generate offspring.

10.3.2.1 Coding

Assume a BPN configuration *n-1-m* where *n* is the number of Neurons in the input layer, *l* is the number of Neurons in the hidden layer and *m* is the number of output layer Neurons. The number of weights to be determined is given by

$$(n + m)i$$

Each weight (which is a gene here) is a real number. Let dbe the number of digits (gene length) in weight. Then a String S of decimal values having string length *(n + m)ld* is randomly generared. It is a string that represenrs weight matrices of inpurhidden and the hidden-output layers in a linear form arranged as tow-major or column-major depending upon the sryle selected. Thereafter a population of *p* (which is the *population size)* chromosomes is randomly generated.

10.3.2.2 Weight Extraction

In order to determine the fitness values, weights are exuacred from each chromosome. Let n3 …., nd, …., nl, represent a chromosome and let npd + dpt + 2 ….. d(p+1)d represent pth gene (p > 0) in the chromosomes.

The actual weight w_p is given by

$$w_p = \begin{cases} -\dfrac{a_{pd+2}10^{d-2} + a_{pd+3}10^{d-3} + \cdots + a_{(p+1)d}}{10^{d-2}} & \text{if } 0 \le a_{pd+1} < 5 \\ +\dfrac{a_{pd+2}10^{d-2} + a_{pd+3}10^{d-3} + \cdots + a_{(p+1)d}}{10^{d-2}} & \text{if } 5 \le a_{pd+1} \le 9 \end{cases}$$

10.3.2.3 Fitness Function

A fimess has to be formulated for each and every problemto be solved. Consider ilie matrix given by

$$\begin{Bmatrix} (x_{11}, x_{21}, x_{31}, \ldots, x_{n1}) & (y_{11}, y_{21}, y_{31}, \ldots, y_{n1}) \\ (x_{12}, x_{22}, x_{32}, \ldots, x_{n2}) & (y_{12}, y_{22}, y_{32}, \ldots, y_{n2}) \\ (x_{13}, x_{23}, x_{33}, \ldots, x_{n3}) & (y_{13}, y_{23}, y_{33}, \ldots, y_{n3}) \\ \vdots & \vdots \\ (x_{1m}, x_{2m}, x_{3m}, \ldots, x_{nm}) & (y_{1m}, y_{2m}, y_{3m}, \ldots, y_{nm}) \end{Bmatrix}$$

where *X* and Y are the inputs and targets, respectively. Compute initial population *I*o of size *'j'*. Let

O_{10}, O_{20}, ... , O_p represem *'j'* chromosomes of the initial population *l*o. Let the weights extracted for each of the chromosomes upto the chromosome be w_{10}, w_{20}, w_{30}, …., w*p*. For a number of inputs and m number of outputs, let the calculated output of the considered BPN be

$$\begin{Bmatrix} (c_{11}, c_{21}, c_{31}, \ldots, c_{n1}) \\ (c_{12}, c_{22}, c_{32}, \ldots, c_{n2}) \\ (c_{13}, c_{23}, c_{33}, \ldots, c_{n3}) \\ \vdots \\ (c_{1m}, c_{2m}, c_{3m}, \ldots, c_{nm}) \end{Bmatrix}$$

As a result, the error here is calculated by

$$\mathbf{ER_1 = (y_{11} - c_{11})^2 + (y_{21} - c_{21})^2 + (y_{31} - c_{31})^2 + ... + (y_{n1} - c_{n1})^2}$$

$$\mathbf{ER_2 = (y_{12} - c_{12})^2 + (y_{22} - c_{22})^2 + (y_{32} - c_{32})^2 + ... + (y_{n2} - c_{n2})^2}$$

$$\mathbf{.....}$$

$$\mathbf{.....}$$

$$\mathbf{ER_m = (y_{1m} - c_{1m})^2 + (y_{2m} - c_{2m})^2 + (y_{3m} - c_{3m})^2 + ... + (y_{nm} - c_{nm})^2}$$

The fitness function is further derivd from this root mean square error given by

$$FF_n = \frac{1}{E_{rmse}}$$

The process has to be carried out for all the total number of chromosomes.

10.3.2.4 Reproduction of Offspring

In this process, before the parents produce the offspring with better fitness, the mating pool has to be formulated. This is accomplished by neglecting the chromosome with minimum fitness and replacing it with a chromosome having maximum fimess, In other words, the fittest individuals among clle chromosomes will be given more chances to participate in the generations and the worst individuals will be eliminated. Once the mating pool is formulated, parent paits are selected' randomly and the chromosomes of respective pairs are combined using crossover technique to reproduce offspring. The selection operator is suitably used to select the best parem to participate in the reproduction process.

10.3.2.5 Convergence

The convergence for genetic algorithm is the number of generations with which the fitness value increases towards the global optimum. Convergence is the progression towards increasing uniformiry. When about 95% of the individuals in the population share the same fitness value then we say that a population has converged.

10.3.3 Advantages of Neuro·Genetic Hybrids

The various advantages of Neuro-genetic hybrid are as follows:

- GA performs optimization of neural network parameters with simplicity, ease of operation, minimal requirememts and global perspective.
- GA helps to find out complex structure of ANN for given input and the output data set by using its learning rule as a fimess function.
- Hybrid approach ensembles a powerful model that could significantly improve the predicrability of the system under construction.

The hybrid approach can be applied to several applications, which include: load forecasting, stock forecasting, cost optimization in textile industries, medical diagnosis, face recognition, multi-processor scheduling, job shop scheduling, and so on.

10.4 Genetic Fuzzy Hybrid and Fuzzy Genetic Hybrid Systems

Curremly, several researches has been performed combining fuzzy logic and genetic algorithms (GAs), and there is an increasing interest in the integration of these two topics. The imegration can be performed in the following two ways:

1. By the use offuzzy logic based techniques for imptoving genetic algorithm behavior and modelling GA components. This is called *fozzy genetic algorithms* (FGk).
2. By the application of genetic algorithms in various optimization and search problems involving fuzzy systems.

An FGA is considered as a genetic algorithm that uses techniques or tools based on fuzzy logic to improve the GA behavior modelling. It may also be defined as an ordering sequence of instructions in which some of the instructions or algorithm components may be designed with tools based on fuzzy logic. For example, fuzzy operators and fuzzy connectives for designing genetic operators with different properties, fuzzy logic control systems for controlling the GA parameters according to some performance measures, stop criteria, representation tasks, ere.

GAs are utilized for solving different fuzzy optimization problems. For example, fuzzy flowshop scheduling problems, vehicle routing problems with fuzzy due-time, fuzzy optimal reliability design problems, fuzzy mixed integer programming applied m resource distribution, job-shop scheduling problem with fuzzy processing time, interactive fuzzy satisfying method for multi-objective 0-1, fuzzy optimization of distribution networks, etc.

10.4.1 Genetic Fuzzy Rule Based Systems (GFRBSs)

For modelling complex systems in which classical tools are unsuccessful, due to them being complex or imprecise, an imponant tool in the form of fuzzy rule based systems has been identified. In this regard, for mechanizing the definicion of the knowledge base of a fuzzy controller GAs have proven to be a powerful tool, since adaptive conrtol, learning, and self-organization may be considered in a Joe of cases as optimization or search processes. Over the last few years their advantages have extended the use of GAs in the development of a wide range of approaches for designing fuzzy controllers. In particular, the application to the design, learning and tuning of knowledge bases has produced quite good results. In general these approaches can be termed as Grnttic Fuzzy Systems (GFSs}. Figure 10-6 shows a system where genetic design and fuzzy processing are the two fundamental constituents. Inside GFRBSs, it is possible to distinguish between either parameter optimization or rule generation processes, that is, adaptation and learning.

The main objectives of optimization in fuzzy rule based system are as follows:

1. The task of finding an apptopriate knowledge base (KB) for a particular problem. This is equivalent to parameterizing the fuzzy KB (rules and membership functions).
2. To find those parameter values that are optimal with respect to the design criteria.

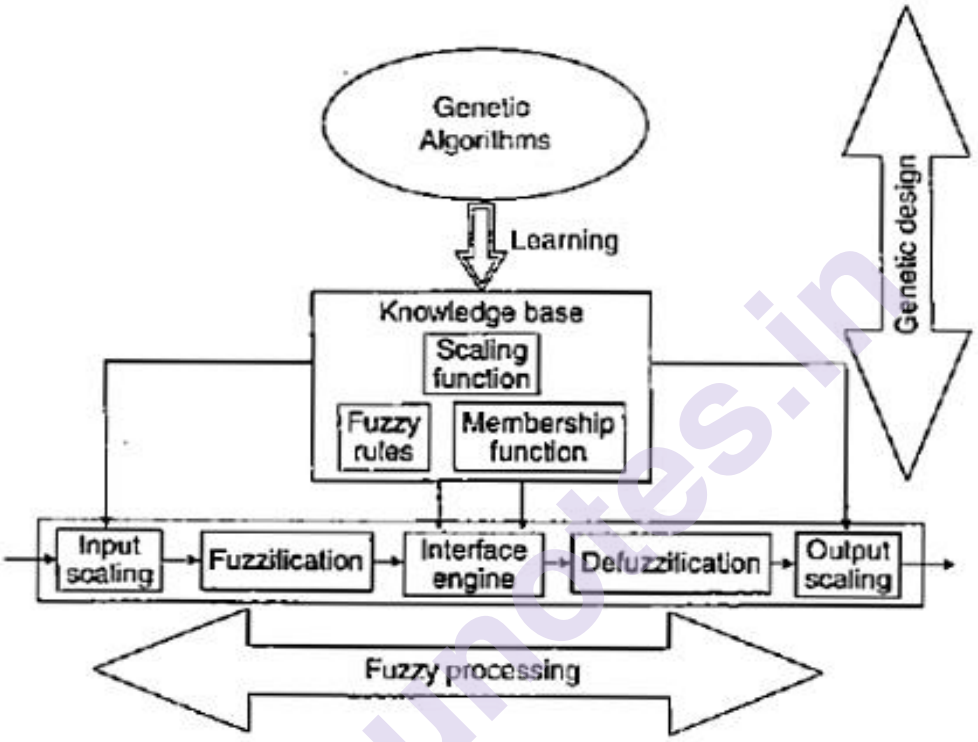


Figure 10·6 Block diagram of genetic fuzzy system

Considering a GFRBS, one has to decide which parts of the knowledge base (KB) are subject to optimization by the GA. The KB of a fuzzy system is the union of qualitatively different components and not a homogeneous structure. As an example, the KB of a descriptive Mathdani-type fuzzy system has two components: a rule base (RB) containing the collection of fuzzy rules and a data base (DB) containing the definitions of the scaling factors and the membership functions of the fuzzy sets associated with the linguistic labels.

In this phase, it is important to distinguish between tuning (alternatively, adaptation) and learning problems. See Table 10-2 for the differences.

10.4.1.1 Genetic Tuning Process

The task of tuning the scaling functions and fuzzy membership function is important in FRBS design. The adoption of parameterized scaling functions and membership functions by the GA is based on the fimess function rhat specifies the design criteria quantitatively. The responsibility of finding a set of optimal parameters for the membership and/or the scaling functions rests with the tuning processes which assume a predefined rule base. The tuning process can he performed a priori also. This can be done if a subsequent process derives the RB once the DR has been obtained, that is a priori genetic DB learning. Figure 5.2-7 illustrates the process of generic tuning.

Tuning Scaling Functions

The universes of discourse where fuzzy membership functions are defined are normalized by scaling functions applied to the input and output variable of FRBSs. In case of linear scaling, the scaling functions are parameterized by a single scaling factor or either by specifying a lower and upper bound. On the other hand, in case of non-linear scaling, the scaling functions are parameterized by one or several contraction / dilation parameters. These parameters are adapted such that the scaled universe of discourse matches the underlying variable range.

Table 10·2 Tuning verses learning problems

Tuning	Learning Problems
It is concrned with optimization of an exisring FRBS.	It constitutes an automated design medhos for fuzzy rule sets that start from scratch.
Tuning processes assume a predefined RB and have the objective to find a et of optimal parameters for the membership and/or the scaling functions, DB parameters.	Learning process perform a more elaborated watch in the space of possible RBs or whole KB and do not depend on a predefined set of rules

Ideally, in these kinds of processes the approach is to adapt one to four parameters per variable: one when using a scaling factor, two for linear scaling, and three or four for non-linear scaling. This approach leads to a fixed length code as the number of variables is predefined as is the number of parameters required to code each scaling function.

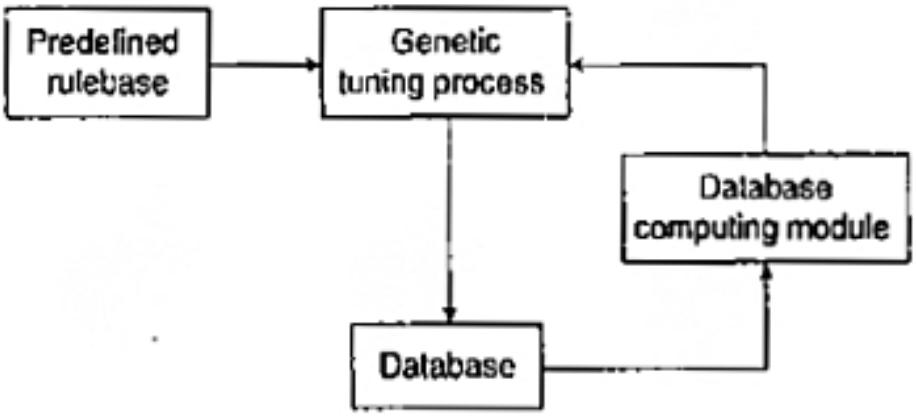


Figure 10·7 Process of ntning the DB.

Tuning Membership Functions

It can be noted that during the runing of membership funcrions, an individual represents the entire DB. This is because its chromosome encodes the parameterized membership functions associated to the linguistic terms in every fuzzy partition considered by the fuzzy rule based system. Triangular (either isosceles or asymmetric), trapezoidal, or Gaussian functions are the most common shapes for ilie membership functions (in GFRBSs). The number of parameters per membership function can vary from one to four and each parameter can be either binary or real coded.

For FRBSs of the descriptive (using linguistic variables) or the approximate (using fuzzy variables) type, the structure of the chromosome is different. In the process of runing the membership functions in a linguistic model, the entire fuzzy partitions are encoded into the chromosome and in order to maintain the global semantic in the RB, it is globally adapted. These approaches usually consider a predefined number of linguistic terms for each variable-with no requirement to be the same for each of them-which leads to a code of fixed length in what concerns membership functions. Despite this, it is possible to evolve the number of linguistic terms associated to a variable; simply define a maximum number (for the length of the code) and let some of the membership functions be located out of the range of the linguistic variable (which reduces the actual number of linguistic terms).

Descriptive fuzzy systems working with sttong fuzzy partitions, is a particular case where the number of parameters to be coded is reduced. Here, the number of parameters to code is reduced to the ones defining the core regions of the fuzzy sets: the modal point for triangles and the extreme points of the core for trapezoidal shapes.

Tuning the membership functions of a model working with funy variables (scatter partitions), on the other hand, is a particular instance of knowledge base learning. This is because, instead of referring to linguistic terms in the DB, the rules are defined completely by their own membership functions.

10.4. 1.2 Genetic *Learning of Rule Bases*

As shown in Figure 10-8, genetic learning of rule bases assumes a predefined set of fuzzy membership functions in the DB to which the rules refer, by means of linguistic labels. As in the approximate approach adapting rules, it only applies to descriptive FRBSs, which is equivalent to modifying the membership functions. When considering a rule based system and focusing on learning rules, there are three main approaches that have been applied in the literature:

1. Pittsburgh approach.
2. Michigan approach.
3. Iterative rule learning approach.

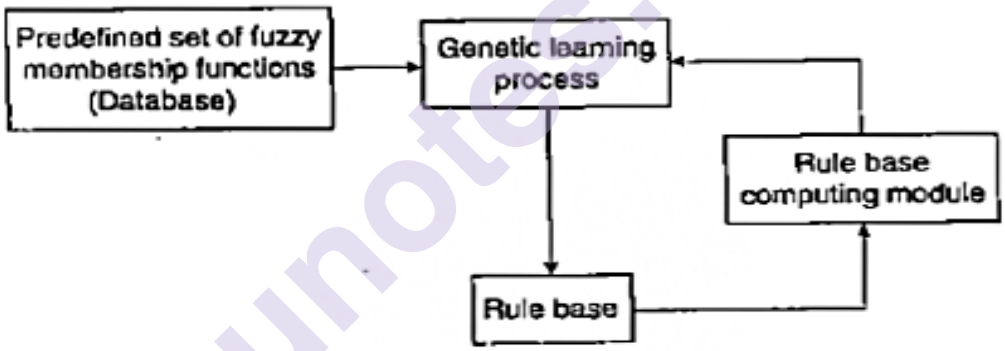


Figure 10·8 Genetic learning of rule base.

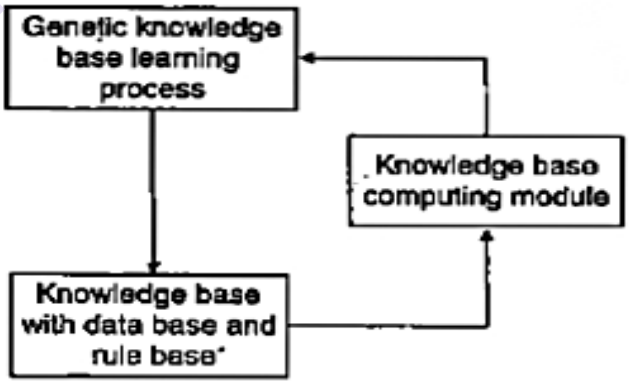


Figure 10-9 Genetic learning of the knowledge base.

The Pittsburgh approach is characterized by representing an entire rule set as a genetic code (chromosome), maintaining a population of candidate rule sets and using selection and genetic operators to produce new generations of rule sets. The

Michigan approach considers a different model where the members of the population are individual rules and a rule set is represented by the entire population. In the third approach, the iterative one, chromosomes code individual rules, and a new rule is adapted and added to the rule set, in an iterative fashion, in every run of the genetic algorithm.

10.4. 1.3 Genetic Learning of Knowledge Base

Genetic learning of a KB includes different genetic represenrarions such as variable length chromosomes, multi-chromosome genomes and chromosomes encoding single rules instead of a whole KB as it deals with heterogeneous search spaces. As the complexity of the search space increases, the computational cost of the genetic search also grows. To combat this issue an option is to maintain a GFRBS that encodes individual rules rather than entire KB. In this manner one can maintain a flexible, complex rule space in which the search for a solution remains feasible and efficient. The three learning approaches as used in case of rule base can also be considered here: Michigan, Pittsburgh, and iterative rule learning approach. Figure 5.2-9 illustrates the genetic learning of KB.

10.4.2 Advantages of Genetic Fuzzy Hybrids

The hybridization between fuzzy systems and GAs in GFSs became an important research area during the last decade. GAs allow us to represent different kinds of structures, such as weights, features together with rule parameters, etc., allowing us m code multiple models of knowledge representation. This provides a wide variety of approaches where it is necessary m design specific genetic components for evolving a specific representation. Nowadays, it is a gtowing research area, where researchers need to reflect in order to advance towards strengths and distinctive features of the GFSs, providing useful advances in the fuzzy systems theory. Genetic algorithm efficiently optimizes the rules, membership functions, DB and KB of fuzzy systems. The methodology adopted is simple and the fittest individual is identified during the process.

10.5 Simplified Fuzzy ARTMAP

The basic concepts of Adaptive Resonance Theory Neural Networks are discussed in Chapter 5. Both the types of ART Networks, ART-1 and ART2, are discussed in derail in Section 5.6.

Apart from these two ART networks, the other two maps are ARTMAP and fuzzy ARTMAP. ARTMAP is also known as Predictive ART. It combines two slightly modified ART-1 or ART-2 units into a supervised learning structure. Here, the first unit rakes the inpm data and the second unit rakes the correct output data. Then

minimum poible adjusrmem of the vigilance parameter in the fim unit is made using the correct output data .o rhar correct classification can be made.

The Fuzzy ARTMAP model has fuzzy-logic-based computations incorporated in the ARTMAP model. Fuzzy ARTMAP is neural network architecture for conducting supervised learning in a multidimensional setting. When Fuzzy ARTMAP is used on learning problem, it is trained till it correctly classifies all uaining data. This feature causes Fuzzy ARTMAP to "overfir" some darasers, especially those in which the underlying panern harm overlap. To avoid the problem of "overfiuing" one must allow for error in the training process.

10.5.1 Supervised ARTMAP System

Figure 5.2-10 shows the super\'ised ARTMAP system. Here, two ART modules are linked by an inrer-ART module called the Map Field. The Map Field forms predictive associations between categories of the ART modules and realizes a march tracking rule. If ARTa and ARTb are disconnected, then each module would be of self-organize category, gtouping their respective in pursers. In supervised mode, the mappings are learned berween input vecmrs *a* and *b.*

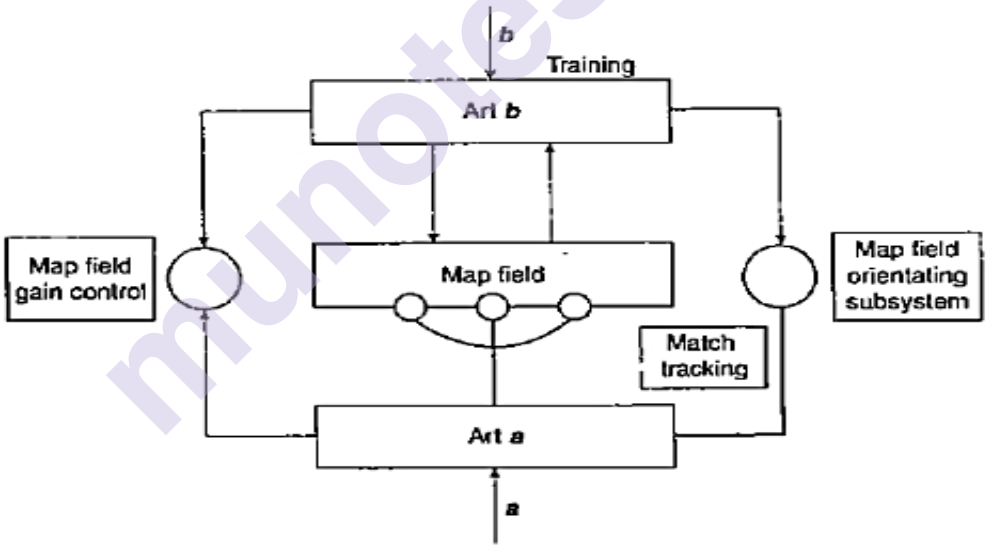


Figure 10·10 Supervised ARTMAP system.

10.5.2 Comparison of ARTMAP with BPN

1. ARTMAP networks are self-stabilizing, while in BPNs the new information gradually washes away old information. A consequence of rhis is rhat a BPN has separate training and performance phases while ART MAP systems perform and learn at the same time.

2. ART MAP networks are designed to work in real-time. While BPNs are typically designed to work off-lilne at least during their training phase.

3. ARTMAP system can learn both in a fast as well as inslow match configuration, while, the BPN can only learn in slow mismatch configuration. This means that an ARTMAP system learns, or adaprts its weights, only when the input matches an established category, while BPNs learn when the input does not match an established category.

4. In BPNs there is always a chance to the system getting trapped in a local minimum while this is impossible for ART systems.

However, the system based on ART modules learning may depend upon the ordering of the input patterns.

10.6 Summary

In this chapter, the various bybrids of individual neural networks, fuzzy logic and genetic algorithm have been discussed in detail. The advantages of each of these techniques are combines together to give a better solution to the problem under consideration. Each of these systems possesses certain limitations when they operate individually and these limitations are met by bringing out the advantages of combining these systems. The hybrid systems are found to provide better solution for complex problems and the advent of hybrid systems makes it applicable to be applied in various application domains.

16.7 Solved Problems using MATLAB

1. Write a MATLAB program to adapt the given input to write wave from using adaptive neun-fuzzy hybrid technique.

```
Epocha = 570;

Creating fuzzy inference engine
Fix = genfisl (trndata, mfs);
Plotfix (fix);
Figure
R=showrule (fis);

Creating adaptive neuro fuxxy inference engine
Nfix – anfis (trndata, fix, epocha);
R1 = showrule (infis);
```

```
Evaluating anfix with given input
Y = evalfix (x, nfis);
Disp ('The output data from ansif : ');
disp(y);

calculating error rate
e=y-t;
plot(e);
title ('Error rate')
figure

ploting given training data and anfix output
plot (x, t, 'o', x, y, …);
title ('Training data vs Out data');
legend ('Training data', 'ANFIS Output');
```

Output

The input data given x is :

```
 0
0.3000
0.6000
0.9000
1.2000
1.5000
1.8000
2.1000
2.4000
2.7000
3.0000
3.3000
3.6000
3.9000
4.2000
4.5000
4.8000
5.1000
5.4000
5.7000
6.0000
```

6.3000
6.6000
6.9000
7.2000
7.5000
7.8000
8.1000
8.4000
8.7000
9.0000
9.3000
9.6000
9.9000
10.2000
10.5000
10.8000
11.1000
11.4000
11.7000
12.0000
12.3000
12.6000
12.9000
13.2000
13.5000
13.8000
14.1000
14.4000
14.7000
15.0000
15.3000
15.6000
15.9000
10.2000
10.5000
10.0000
17.1000
17.4000
17.7000

18.0000
18.3000
18.6000
18.9000
19.2000

0.9437
0.9993
0.9657
0.8457
0.6503
0.3967
0.1078
-0.1909
-0.4724
-0.7118
-0.8876
-0.9841
-0.9927
-0.9126
-0.7510
-0.5223
-0.2470
0.0504
0.3433
0.6055
0. 8137
ANFIS info:
Number of nodes: 32

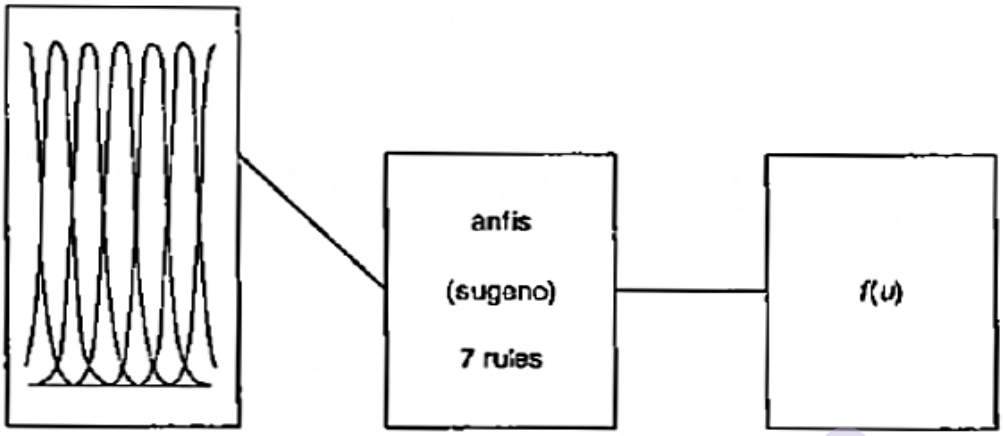
Number of linear parameters: 14
Number of nonlinear parameters: 21
Total number of parameters: 35
Number of training data paits: 67
Number of checking data paits: 0
Number of fuzzy rules: 7
Start training ANFIS
1 0.0517485
2 0. 0513228
3 0.0508992

4 0.0504776
5 0.0500581
Step size increases to 0.011000 after epoch 5.
6 0.0496406
7 0.0491837
a o.o4S7291
568 0.00105594 -

Hybrid Soft Comp"t;og Techo;q,es
Designated epqch number reahed--->ANFIS training completed at epoch 570.
The output data from anfis':
-0.0014
0.2981
0.5647
0.7817
0.9314
0.9984
0.9747
0.8629
0.6746
0. 4271
0.145.2
-0.1571
-0.4425
-0.6884
-0.8720
-0.9772
-0.9955
-0.9260
-0.7735
-0.5509
-0.2788
0.0174
0. 3112
0. 5777
0.7935
0.9387
0.9991
0.9697

0.8540
0.6627
0.4122
0.1247
-0.1741
-0.4574
-0.7000
-0.8801
-0.9812
-0.9941
-0.9189
-0.7623
-0.5371
-0.2629
0.0346
0.3277
0.5908
0.8024
0.9442
1.0014
0.9667
0.8443
0.6484
0.3969
0.1093
-0.1900
-0.4731
-0.7130
-0.8879
-0.9833
-0.995.2
-0.9125
-0.7521
-0.5232
-0.2457
0.0526
0.3426
0.6015
0.85.23
<end of program>

Figure 10-11 illustrates the ANFIS system module; figure 10-12 the error me; and Figure 10-13 the performance of training dam and output data. Thus ir can be noted from Figure 10-13, that an f is has adapted the given inpur to sine wave form.



Input 1 (7) output (7)

System anfix: 1 inputs, 1 outputs, 7 rules

Figure 10-11 ANFIS system module.

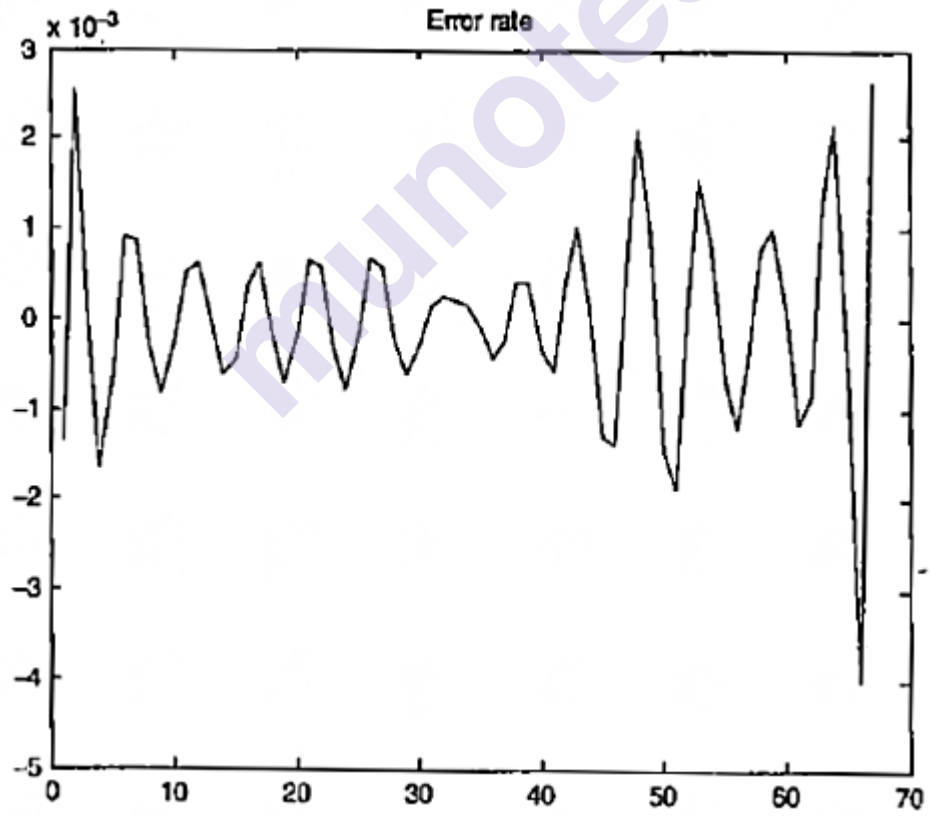


Figure 10-12 Arror rate.

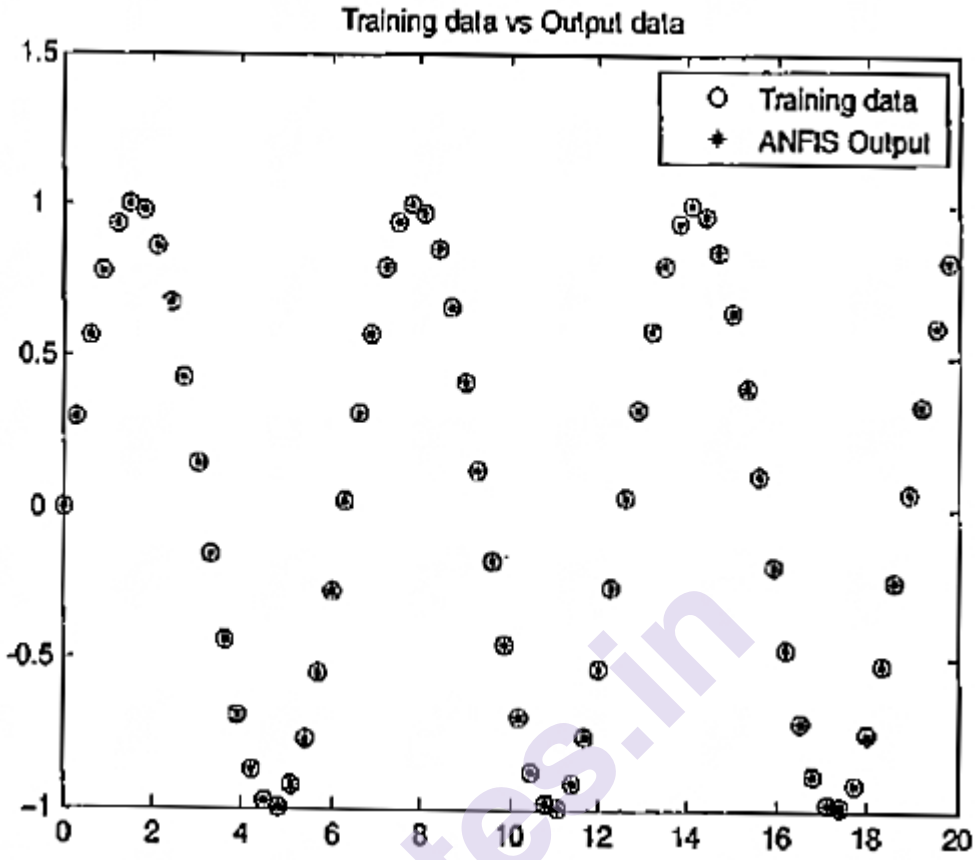


Figure 10-13 Performance of training data and output data.

2. Write a MATLAB program to recognize the given input of alphabets to its respective output using adaptive Neuro-fuzzy hybrid technique.

Source code

```
%program to recognize the given input of .alphabets to its respective
%outputs using adaptive Neuro fuzzy hybrid technique.
'clc;
clear all;
close all;

%input data
x= [0,l,0,0;l,0,l,l;l,l,l,2;1,0,1,3;1,0,1,4;
    1,1,0,5;1,0,1,6;1,1,0,7;1.0,1,8;1,1,0,9;
    0,1,1,10;1,0,0,11;1,0,0,12;1,0,0,13;0,1,1,14;
    1,1,0,15;1,0,1,5.2;1,0,1,17;1,0,1,18;1,1,0,19;
    1, 1, 1, 20; 1, 0, 0, 21; 1, 1, 0, 22; 1, 0, 0, 23; 1, 1, 1, 24; ]
```

```
%target data
t:::[0;0;0;0;0;
      1;1;1;1;1;
      2;2;2;2;2;
      3;3;3;3;3;
      4;4;4;4;4; 1

%training data
trndata= [x, t);
mfs::o3;
epochs=400;

%creating fuzzy inference engine
fis=genfis1(trndata,mfs);
plotmf ( fis, 'input' , 1) ;
r=showrule(fis);

%creating adaptive Neuro fuzzy inference engine
nfis = anfis(trndata,fis,epochs);
surfview(nfis);
figure
r1=showru1e(nfis);

%evaluating anfis with given input
Y=eva1fis(x,nfis);
disp('The output data from anfis:'):
disp(y);

%calculating error rate
esy-t;
plot (e);
title('Error rate');
figure

%ploting given training data and anfis output
Plot (x,t, ‘or’, x,y, ‘kx’);
Title (Training data vs Output data’)
Legend (‘Training data’, ANFIS Output’, ‘location’, North’);
```

Output

X =

0	1	0	0
1	0	1	1
1	1	1	2
1	0	1	3
1	0	1	4
1	1	0	5
1	0	1	6
1	1	0	7
1	0	1	8
1	1	0	9
0	1	1	10
1	0	0	11
1	0	0	12
1	0	0	13
c	1	1	14
1	1	0	15
1	0	1	5.2
1	0	1	17
1	0	1	18
1	1	0	19
1	1	1	20
1	0	0	21
1	1	0	22
1	0	0	23
1	1	1	24

t =

0
0
0
0
0
1
1
1
1
1
2

2
2
2
2
3
3
3
3
3
4
4
4
4
4

ANFIS info:

Number of nodes: 193
Number of linear parameters: 405
Number of nonlinear parameters: 36
Total number of parameters: 441
Number of training data paits: 25
Number of checking data paits: 0
Number of fuzzy rules: 81

Start training ANFIS

1 0.08918
2 0.0889038
3 0.0886229
4 0.0883371
5 0.0880464

Step size increases to 0.011000 after epoch 5.

6 0.0877506
7 0.0874193
.
.
.
.
398 0.001025.21
399 0.00102102
400 0.0010191

Step size increases to 0.003347 after epoch 400.

Designated epoch number reached --> ANFIS training completed at epoch 400.

The output data from anfis:

-0.0000
0.0009
0.0000
-0.0031
0.0024
1. 0000
0.9997
1. 0000
1.0002
1.0001
2.0000
2.0001
1. 9998
2.0001
2.0000
2.9999
2.9982
3.0022
2.9994
3.0001
4.0000
4.0000
3.9999
4.0000
4.0000

<end of program>

Figure 10.14 shows the degree of membership. Figure 10.15 illusrmes the surface view of the given system; Figure 10.16 the error rate; and Figure 10.17 the performance of training data with output data.

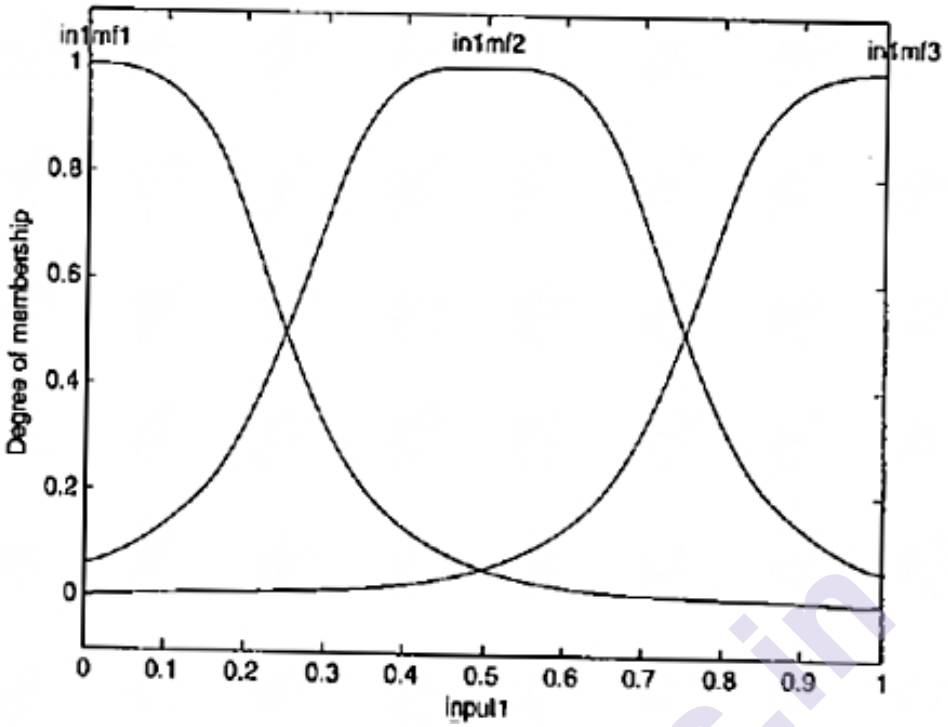


Figure 10.14 Degree of membership.

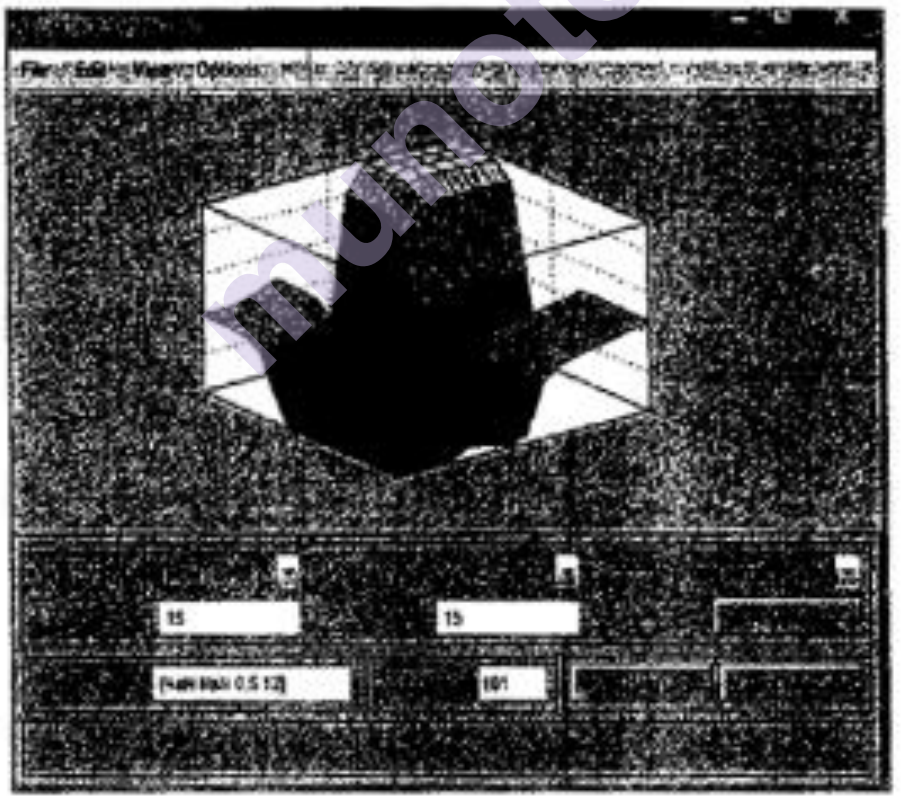


Figure 10.15 Surface view of the given system.

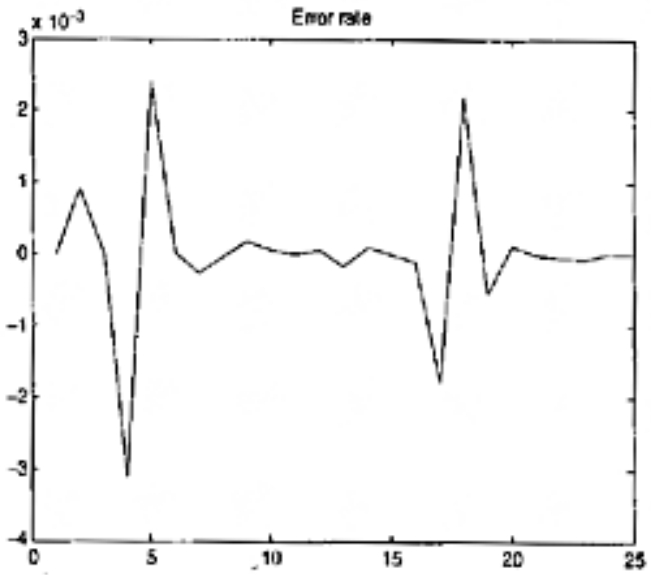


Figure 16-16 Error rate.

Figure 10.16 Error rate.

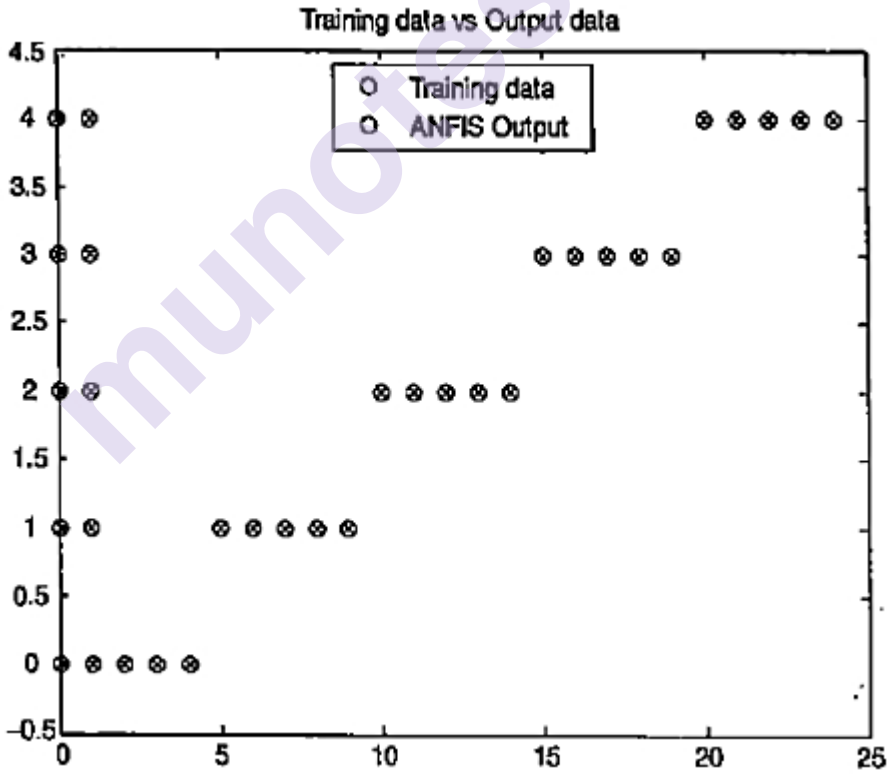


Figure 10·17 Performance of training data with output data.

3. Write a MATI.AB program m train the given trmh table using adaptive Neuro-fuuy hybrid technique. Source code

```
% Program to train the given truth table using adaptive Neuro fuzzy %hybrid technique.
clc;
clear all;
close all;
%input data
X = [ 0, 0, 0; 0, 0, 1; 0, 1, 1;  0, 0,1,1,1,0,0,1,1,1,0,1,1,1;]

%target data
c=[0,0,0,1,0,1,1,1]

%training data
trndata= [x, t];
mfs=3;
mfType = 'gbellmf';
epochs=49;

%creating fuzzy inference engine
Fis=genfisl (trndata, mfs, mfType);
Plotfis (fis);
title ('The created fuzzy logic');

figure
plotmf(fis, 'input',l);
title ('The membership function of the fuzzy'):
surfview ( fis) ;
figure
ruleview ( fis) ;
r=showrule(fis);

%creating adaptive Neuro fuzzy inference engine
nfis = anfis (trndata, fis, epochs);
plotfis (nfis);
title ('The created anfis');
figure
plotmf (nfis,'input',l);
title ('The membership function of the anfis');
surfview (nfis);
figure
ruleview(nfis);
rl=showrule(nfis);
```

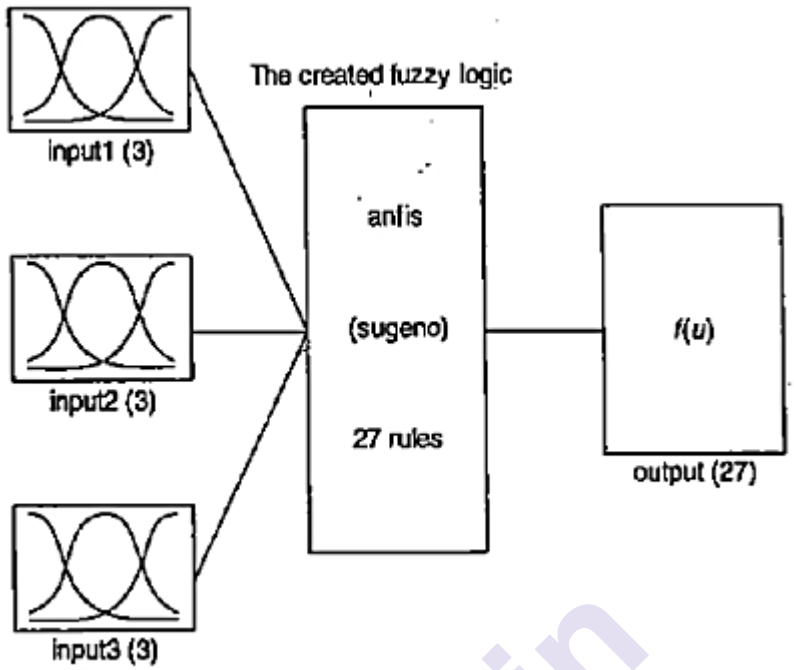
```
%evaluating anfis with given input
y=evalfis (x,nfis);
disp ('The output data from anfis:');
disp (y);
%calculating error rate
e=y-t;
plot(e);
title('Error rate');
figure

%plating given training data and anfis output
plot (x, t, 'o' ,x,y, '*' l;
title ('Training data vs Output data');
legend ('Training data','ANFIS Output');
```

Output

```
X =
     0     0     0
     0     0     0
     0     1     0
     0     1     1
     1     0     0
     1     0     1
     1     1     0
     1     1     1

T =
     0
     0
     0
     1
     0
     1
     1
     1

ANFIS info:
     Number of nodes: 78
     Number of linear parameters: 108
     Number of nonlinear parameters: 27
     Total number of parameters: 135
```

```
    Number of training data paits: 8
    Number of checking data paits: 0
    Number of fuzzy rules: 27

Start training ANFIS …
    1    3.13863e-007
    2    3.0492e-007
    3    2.9784le-007
    4    2. 90245e-007
    5    2.84305e-007

Step size increases to 0.011000 after epoch 5
    6    2.78077e-007
    .
    .
    .
    .
    47   2.22756e-007
    48   2.22468e-007
    49   2.22431e-007

Step size increases to 0.015627 after epoch 49.

Hybrid Soft Computing Techniques

Designated epoch number reached --> ANFIS training completed at epoch 49.

The output data from anfis:
    -0.0000
    0.0000
    0.0000
    1.0000
    0.0000
    1.0000
    1.0000
    1.0000
<end of program>
```

Figure 10-18 shows the ANFIS module for the given system with specified inputs. Figure 10-19 illustrates the rule viewer for the ANFIS module. Figure 10-20 gives the error rate. Figure 10-21 shows the performance of Training data and ourpur data.



System anfis : 3 inputs, 1 outputs, 27 rules.

Figure 10-18 ANFIS module for the given system with specified inputs.

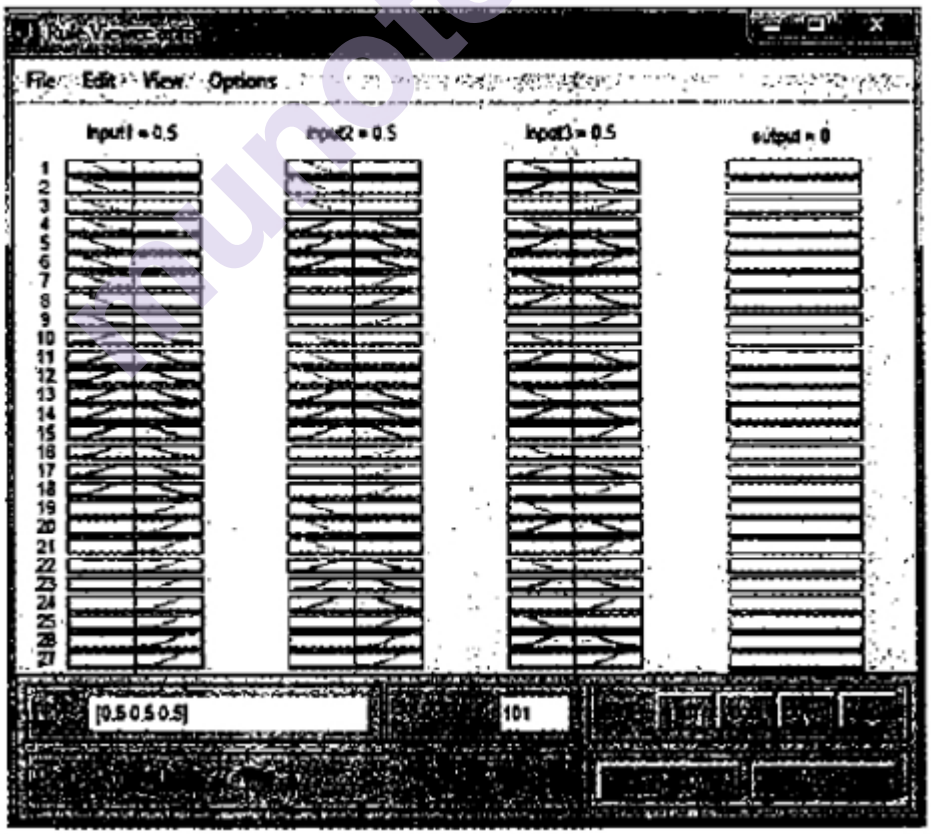


Figure 10-19 Rule viewer for the ANFIS module.

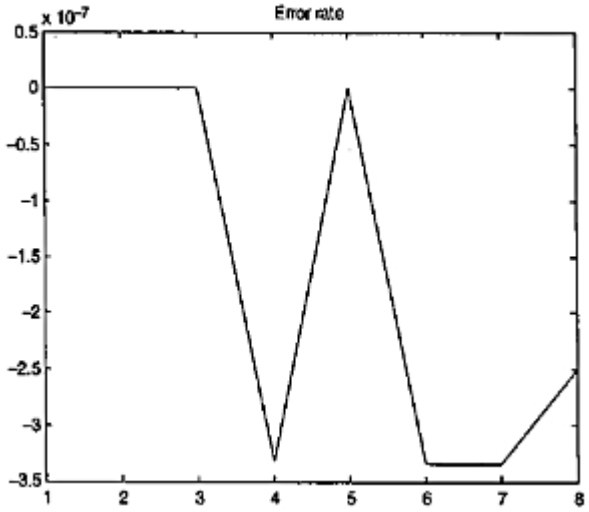


Figure 10-20 Error rate.

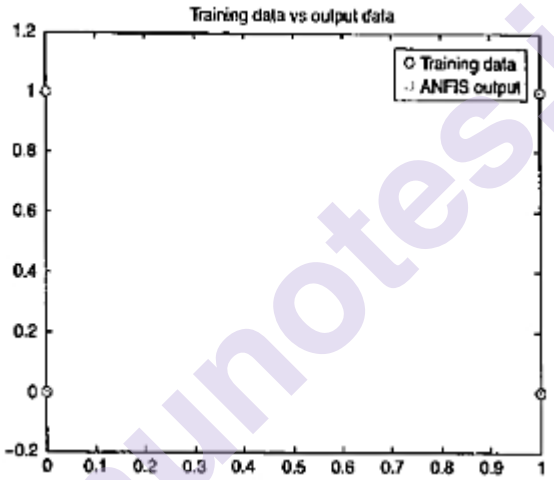


Figure 10-21 Performance of training data and output data.

4. Write a MATLAB program to optimize the neural network parameters for the given truth table using genetic algorithm.

Source code

```
%Program to optimize the neural network parameters from given truth table
%using genetic algorithm

clc;
clear all;
close all;

%input data
p = [ 0 0 1 1; 0 1 0 1 ];
```

```
%target data
T  =  [ -1  1  -1  1 ];

%creating a feedforeward neural network
net=newff (minrnax(p), [2,1]);

%creating two layer net with two Neurons in hidden (1) layer
net.inputs (l).size = 2;
net.numLayers = 2;

%initializing network
net= init(net);
net.initFcn = 'initlay';

%initializing weights and bias
net.layers{1}.initFcn = 'initwb';
net.layers{2).initFcn = 'initwb';

%Assigning weights and bias from function 'gawbinit'
net.inputWeights{1,1).initFcn = 'gawbinit';
net.layerWeights{2,1).initFcn = 'gawbinit';
net.biases{l}.initFcn='gawbinit';
net.biases{2).initFcn='gawbinit';

%configuring training parameters
net.trainParam.lr = 0.05; %learning rate
net.trainParam.min_grad=Oe-10; %min. gradient
net.trainParam.epochs = 60; %No. of iterations
%Training neural net
net=train(net,p,t);

%simulating the net with given input
y = sim (net,p);
disp ('The output of the net is : ');
disp (y};
plating given training data and anfis output
plot (p,t,'o',p,y, '*');
title ('Training data vs Output data');

%calculating error rate
e= gsubtract (t,y); % e=t-y
disp ('The error (t-y) of the net is :');
disp (e);
```

```
%program to calculate weights and bias of the net
function outl  = gawbinit (inl, in2, in3, in4, in5,-)
%%=========================================================

%Implementng genetic algorithm

%configuring ga arguments
A= [ ]; b = [ ]; %:              linear constraints
Aeq = [ ]; beq = [ ];            %linear inequalities
lb = {-2 -2 -2 -2 -2 -2];        %lower bound
ub = [2 2 2 2 2 2];              %upper bound

%ploting ga parameters
Options = gaoptimset ('PlotFcns',{Egaplotscorediversity,Egaplotbest f)};

%creating a multi objective genetic algorithm
%number of variables , for 2 layer 1 output 5 Neuron net there are
%6 weights and 3 biases (6+3=9)
nvars=9;
[X, fval, exitFlag, Output]=gamultiobj{@fitnesfun,nvars,A,b,Aeq,beq,lb,

figure

%displaying the ga output parameters
disp(X);
fprint f ('The number of generations was : %d\n', Output.generations);
fprint f ('The number of function evaluations was : %d\n', OUtput.funccount);
fprint f ('The best function value found was %g\n', fval);
%%=========================================================

%Assigning the values of weights and bias respectively
%getting information of the net
persistent INFO;
if isempty (INFO), INFO= nnfcnWeightinit(mfilename,'Random Symmetric',
7.0, ... true, true, true, true, true, true, true, true); end
if ischar (inl)
switch lower (inl)
case 'info', outl =INFO;
%configuring function
case 'configure'
outl = struct;
```

```
case 'initialize'
%selecting input weights , layer·weights and bias separately
switch(upper(in3))
case {'IW') %for input weights·
if INFO.initinputWeight
if in2.inputConnect(in4,in5)
x=X; %Assigning ga output 'X' to input weights
%Taking first 4 ga outputs to cFeate input weight matrix 'wi'

wi(l,l)=x(l,l);
wi{1,2)=x{1,2);
wi(2,l)=x(l,3);
wi(2,2)=x(1,4);
disp(wil;
outl = wi;%Returning input layer matrix
else
outl = [ ];
end
else
505
nerr.thtow([upper(mfilename) ' does not initialize input weights.']);
end
case {'LW'} %for layer weights
if INFO.initLayerWeight
if i2.layerConnect{in4,in5)
x=X; %Assigning ga output 'X' to layer weights
%Taking 7th and 8th ga outputs to create layer weight matrix 'wl'
wl(l,l)=x{l, 7);
wl{1,2)=x(l,Bl;
disp (wl);
outl = wl;%Returning layer 1•1eight matrix
else
outl [ ];
end
else
nnerr.thtow([upper(mfilename) ' does not initialize input weights.']);
end
case {'B'} %for bias
if INFO.initBias
```

```
if in2.biasConnect{in4)
x=X; %Assigning ga output 'X' to bias
%Taking 5th, 6th and 9th ga outputs to create bias matrix 'bl'
bl[l)=x{l,5);
bl[2)=x[1,6);
bl [3) =x(l, 9);
disp(bl);
outl = bl;
else



end
 [];%Returning bias matrix
nnerr.thtow([upper(mfilename) ' does not initialize biases.']);
end
otherwise,
end
end
end
end
nnerr.thtow('Unrecognized value type.');
%Creating fitness function for genetic algorithm
function z = fitnesfun(e)
%The error(t-y) for all 4 i/o paits are summed to get overall error
%For 4 input target paits the overall error is divided by 4 to get average
%error value (1/4=0.25)
z=0.25*surn(abs(e));
end
```

Output

```
Optimization terminated: average change in the spread of Pareto solutions
less than options.TolFun.
Columns 1 thtough 7
0.0280 0.0041 0.0112 0.0069 0.0050
Columns 8 thtough 9
0.0018 0.0003
The number of generations was : 102
The number of function evaluations was : 13906
The best function value found was : 0.0177734
0.0062 0.0075
```

Optimization terminated: average change in the spread of Pareto solutions less than options.TolFun.
Columns 1 thtough 7
0.0012 0.0020 0.0096 0.0014 0.0018
Columns a thtough 9
0.0084 0.0025
The number of generations was 102
The number of function evaluations was : 13906
The best function value found was 0.00988699
The output of the net is :
-1.0000 1.0000 -1.0000 1.0000
The error (t-y) of the net is :
1.0e-011
-0.3097 0.2645 -0.2735 0.3006
0.0044 0.0084

Figure 10.22 shows the plot of the generations versus fitness value and histogram. Figure 10-23 illustrates the Neural Nenvork Training Tool for the given input and output paits. Figure 10-24 shows the neural network training p.erf'Oimance. Neural necwork training state is shown in Figure 10-25. Figure 10-26 displays the performance of uaining data versus output data.

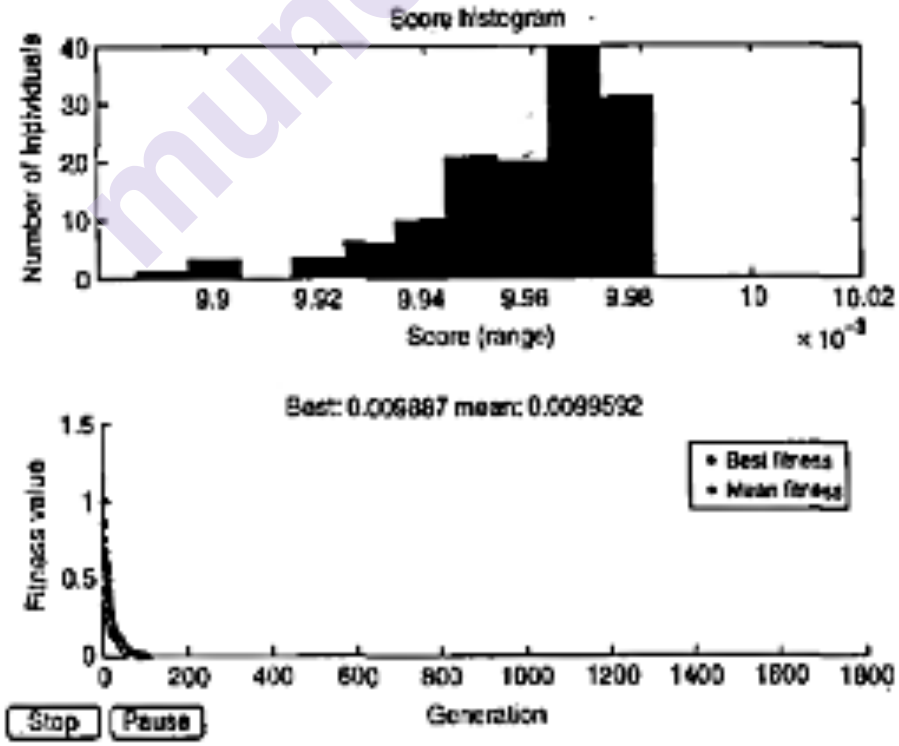


Figure 16-22 Plot of the generations versus fitness value and histogram.

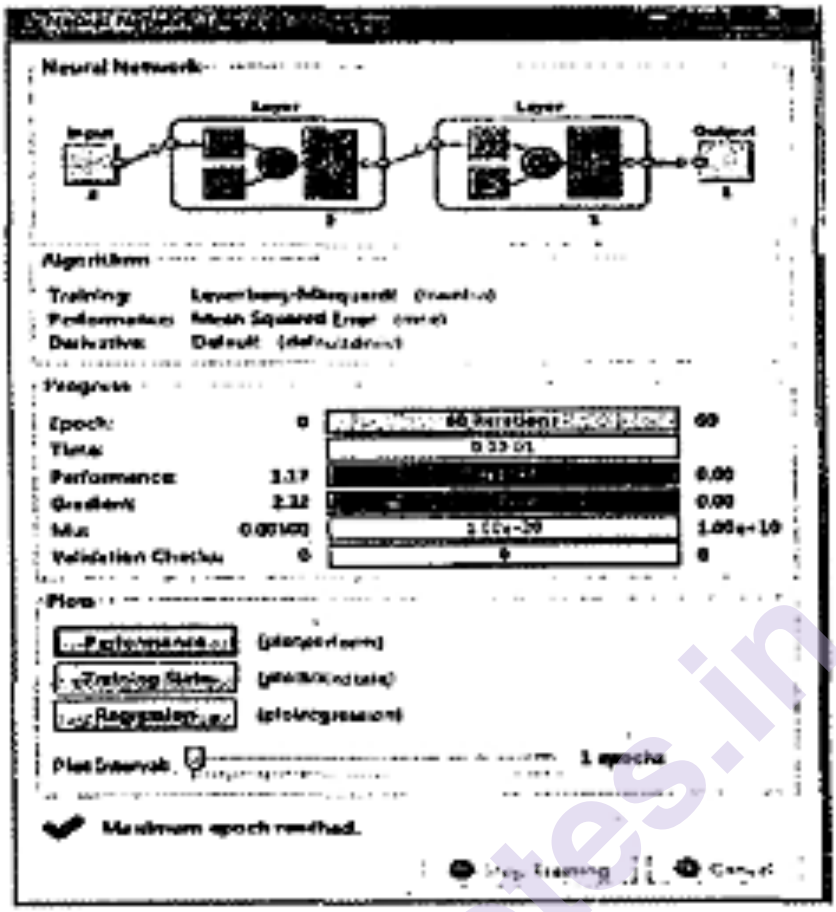


Figure 16-23 Neural Network Training Tool for the given input and output pairs.

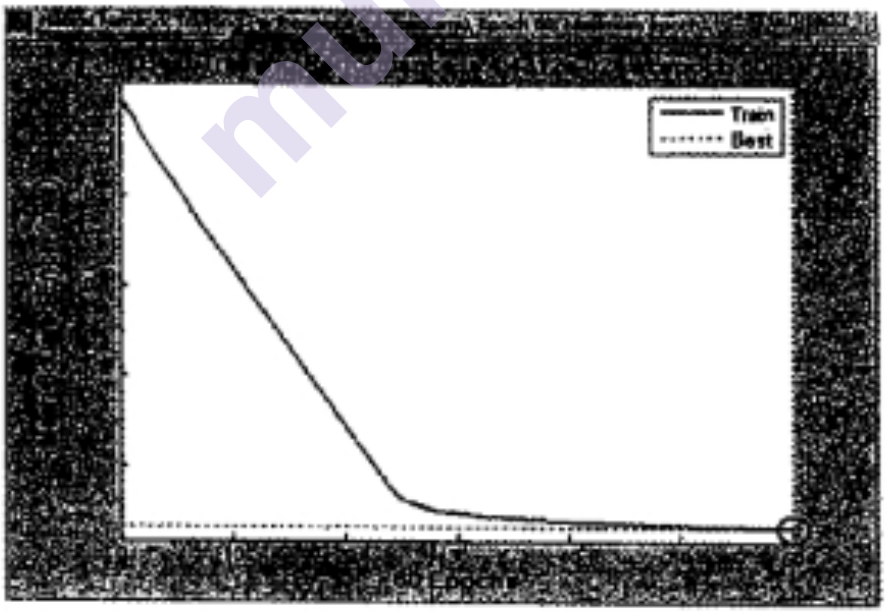


Figure 16-24 Neural network training performance.

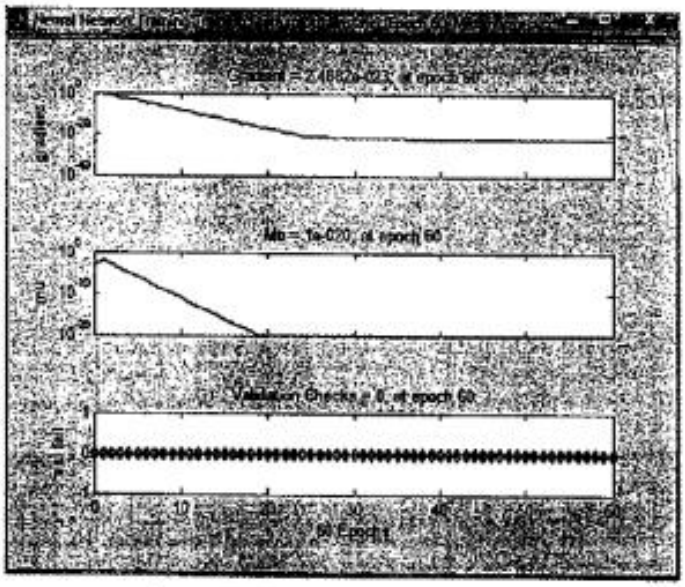


Figure 16-25 Neural network training state.

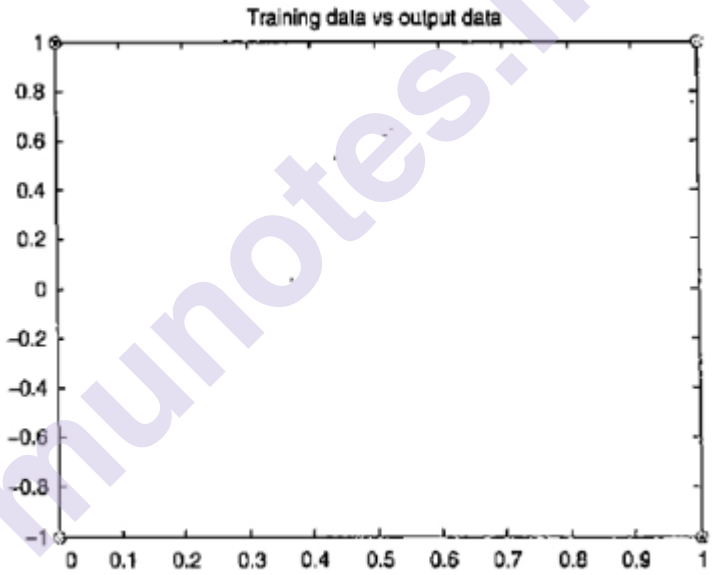


Figure 16-26 Performance of training data versus output data.

10.8 Review Questions

1. State the limitations of neural networks and fuzzy systems when operated individually.
2. List the various cypes of hybrid systems.
3. Mention the characteristics and properties of Neuro-fuzzy hybrid systems.
4. What are the classifications of Neuro-fuzzy hybrid sysrems? Explain in derail any one of the Neuro-fuzzy hybrid systems.

5. Give derails on the various applications of ncutofuzzy hybrid systems.
6. How are genetic algorithms utilized for optimizing the weights in neural network archirecmre?
7. Explain in derail the concepts of fuzzy genetic hybrid *systems.*
8. Differentiate: ARTMAP and Fuzzy ARTMAP, Fuzzy ARTMAP and back-Propagation neural networks.
9. Write nares on the supervised fuzzy ARTMAPs.
10. Give description on the operation of ANFIS Editor in MATI.AB.

Exercise Problems

1. Write a MATLAB program m train NAND gate with binary inputs and targets (rwo input-one Output) using adaptive Neuro-fuzzy hybrid technique.
2. Consider some alphabets of your own and recognize the assumed characters using ANFIS Editor module in MATLAB
3. Perform Problem 2 for any assumed numeral charaaers. .
4. Design a genetic algoriilim to optimize the weights of a neural network model while training Hybrid Soft Computing Techniques an OR gate wiili 2 bipolar inputs and 1 bipolar targets.
5. Write a MATLAB M·file program for the working of washing machine using fuzzy genetic hybrids**.**

REFERENCES:

S.Rajasekaran, G. A. Vijayalakshami , Neural Networks, Fuzzy Logic and Genetic Algorithms: Synthesis & Applications, Prentice Hall of India, 2004

https://neptune.ai/blog/adaptive-mutation-in-genetic-algorithm

https://www.cs.ucdavis.edu/~vemuri/classes/ecs271/The%20GP%20Tutorial.htm

https://link.springer.com/article/10.1007/BF00175354

❖❖❖