LECTURE

NOTES ON

SOFT COMPUTING

M. Tech II semester

Prepared by

Ms. K. Sai Saranya, Assistant Professor, Dept. of CSE

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

DUNDIGAL, HYDERABAD - 500 043

UNIT I

INTRODUCTION TO NEURAL NETWORKS

Soft Computing refers to a partnership of computational techniques in computer science, artificial intelligence, machine learning and some engineering disciplines, which attempt to study, model, and analyze complex phenomena. The principle partners at this juncture are fuzzy logic, neuron-computing, probabilistic reasoning, and genetic algorithms. Thus the principle of soft computing is to exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low cost solution, and better rapport with reality. Learn more in: Adaptive Neuro-Fuzzy Systems.

FUNDAMENTAL CONCEPTS:

Neural networks are parallel computing devices, which is basically an attempt to make a computer model of the brain.

The main objective is to develop a system to perform various computational tasks faster than the traditional systems. These tasks include pattern recognition and classification, approximation, optimization, and data clustering.

What is Artificial Neural Network?

Artificial Neural Network (ANN) is an efficient computing system whose central theme is borrowed from the analogy of biological neural networks. ANNs are also named as "artificial neural systems," or "parallel distributed processing systems," or "connectionist systems." ANN acquires a large collection of units that are interconnected in some pattern to allow communication between the units. These units, also referred to as nodes or neurons, are simple processors which operate in parallel.

Every neuron is connected with other neuron through a connection link. Each connection link is associated with a weight that has information about the input signal. This is the most useful information for neurons to solve a particular problem because the weight usually excites or inhibits the signal that is being communicated. Each neuron has an internal state, which is called an activation signal. Output signals, which are produced after combining the input signals and activation rule, may be sent to other units.

EVOLUTION OF ARTIFICIAL NEURAL NETWORKS:

The history of ANN can be divided into the following three eras −

ANN during 1940s to 1960s

Some key developments of this era are as follows −

- **1943** − It has been assumed that the concept of neural network started with the work of physiologist, Warren McCulloch, and mathematician, Walter Pitts, when in 1943 they modeled a simple neural network using electrical circuits in order to describe how neurons in the brain might work.
- **1949** − Donald Hebb's book, *The Organization of Behavior*, put forth the fact that repeated activation of one neuron by another increases its strength each time they are used.
- **1956** − An associative memory network was introduced by Taylor.
- **1958** − A learning method for McCulloch and Pitts neuron model named Perceptron was invented by Rosenblatt.
- **1960** − Bernard Widrow and Marcian Hoff developed models called "ADALINE" and "MADALINE."

ANN during 1960s to 1980s

Some key developments of this era are as follows −

- **1961** − Rosenblatt made an unsuccessful attempt but proposed the "backpropagation" scheme for multilayer networks.
- **1964** − Taylor constructed a winner-take-all circuit with inhibitions among output units.
- **1969** − Multilayer perceptron (MLP) was invented by Minsky and Papert.
- **1971** − Kohonen developed Associative memories.
- **1976** − Stephen Grossberg and Gail Carpenter developed Adaptive resonance theory.

ANN from 1980s till Present

Some key developments of this era are as follows −

- **1982** − The major development was Hopfield's Energy approach.
- **1985** − Boltzmann machine was developed by Ackley, Hinton, and Sejnowski.
- **1986** − Rumelhart, Hinton, and Williams introduced Generalised Delta Rule.
- **1988** − Kosko developed Binary Associative Memory (BAM) and also gave the concept of Fuzzy Logic in ANN.

Soft computing characterstics:

The historical review shows that significant progress has been made in this field. Neural network based chips are emerging and applications to complex problems are being developed. Surely, today is a period of transition for neural network technology.

Human expertise Soft computing utilizes human expertise in the form of fuzzy if-then rules, as well as in conventional knowledge representations, to solve practical problems. Biologically inspired computing models Inspired by biological neural networks, artificial neural networks are employed extensively in soft computing to deal with perception, pattern recognition, and nonlinear regression and classification problems. New optimization techniques Soft computing applies innovative optimization methods arising from various sources; they are genetic algorithms (inspired by the evolution and selection process), simulated annealing (motivated by thermodynamics), the random search method, and the downhill Simplex method. These optimization methods do not require the gradient vector of an objective function, so they are more flexible in dealing with complex optimization problems. Numerical computation Unlike symbolic AI, soft computing relies mainly on numerical computation. Incorporation of symbolic techniques in soft computing is an active research area within this field. New application domains Because of its numerical computation, soft computing has found a number of new application domains besides that of AI approaches. These application domains are mostly computation intensive and include adaptive signal processing, adaptive control, nonlinear system identification, nonlinear regression, and pattern recognition. Model-free learning Neural networks and adaptive fuzzy inference systems have the ability to construct models using only target system sample data. Detailed insight into the target system helps set up the initial model structure, but it is not mandatory. Intensive computation Without assuming too much background knowledge of the problem being solved, neuro-fuzzy and soft computing rely heavily on high-speed number-crunching computation to find rules or regularity in data sets. This is a common feature of all areas of computational intelligence. Fault tolerance Both neural networks and fuzzy inference systems exhibit fault tolerance. The deletion of a neuron in a neural network, or a rule in a fuzzy inference system, does not necessarily destroy the system. Instead, the system continues pedorming because of its parallel and redundant architecture, although performance quality gradually deteriorates. Goal driven characteristics Neuro-fuzzy and soft computing are goal driven; the path leading from the current state to the solution does not really matter as long as we are moving toward the goal in the long run. This is particularly true when used with derivative-free optimization schemes, such as genetic algorithms, simulated annealing, and the random search method. Domainspecific knowledge helps reduces the amount of computation and search time, but it is not a requirement. Real-world applications Most real-world problems are large scale and inevitably incorporate built-in uncertainties; this precludes using conventional approaches that require detailed description of the problem being solved. Soft computing is an integrated approach that can usually utilize specific techniques within subtasks to construct generally satisfactory solutions to real-world problems. The field of soft computing is evolving rapidly; new techniques and applications are constantly being proposed. We can see that a firm foundation for soft

computing is being built through the collective efforts of researchers in various disciplines all over the world. The underlying driving force is to construct highly automated, intelligent machines for a better life tomorrow, which is already just around the corner

MODEL OF ARTIFICIAL NEURAL NETWORK:

The following diagram represents the general model of ANN followed by its processing.

The inventor of the first neuro computer, Dr. Robert Hecht-Nielsen, defines a neural network as − "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." Basic Structure of ANNs The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living neurons and dendrites. The human brain is composed of 86 billion nerve cells called neurons. They are connected to other thousand cells by Axons. Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.

ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value. Each link is associated with weight. ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN

Types of Artificial Neural Networks There are two Artificial Neural Network topologies − FeedForward and Feedback. FeedForward ANN The information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.

FeedBack ANN: Here, feedback loops are allowed. They are used in content addressable memories.

Working of ANNs In the topology diagrams shown, each arrow represents a connection between two neurons and indicates the pathway for the flow of information. Each connection has a weight, an integer number that controls the signal between the two neurons. If the network generates a "good or desired" output, there is no need to adjust the weights. However, if the network generates a "poor or undesired" output or an error, then the system alters the weights in order to improve subsequent results.

Machine Learning in ANNs :ANNs are capable of learning and they need to be trained. There are several learning strategies − Supervised Learning − It involves a teacher that is scholar than the ANN itself. For•example, the teacher feeds some example data about which the teacher already knows the answers. For

example, pattern recognizing. The ANN comes up with guesses while recognizing. Then the teacher provides the ANN with the answers. The network then compares it guesses with the teacher's "correct" answers and makes adjustments according to errors. Unsupervised Learning − It is required when there is no example data set with known answers. For example, searching for a hidden pattern. In this case, clustering i.e. dividing a set of elements into groups according to some unknown pattern is carried out based on the existing data sets present. Reinforcement Learning − This strategy built on observation. The ANN makes a

- decision by observing its environment. If the observation is negative, the network adjusts its weights to be able to make a different required decision the next time.

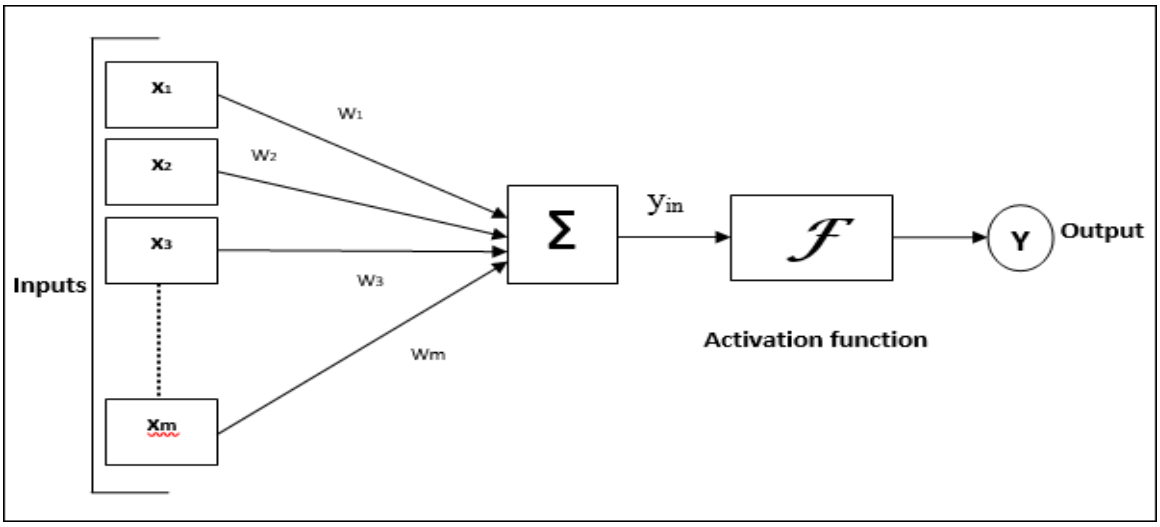
Back Propagation Algorithm It is the training or learning algorithm. It learns by example. If you submit to the algorithm the example of what you want the network to do, it changes the network's weights so that it can produce desired output for a particular input on finishing the training. Back Propagation networks are ideal for simple Pattern Recognition and Mapping Tasks. Bayesian Networks (BN) These are the graphical structures used to represent the probabilistic relationship among a set of random variables. Bayesian networks are also called Belief Networks or Bayes Nets. BNs reason about uncertain domain. In these networks, each node represents a random variable with specific propositions. For example, in a medical diagnosis domain, the node Cancer represents the proposition that a patient has cancer. The edges connecting the nodes represent probabilistic dependencies among those random variables. If out of two nodes, one is affecting the other then they must be directly connected in the directions of the effect. The strength of the relationship between variables is quantified by the probability associated with each node. There is an only constraint on the arcs in a BN that you cannot return to a node simply by following directed arcs. Hence the BNs are called Directed Acyclic Graphs (DAGs). BNs are capable of handling multivalued variables simultaneously. The BN variables are composed of two dimensions − Range of prepositions

- Probability assigned to each of the prepositions.

- Consider a finite set X = {X1, X2, …,Xn} of discrete random variables, where each variable Xi may take values from a finite set, denoted by Val(Xi). If there is a directed link from variable Xi to variable, Xj, then variable Xi will be a parent of variable Xj showing direct dependencies between the variables. The structure of BN is ideal for combining prior knowledge and observed data. BN can be used to learn the causal relationships and understand various problem domains and to predict future events, even in case of missing data. Building a Bayesian Network A knowledge engineer can build a Bayesian network. There are a number of steps the knowledge engineer needs to take while building it. Example problem − Lung cancer. A patient has been suffering from breathlessness. He visits the doctor, suspecting he has lung cancer. The doctor knows that barring lung cancer, there are various other possible diseases the patient might have such as tuberculosis

and bronchitis. Gather Relevant Information of Problem Is the patient a smoker? If yes, then high chances of cancer and bronchitis.

- Is the patient exposed to air pollution? If yes, what sort of air pollution?
- Take an X-Ray positive X-ray would indicate either TB or lung cancer.
- Identify Interesting Variables The knowledge engineer tries to answer the questions − Which nodes to represent?
- What values can they take? In which state can they be?
- For now let us consider nodes, with only discrete values. The variable must take on exactly one of these values at a time. Common types of discrete nodes are − Boolean nodes − They represent propositions, taking binary values TRUE (T) and
- FALSE (F). Ordered values − A node Pollution might represent and take values from {low, medium,
- high} describing degree of a patient's exposure to pollution. Integral values − A node called Age might represent patient's age with possible values
- from 1 to 120. Even at this early stage, modeling choices are being made.



For the above general model of artificial neural network, the net input can be calculated as follows −

$$Y_{in}=x_1.w_1+x_2.w_2+x_3.w_3\ldots x_m.w_m$$

i.e., i.e., Net input $Y = \sum_i^m xi.wi$

The output can be calculated by applying the activation function over the net input.

$$Y=F\ (Y_{in})$$

Output = function (net input calculated)

Areas of Application

Followings are some of the areas, where ANN is being used. It suggests that ANN has an interdisciplinary approach in its development and applications.

Speech Recognition

Speech occupies a prominent role in human-human interaction. Therefore, it is natural for people to expect speech interfaces with computers. In the present era, for communication with machines, humans still need sophisticated languages which are difficult to learn and use. To ease this communication barrier, a simple solution could be, communication in a spoken language that is possible for the machine to understand.

Great progress has been made in this field, however, still such kinds of systems are facing the problem of limited vocabulary or grammar along with the issue of retraining of the system for different speakers in different conditions. ANN is playing a major role in this area. Following ANNs have been used for speech recognition −

- Multilayer networks

- Multilayer networks with recurrent connections
- Kohonen self-organizing feature map

The most useful network for this is Kohonen Self-Organizing feature map, which has its input as short segments of the speech waveform. It will map the same kind of phonemes as the output array, called feature extraction technique. After extracting the features, with the help of some acoustic models as back-end processing, it will recognize the utterance.

Character Recognition

It is an interesting problem which falls under the general area of Pattern Recognition. Many neural networks have been developed for automatic recognition of handwritten characters, either letters or digits. Following are some ANNs which have been used for character recognition −

- Multilayer neural networks such as Backpropagation neural networks.
- Neocognitron

Though back-propagation neural networks have several hidden layers, the pattern of connection from one layer to the next is localized. Similarly, neocognitron also has several hidden layers and its training is done layer by layer for such kind of applications.

Signature Verification Application

Signatures are one of the most useful ways to authorize and authenticate a person in legal transactions. Signature verification technique is a non-vision based technique.

For this application, the first approach is to extract the feature or rather the geometrical feature set representing the signature. With these feature sets, we have to train the neural networks using an efficient neural network algorithm. This trained neural network will classify the signature as being genuine or forged under the verification stage.

Human Face Recognition

It is one of the biometric methods to identify the given face. It is a typical task because of the characterization of “non-face” images. However, if a neural network is well trained, then it can be divided into two classes namely images having faces and images that do not have faces.

First, all the input images must be preprocessed. Then, the dimensionality of that image must be reduced. And, at last it must be classified using neural network training algorithm. Following neural networks are used for training purposes with preprocessed image −

- Fully-connected multilayer feed-forward neural network trained with the help of back-propagation algorithm.
- For dimensionality reduction, Principal Component Analysis (PCA) is used.

McCullogh-Pitts Model

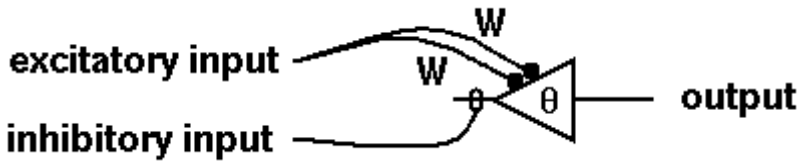
In 1943 two electrical engineers, Warren McCullogh and Walter Pitts, published the first paper describing what we would call a neural network. Their "neurons" operated under the following assumptions:

1. They are binary devices (V_i = [0,1])
2. Each neuron has a fixed threshold, theta
3. The neuron receives inputs from excitatory synapses, all having identical weights. (However it my receive multiple inputs from the same source, so the excitatory weights are effectively positive integers.)
4. Inhibitory inputs have an absolute veto power over any excitatory inputs.
5. At each time step the neurons are simultaneously (synchronously) updated by summing the weighted excitatory inputs and setting the output (V_i) to 1 iff the sum is greater than or equal to the threhold AND if the neuron receives no inhibitory input.

We can summarize these rules with the McCullough-Pitts output rule

$$V_i = \begin{cases} 1 & : \quad \sum_j W V_j \geq \theta \text{ AND no inhibition} \\ 0 & : \quad \text{otherwise} \end{cases}$$

and the diagram

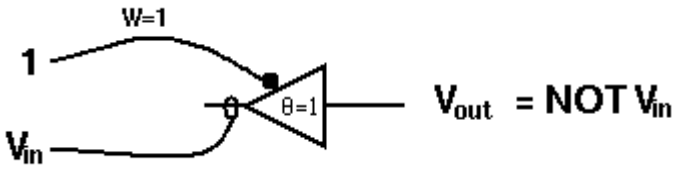


Using this scheme we can figure out how to implement any Boolean logic function. As you probably know, with a NOT function and either an OR or an AND, you can build up XOR's, adders, shift registers, and anything you need to perform computation.

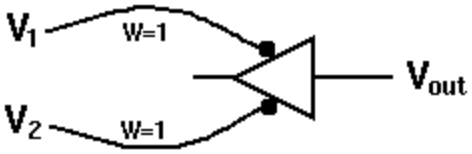
We represent the output for various inputs as a **truth table**, where 0 = FALSE, and 1 = TRUE. You should verify that when W = 1 and theta = 1, we get the truth table for the logical NOT,

```
Vin  |  Vout
-----+------
 1   |   0
 0   |   1
```

by using this circuit:



With two excitatory inputs V_1 and V_2, and W =1, we can get either an OR or an AND, depending on the value of theta:



if $\theta = 1 \Longrightarrow V_{out} = V_1 \text{ OR } V_2$

if $\theta = 2 \Longrightarrow V_{out} = V_1 \text{ AND } V_2$

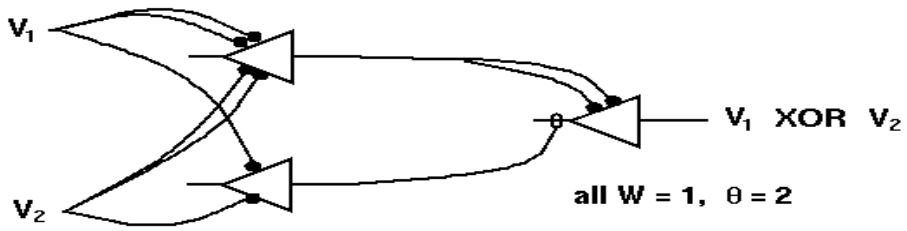
Can you verify that with these weights and thresholds, the various possible inputs for V_1 and V_2 result in this table?

```
V1 | V2 | OR | AND
---+----+----+----
 0 |  0 |  0 |  0
 0 |  1 |  1 |  0
 1 |  0 |  1 |  0
 1 |  1 |  1 |  1
```

The **exclusive OR** (XOR) has the truth table:

```
V1 | V2 | XOR
---+----+----
 0 |  0 |  0
 0 |  1 |  1      (Note that this is also a
 1 |  0 |  1       "1 bit adder".)
 1 |  1 |  0
```

It cannot be represented with a single neuron, but the relationship
XOR = (V_1 OR V_2) AND NOT (V_1 AND V_2) suggests that it can be represented with the network



Exercise: Explain to your own satisfaction that this generates the correct output for the four combinations of inputs. What computation is being made by each of the three "neurons"?

These results were very encouraging, but these networks displayed no learning. They were essentially "hard-wired" logic devices. One had to figure out the weights and connect up the neurons in the appropriate manner to perform the desired computation. Thus there is no real advantage over any conventional digital logic circuit. Their main importance was that they showed that networks of simple neuron-like elements could compute.

Hebbian Learning Rule

This rule, one of the oldest and simplest, was introduced by Donald Hebb in his book *The Organization of Behavior* in 1949. It is a kind of feed-forward, unsupervised learning.

Basic Concept − This rule is based on a proposal given by Hebb, who wrote −

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

From the above postulate, we can conclude that the connections between two neurons might be strengthened if the neurons fire at the same time and might weaken if they fire at different times.

Mathematical Formulation − According to Hebbian learning rule, following is the formula to increase the weight of connection at every time step.

$$\Delta wji\ (t)=\alpha xi(t).yj(t)\Delta wji(t)=\alpha xi(t).yj(t)$$

Here, Δwji (t) Δwji(t) = increment by which the weight of connection increases at time step **t**

αα = the positive and constant learning rate

xi(t)xi(t) = the input value from pre-synaptic neuron at time step **t**

yi(t)yi(t) = the output of pre-synaptic neuron at same time step **t**

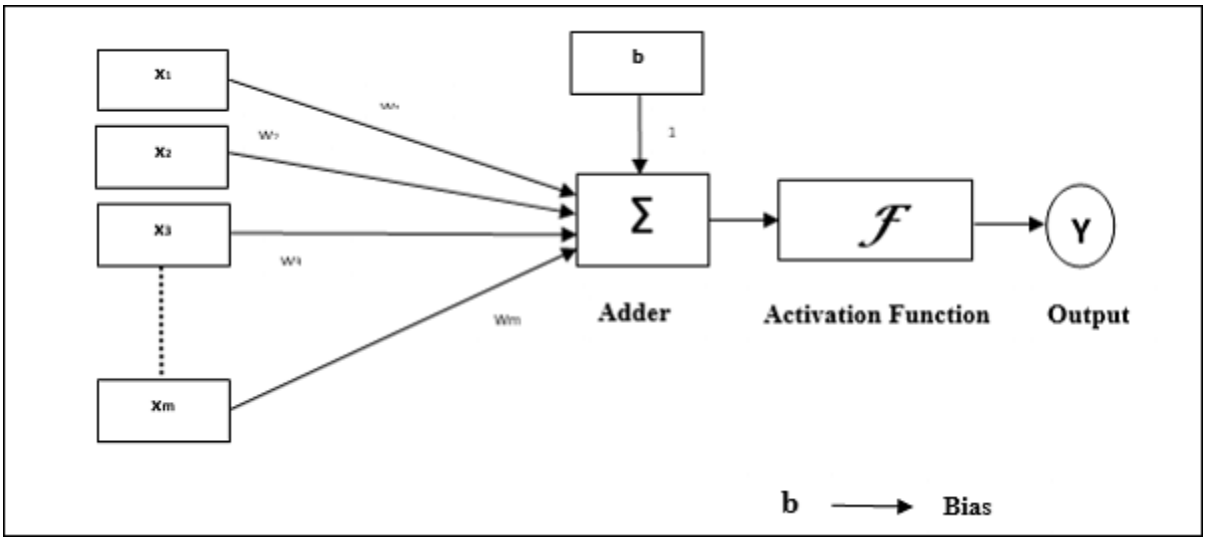
Supervised learning takes place under the supervision of a teacher. This learning process is dependent. During the training of ANN under supervised learning, the input vector is presented

to the network, which will produce an output vector. This output vector is compared with the desired/target output vector. An error signal is generated if there is a difference between the actual output and the desired/target output vector. On the basis of this error signal, the weights would be adjusted until the actual output is matched with the desired output.

Perceptron

Developed by Frank Rosenblatt by using McCulloch and Pitts model, perceptron is the basic operational unit of artificial neural networks. It employs supervised learning rule and is able to classify the data into two classes.

Operational characteristics of the perceptron: It consists of a single neuron with an arbitrary number of inputs along with adjustable weights, but the output of the neuron is 1 or 0 depending upon the threshold. It also consists of a bias whose weight is always 1. Following figure gives a schematic representation of the perceptron.



Perceptron thus has the following three basic elements −

- **Links** − It would have a set of connection links, which carries a weight including a bias always having weight 1.
- **Adder** − It adds the input after they are multiplied with their respective weights.
- **Activation function** − It limits the output of neuron. The most basic activation function is a Heaviside step function that has two possible outputs. This function returns 1, if the input is positive, and 0 for any negative input.

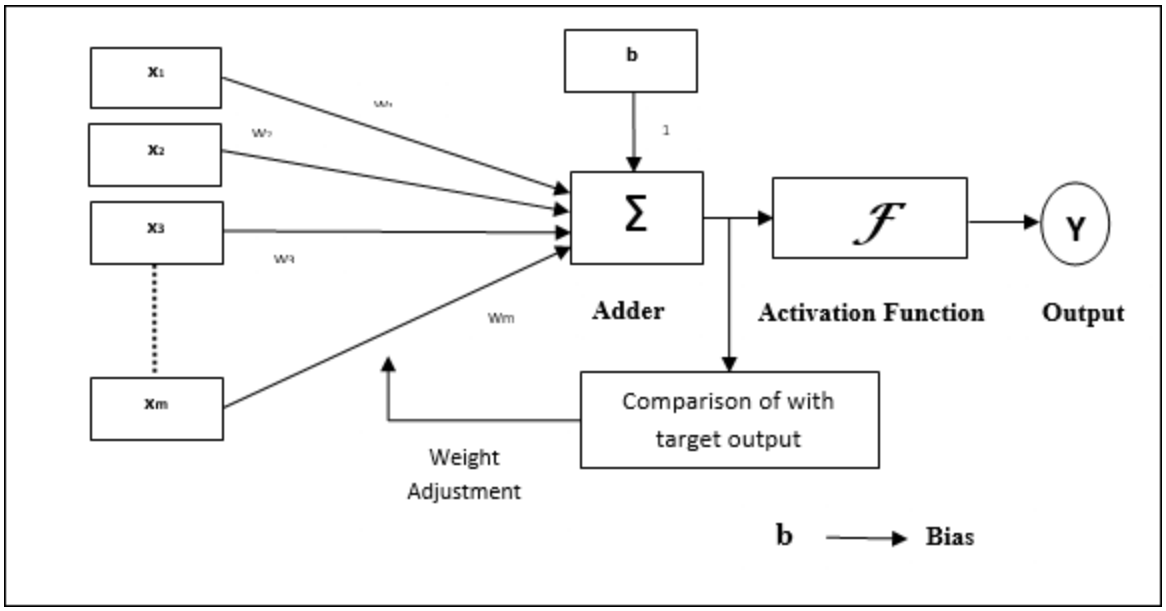
Adaptive Linear Neuron (Adaline)

Adaline which stands for Adaptive Linear Neuron, is a network having a single linear unit. It was developed by Widrow and Hoff in 1960. Some important points about Adaline are as follows −

- It uses bipolar activation function.
- It uses delta rule for training to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.
- The weights and the bias are adjustable.

Architecture

The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared with the desired/target output. After comparison on the basis of training algorithm, the weights and bias will be updated.



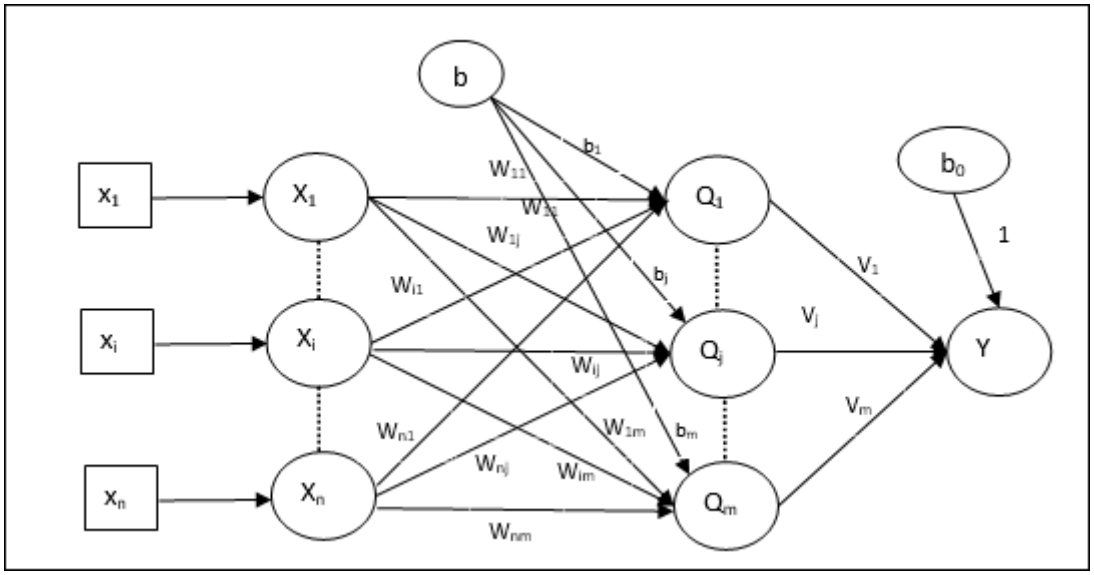
Multiple Adaptive Linear Neuron (Madaline):

Madaline which stands for Multiple Adaptive Linear Neuron, is a network which consists of many Adalines in parallel. It will have a single output unit. Some important points about Madaline are as follows −

- It is just like a multilayer perceptron, where Adaline will act as a hidden unit between the input and the Madaline layer.

- The weights and the bias between the input and Adaline layers, as in we see in the Adaline architecture, are adjustable.
- The Adaline and Madaline layers have fixed weights and bias of 1.

Training can be done with the help of Delta rule The architecture of Madaline consists of "n" neurons of the input layer, "m"neurons of the Adaline layer, and 1 neuron of the Madaline layer. The Adaline layer can be considered as the hidden layer as it is between theinput layer and the output layer, i.e. the Madaline layer**.**



Back Propagation Neural Networks:

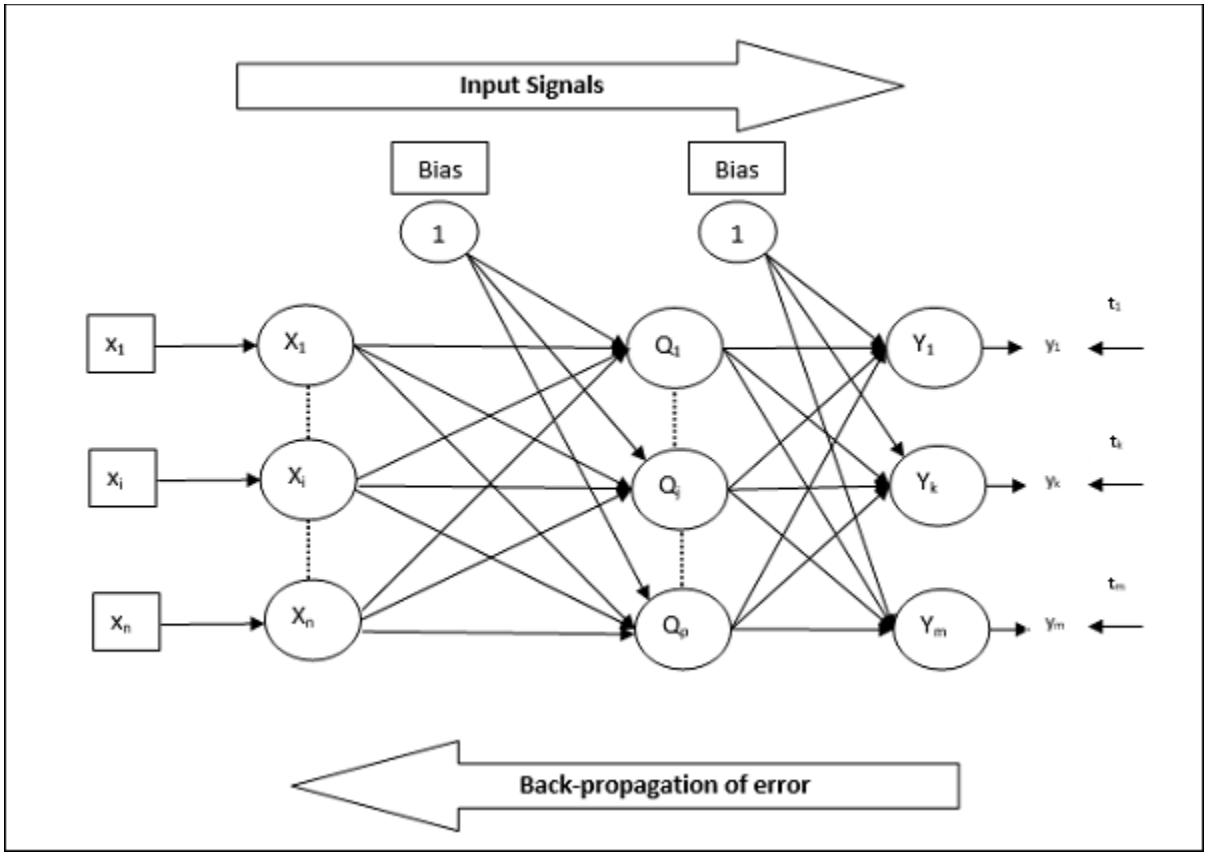
Back Propagation Neural (BPN) is a multilayer neural network consisting of the input layer, at least one hidden layer and output layer. As its name suggests, back propagating will take place in this network. The error which is calculated at the output layer, by comparing the target output and the actual output, will be propagated back towards the input layer.

Architecture

As shown in the diagram, the architecture of BPN has three interconnected layers having weights on them. The hidden layer as well as the output layer also has bias, whose weight is always 1, on them. As is clear from the diagram, the working of BPN is in two phases. One phase sends the signal from the input layer to the output layer, and the other phase back propagates the error from the output layer to the input layer.In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back propagation algorithm is the most widely used method for determining the EW.

The back-propagation algorithm is easiest to understand if all the units in the network are linear.

The algorithm computes each EW by first computing the EA, the rate at which the error changes as the activity level of a unit is changed. For output units, the EA is simply the difference between the actual and the desired output. To compute the EA for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the EAs of those output units and add the products. This sum equals the EA for the chosen hidden unit. After calculating all the EAs in the hidden layer just before the output layer, we can compute in like fashion the EAs for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the EA has been computed for a unit, it is straight forward to compute the EW for each incoming connection of the unit. The EW is the product of the EA and the activity through the incoming connection.



Radial basis function network:

In the field of mathematical modeling, a radial basis function network is an artificial neural network that uses radial basis functions as activation functions. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters.

Architecture

Locally tuned and overlapping receptive fields are well-known structures that have been studied

in regions of the cerebral cortex, the visual cortex, and others. Drawing on knowledge of biological receptive fields, Moody and Darken proposed a network structure that employs local receptive fields to perform function mappings. Similar schemes have been proposed by Powell, Broom head and Lowe and many others in the areas of interpolation and approximation theory; these schemes are collectively called radial basis function approximations. Here we shall call the network structure the radial basis function network or RBFN.

Wi = ~(x) = ~(llx - uill/o-i), i = 1, 2, ... , H,

where x is a multidimensional input vector, lli is a vector with the same dimension as x, H is the number of radial basis functions (or, equivalently, receptive field units), and ~ (·) is the ith radial basis function with a single maximum at the origin. There are no connection weights between the input layer and the hidden

Radial Basis Function Networks layer. Typically, R; (·) is a Gaussian function or a logistic function

R;,(x) = exp (- llx - uill2) 2o}

1 R;, (x) = _l _+_ex_p_[l-lx ___ u_i_ll2_/_a~ff

Thus, the activation level of radial basis function wi computed by the ith hidden unit is maximum when the input vector x is at the center ui of that unit. The output of an RBFN can be computed in two ways. In the simpler method, as shown in Figure 9.lO(a), the final output is the weighted sum of the output value associated with each receptive field:

H H d(x) = L CiWi = L ciRi(x), (9.16) i=l i=l

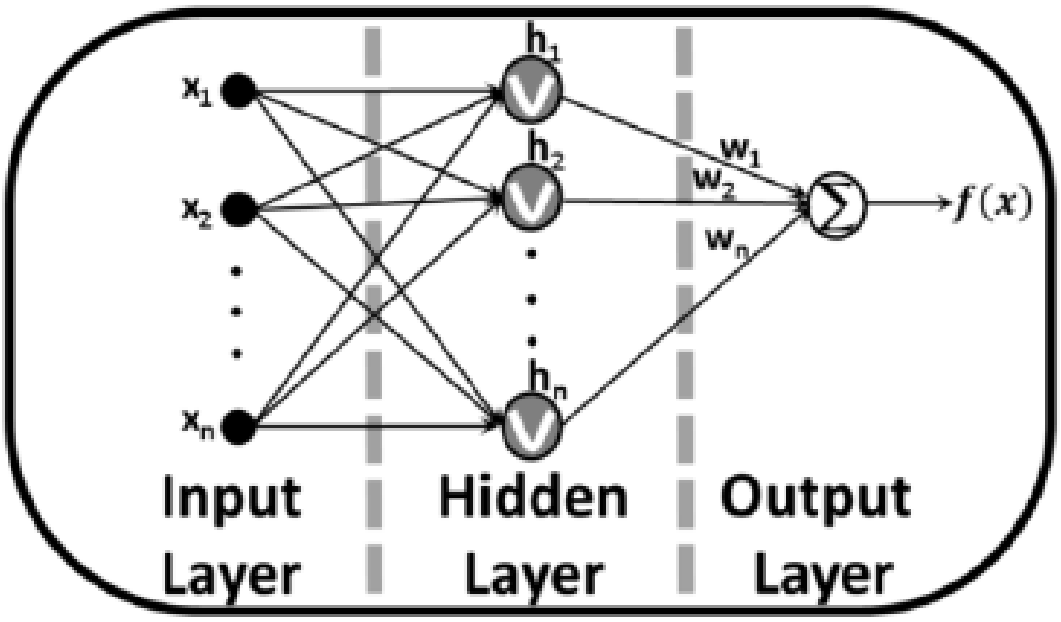
where ci is the output value associated with the ith receptive field. We can a.lso view ci as the connection weight between the receptive field i and the output unit. A more complicated method for calculating the overall output is to take the weighted average of the output associated with each receptive field:

Weighted average has a higher degree of computational complexity, but it is advantageous in that points in the areas of overlap between two or more receptive fields will have a well-interpolated overall output between the outputs of the overlapping receptive fields. For representation purposes, if we change the radial basis function Ri(x) in each node of layer 2 in to its normalized counterpart R;,(x)/ Li Ri(x), then the overall output is specified by Equation . A more explicit representation is shown in Figure 9.lO(b), where the division of the weighted sum (:Ei ciwi) by the activation total (:Ei wi) is indicated in the division node in the last layer. In Figure 9.10, plots (c) and (d) are the two-output counterparts of the RBFNs

in (a) and (b). Moody-Darken's RBFN may be extended by assigning a linear function to the output function of each receptive field-that is, making ci a linear combination of the input variables plus a constant: Ci= a; X + bi.

where ai is a parameter vector and bi is a scalar parameter. Stokbro et al. used this structure to model the Mackey-Glass chaotic time series and found

An RBFN's approximation capacity may be further improved with supervised adjustments of the center and shape of the receptive field (or radial basis) functions Several learning algorithms have been proposed to identify the parameters (ui, <Ti, and Ci) of an RBFN. Besides using a supervised learning scheme alone to update all modifiable parameters, a variety of sequential training algo Radial Basis Function Networks for RBFNs have been reported. The receptive field functions are first fixed, and then the weights of the output layer are adjusted. Several schemes have been proposed to determine the center positions (ui) of the receptive field functions. Lowe proposed a way to determine the centers based on standard deviations of training data. Moody and Darken selected the centers ui by means ofdata clustering techniques that assume that similar input vectors produce similar outputs; <Ti's are then obtained heuristically by taking the average distance to the several nearest neighbors of u/s. In another variation, Nowlan employed the so-called soft competition among Gaussian hidden units to locate the centers. This soft competition method is based on a "maximum likelihood estimate" for the centers, in contrast to the so-called hard competitions such as the k-means winner-take-all algorithm. Once these nonlinear parameters are fixed and the receptive fields are frozen, the linear parameters (i.e., the weights of the output layer) can be updated using either the least-squares method or the gradient method. Alternatively, we can apply the pseudoinverse method in solving Equations to determine these weights .used another method that employs the orthogonal least-squares algorithm to determine the u/s and Ci'S while keeping the <1/s at predetermined values. There are many other schemes as well, such as generalization properties, and sequential adaptation among others .



Applications of Neural Networks:

They can perform tasks that are easy for a human but difficult for a machine −

- **Aerospace** − Autopilot aircrafts, aircraft fault detection.
- **Automotive** − Automobile guidance systems.
- **Military** − Weapon orientation and steering, target tracking, object discrimination, facial recognition, signal/image identification.
- **Electronics** − Code sequence prediction, IC chip layout, chip failure analysis, machine vision, voice synthesis.
- **Financial** − Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, portfolio trading program, corporate financial analysis, currency value prediction, document readers, credit application evaluators.
- **Industrial** − Manufacturing process control, product design and analysis, quality inspection systems, welding quality analysis, paper quality prediction, chemical product design analysis, dynamic modeling of chemical process systems, machine maintenance analysis, project bidding, planning, and management.
- **Medical** − Cancer cell analysis, EEG and ECG analysis, prosthetic design, transplant time optimizer.
- **Speech** − Speech recognition, speech classification, text to speech conversion.

- **Telecommunications** − Image and data compression, automated information services, real-time spoken language translation.
- **Transportation** − Truck Brake system diagnosis, vehicle scheduling, routing systems.
- **Software** − Pattern Recognition in facial recognition, optical character recognition, etc.
- **Time Series Prediction** − ANNs are used to make predictions on stocks and natural calamities.
- **Signal Processing** − Neural networks can be trained to process an audio signal and filter it appropriately in the hearing aids.
- **Control** − ANNs are often used to make steering decisions of physical vehicles.
- **Anomaly Detection** − As ANNs are expert at recognizing patterns, they can also be trained to generate an output when something unusual occurs that misfits the pattern.

The most influential work on neural nets in the 60's went under the heading of 'perceptrons' a term coined by Frank Rosenblatt. The perceptron (figure 4.4) turns out to be an MCP model (neuron with weighted inputs) with some additional, fixed, pre--processing. Units labelled A1, A2, Aj , Ap are called association units and their task is to extract specific, localised featured from the input images. Perceptrons mimic the

basic idea behind the mammalian visual system. They were mainly used in pattern recognition even though their capabilities extended a lot more.

In 1969 Minsky and Papert wrote a book in which they described the limitations of single layer Perceptrons. The impact that the book had was tremendous and caused a lot of neural network researchers to loose their interest. The book was very well written and showed mathematically that single layer perceptrons could not do some basic pattern recognition operations like determining the parity of a shape or determining whether a shape is connected or not. What they did not realised, until the 80's, is that given the appropriate training, multilevel perceptrons can do these operations.

Hebb’s rule:

Hebb’s rule is a postulate proposed by Donald Hebb in 1949 [1]. It is a learning rule that describes how the neuronal activities influence the connection between neurons, i.e., the synaptic plasticity. It provides an algorithm to update weight of neuronal connection within

neural network. Hebb's rule provides a simplistic physiology-based model to mimic the activity dependent features of synaptic plasticity and has been widely used in the area of artificial neural network. Different versions of the rule have been proposed to make the updating rule more realistic. The weight of connection between neurons is a function of the neuronal activity. The classical Hebb's rule indicates "neurons that fire together, wire together". In the simplest form of Hebb's rule, wij stands for the weight of the connection from neuron j to neuron i

The plasticity within neural network. (A) The single connection between neuron i and neuron j. (B) A network of neurons connecting to neuron i. The perceptron is type of artificial neural network. It can be seen as the simple feedforward.

UNIT-2

ASSOCIATIVE MEMORY AND UNSUPERVISED LEARNING NETWORKS

Associate Memory Network:

These kinds of neural networks work on the basis of pattern association, which means they can store different patterns and at the time of giving an output they can produce one of the stored patterns by matching them with the given input pattern. These types of memories are also called Content-Addressable Memory (CAM). Associative memory makes a parallel search with the stored patterns as data files.

Following are the two types of associative memories we can observe −

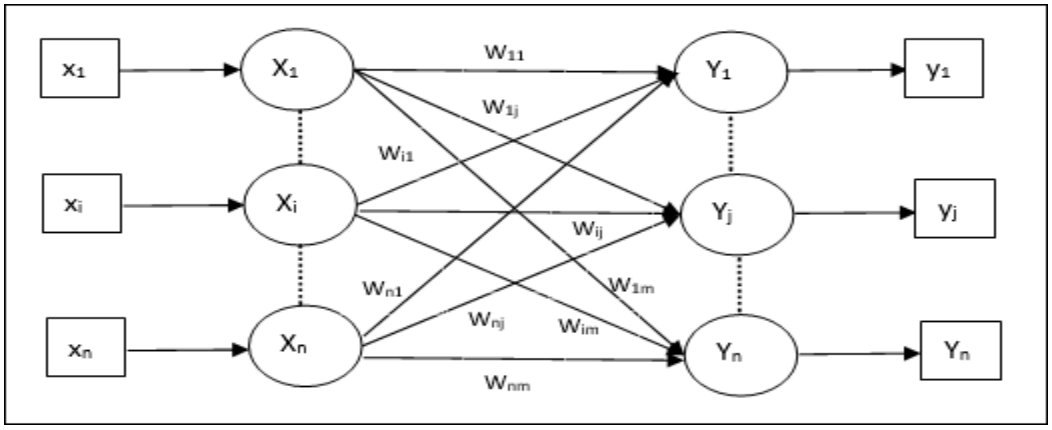
- Auto Associative Memory
- Hetero Associative memory

Auto Associative Memory

This is a single layer neural network in which the input training vector and the output target vectors are the same. The weights are determined so that the network stores a set of patterns.

Architecture

As shown in the following figure, the architecture of Auto Associative memory network has **'n'** number of input training vectors and similar **'n'** number of output target vectors.



Training Algorithm

For training, this network is using the Hebb or Delta learning rule.

Step 1 − Initialize all the weights to zero as $w_{ij} = 0$ (i = 1 to n, j = 1 to n)

Step 2 − Perform steps 3-4 for each input vector.

Step 3 − Activate each input unit as follows −

$$xi=si(i=1ton)xi=si(i=1ton)$$

Step 4 − Activate each output unit as follows −

$$yj=sj(j=1ton)yj=sj(j=1ton)$$

Step 5 − Adjust the weights as follows −

$$wij(new)=wij(old)+xiyjwij(new)=wij(old)+xiyj$$

Testing Algorithm

Step 1 − Set the weights obtained during training for Hebb's rule.

Step 2 − Perform steps 3-5 for each input vector.

Step 3 − Set the activation of the input units equal to that of the input vector.

Step 4 − Calculate the net input to each output unit j = 1 to n

$$yinj=\sum i=1nxiwijyinj=\sum i=1nxiwij$$

Step 5 − Apply the following activation function to calculate the output

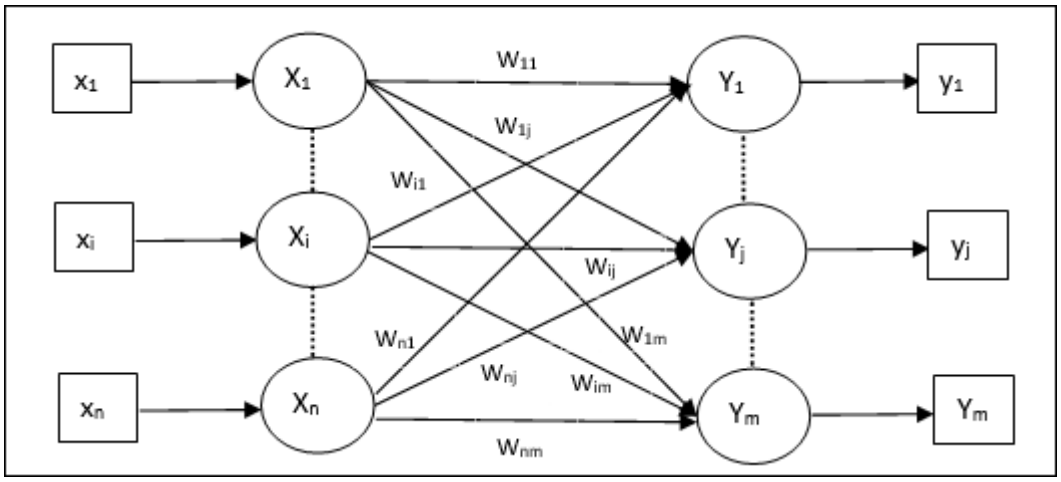
$$yj=f(yinj)=\{+1-1ifyinj>0ifyinj\leqslant 0$$

Hetero Associative memory

Similar to Auto Associative Memory network, this is also a single layer neural network. However, in this network the input training vector and the output target vectors are not the same. The weights are determined so that the network stores a set of patterns. Hetero associative network is static in nature, hence, there would be no non-linear and delay operations.

Architecture

As shown in the following figure, the architecture of Hetero Associative Memory network has **'n'** number of input training vectors and **'m'** number of output target vectors.



Training Algorithm

For training, this network is using the Hebb or Delta learning rule.

Step 1 − Initialize all the weights to zero as $w_{ij} = 0$ (i = 1 to n, j = 1 to m)

Step 2 − Perform steps 3-4 for each input vector.

Step 3 − Activate each input unit as follows −

$$xi=si(i=1ton)xi=si(i=1ton)$$

Step 4 − Activate each output unit as follows −

$$yj=sj(j=1tom)yj=sj(j=1tom)$$

Step 5 − Adjust the weights as follows −

$$wij(new)=wij(old)+xiyjwij(new)=wij(old)+xiyj$$

Testing Algorithm

Step 1 − Set the weights obtained during training for Hebb's rule.

Step 2 − Perform steps 3-5 for each input vector.

Step 3 − Set the activation of the input units equal to that of the input vector.

Step 4 − Calculate the net input to each output unit j = 1 to m;

$$yinj=\sum i=1nxiwijyinj=\sum i=1nxiwij$$

Step 5 − Apply the following activation function to calculate the output

$$yj=f(yinj)=\begin{cases}+1 & ifyinj>0\\ 0 & ifyinj=0\\ -1 & ifyinj<0\end{cases}$$

Hopfield networks:

Hopfield neural network was invented by Dr. John J. Hopfield in 1982. It consists of a single layer which contains one or more fully connected recurrent neurons. The Hopfield network is commonly used for auto-association and optimization tasks.

Hopfield network which operates in a discrete line fashion or in other words, it can be said the input and output patterns are discrete vector, which can be either binary (0,1) or bipolar (+1, -1) in nature. The network has symmetrical weights with no self-connections i.e., $\mathbf{w_{ij} = w_{ji}}$ and $\mathbf{w_{ii} = 0}$.

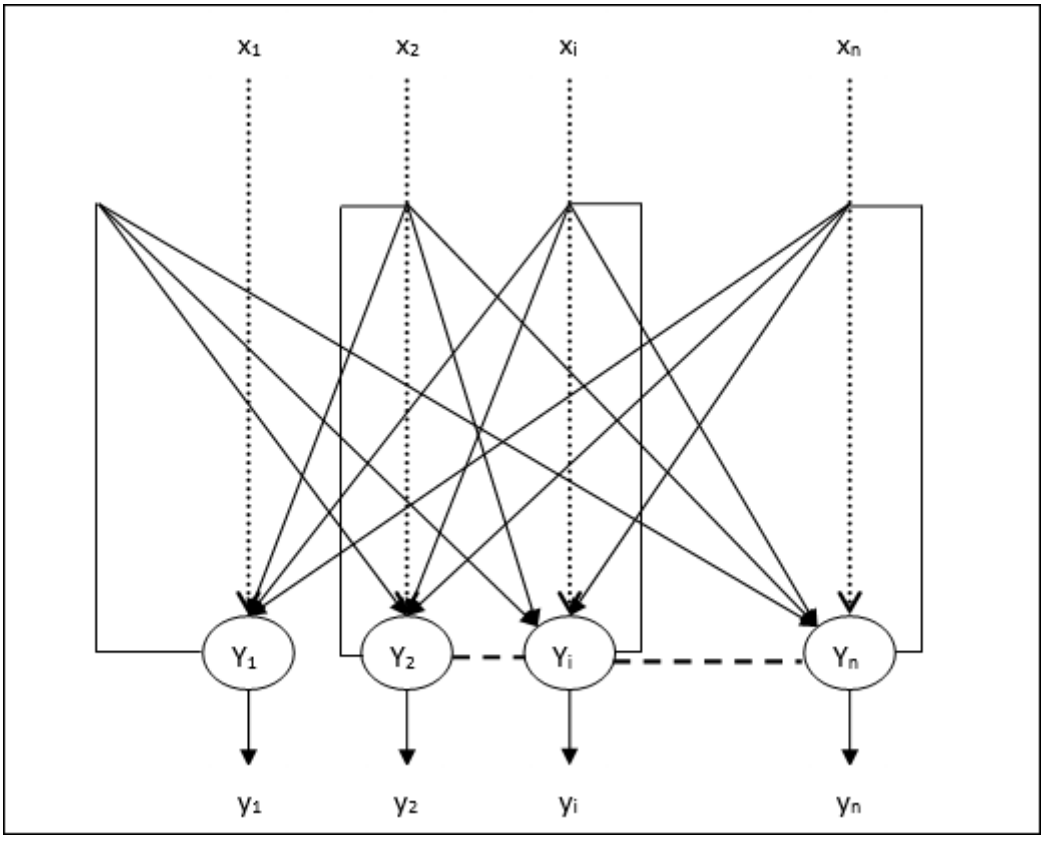
Architecture

Following are some important points to keep in mind about discrete Hopfield network −

- This model consists of neurons with one inverting and one non-inverting output.
- The output of each neuron should be the input of other neurons but not the input of self.
- Weight/connection strength is represented by $\mathbf{w_{ij}}$.

- Connections can be excitatory as well as inhibitory. It would be excitatory, if the output of the neuron is same as the input, otherwise inhibitory.
- Weights should be symmetrical, i.e. $\mathbf{w_{ij} = w_{ji}}$

The output from $\mathbf{Y_1}$ going to $\mathbf{Y_2}$, $\mathbf{Y_i}$ and $\mathbf{Y_n}$ have the weights $\mathbf{w_{12}}$, $\mathbf{w_{1i}}$ and $\mathbf{w_{1n}}$ respectively. Similarly, other arcs have the weights on them.



Training Algorithm

During training of discrete Hopfield network, weights will be updated. As we know that we can have the binary input vectors as well as bipolar input vectors. Hence, in both the cases, weight updates can be done with the following relation

Case 1 − Binary input patterns

For a set of binary patterns s(p), p = 1 to P

Here, $s(p) = s_1(p), s_2(p),..., s_i(p),..., s_n(p)$

Weight Matrix is given by

$$wij=\sum p=1P[2si(p)-1][2sj(p)-1]fori\neq jwij=\sum p=1P[2si(p)-1][2sj(p)-1]fori\neq j$$

Case 2 − Bipolar input patterns

For a set of binary patterns s(p), p = 1 to P

Here, $s(p) = s_1(p), s_2(p),..., s_i(p),..., s_n(p)$

Weight Matrix is given by

$wij=\sum p=1P[si(p)][sj(p)]fori\neq j$

Testing Algorithm

Step 1 − Initialize the weights, which are obtained from training algorithm by using Hebbian principle.

Step 2 − Perform steps 3-9, if the activations of the network is not consolidated.

Step 3 − For each input vector X, perform steps 4-8.

Step 4 − Make initial activation of the network equal to the external input vector X as follows −

$$yi=xifori=1tonyi=xifori=1ton$$

Step 5 − For each unit Y_i, perform steps 6-9.

Step 6 − Calculate the net input of the network as follows −

$$yini=xi+\sum jyjwjiyini=xi+\sum jyjwji$$

Step 7 − Apply the activation as follows over the net input to calculate the output −

$$yi=\begin{cases}1yi0ifyini>\theta iifyini=\theta iifyini<\theta iyi=\{1ifyini>\theta iyiifyini=\theta i0ifyini<\theta i\end{cases}$$

Here $\theta i\theta i$ is the threshold.

Step 8 − Broadcast this output y_i to all other units.

Step 9 − Test the network for conjunction.

Unsupervised Learning:

As the name suggests, this type of learning is done without the supervision of a teacher. This learning process is independent. During the training of ANN under unsupervised learning, the input vectors of similar type are combined to form clusters. When a new input pattern is applied, then the neural network gives an output response indicating the class to which input pattern belongs. In this, there would be no feedback from the environment as to what should be the desired output and whether it is correct or incorrect. Hence, in this type of learning the network itself must discover the patterns, features from the input data and the relation for the input data over the output.

Kohonen self organizing feature maps:

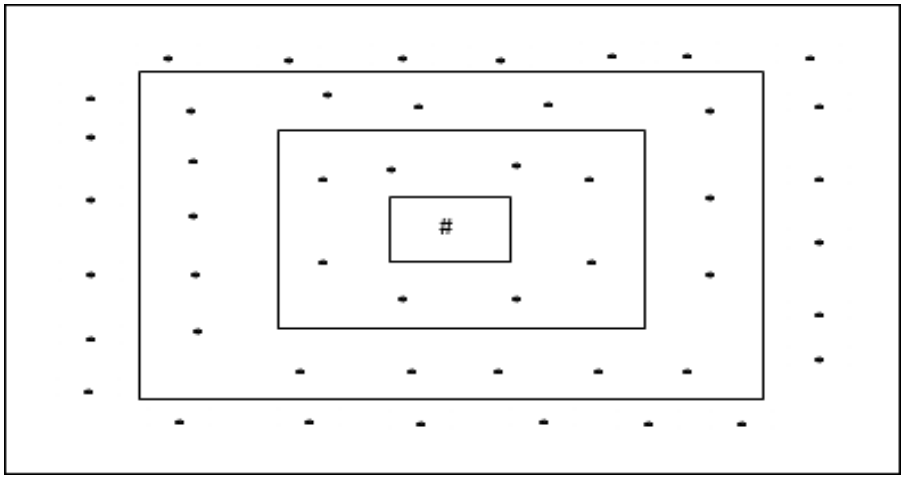
Suppose we have some pattern of arbitrary dimensions, however, we need them in one dimension or two dimensions. Then the process of feature mapping would be very useful to convert the wide pattern space into a typical feature space. Now, the question arises why do we require self-organizing feature map? The reason is, along with the capability to convert the arbitrary dimensions into 1-D or 2-D, it must also have the ability to preserve the neighbor topology.

Neighbor Topologies in Kohonen SOM

There can be various topologies, however the following two topologies are used the most −

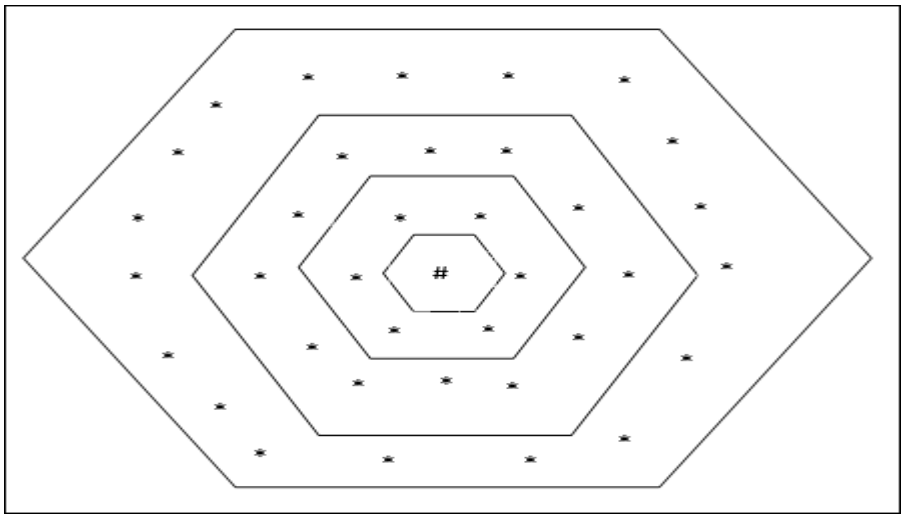
Rectangular Grid Topology

This topology has 24 nodes in the distance-2 grid, 16 nodes in the distance-1 grid, and 8 nodes in the distance-0 grid, which means the difference between each rectangular grid is 8 nodes. The winning unit is indicated by #.



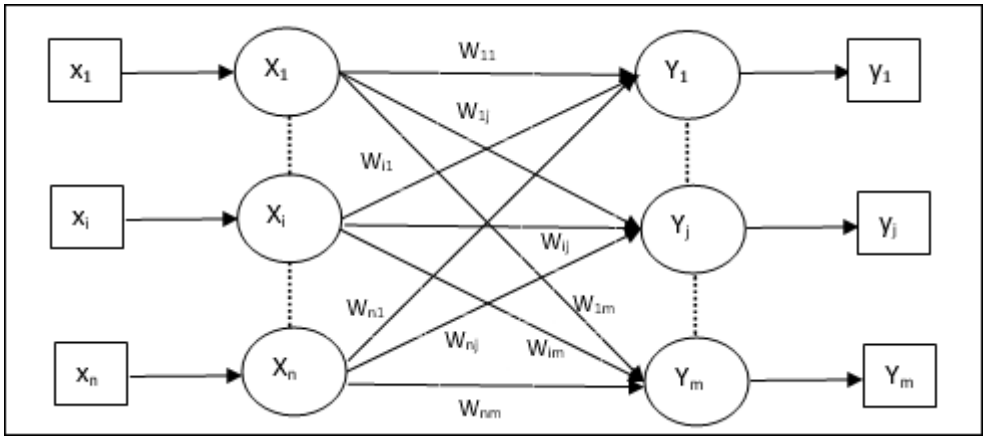
Hexagonal Grid Topology

This topology has 18 nodes in the distance-2 grid, 12 nodes in the distance-1 grid, and 6 nodes in the distance-0 grid, which means the difference between each rectangular grid is 6 nodes. The winning unit is indicated by #.



Architecture

The architecture of KSOM is similar to that of the competitive network. With the help of neighborhood schemes, discussed earlier, the training can take place over the extended region of the network.



Algorithm for training

Step 1 − Initialize the weights, the learning rate α and the neighborhood topological scheme.

Step 2 − Continue step 3-9, when the stopping condition is not true.

Step 3 − Continue step 4-6 for every input vector x.

Step 4 − Calculate Square of Euclidean Distance for j = 1 to m

$$D(j)=\sum i=1n\sum j=1m(xi-wij)2D(j)=\sum i=1n\sum j=1m(xi-wij)2$$

Step 5 − Obtain the winning unit J where *D(j)* is minimum.

Step 6 − Calculate the new weight of the winning unit by the following relation −

$$wij(new)=wij(old)+\alpha[xi-wij(old)]wij(new)=wij(old)+\alpha[xi-wij(old)]$$

Step 7 − Update the learning rate α by the following relation −

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

Step 8 − Reduce the radius of topological scheme.

Step 9 − Check for the stopping condition for the network.

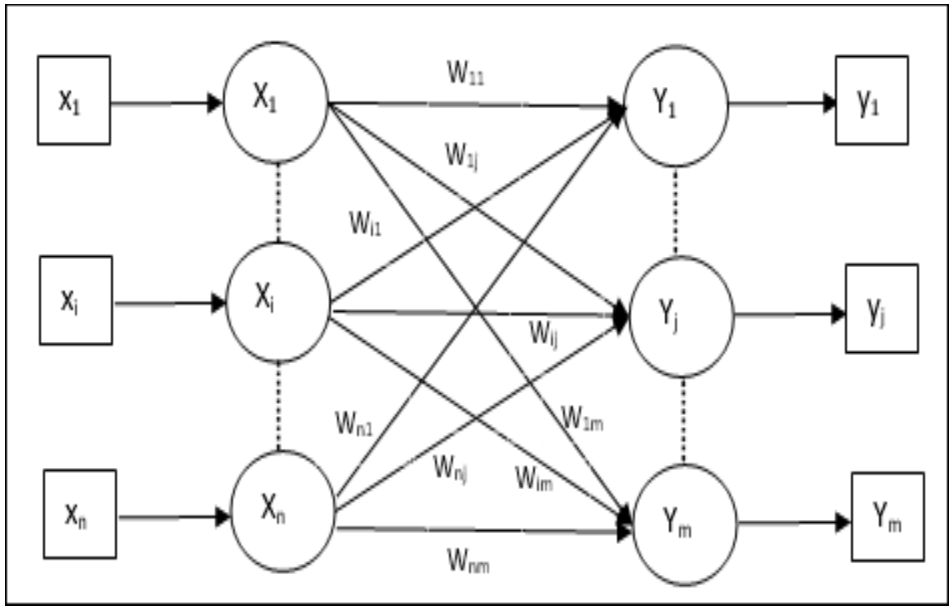
learning vector quantization

Learning Vector Quantization (LVQ), different from Vector quantization (VQ) and Kohonen Self-Organizing Maps

(KSOM), basically is a competitive network which uses supervised learning. We may define it as a process of classifying the patterns where each output unit represents a class. As it uses supervised learning, the network will be given a set of training patterns with known classification along with an initial distribution of the output class. After completing the training process, LVQ will classify an input vector by assigning it to the same class as that of the output unit.

Architecture

Following figure shows the architecture of LVQ which is quite similar to the architecture of KSOM. As we can see, there are **"n"** number of input units and **"m"** number of output units. The layers are fully interconnected with having weights on them.



Parameters Used

Following are the parameters used in LVQ training process as well as in the flowchart

x = training vector $(x_1,...,x_i,...,x_n)$

T = class for training vector **x**

$\mathbf{w_j}$ = weight vector for $\mathbf{j^{th}}$ output unit

$\mathbf{C_j}$ = class associated with the $\mathbf{j^{th}}$ output unit

Training Algorithm

Step 1 − Initialize reference vectors, which can be done as follows −

Step 1(a) − from the given set of training vectors, take the first "m" (number of clusters) training vectors and use

Them as weight vectors. The remaining vectors can be used for training.

Step 1(b) − Assign the initial weight and classification randomly.

Step 1(c) − Apply K-means clustering method.

Step 2 − Initialize reference vector $\alpha\alpha$

Step 3 − Continue with steps 4-9, if the condition for stopping this algorithm is not met.

Step 4 − Follow steps 5-6 for every training input vector x.

Step 5 − Calculate Square of Euclidean Distance for j = 1 to m and i = 1 to n

$$D(j)=\sum i=1n\sum j=1m(xi-wij)2D(j)=\sum i=1n\sum j=1m(xi-wij)2$$

Step 6 − Obtain the winning unit J where D(j) is minimum.

Step 7 − Calculate the new weight of the winning unit by the following relation

if T = C_j then $wj(new)=wj(old)+\alpha[x-wj(old)]$wj(new)=wj(old)+α[x−wj(old)]
if T ≠ C_j then $wj(new)=wj(old)-\alpha[x-wj(old)]$wj(new)=wj(old)−α[x−wj(old)]

Step 8 − Reduce the learning rate αα.

Step 9 − Test for the stopping condition. It may be as follows –

Maximum number of epochs reached and Learning rate reduced to a negligible value.

adaptive resonance theory network:

This network was developed by Stephen Grossberg and Gail Carpenter in 1987. It is based on competition and uses unsupervised learning model. Adaptive Resonance Theory (ART) networks, as the name suggests, is always open to new learning (adaptive) without losing the old patterns (resonance). Basically, ART network is a vector classifier which accepts an input vector and classifies it into one of the categories depending upon which of the stored pattern it resembles the most.

Operating Principal

The main operation of ART classification can be divided into the following phases −

- **Recognition phase** − The input vector is compared with the classification presented at every node in the output layer. The output of the neuron becomes "1" if it best matches with the classification applied, otherwise it becomes "0".
- **Comparison phase** − In this phase, a comparison of the input vector to the comparison layer vector is done. The condition for reset is that the degree of similarity would be less than vigilance parameter.
- **Search phase** − In this phase, the network will search for reset as well as the match done in the above phases. Hence, if there would be no reset and the match is quite good, then the classification is over. Otherwise, the process would be repeated and the other stored pattern must be sent to find the correct match.

ART1

It is a type of ART, which is designed to cluster binary vectors. We can understand about this with the architecture of it.

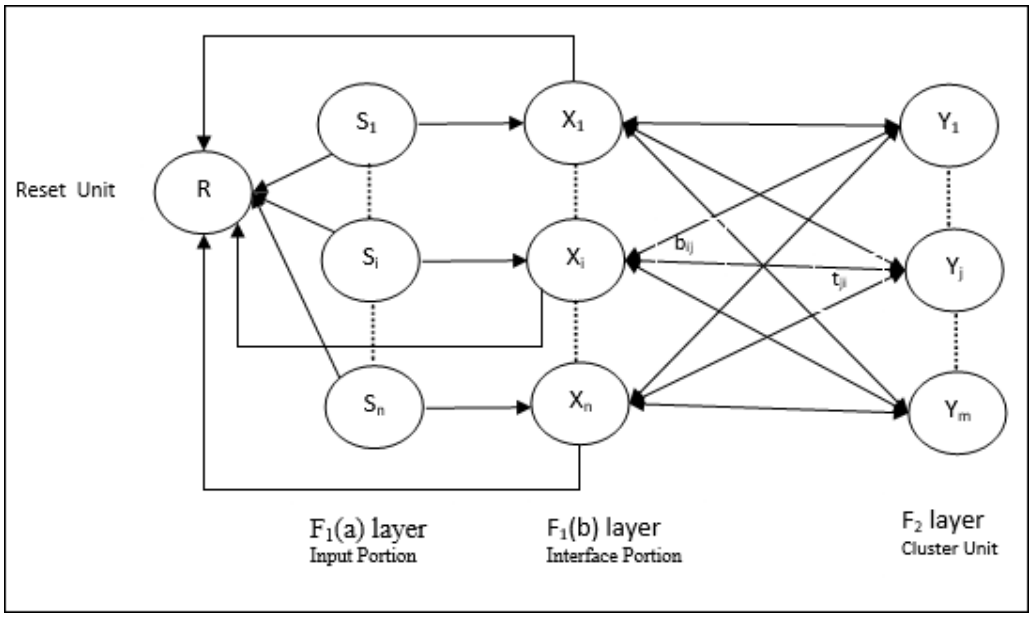
Architecture of ART1

It consists of the following two units –

Computational Unit − It is made up of the following −

- Input unit (F_1 layer) − It further has the following two portions −
    - F_1(a) layer (Input portion) − In ART1, there would be no processing in this portion rather than having the input vectors only. It is connected to F_1(b) layer (interface portion).
    - F_1(b) layer (Interface portion) − This portion combines the signal from the input portion with that of F_2 layer. F_1(b) layer is connected to F_2 layer through bottom up weights b_{ij} and F_2 layer is connected to F_1(b) layer through top down weights t_{ji}.
- Cluster Unit (F_2 layer) − This is a competitive layer. The unit having the largest net input is selected to learn the input pattern. The activation of all other cluster unit are set to 0.
- Reset Mechanism − The work of this mechanism is based upon the similarity between the top-down weight and the input vector. Now, if the degree of this similarity is less than the vigilance parameter, then the cluster is not allowed to learn the pattern and a rest would happen.

Supplement Unit − Actually the issue with Reset mechanism is that the layer F_2 must have to be inhibited under certain conditions and must also be available when some learning happens. That is why two supplemental units namely, G_1 and G_2 is added along with reset unit, R. They are called gain control units. These units receive and send signals to the other units present in the network. '+' indicates an excitatory signal, while '−' indicates an inhibitory signal.



Parameters Used

Following parameters are used −

n − Number of components in the input vector

m − Maximum number of clusters that can be formed

$\mathbf{b_{ij}}$ − Weight from F_1(b) to F_2 layer, i.e. bottom-up weights

$\mathbf{t_{ji}}$ − Weight from F_2 to F_1(b) layer, i.e. top-down weights

ρ − Vigilance parameter

||x|| − Norm of vector x

Algorithm

Step 1 − Initialize the learning rate, the vigilance parameter, and the weights as follows −

$$\alpha>1 and 0<\rho\leq 1 \alpha>1 and 0<\rho\leq 1$$
$$0<bij(0)<\alpha\alpha-1+nandtij(0)=10<bij(0)<\alpha\alpha-1+nandtij(0)=1$$

Step 2 − Continue step 3-9, when the stopping condition is not true.

Step 3 − Continue step 4-6 for every training input.

Step 4 − Set activations of all F_1(a) and F_1 units as follows

F_2 = 0 and F_1(a) = input vectors

Step 5 − Input signal from F_1(a) to F_1(b) layer must be sent like

$$si=xisi=xi$$

Step 6 − For every inhibited F_2 node

$yj=\sum ibijxiyj=\sum ibijxi$ the condition is $y_j \neq -1$

Step 7 − Perform step 8-10, when the reset is true.

Step 8 − Find J for $y_J \geq y_j$ for all nodes j

Step 9 − Again calculate the activation on F_1(b) as follows

$$xi=sitJixi=sitJi$$

Step 10 − Now, after calculating the norm of vector x and vector s, we need to check the reset condition as follows −

If ||x||/ ||s||< vigilance parameter ρ, then inhibit node J and go to step 7

Else If ||x||/ ||s|| ≥ vigilance parameter ρ, then proceed further.

Step 11 − Weight updating for node J can be done as follows −

$$bij(new)=\alpha xi\alpha-1+||x||bij(new)=\alpha xi\alpha-1+||x||$$
$$tij(new)=xitij(new)=xi$$

Step 12 − The stopping condition for algorithm must be checked and it may be as follows −

Do not have any change in weight.

Reset is not performed for units.

Maximum number of epochs reached.

UNIT -3

FUZZY LOGIC

Fuzzy logic system

Today control systems are usually described by mathematical models that follow the laws of physics, stochastic models or models which have emerged from mathematical logic. A general difficulty of such constructed model is how to move from a given problem to a proper mathematical model. Undoubtedly, today's advanced computer technology makes it possible; however managing such systems is still too complex.

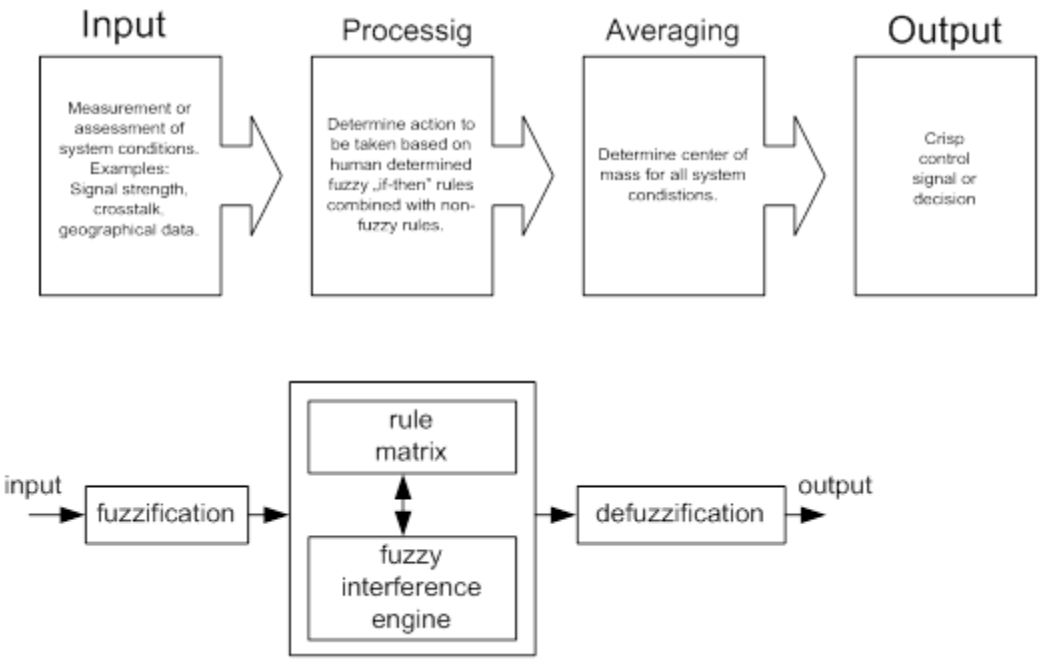
These complex systems can be simplified by employing a tolerance margin for a reasonable amount of imprecision, vagueness and uncertainty during the modelling phase. As an outcome, not completely perfect system comes to existence; nevertheless in most of the cases it is capable of solving the problem in appropriate way. Even missing input information has already turned out to be satisfactory in knowledge-based systems.

Fuzzy logic allows to lower complexity by allowing the use of imperfect information in sensible way. It can be implemented in hardware, software, or a combination of both. In other words, fuzzy logic approach to problems' control mimics how a person would make decisions, only much faster.

The fuzzy logic analysis and control methods shown in Figure 1 can be described as:

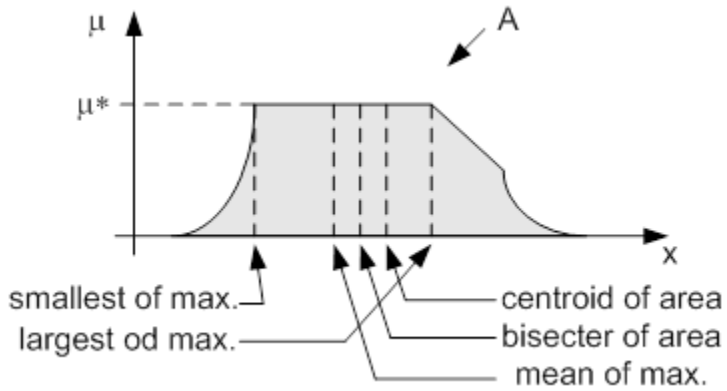
1. Receiving one or large number of measurements or other assessment of conditions existing in some system that will be analysed or controlled.
2. Processing all received inputs according to human based, fuzzy "if-then" rules, which can be expressed in simple language words, and combined with traditional non-fuzzy processing.
3. Averaging and weighting the results from all the individual rules into one single output decision or signal which decides what to do or tells a controlled system what to do. The result output signal is a precision as follows;

Usually fuzzy logic control system is created from four major elements presented on Figure 2: fuzzification interface, fuzzy inference engine, fuzzy rule matrix and defuzzification interface. Each part along with basic fuzzy logic operations will be described in more detail below.



Defuzzification Mechanisms

Defuzzification task is to find one single crisp value that summarises the fuzzy set. There are several mathematical techniques available: centroid, bisector, mean, maximum, maximum and weighted average. Figure 5 demonstrate illustration of how values for each method are chosen**.**



Graphical demonstration of defuzzification methods

Centroid defuzzification is the most commonly used method, as it is very accurate. It provides centre of the area under the curve of membership function. For complex membership functions it puts high demands on computation. It can be expressed by the following formula

$$z_0 = \frac{\int \mu_i(x) x dx}{\int \mu_i(x) dx}$$

where z_0 is defuzzified output, u_i is a membership function and x is output variable.

Bisector defuzzification uses vertical line that divides area under the curve into two equal areas.

$$\int_{\alpha}^{z} \mu_A(x) dx = \int_{z}^{\beta} \mu_A(x) dx$$

Mean of maximum defuzzification method uses the average value of the aggregated membership function outputs.

$$z_0 = \frac{\int_{x'}^{} x dx}{\int_{x'}^{'} dx}$$

where $x' = \{x; \mu_A(x) = \mu^*\}$.

Smallest of maximum defuzzification method uses the minimum value of the aggregated membership function outputs.

$$z_0 \in \left\{ x \middle| \mu(x) = \min_{\omega} \mu(\omega) \right\}$$

Largest of maximum defuzzification method uses the maximum value of the aggregated membership function outputs.

$$z_0 \in \left\{ x \middle| \mu(x) = \max_{\omega} \mu(\omega) \right\}$$

Weighted average defuzzification method, based on peak value of each fuzzy sets, calculates weighted sum of these peak values [4]. According to these weight values and the degree of membership for fuzzy output, the crisp value of output is determined by the following formula.

$$z_0 = \frac{\sum \mu(x)_i \times W_i}{\sum \mu(x)_i}$$

where μ_i is the degree of membership in output singleton i, W_i and is the fuzzy output weight value for the output singleton

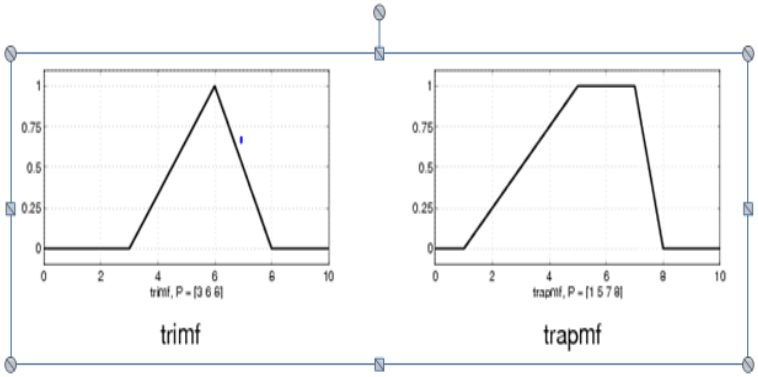
Member ship functions:

The MATLAB toolbox includes 11 built-in membership function types. These 11 functions are, in turn, built from several basic functions:

- piecewise linear functions
- the Gaussian distribution function
- the sigmoid curve
- quadratic and cubic polynomial curves

The simplest membership functions are formed using straight lines. The simplest is the triangular membership function, and it has the function name trimf.

- The trapezoidal membership function, trapmf, has a flat top and really is just a truncated triangle curve. These straight line membership functions have the advantage of simplicity.



Membership FunctionsAlthough the Gaussian membership functions and bell membership functions achieve smoothness, they are unable to specify asymmetric membership functions, which are important in certain applications. the sigmoidal membership functionis defined, which is either open left or right. Asymmetric and closed (i.e. not open to the left or right) membership functions can be synthesized using two sigmoidal functions, so in addition to the basic sigmf,

you also have the difference between two sigmoidal functions, dsigmf, and the product of two sigmoidal functions psigmf.

Methods of membership value assignments:

Since the membership function essentially embodies all fuzziness for a particular fuzzy set, its description is the essence of a fuzzy property or operation.

Membership Value assignments Methods

1. Intuition

2. Inference

3. Rank ordering

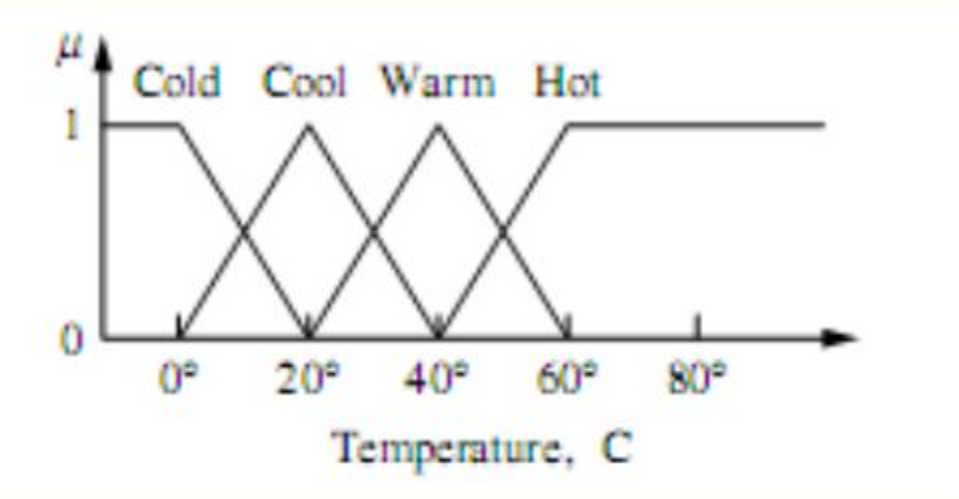
4. Neural networks

5. Genetic algorithms

6. Inductive reasoning

Intuition:

Intuition involves contextual and semantic knowledge about an issue; it can also involve linguistic truth values about this knowledge



For example, each curve is a membership function corresponding to various fuzzy variables, such as very cold, cold, normal, hot, and very hot.

Inference:

In the inference method we use knowledge to perform deductive reasoning.

• let A, B, and C be the inner angles of a triangle, in the order A ≥ B ≥ C ≥ 0, and

let U be the universe of triangles, i.e., U = {(A, B,C) | A ≥ B ≥ C ≥ 0; A + B + C = 180 ◦}

$\underset{\sim}{I}$	Approximate isosceles triangle
$\underset{\sim}{R}$	Approximate right triangle
$\underset{\sim}{IR}$	Approximate isosceles *and* right triangle
$\underset{\sim}{E}$	Approximate equilateral triangle
$\underset{\sim}{T}$	Other triangles

For the approximate isosceles triangle,

$$\mu_{\underset{\sim}{I}}(A,B,C) = 1 - \frac{1}{60^\circ}\min(A-B, B-C) = \begin{Bmatrix} 1, if A = B or B = C \\ 0, if A = 120^\circ, B = 60^\circ, C = 0^\circ \end{Bmatrix}$$

For the approximate right triangle,

$$U_{\underset{\sim}{R}}(A,B,C) = 1 - \frac{1}{90^\circ}|A - 90^\circ| = \begin{Bmatrix} 1, \text{ if } A = 90^\circ \\ 0, \text{ if } A = 180^\circ \end{Bmatrix}$$

For approximate isosceles and right triangle

the logical intersection (and operator) of the isosceles and right triangle membership functions,

or

$$\underset{\sim}{IR} = \underset{\sim}{I} \cap \underset{\sim}{R}$$

$$\mu_{\underset{\sim}{IR}}(A,B,C) = \min[\mu_{\underset{\sim}{I}}(A,B,C), \mu_{\underset{\sim}{R}}(A,B,C)]$$

$$= 1 - \max\left[\frac{1}{60}\min(A-B, B-C), \frac{1}{90}|A-90|\right]$$

A fuzzy equilateral triangle

$$\mu_{\tilde{E}}(A, B, C) = 1 - \frac{1}{180°}(A - C)$$

When $A = B = C$, $\mu_E(A, B, C) = 1$;
When $A = 180°$, $\mu_E = 0$.

All other triangles

(all triangular shapes other than $\tilde{I}, \tilde{R}, \tilde{E}$)

$$\tilde{T} = \overline{(\tilde{I} \cup \tilde{R} \cup \tilde{E})} = \bar{\tilde{I}} \cap \bar{\tilde{R}} \cap \bar{\tilde{E}}$$

$$U_{\tilde{T}}(A,B,C) = \min\{1-\mu_{\tilde{I}}(A,B,C), 1-\mu_{\tilde{E}}(A,B,C), 1-\mu_{\tilde{R}}(A,B,C)\}$$

$$= \frac{1}{180°}\min\{3(A-B), 3(B-C), 2|A-90°|, A-C\}$$

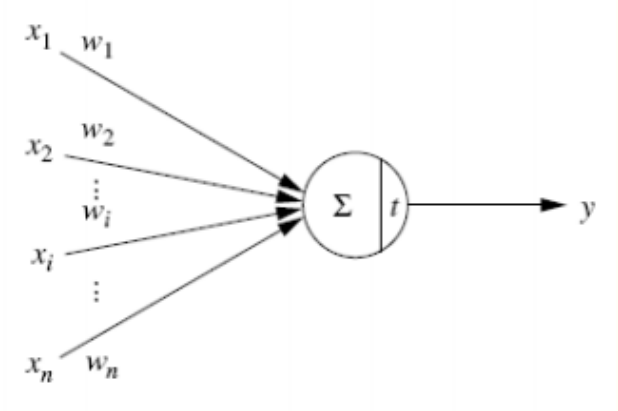
Rank Ordering"

Assessing preferences by a single individual, a committee, a poll, and other opinion methods can be used to assign membership values to a fuzzy variable. • Preference is determined by pairwise comparisons, and these determine the ordering of the membership.

Neural Network:

It is a technique that seeks to build an intelligent program using models that simulate the working network of the neurons in the human brain.

A threshold element as an analog to a Neuron



$$y = F\left(\sum w_i x_i - t\right)$$

$$F(s) = \frac{1}{1 + e^{-s}}$$

x_i signal input (i = 1, 2, . . . , n)

w_i weight associated with the signal input x_i

t threshold level prescribed by user

$F(s)$ is a nonlinear function, e.g., a sigmoid function

A simple neural network for a system

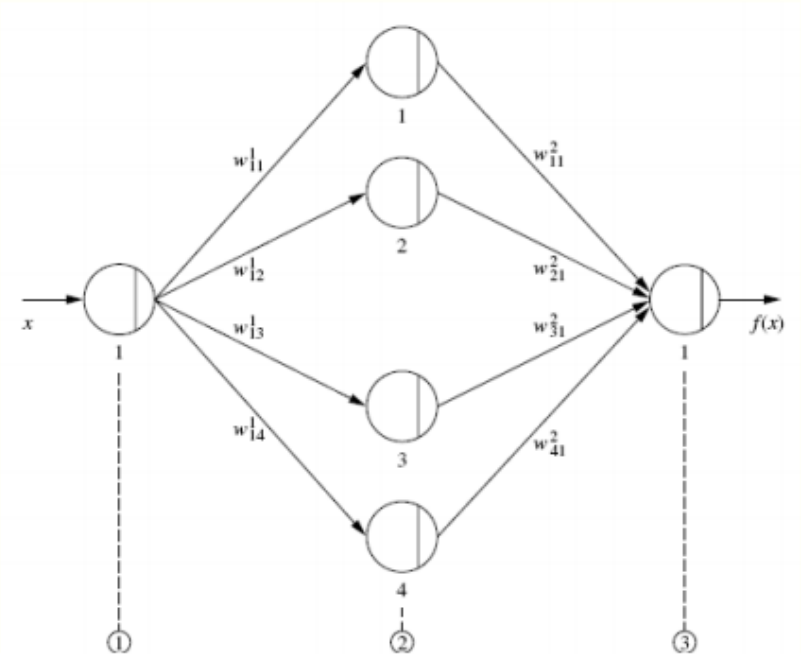


FIGURE 6.6
A simple $1 \times 4 \times 1$ neural network, where w_{jk}^i represents the weight associated with the path connecting the jth element of the ith layer to the kth element of the $(i + 1)$th layer.

UNIT 4

FUZZY AIRTHEMATIC

Fuzzy rule base and approximate reasoning

Fuzzy systems models form a special class of systems models that use the apparatus of fuzzy logic to represent the essential features of a system. From a formal point of view, fuzzy systems can be regarded as one alternative to the linear, nonlinear and neural modeling paradigms. Fuzzy systems models, however, possess a unique characteristic that is not available in most other types of formal modeling techniques — this is the ability to mimic the mechanism of approximate reasoning performed in the human mind. The most common fuzzy systems models consist of collections of logical IF — THEN rules with vague predicates; these rules along with the reasoning mechanism are the kernel of a fuzzy model.

Fuzzy Logic (FL) is a method of reasoning that resembles human reasoning. The approach of FL imitates the way of decision making in humans that involves all intermediate possibilities between digital values YES and NO.

The conventional logic block that a computer can understand takes precise input and produces a definite output a TRUE or FALSE, which is equivalent to human's YES or NO.

The inventor of fuzzy logic, Lotfi Zadeh, observed that unlike computers, the human decision making includes a range of possibilities between YES and NO, such as –

CERTAINLY YES
POSSIBLY YES
CANNOT SAY
POSSIBLY NO
CERTAINLY NO

The fuzzy logic works on the levels of possibilities of input to achieve the definite output.

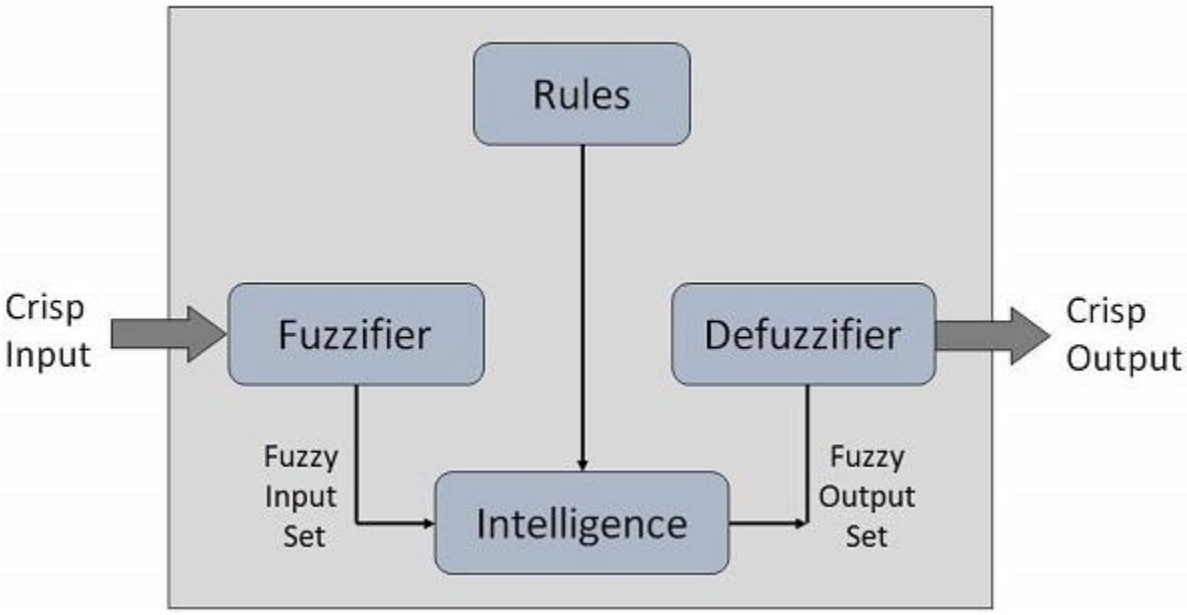
Implementation

- It can be implemented in systems with various sizes and capabilities ranging from small micro-controllers to large, networked, workstation-based control systems.
- It can be implemented in hardware, software, or a combination of both.

Fuzzy logic is useful for commercial and practical purposes.

- It can control machines and consumer products.
- It may not give accurate reasoning, but acceptable reasoning.

- Fuzzy logic helps to deal with the uncertainty in engineering.



Fuzzy Logic Systems Architecture
It has four main parts as shown −
Fuzzification Module − It transforms the system inputs, which are crisp numbers, into fuzzy Sets. It splits the input signal into five steps such as −LP x is Large Positive, MP x is Medium Positive,S x is Small,MN x is Medium Negative

Knowledge Base − It stores IF-THEN rules provided by experts.
Inference Engine − It simulates the human reasoning process by making fuzzy inference onthe inputs and IF-THEN rules.Defuzzification Module − It transforms the fuzzy set obtained by the inference engine into.
The membership functions work on fuzzy sets of variables.
Membership functions allow you to quantify linguistic term and represent a fuzzy set graphically. AMembership function for a fuzzy *set A* on the universe of discourse X is defined as $\mu A:X \rightarrow [0,1]$. Here, each element of *X* is mapped to a value between 0 and 1. It is called membership value or Degree of membership. It quantifies the degree of membership of the element in *X* to the fuzzy set *A.*

x axis represents the universe of discourse.y axis represents the degrees of membership in the [0, 1] interval. There can be multiple membership functions applicable to fuzzify a numerical value. Simple
Membership functions are used as use of complex functions does not add more precision in the
Output. All membership functions for LP, MP, S, MN, and LN are shown as below –
The triangular membership function shapes are most common among various other membershipsFunction shapes such as trapezoidal, singleton, and Gaussian.
Here, the input to 5-level fuzzifier varies from -10 volts to +10 volts. Hence the corresponding output also changes.

Advantages of FLSs:

Mathematical concepts within fuzzy reasoning are very simple. You can modify a FLS by just adding or deleting Rules due to flexibility of fuzzy logic.Fuzzy logic Systems can take imprecise, distorted, noisy input information.FLSs are easy to construct and understand. Fuzzy logic is a solution to complex problems in all fields of life, including medicine, as it resembles human reasoning and decision making.

Disadvantages of FLSs:

There is no systematic approach to fuzzy system designing. They are understandable only when simple. They are suitable for the problems which do not need high accuracy.

Application Areas of Fuzzy Logic;

The key application areas of fuzzy logic are as given −
Automotive Systems
Automatic Gearboxes
Four-Wheel Steering
Vehicle environment control
Consumer Electronic Goods
Hi-Fi Systems
Photocopiers
Still and Video Cameras
Television
Domestic Goods

Example of a Fuzzy Logic System
Let us consider an air conditioning system with 5-lvel fuzzy logic system. This system adjusts the temperature of air conditioner by comparing the room temperature and the target temperature value.

Algorithm:

- Define linguistic variables and terms.
- Construct membership functions for them.
- Construct knowledge base of rules.
- Convert crisp data into fuzzy data sets using membership functions. *Fuzzification*
- Evaluate rules in the rule base. *Interfaceengine*
- Combine results from each rule. *Interfaceengine*

Convert Logic Development:

Step 1: Define linguistic variables and terms
Linguistic variables are input and output variables in the form of simple words or sentences. For room Temperature, cold, warm, hot, etc., are linguistic terms.
Temperature *t* = {very-cold, cold, warm, very-warm, hot}

Every member of this set is a linguistic term and it can cover some portion of overall temperature values.

Step 2: Construct membership functions for them

The membership functions of temperature variable are as shown −utput data into non-fuzzy values. *defuzzification*

Step3: Construct knowledge base rules

Create a matrix of room temperature values versus target temperature values that an air Conditioning system is expected to provide. Build a set of rules into the knowledge base in the form of IF-THEN-ELSE structures.

Step 4: Obtain fuzzy value

Fuzzy set operations perform evaluation of rules. The operations used for OR and AND are Max and Min respectively. Combine all results of evaluation to form a final result. This result is a fuzzy value.

Step 5: Perform defuzzification

Defuzzification is then performed according to membership function for output variable.

Foundations of fuzzy theory:

Mathematical foundations of fuzzy logic rest in fuzzy set theory, which can be seenas a generalization of classical set theory. Fuzziness is a language concept; its main strength is its vagueness using symbols and defining them. Consider a set of tables in a lobby. In classical set theory we would ask: Is it a table? And we would have only two answers, *yes* or *no*. If we code *yes* with a 1 and *no* with a 0 then we would have the pair of answers as {0,1}. At the end we Would collect all the elements with 1 and have the set of tables in the lobby. We may then ask what objects in the lobby can *function* as a table. We could answer those tables, boxes, desks, among others can function as a table. The set is not uniquely defined, and it all depends on what we mean by the word *function*. Words like this have many shades of meaning and depend on the circumstances of the situation. Thus, we may say that the set of objects in the lobby that can Function as a table is a *fuzzy set*, because we have not crisply defined the criteria defines the *membership* of an element to the set. Objects such as tables,desks, boxes may function as a table with a certain degree, although the fuzziness is a feature of their representation in symbols and is normally a property of models, or languages.

Fuzzy Relations

In fuzzy relations we consider n-tuples of elements that are related to a *degree*. Just as the question of whether some element belongs to a set may be considered a matter of degree, whether some elements are associated may also be a matter of degree. Fuzzy relations are fuzzy sets defined on Cartesian products. While fuzzy sets are defined on a single universe of discourse, fuzzy relations are defined on higher-dimensional universes of discourse. If **S** is the universe set and A and B are subsets, A _ B will denote a product set in the universe **S** _ **S**. A fuzzy relation is a relation between elements of A and

elements of B, described by a membership function _AxB .a; b/, a 2 A and b 2 B. A discrete example of a fuzzy relation can be defined as: **S**D R, A D fa1; a2; a3;a4g D f1; 2; 3; 4g and B D fb1; b2; b3g D f0; 0:1; 2g. defines a Fuzzy.

relation: a is considerably larger than b.Definition of relation: a is considerably larger than

	b1	b2	b3
a1	0.6	0.6	0.0
a2	0.8	0.7	0.0
a3	0.9	0.8	0.4
a4	1.0	0.9	0.5

Properties of Relations:

Fuzzy relations can be represented in many ways: linguistically, listed, in a directed graph, tabular, matrix, among others. Crisp and fuzzy relations are classified on the basis of the mathematical properties they possess. In fuzzy relations, different properties call for different requirements for the membership function of a relation.

The following are some of the properties that a relation can have:

- *Reflexive*.We say that a relation R is *reflexive* if any arbitrary element x in S for which xRx is *valid*.
  Anti-reflexive. A relation R is *anti-reflexive* if there is no x in S for which xRx is valid.
- *Symmetric*. A relation R is *symmetric* if for all x and y in S, the following is true: if xRy then yRx is valid also.
- *Anti-symmetric*. A relation R is *anti-symmetric* if for all x and y in S, when xRy is valid and yRx is also valid, then x D y.
- *Transitive.* A relation R is called *transitive* if the following for all x; y; z in S: if xRy is valid and yRx is also valid, then xRz is valid as well.
- *Connected.* A relation R is *connected* when for all x; y in S, the following is true: if x . y, then either xRy is valid or yRx is valid.
- *Left unique.* A relation R is *left unique* when for all x; y; z in S the following is true: if xRy is valid and yRx is also valid, then we can infer that x D y.
- *Right unique.* A relation R is *right unique* when for all x; y; z in S the following is true: if xRy is valid and xRz is also valid, then we can infer that y D z.
- *Biunique.*A relationR that is both *left unique* and *right unique* is called *biunique.*

Fuzzy Linguistic Descriptions:

Fuzzy linguistic descriptions are often called *fuzzy systems* or *linguistic descriptions*. They are formal representations of systems made through fuzzy *IF–THEN*rules. A linguistic variable is a variable whose arguments are words modeled by *fuzzy sets*, which are called *fuzzy values*. They are an alternative to analytical modeling systems. Informal linguistic descriptions used by humans in daily life as wellas in the performance of skilled tasks are usually the starting point for the development of fuzzy linguistic descriptions. Although fuzzy linguistic descriptions are formulated in a human-like language, they have rigorous mathematical foundations involving fuzzy sets and relations. The knowledge is encoded in a statement of theform shown in

IF (a set of conditions is satisfied) *THEN* (a set of consequences can be inferred) :

A *general fuzzy IF–THEN rule* has the form:

IF a1 is A1 *AND* : : : *AND* an is An *THEN* b is B

Reasoning with Fuzzy Rules

Fuzzy reasoning includes two distinct parts: evaluating the rule antecedent (*IF* part of the rule) and implication or applying the result to the consequent, the *THEN* part of the rule.While in classical rule-based systems if the antecedent of the rule is true, then the consequent is also true, but in fuzzy systems the evaluation is different. In fuzzy systems the antecedent is a fuzzy statement, this means all the rules fire at some extent. If the antecedent is true in some degree of membership, then the consequent is also true in some degree.

Example 2.11. Consider two fuzzy sets, *tall men* and *heavy men* represented in. These fuzzy sets provide the basis for a weight estimation model. Themodel is based on a relationship between a man's height and his weight, which can be expressed with the following rule: *IF* height is *tall*, *THEN* weight is *heavy*. The value of the output or the membership grade of the rule consequent can be estimated directly from a corresponding membership grade in the antecedent. Fuzzy rules can have multiple antecedents, as the consequent of the rule, which can also includmultiple parts. UtIn general, fuzzy expert systems incorporate not one but several rules that describeexpert knowledge. The output of each rule is a fuzzy set, but usually we need to obtain a single number representing the expert system output, the crisp solution. To obtain a single crisp output a fuzzy expert system first aggregates all output fuzzy sets into a single output fuzzy set, and then defuzzifies the resulting set int

The Fuzzy Logic Controller

With traditional sets an element either belongs to the set or does not belong to the set {0,1}, while in fuzzy sets the degree to which the element belongs to the set is analyzed and it is called the membership degree, giving values in the range [0,1], where 1 indicates that the element belongs completely to the set. The*fuzzy logic controllers* (FLC) make a non-linear mapping between the input and the output using *membership functions* and *linguistic rules* (normally in the form if__then__). In order to use a FLC, knowledge is needed and this can be represented as two different types:

1. *Objective* information is what can be somehow quantifiable by mathematical models and equations.

2. *Subjective* information is represented with linguistic rules and design requirements

Linguistic Variables:

Just like in human thinking, in fuzzy logic systems (FLS) linguistic variables are utilized to give a "value" to the element, some examples are *much*, *tall*, *cold*, etc. FLS require the linguistic variables in relation to their numeric values, their quantification and the connections between variables and the possible implications.

Membership Functions
In FLS the membership functions are utilized to find the degree of membership of the element in a given set.
Rules Evaluation:

The rules used in the FLS are of the *IF–THEN* type, for example, *IF* x1 is big

THEN y1 is small. To define the rules you need an expert, or you must be able to extract the information from a mathematic formula. The main elements of a FLC are fuzzification, rules evaluation, and defuzzification.

Rules Evaluation:

After getting the membership values of the inputs, they are evaluated using *IF–THEN* rules: *IF* a is x *AND* b is y *AND c* is z *THEN* w, where a, b and c are thecrisp inputs, x, y and z are the fuzzy clusters to which the inputs may correspond, and w is the output fuzzy cluster used to defuzzify. To be able to obtain the fuzzy values of the outputs, the system has to use an inference engine. The min–max composition is used which takes the minimum of the premises and the maximum of the consequences.

Fuzzy Numbers:
Fuzzy numbers are the basis for fuzzy arithmetic. A fuzzy number is a fuzzy subset of the universe of numerical numbers e.g. afuzzy integer is a fuzzysubset of the domain of integers. While Fig. a is crisp Number 1.3, depicts b the fuzzy number 1.3, or in other words the fuzzy set "around 1.3"or "close to 1.3" Fig. b,d are for the Interval 1.25 to1.35

The extension Principle :
In a mapping provided by the general function *f: y = f (x)*, if the input, *x* is crisp, then the resulting output, *y*, is also crisp. An extension principle developed by Zadeh enables us to extend the domain ofa function on fuzzy sets. It thus generalizes a common point-topoint mapping of a function f(.) to a mapping between fuzzy sets.

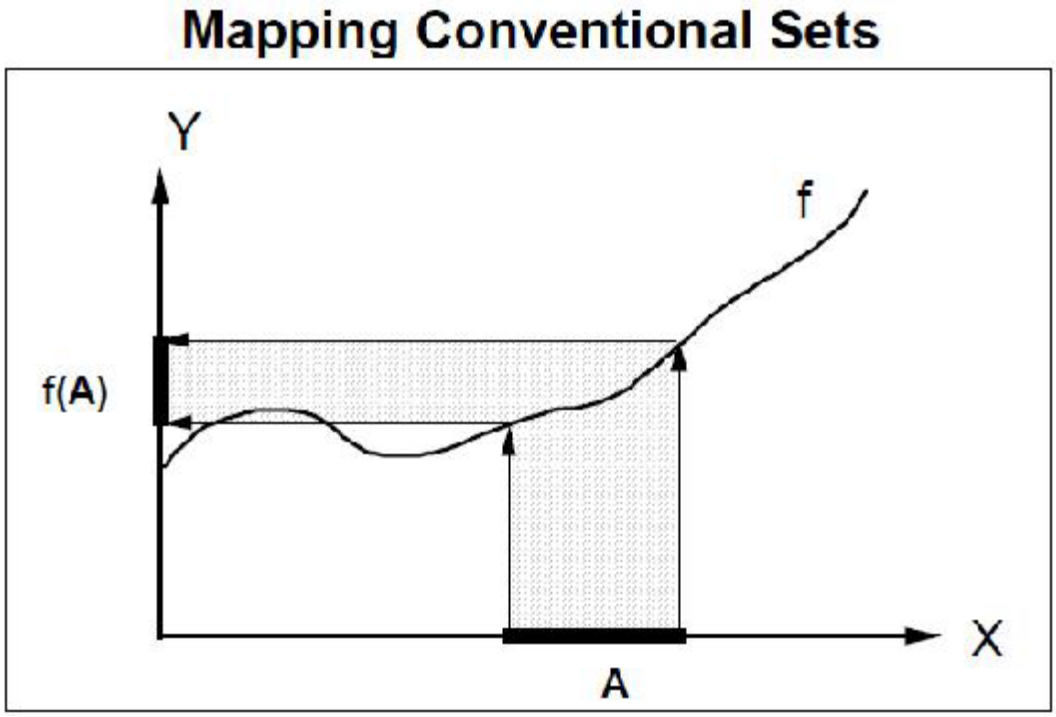
Let *A ,B* be two fuzzy sets, defined in universe of discourse *X,Y.*And let '*f ' be* a nonfuzzy transformation function between universes X and Y, so that *f* : X □ Y. We say that the crisp function
f: X □ Y

is fuzzified when it is extended to act on fuzzy sets defined on *X and Y.*

Fuzzy Arithmetic Operations:
Applying the extension principle to arithmetic operations it is possible to define fuzzy arithmetic operations.
Let *x and y be the operands, z the* result. Let *A and B denote the fuzzy sets that* represent the operands *x and y* respectively. the symbol ∨ is the maximum operator and ∧ is the minimum operator

. Fuzzy Addition

$$\mu_{A+B}(z) = \vee_{\substack{x,y \\ x+y=z}} (\mu_A(x) \wedge \mu_B(y))$$

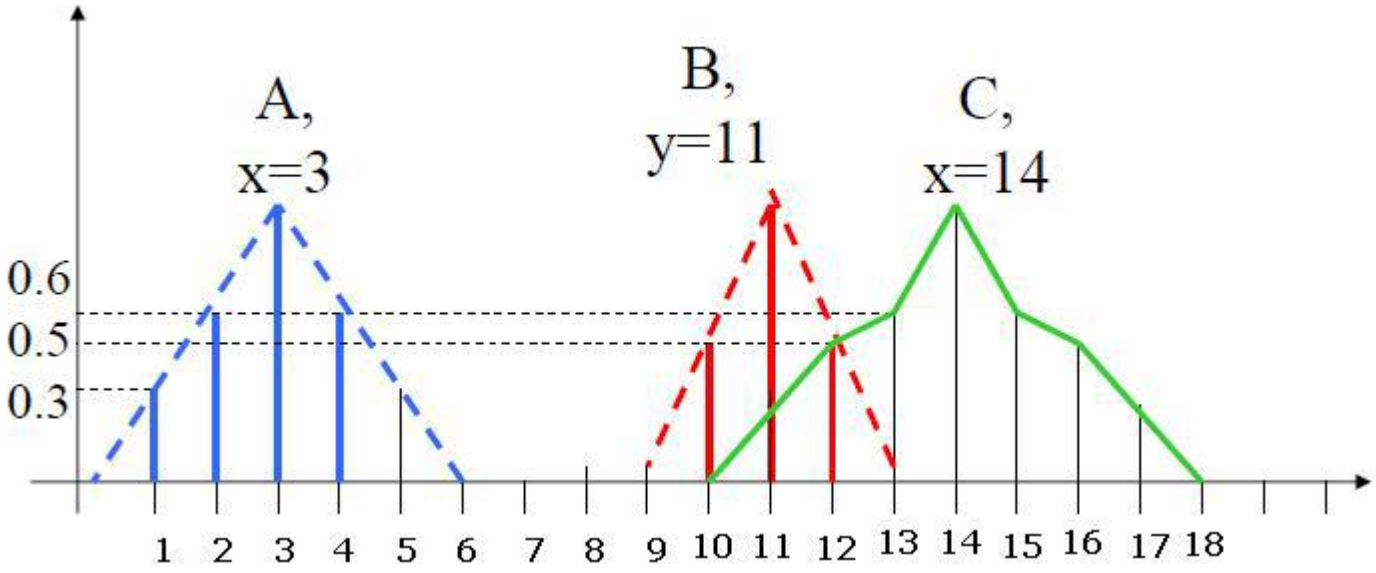


= fuzzy number 3 = 0.3/1+0.6/2+1/3+0.6/4+0.3/5
B = fuzzy number 11= 0.5/10 + 1/11 + 0.5/12

A+B= (0.3^0.5)/(1+10) + (0.3^1)/(1+11) + (0.3^0.5)/(1+12) +(0.6^0.5)/(2+10) + (0.6^1)/(2+11) + (0.6^0.5)/(2+12) + (1^0.5)/(3+10) + (1^1)/(3+11) + (1^0.5)/(3+12) +(0.6^0.5)/(4+10) +(0.6^1)/(4+11) + (0.6^0.5)/(4+12) + (0.3^0.5)/(5+10) + (0.3^1)/(5+11) + (0.3^0.5)/(5+12)

Getting the minimum of the membership values:
A+B=0.3/11 + 0.3/12 + 0.3/13 +0.5/12 + 0.6/13 + 0.5/14 + 0.5/13 + 1/14 + 0.5/15 +0.5/14 + 0.6/15 + 0.5/16 + +0.3/15 + 0.3/16 + 0.3/17
Getting the maximum of the duplicated values:

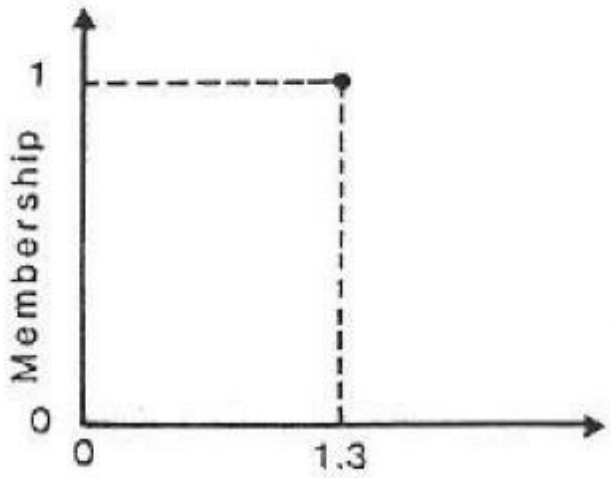
A+B=0.3/11 + (0.3 V 0.5)/12 + (0.3 V 0.6 V 0.5)/13 + (0.5 V 1 V 0.5)/14 +(0.5 V 0.6 V 0.3)/15 + (0.5 V 0.3)/16 + 0.3/17
A+B=0.3 / 11 + 0.5 / 12 + 0.6 / 13 + 1 / 14 + 0.6 / 15 + 0.5 / 16 + 0.3 / 17



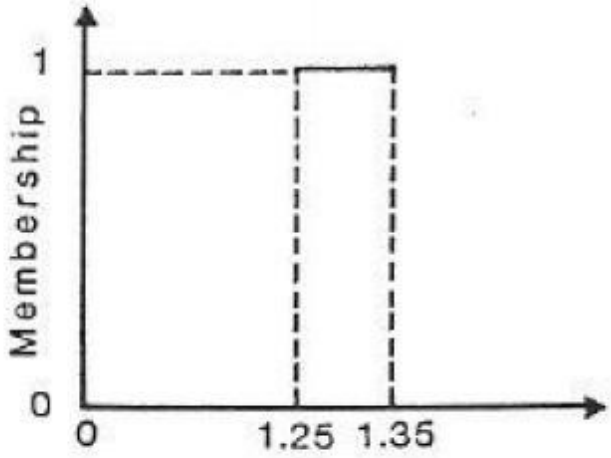
Fuzzy Subtraction

$$\mu_{A-B}(z) = \vee_{\substack{x,y \\ x-y=z}} \mu_A(x) \wedge \mu_B(y)$$

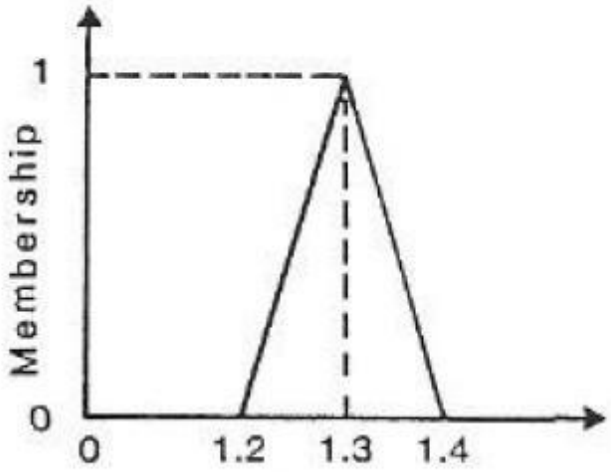
Fuzzy Multiplication


1
Membership
0
0
1.3

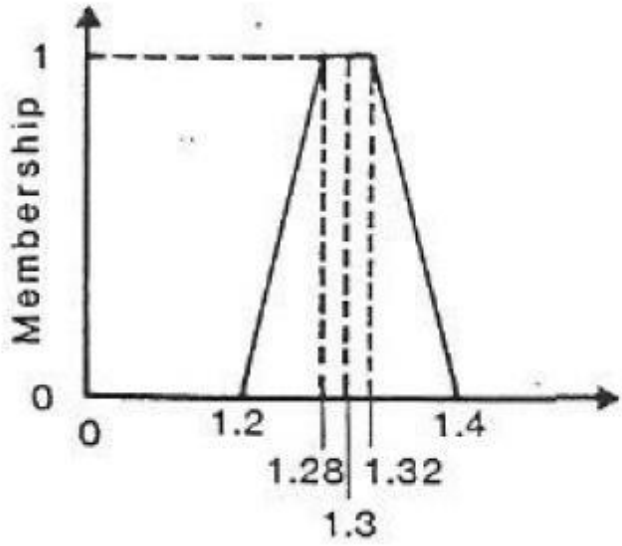
(a)


1
Membership
0
0
1.25
1.35

(b)


1
Membership
0
0
1.2
1.3
1.4

(c)


1
Membership
0
0
1.2
1.4
1.28
1.32
1.3

(d)

UNIT-5

GENETIC ALGORITHRMS

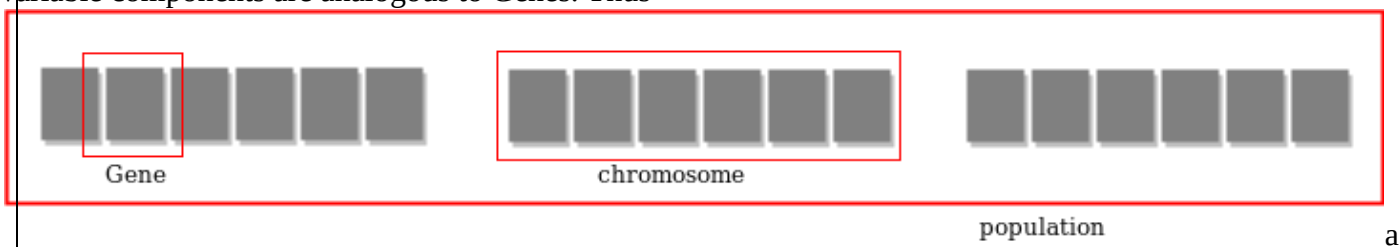
Genetic algorithm and search space:
Genetic Algorithms (GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate "survival of the fittest" among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.
Foundation of Genetic Algorithms
Genetic algorithms are based on an analogy with genetic structure and behavior of chromosome of the population. Following is the foundation of GAs based on this analogy –

1. Individual in population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from "fittest" parent propagate throughout the generation that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.

Search space

The populations of individuals are maintained within search space. Each individual represent a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus



a chromosome (individual) is composed of

Fitness Score
A Fitness Score is given to each individual which shows the ability of an individual to "compete". The individual having optimal fitness score (or near optimal) are sought.
The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness scores.The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are

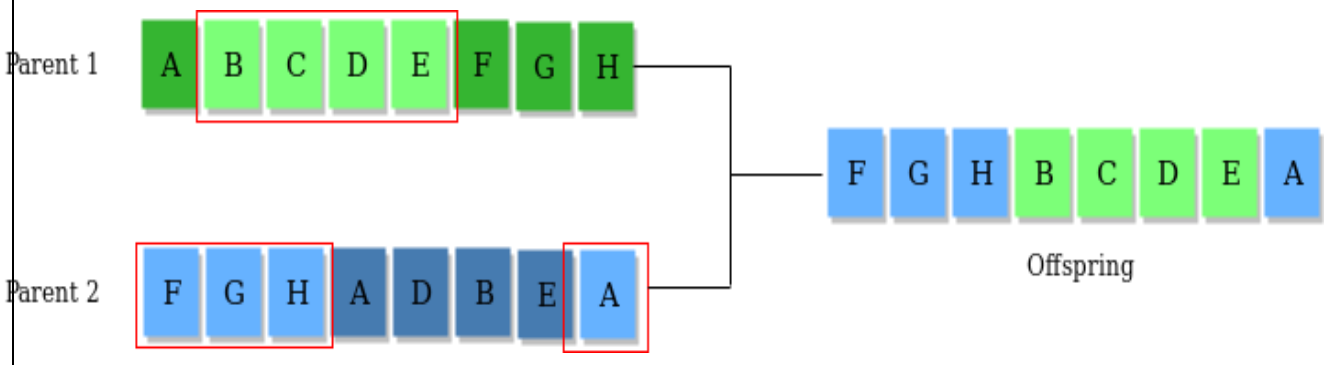
selected who mate and produce better offspring by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die.
Each new generation has on average more "better genes" than the individual (solution) of previous generations. Thus each new generation have better "partial solutions" than previous generations. Once the off springs produced having no significant difference than offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

General genetic algorithm:

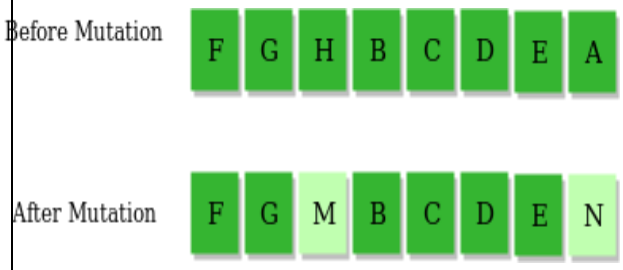
Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.
GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

Operators of Genetic Algorithms:

Once the initial generation is created, the algorithm evolves the generation using following operators –
1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to the successive generations.
2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring). For example



3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in population to avoid the premature convergence. For example **–**



The whole algorithm can be summarized as –

1) Randomly initialize population's p

2) Determine fitness of population

3) Untill convergence repeat:

a) Select parents from population

b) Crossover and generate new population

c) Perform mutation on new population

d) Calculate fitness for new population

Example problem and solution using Genetic Algorithms:

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

- Characters A-Z, a-z, 0-9 and other special symbols are considered as genes
- A string generated by these character is considered as chromosome/solution/Individual

Fitness score is the number of characters which differ from characters in target string at a particular index. So individual having lower fitness value is given more preference.

Generational cycle:

In genetic algorithms and evolutionary computation, crossover, also called recombination, is a genetic operator used to combine the genetic information of two parents to generate new offspring. It is one way to stochastically generate new solutions from an existing population, and analogous to the crossover that happens during sexual reproduction in biology. Solutions can also be generated by cloning an existing solution, which is analogous to asexual reproduction. Newly generated solutions are typically mutated before being added to the population.

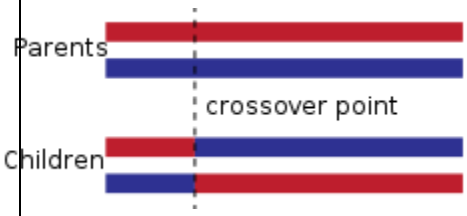
Different algorithms in evolutionary computation may use different data structures to store genetic information, and each genetic representation can be recombined with different crossover operators. Typical data structures that can be recombined with crossover are bit arrays, vectors of real numbers, or trees.

Traditional genetic algorithms store genetic information in a chromosome represented by a bit array. Crossover methods

for bit arrays are popular and an illustrative example of genetic recombination.

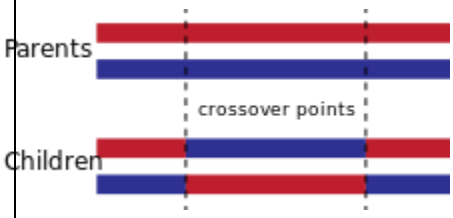
Single-point crossover

A point on both parents' chromosomes is picked randomly, and designated a 'crossover point'. Bits to the right of that point are swapped between the two parent chromosomes. This results in two offspring, each carrying some genetic information from both parents.



Two-point and k-point crossover

In two-point crossover, two crossover points are picked randomly from the parent chromosomes. The bits in between the two points are swapped between the parent organisms.



Two-point crossover is equivalent to performing two single-point crossovers with different crossover points. This strategy can be generalized to k-point crossover for any positive integer k, picking k crossover points.

Uniform crossover

In uniform crossover, each bit from the offspring's genome is independently chosen from the two parents according to a given distribution. In contrast to k-point crossover, uniform crossover exchanges individual bits and not segments of the bit array. This means there is no bias for two bits that are close together in the array to be inherited together.

Typically, each bit is chosen from either parent with equal probability. Other mixing ratios are sometimes used, resulting in offspring which inherit more genetic information from one parent than the other.

Crossover for ordered lists

In some genetic algorithms, not all possible chromosomes represent valid solutions. In some cases, it is possible to use specialized crossover and mutation operators that are designed to avoid violating the constraints of the problem.

For example, a genetic algorithm solving the travelling salesman problem may use an ordered list of cities to represent a solution path. Such a chromosome only represents a valid solution if the list contains all the cities that the salesman must visit. Using the above crossovers will often result in chromosomes that violate that constraint. Genetic algorithms optimizing the ordering of a given list thus require different crossover operators that will avoid generating invalid solutions. Many such crossovers have been published:[1]

1. partially matched crossover (PMX)
2. cycle crossover (CX)
3. order crossover operator (OX1)
4. order-based crossover operator (OX2)

5. position-based crossover operator (POS)
6. voting recombination crossover operator (VR)
7. alternating-position crossover operator (AP)
8. sequential constructive crossover operator (SCX)

Other possible methods include the edge recombination operator.

Stopping condition:

The stopping condition of a Genetic Algorithm is important in determining when a GA run will end. It has been observed that initially, the GA progresses very fast with better solutions coming in every few iteration, but this tends to saturate in the later stages where the improvements are very small. We usually want a termination condition such that our solution is close to the optimal, at the end of the run.

Usually, we keep one of the following termination conditions −

- When there has been no improvement in the population for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value.

For example, in a genetic algorithm we keep a counter which keeps track of the generations for which there has been no improvement in the population. Initially, we set this counter to zero. Each time we don't generate off-springs which are better than the individuals in the population, we increment the counter.

However, if the fitness any of the off-springs is better, then we reset the counter to zero. The algorithm terminates when the counter reaches a predetermined value.

Like other parameters of a GA, the termination condition is also highly problem specific and the GA designer should try out various options to see what suits his particular problem the best.

Constraints:
In mathematics, a constraint is a condition of an optimization problem that the solution must satisfy. There are several types of constraints—primarily equality constraints, inequality constraints, and integer constraints. The set of candidate solutions that satisfy all constraints is called the feasible set.
Example:
The following is a simple optimization problem:
Min $f(x) = x_1^2 + x_2^4$
Subject to $x_{1>1 \text{ and }} x2_{=1}$

Where x denotes the vector (x_1, x_2).

In this example, the first line defines the function to be minimized (called the objective function, loss function, or cost

function). The second and third lines define two constraints, the first of which is an inequality constraint and the second of which is an equality constraint. These two constraints are hard constraints, meaning that it is required that they are satisfied; they define the feasible set of candidate solutions.

Without the constraints, the solution would be (0, 0), where f(x) has the lowest value. But this solution does not satisfy the constraints. The solution of the constrained optimization problem. stated above is , x=(1,1),which is the point with the smallest value of f(x) that satisfies the two constraints.

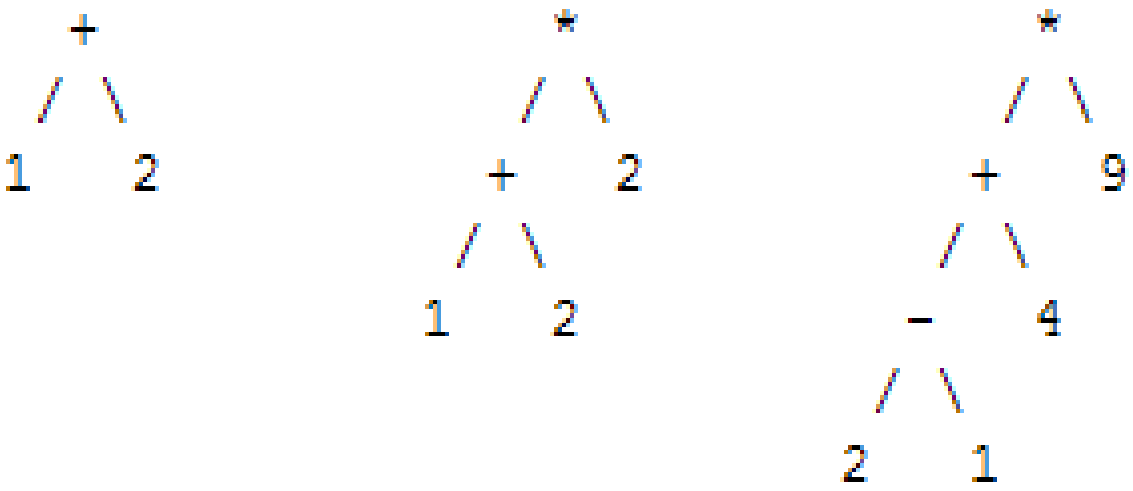
Hard and Soft constraints:

If the problem mandates that the constraints be satisfied, as in the above discussion, the constraints are sometimes referred to as *hard constraints*. However, in some problems, called flexible constraint satisfaction problems, it is preferred but not required that certain constraints be satisfied; such non-mandatory constraints are known as *soft constraints*. Soft constraints arise in, for example, preference-based planning. In a MAX-CSP problem, a number of constraints are allowed to be violated, and the quality of a solution is measured by the number of satisfied constraints.

Genetic programming:

- In programming languages such as LISP, the mathematical notation is not written in standard notation, but in prefix notation. Some examples of this:
- + 2 1
- * + 2 1 2
- * + - 2 1 4 9 :
- Notice the difference between the left-hand side to the right? Apart from the order being different, no parenthesis! The prefix method makes it a lot easier for programmers and compilers alike, because order precedence is not an issue.
- You can build expression trees out of these strings that then can be easily evaluated, for example, here are the trees for the above three expressions.

- You can see how expression evaluation is thus a lot easier.
- What this have to do with GAs? If for example you have numerical data and 'answers', but no expression to conjoin the data with the answers.
- A genetic algorithm can be used to 'evolve' an expression tree to create a very close fit to the data.
- By 'splicing' and 'grafting' the trees and evaluating the resulting expression with the data and testing it to the answers, the fitness function can return how close the expression is.
- The limitations of genetic programming lie in the **huge** search space the GAs have to search for - an infinite number of equations.
- Therefore, normally before running a GA to search for an equation, the user tells the program which operators and numerical ranges to search under.
- Uses of genetic programming can lie in stock market prediction, advanced mathematics and military applications



Multilevel optimization:

The same is true for other areas such as multi-objective programming (there are always several goals in a real application), stochastic programming (all data is uncertain and therefore stochastic models should be used), and so forth. In this spirit we claim: The word is multilevel. In many decision processes there is a hierarchy of decision makers, and decisions are made at different levels in this hierarchy. One way to handle such hierarchies is to focus on one level and include other levels' behaviors as assumptions. Multilevel programming is the research area that focuses on the whole hierarchy structure. In terms of modeling, the constraint domain associated with a multilevel programming problem is implicitly determined by a series of optimization problems which must be solved in a predetermined sequence. If only two levels are considered, we have one leader (associated with the upper level) and one follower (associated with the lower level).

A multilevel genetic algorithm (MLGA) is proposed in this paper for solving the kind of optimization problems which are multilevel structures in nature and have features of mixed-discrete design variables, multi-modal and non-continuous objective functions, etc. Firstly, the formulation of the mixed-discrete multilevel optimization problems is presented. Secondly, the architecture and implementation of MLGA are described. Thirdly, the algorithm is applied to two multilevel optimization problems. The rst one is a three-level optimization problem in which the optimization of the number of actuators, the positions of actuators and the control parameters are considered in dierent levels. An actively controlled tall building subjected to strong wind action is considered to investigate the eectiveness of the proposed algorithm. The second application considers a combinatorial optimization problem in which the number and conguration of actuators are optimized simultaneously, an actively controlled building under earthquake excitations is adopted for this case study. Finally, some results and discussions about the application of the proposed algorithm are presented.

Application of Genetic Algorithm:

Genetic algorithms have been used for difficult problems (such as NP-hard problems), for machine learning and also for evolving simple programs. They have been also used for some art, for evolving pictures and music. A few applications of GA are as follows:

- Business: Genetic Algorithms have been used to solve many different types of business problems in functional areas such as finance, marketing, information systems, and production/ operations. Within these functional areas, GAs has performed a variety of applications such as tactical asset allocation, job scheduling, machine-part grouping, and computer network design.
- 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 , job shop scheduling and video & sound quality optimization, Telecommunication routing, State assignment problem, Time tabling problem, Traffic and Shipment routing etc.
- Automatic Programming: They are used to evolve computer programs for specific tasks and to design other computational structures as in Cellular automata and sorting networks.
- Design: They are also used to optimize the structure and operational design of buildings, factories, machines etc. They are used to design heat exchangers, robot gripping arms, flywheels, turbines etc.
- Robotics: Robot’s design is dependent on the job it is intended to do. A range of optimal designs and components can be searched with the help of genetic algorithms for each specific use and return entirely new type of robots.

- Machine Learning: These algorithms are used for machine learning applications like and prediction, protein structure prediction etc. They are also used to design neural networks, to evolve rules for learning classifier systems and symbolic production systems.
- Evolvable Hardware: Genetic algorithms are used develop computer models that use stochastic operators to evolve new configurations from old ones so as develop new electronic circuits that can be termed as evolvable hardware.
- Game Playing: Genetic algorithms are also applied in Game theory and so they are widely used in developing computer games, simulated environments.
- Encryption and code breaking: Genetic algorithms can be used both to create encryption for sensitive data as well as to break those codes.

  Image processing: With medical X-rays or satellite images, there is often a need to align two images of the same area, taken at different times. By comparing a random sample of points on the two images, a GA can efficiently and a set of equations which transform one image to fit onto the other.

Optimization of travelling salesman problem using genetic algorithm approach:

The Travelling Salesman Problem (TSP) is a classic combinatorial optimization problem, which is simple to state but very difficult t o solve. This problem is known to be NP-hard, and cannot be solved exactly in polynomial time. Many exact and heuristic algorithms have been developed in the field of operations research (OR) to solve this problem. The problem is to find the shortest possible tour through a set of n vertices so that each vertex is visited exactly once. The traveling salesman first gained fame in a book written by German salesman BF Voigt in 1832 on how to be a successful traveling salesman. He mentions the TSP, although not by that name, by suggesting that to cover as many locations as possible without visiting any location twice is the most important aspect of the scheduling of a tour.

The origins of the TSP in mathematics are not really known -all we know for certain is that it happened around 1931. On the basis of the structure of the cost matrix, the TSPs are classified into two groups – symmetric and asymmetric. The TSP is symmetric if cij = cji, for all i, j and asymmetric otherwise. For an n-city asymmetric TSP, there are (n −)!1 possible solutions, one or more of which gives the minimum cost. For an n-city symmetric TSP, there are (n − 1)!/2 possible solutions along with their reverse cyclic permutations having the same total cost. In either case the number of solutions becomes extremely large for even moderately large n so that an exhaustive search is impracticable.

Genetic algorithm based internet search technique:

Abstract: Internet search is becoming problematic due to Information overload on the Internet. In order to help users in information searching, various applications appeared. There are two basic approaches to Internet search: indexing and agent search. An agent presented in this paper uses genetic algorithm for global search. It is written in the Java programming language. For a set of input documents, the agent finds and displays similar documents and the information regarding how similar they are. It uses database of topic sorted URLs to perform mutation.

Introduction:

Each day number of documents and servers on the Internet grows rapidly. In such an abundance of data it is hard to get the information needed in the shortest time possible. There are two approaches to Internet search:

1. Indexing and database searching (e.g. Altavista, Yahoo...) - tools for indexing fetch all available documents on the Internet, parse their contents, and store it in an indexed database. Based on the keywords, submitted by the user, the tool searches the database and displays all documents containing desired keywords. Advantage of this approach is that it covers a big search space, but these tools have usually poor evaluation function, and keywords that user submits have to be carefully picked up, in order to obtain a desired number and quality of resulting documents.
2. Agent search - autonomous program (agent) performs the search without user supervision. It takes the number of URLs as input parameters and follows their links in order to find similar documents on the Web.

Program package Tropical, developed in this B.Sc. thesis is agent that performs Internet search using genetic algorithm with database mutation. It is written in Java programming language.

Problem Definition:

Client-based searching agents can be agents for local search (which means that they search for documents linked to the input documents) and agents for non-local search (which can find documents that are not linked to the input documents).

Broadly defined, an agent is a program that can operate autonomously to accomplish unique tasks without direct human supervision.

The agent presented in this paper named Tropical is an agent for global search, written in the Java programming language. For set of input documents, the agent finds and displays similar documents and the information regarding how similar they are. It uses services of following packages:

- Spider for document fetching
- Agent for Best First Search algorithm
- Generator for designing and communication with the database of topic sorted URLs.

Basic Genetic Algorithm

Genetic algorithm is a search method that is used for covering a big search space, and finding the optimal solution. Its application is especially important in AI. It is frequently used for finding optimal schedule of resource. Basic steps in the workflow of genetic algorithm are:

1. representing each solution uniformly in a way suitable for computer processing (often like arrays or strings)
2. initialization of the current set of solutions that is examined (current configuration set) by picking randomly from the whole search space desired number of potential solutions
3. Forming set of promising solutions for further examination (mating pool) by picking the best ones from current configuration set. How good is the solution, determines the fitness function which is defined for each problem separately.
4. Performing operators of crossover and mutation on mating pool and thus generating new solutions. Crossover operator generates new solution by using genetic material of existing solutions in the mating pool. Mutation operator generates a whole new solution randomly. New solutions are then inserted in the current configuration set.
5. If stopping criterion is satisfied algorithm ends, otherwise steps 3-5 are repeated.

Genetic Algorithm for Internet Search

Algorithm performs following steps:

1. User submits input set of documents like string of URLs.

5. From the Web are fetched the input documents and parsed in order to extract keywords.
6. From the Web are fetched documents that are pointed to by the hyperlinks in input documents and they form the current configuration set H.
7. From the database is randomly picked desired number of URLs that cover the same topic as do the input documents (this topic is specified by the user at the beginning of the program) and corresponding documents are fetched from the Web and inserted in the set H.
8. Determining the documents from the set H that are most similar to the input ones by calculating Jaccard Score for each of them.
9. Inserting the most similar document in the output set and adding the documents linked to it in the set H.

   Repeating steps 4-6 until desired number of output documents is obtained

Some Relevant Details

Linking the database - just once before you first start the program

Start button in Windows environment, Settings, Control Panel, ODBC, User DSN Window, Add, Microsoft Access Driver, Finish, In the field Data Source Name put URL base, Select, select the database url.mdb in the tropical directory, Advanced, in the fields for username and password type mladen and java

Starting the program

In JDK 1.1.4 Package:

java tropical -dProxySet=true

-dProxyHost=proksi

-dProxyPort=port

proksi is IP address of proxy server you are using
port port number of this proxy server

I suggest you put this line in tropical.bat file and then just type tropical each time.

Example of the command line is:

Java tropical -dProxySet=true

-dProxyHost=147.91.8.62

-dProxyPort=8080

Input parameters

- Input Set - field for input set of URLs. It is necessary that you enter either a set of URLs or a name of the file containing URLs or both.
- Input file - field for the name of the file with URLs (each in the new line)
- Number of outputs - field for the required number of output documents. Default number is 10.
- Database mutation - list for desired number of mutated documents per iteration. Default number is 0.
- Spatial mutation - not realized yet.
- Temporal mutation - not realized yet.
- Topic - list for choosing one or more topics covered by mutation.
- Comments - comments on/off field.

Output window

- Program Flow - some program messages that keep you informed about program execution.
- Result Set - result set of URLs.
- Medium Jaccard's Score - medium Jaccard's Score for the result set.

Result of the program work can also be found in the file izlaz.html in the Tropical directory and can be viewed during program execution.

Conclusion:

Agent for global search was meant to be a part of a bigger project, but also to be able to do the autonomous work. During "Tropical" development, it was noticed that it could be improved. For example, improvement of class which performs extraction of keywords, creation of mobile agents, intelligent search, cooperation with Altavista tool in order to cover a bigger search space... Agent presented in this paper can give better search results than index engines in cases when desired URLs are in the database regarding time of search and quality of the documents found.

*

■