RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Information Technology, VII- semester

IT 701 Soft Computing

Course Objective:
The objective of this course is to familiarize the students with different soft computing tools to use them to be able to solve complex problems

Unit I Introduction to Neural Network: Concept, biological neural network, comparison of ANN with biological NN, evolution of artificial neural network, Basic models, Types of learning, Linear separability, XOR problem, McCulloch-Pitts neuron model, Hebb rule.

Unit II Supervised Learning: Perceptron learning, Single layer/multilayer, Adaline, Madaline, Back propagation network, RBFN, Application of Neural network in forecasting, data compression and image compression.

Unit III Unsupervised learning: Introduction, Fixed weight competitive nets, Kohonen SOM, Counter Propagation networks, (Theory, Architecture, Flow Chart, Training Algorithm and applications). Introduction to Convolutional neural networks (CNN) and Recurrent neural networks (RNN).

Unit IV Fuzzy Set: Introduction, Basic Definition and Terminology, Properties and Set-theoretic Operations, , Fuzzy Relations , Membership Functions and their assignment, Fuzzy rules and fuzzy Reasoning, Fuzzy if-then Rules, Fuzzy Inference Systems. Application of Fuzzy logic in solving engineering problems.

Unit V Genetic Algorithm: Introduction to GA, Simple Genetic Algorithm, terminology and operators of GA (individual, gene, fitness, population, data structure, encoding, selection, crossover, mutation, convergence criteria). Reasons for working of GA and Schema theorem, GA optimization problems like TSP (Travelling salesman problem), Network design routing. Introduction to Ant Colony optimization (ACO) and Particle swarm optimization (PSO).

References-

1. S.N. Shivnandam, “Principle of soft computing”, Wiley.
2. S. Rajshekaran and G.A.V. Pai, “Neural Network , Fuzzy logic And Genetic Algorithm”, PHI.
3. Jack M. Zurada, “Introduction to Artificial Neural Network System” JAico Publication.
4. Simon Haykins, “Neural Network- A Comprehensive Foudation”
5. Timothy J.Ross, “Fuzzy logic with Engineering Applications”, McGraw-Hills 1.

Suggested List of Experiments-

1. Form a perceptron net for basic logic gates with binary input and output.
2. Using Adaline net, generate XOR function with bipolar inputs and targets.
3. Calculation of new weights for a Back propagation network, given the values of input pattern, output pattern, target output, learning rate and activation function.
4. Design fuzzy inference system for a given problem.
5. Maximize the function $y = 3x^2 + 2$ for some given values of x using Genetic algorithm.
6. Implement Travelling salesman problem using Genetic Algorithm.
7. Optimisation of problem like Job shop scheduling using Genetic algorithm

Course Outcomes:

After the completion of this course, the students will be able to:

1. Understand concept of ANN and explain the XOR problem
2. Use supervised neural networks to classify given inputs
3. Understand unsupervised neural networks for clustering data .
4. Build Fuzzy inference system using concepts of fuzzy logic.
5. Obtain an optimized solution to a given problem using genetic algorithm.

Subject Name: Soft Computing **Subject Code: IT 701**

Subject Notes

Syllabus:

Supervised Learning: Perceptron learning, Single layer/multilayer, Adaline, Madaline, Back propagation network, RBFN, Application of Neural network in forecasting, data compression and image compression.

Unit-2

Course Objectives

1. The objective of this course is to understand supervised learning and its type.
2. To help student to understand neural network work in forecasting and data compression, image compression .

Supervised Learning:

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

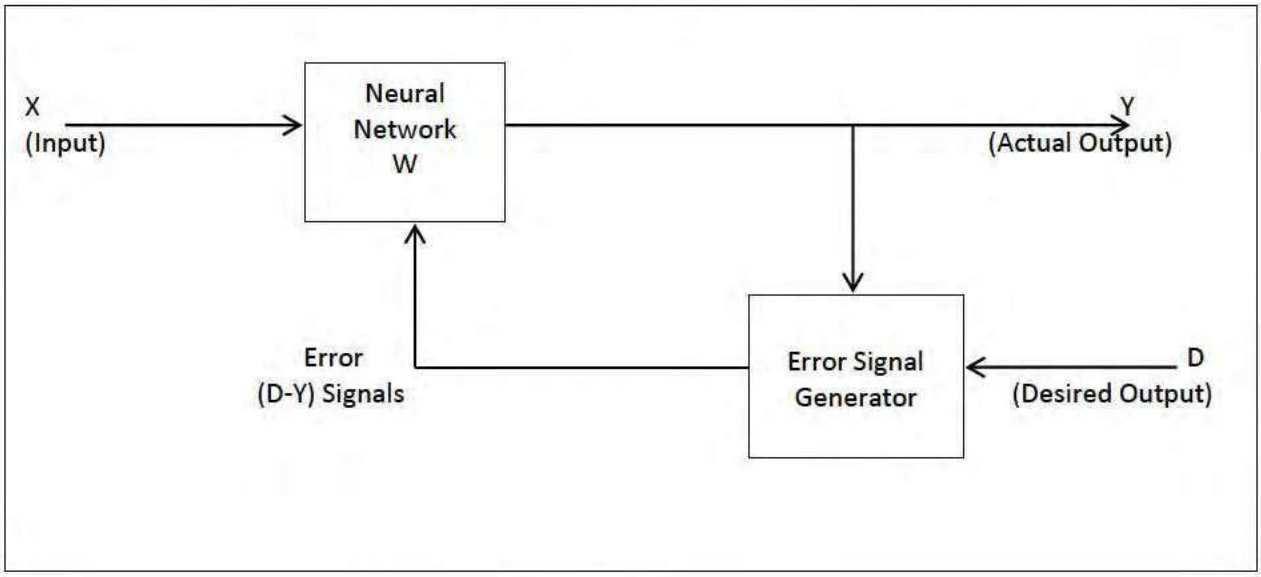


Figure 2.1: Supervised Learning

It is called supervised learning because the process of algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers; the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

During training, the input vector is presented to the network, which results in an output vector. This output vector is the actual output vector. Then the actual output vector is compared with the desired (target) output vector. If there exists a difference between the two output vectors then an error signal is generated by the network. This error is used for adjustment of weights until the actual output matches the desired (target) output.

In this type of learning, a supervisor or teacher is required for error minimization. Hence, the network trained by this method is said to be using supervised training methodology. In supervised learning it is assumed that the correct "target" output values are known for each input pattern.

Perceptron Learning:

The perceptron is an algorithm for supervised learning of binary classifier. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. Perceptron network come under single-layer feed-forward networks.

The key points to be noted:

The perceptron network consists of three units, namely, sensory unit (input unit), associator unit (hidden unit), and response unit (output unit).

The sensory unit are connected to associator units with fixed weights having values 1, 0, or -1, which are assigned at random.

The binary activation function is used in sensory unit and associator unit.

The response unit has an activation function of 1, 0, or -1. The binary step with fixed threshold θ is sued as activation for associator. The output signals that are sent from the associator unit to the response time are only binary.

The output of the perceptron network is given by $y=f(y_{in})$

Where $f(y_{in})$ is activation function and is defined as:

$f(y_{in})$	=	1	If $y_{in} > \theta$
		0	If $-\theta <= y_{in} <= \theta$
		-1	If $y_{in} < -\theta$

Table 2.1: Perceptron Network

Perceptron Learning Rule:

In case of the perceptron learning rule, the learning signal is the difference between the desired and actual response of a neuron.

Consider a finite "n" number of input training vectors, with their associated target (desired) values x(n) and t(n), where "n" ranges from 1 to N. The target is either +1 or -1. The output "y"

is obtained on the basis of the net input calculated and activation function being applied over the net input.

$f(y_{in})$	=	1	If $y_{in} > \theta$
		0	If $-\theta <= y_{in} <= \theta$
		-1	If $y_{in} < -\theta$

Table 2.2: Perceptron Learning Rule

In original perceptron network, the output obtained from the associator unit is a binary vector, and hence the output can be taken as input signal to the response unit, and classification can be performed. Here only weights between the associator unit and the output unit can be adjusted, and the weights between the sensory and associator units are fixed.

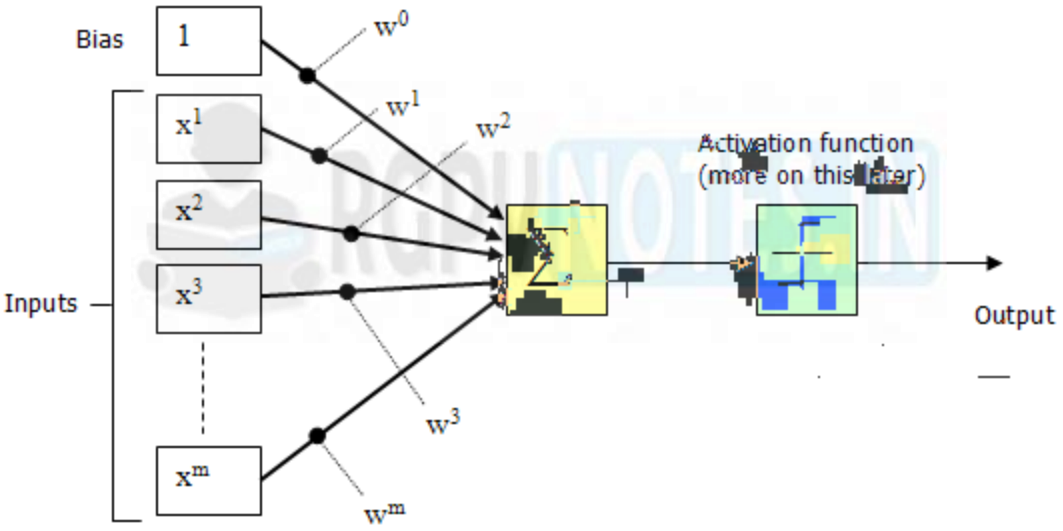


Figure 2.2: Perceptron Network

In above figure, there are n input neurons, 1 output neuron and a bias. The input-layer and output-layer neurons are connected through a directed communication link, which is associated with weights. The goal of the perceptron net is to classify the input pattern as a member or not a member to a particular class.

Perceptron Training Algorithm for Single Output Class:

The perceptron algorithm can be used for either binary input vectors, having bipolar targets, threshold being fixed or variable bias. In the algorithm, initially the inputs are assigned, then the net input is calculated. The output of the network is obtained by applying the activation function over the calculated net input. On performing comparison over the calculated and the

desired output, the weight updation process is carried out. The entire network is trained based on the mentioned stopping criteria.
The algorithm of a perceptron network is as follows:
Step 0: Initialize the weights and the bias (for easy calculation they can be set to zero). Also initialize the learning rate α (0<α<=1). For simplicity α is set to 1.

Step 1: Perform steps 2-6 until the final stopping condition is false.
Step 2: Perform steps 3-5 for each training pair indicated by s:t.
Step 3: The input layer containing input units is applied with identity activation function:
$x_i = s_i$
Step 4: Calculate the output of the network. To do so, first obtain the net input:

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

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

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

Step 5: Weight and bias adjustment: Compare the value of the actual (calculated) output and desired (target) output.
If y != t, then
$w_i(new) = w_i(old) + \alpha t x_i$
$b(new) = b(old) + \alpha t$
else, we have
$w_i(new) = w_i(old)$
$b(new) = b(old)$
Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again form step 2.

Perceptron Training Algorithm for Multiple Output Class:
For multiple output classes, the perceptron training algorithm is as follows:
Step 0: Initialize the weights, biases and learning rate suitably.
Step 1: Check for stopping condition; if it is false, perform step 2-6.
Step 2: Perform step 3-5 for each bipolar or binary training vector pair s:t.
Step 3: Set activation (identity) of each input unit i=1 to n:
Xi = si
Step 4: Calculate output response of each output unit j=1 to m: First, the net input is calculated as

$$Y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

The activation are applied over the net input to calculate the output response:

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

Step 5: Make adjustment in weight and bias for j=1 to m and i=1 to n
If t_j != y_j, then
w_{ij} (new) = w_{ij} (old) + $\alpha t_j x_i$
b_j (new) = b_j (old) + αt_j
else, we have
w_{ij} (new) = w_{ij} (old)
b_j (new) = b_j (old)
Step 6: Test for the stopping condition, if there is no change in weights then stop the training process, else start again from step 2.

Single Layer / Multilayer Perceptron
Single layer perceptron is the first proposed neural model created. The content of the local memory of the neuron consists of a vector of weights. The computation of a single layer perceptron is performed over the calculation of sum of the input vector each with the value multiplied by corresponding element of vector of the weights. The value which is displayed in the output will be the input of an activation function.
In the Multilayer perceptron, there can more than one linear layer (combinations of neurons). If we take the simple example the three-layer network, first layer will be the input layer and last will be output layer and middle layer will be called hidden layer. We feed our input data into the input layer and take the output from the output layer. We can increase the number of the hidden layer as much as we want, to make the model more complex according to our task.

Linear Separability:
Consider two-input patterns (X1, X2) being classified into two classes as shown in figure. Each point with either symbol of x or o represents a pattern with a set of values (X1, X2). Each pattern is classified into one of two classes. Notice that these classes can be separated with a single line L. They are known as linearly separable patterns. Linear separability refers to the fact that classes of patterns with n-dimensional vector {x} = (x1, x2, …, xn) can be separated with a single decision surface. In the case above, the line L represents the decision surface.

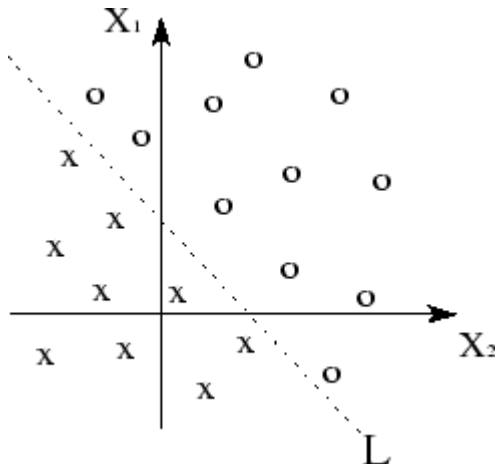


Figure 2.3: Linear Separability

The processing unit of a single-layer perceptron network is able to categorize a set of patterns into two classes as the linear threshold function defines their linear separability. Conversely, the two classes must be linearly separable in order for the perceptron network to function correctly. Indeed, this is the main limitation of a single-layer perceptron network.

The most classic example of linearly inseparable pattern is a logical exclusive-OR (XOR) function; is the illustration of XOR function that two classes, 0 for black dot and 1 for white dot, cannot be separated with a single line. The solution seems that patterns of (X1,X2) can be logically classified with two lines L1 and L2.

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = X_1 \oplus X_2$

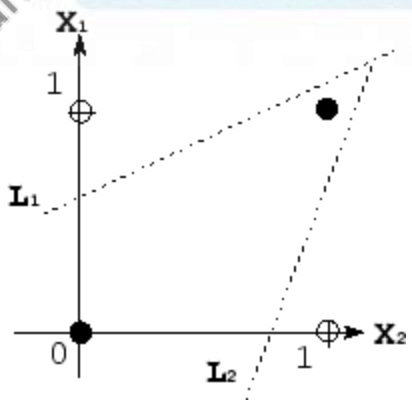


Figure 2.4: Example of Linear Separability

Adaptive Linear Neuron (Adaline):

The units with linear activation function are called linear units. A network with a single linear unit is called an Adaline. In an Adaline, the input-output relationship is linear. Adaline uses bipolar activation for its input signals and its target output. The Adaline network may be trained using delta rule. The delta rule also be called as least mean square (LMS) rule or Widrow-Hoff rule.

Architecture of Adaline:

The basic Adaline model consists of trainable weights. Inputs are either of the two values (+1 or -1) and the weights have sign (positive or negative). Initially, random weights are assigned. The net input calculated is applied to a quantizer transfer function that restores the output to +1 or -1. The Adaline model compares the actual output with the target output and on the basis of the training algorithm, the weights are adjusted.

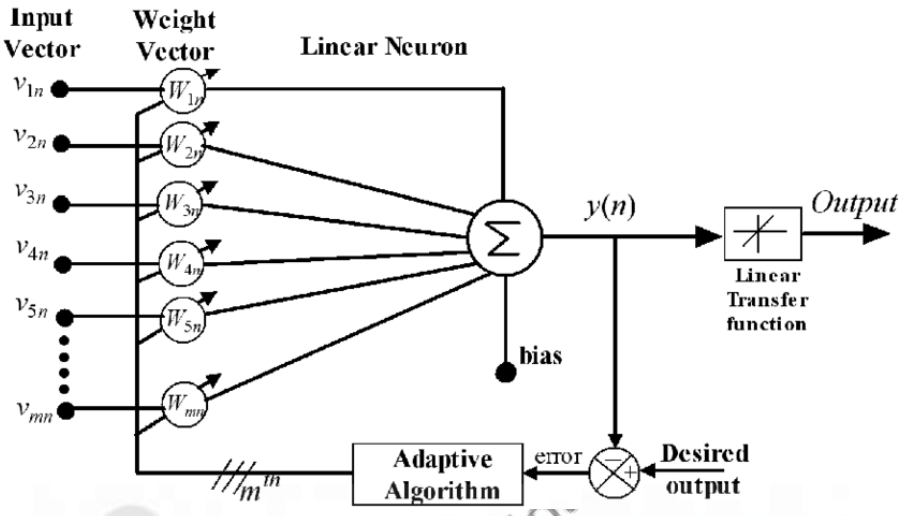


Figure 2.5: Architecture of Adaline

Training Algorithm:

The Adaline network training algorithm is as follows:

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

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

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

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

Step4: Calculate the net input to the output unit.

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

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

Testing Algorithm:

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

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

Step 1: Perform steps 2-4 for each bipolar input vector x.

Step 2: Set the activations of the input units to x.

Step 3: Calculate the net input to the output units

Step 4: Apply the activation function over the net input calculated.

Multiple Adaptive Linear Neurons (Madaline):
It consists of many Adaline in parallel with a single output unit whose value is based on certain selection rules. It uses the majority vote rule. On using this rule, the output unit would have an answer either true or false. On the other hand, if AND rule is used, the output is true if and only if both the inputs are true and so on. The training process of Madaline is similar to that of Adaline

Architecture:
It consists of "n" units of input layer and "m" units of Adaline layer and "1"Unit of the Madaline layer. Each neuron in the Adaline and Madaline layers has a bias of excitation "1". The Adaline layer is present between the input layer and the Madaline layer; the Adaline layer is considered as the hidden layer. The use of hidden layer gives the net computational capability which is not found in the single-layer nets, but this complicates the training process to some extent.

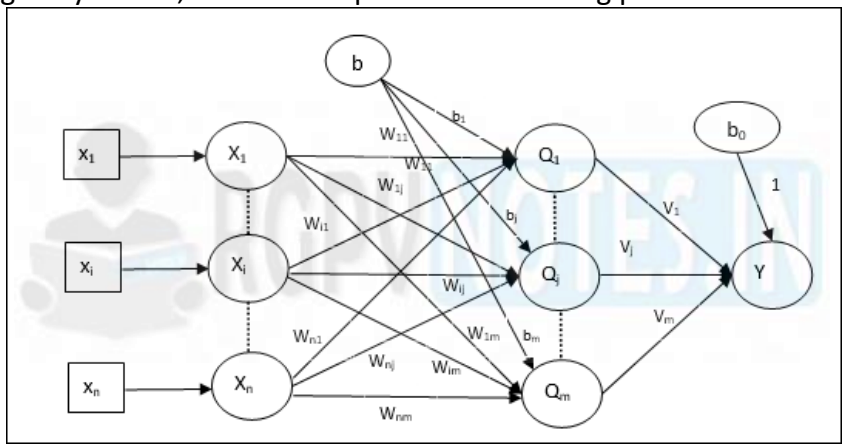


Figure 2.6: Architecture of Madaline

Training Algorithm:
In this training algorithm, only the weights between the hidden layers are adjusted, and the weights for the output units are fixed. The weights v1, v2……vn and the bias b0 that enter into output unit Y are determined so that the response of unit Y is 1.
Thus the weights entering Y unit may be taken as

$$v1=v2=….=vn=1/2$$

And the bias can be taken as

$$b_0 = 1/2$$

Step 0: Initialize the weights. The weights entering the output unit are set as above. Set initial small random valued for Adaline weights. Also set initial learning rate α.
Step 1: When stopping condition is false, perform steps 2-3.
Step 2: For each bipolar training pair s:t, perform steps 3-7.
Step 3: Activate input layer units, For i=1 to n.

$$xi = si$$

Step 4: Calculate net input to each hidden Adaline unit:

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

Step 5: Calculate output of each hidden unit:

$$z_j = f(z_{inj})$$

Step 6: Find the output of the net:

$$y_{in} = b_0 + \sum_{j=1}^{m} z_j v_j$$

Step 7: Calculate the error and update the weights.

1. If t = y, no weight updation is required.
2. If t != y and t =+1, update weights in zj, where net input is closest to 0 (zero):

$$b_j(new) = {}_{bj}(old) + \alpha\,(1-z_{inj})$$

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

3. If t != y and t =-1, update weights on units zk whose net input is positive:

$$w_{ik}(new) = w_{ik}(old) + \alpha(-1-z_{ink})x_i$$

$$b_k(new) = b_k(old) + \alpha(-1-z_{ink})$$

Step 8: Test for the stopping condition. (If there is no weight change or weight reaches a satisfactory level, or if a specified maximum number of iterations of weights updation have been performed then stop, or else continue).

Back Propagation Network:

Back propagation learning algorithm is one of the most important developments in neural networks. This learning algorithm is applied to multilayer feed-forward networks consisting of processing elements with continuous differentiable activation functions. The network associated with back propagation learning algorithm is called back-propagation networks (BPNs).

Architecture:

A back propagation neural network is a multilayer, feed-forward neural network consisting of an input layer, a hidden layer and an output layer. The neurons present in the hidden and output layers have biases, which are the connections from the units whose activation is always 1. The bias terms also acts as weights.

The inputs are sent to the BPN and the output obtained from the net could be either binary (0,1) or bipolar (-1, +1). The activation function could be any function which increases monotonically and is also differentiable.

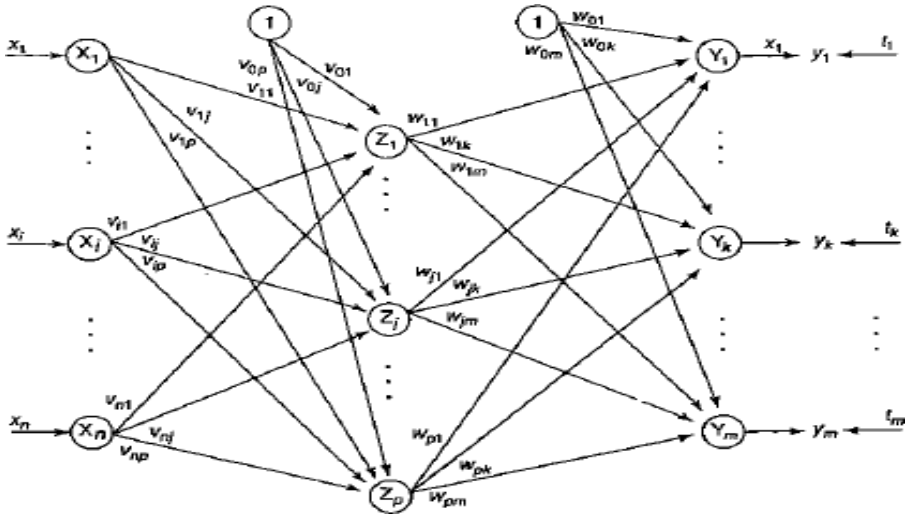


Figure 2.7: Architecture of Back Propagation Network

Training Algorithm:

Step 0: Initialize weights and learning rate (take some small random values).

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

Step 2: Perform Steps 3-8 for training pair.

Feed-forward phase (Phase-I):

Step 3: Each input unit receives input signal x; and sends it to the hidden unit (i = l to n}.

Step 4: Each hidden unit Zj (j = 1 to p) sums its weighted input signals to calculate net input:

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

Calculate output of the hidden unit by applying its activation functions over Zinj (binary or bipolar sigmoidal activation function}: -

$$z_j = f(z_{inj})$$

and send the output signal from the hidden unit to the input of output layer units.

Step 5: For each output unity$_k$(k = I to m), calculate the net input:

$$y_{ink} = w_{ok} + \sum_{j=1}^{p} z_i w_{jk}$$

and apply the activation function to compute output signal

$$y_k = f(y_{ink})$$

Back-propagation of error (Phase-II):

Step 6: Each output unit y_k(k = I to m) receives a target pattern corresponding to the input training pattern and computes the error correction term:

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

The derivative f(y_{ink}) can be calculated. On the basis of the calculated error correction term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j;$$

$$\Delta w_{0k} = \alpha \delta_k;$$

Also, send δ_k to the hidden layer backwards.

Step 7: Each hidden unit (zj,j = I top) sums its delta inputs from the output units:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

The term δ_{inj} gets multiplied with the derivative of $f(Z_{inj})$ to calculate the error term:

$$\delta_j = \delta_{inj}\, f'(z_{inj})$$

The derivative $f'(z_{inj})$ can be calculated. Depending on whether binary or bipolar sigmoidal function is used. On the basis of the calculated δ_j, update the change in weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i;$$

$$\Delta v_{0j} = \alpha \delta_j;$$

Weight and bias updation (Phase-III):

Step 8: Each output unit (yk, k=1 to m) updates the bias and weights:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

$$w_{ok}(new) = w_{ok}(old) + \Delta w_{ok}$$

Each hidden unit (zj, j= 1 to p) updates its bias and weights:

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$

$$v_{oj}(new) = v_{oj}(old) + \Delta v_{oj}$$

Step 9: Check for the stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equals the target output.

Radial Basis Function Network:

The radial basis function (RBF) is a classification and functional approximation neural network developed

by M.J.D. Powell. The network uses the most common nonlinearities such as sigmoidal and Gaussian kernel functions. The Gaussian functions are also used in regularization networks. The response of such a function is positive for all values of y; the response decreases to 0 as lyl→ 0. The Gaussian function is generally defined as

$$f(y) = e^{-y2}$$

The derivative of this function is given by:

$$f'(y) = -2ye^{-y2} = -2yf(y)$$

Architecture:

The architecture consist of two layers whose output nodes form a linear combination of the kernel (or basis) functions computed by means of the RBF nodes or hidden layer nodes. The basis function (nonlinearity) in the hidden layer produces a significant nonzero response w the input stimulus it has received only when the input of it falls within a small localized region of the input space. This network can also be called as localized receptive field network.

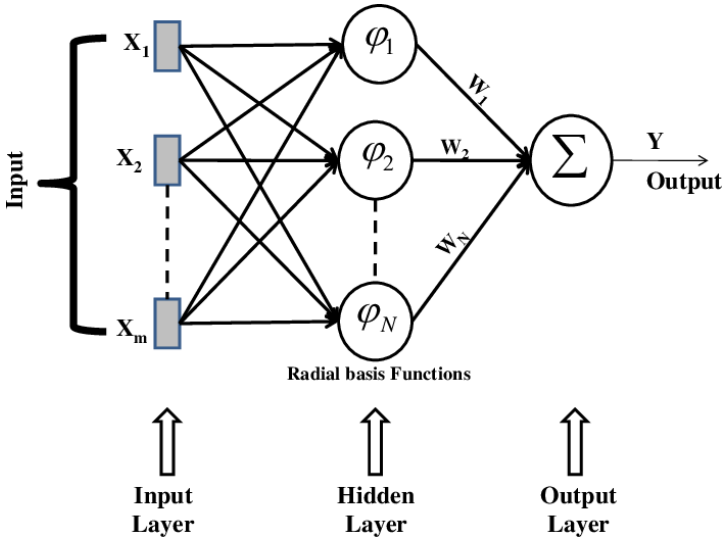


Figure 2.8: Architecture of RBF

Training Algorithm:

Step 0: Set the weight to small random values.

Step 1: Perform steps 2-8 when the stopping condition is false.

Step 2: Perform steps 3-7 for each input.

Step 3: Each input unit (xi for all I = 1 to n) receives input signals and transmits to the next hidden layer unit.

Step 4: Calculate the radial basis function.

Step 5: Select the centers for the radial basis function. The centers are selected from the set of input vectors. It should be noted that a sufficient number of centers have to be selected to ensure adequate sampling of the input vector space.

Step 6: Calculate the output from the hidden layer unit:

$$v_i(x_i) = \frac{\exp\left[-\sum_{j=1}^{r}(x_{ji} - \hat{x}_{ji})^2\right]}{\sigma_i^2}$$

^

where $x_{ji}$is the center of the RBF unit for input variable; $\sigma_i$the width of ith RBF unit; $x_{ji}$the jth variable of input pattern.

Step 7: Calculate the output of the neural network:

$$y_{net} = \sum_{i=1}^{k} w_{im} v_i(x_i) + w_0$$

where k is the number of hidden layer nodes (RBF function);y_{net} the output value of mth node in output layer for the nth incoming pattern; $W_{im}$the weight between ith RBF unit and mth output node; W_o the biasing term at nth output node.

Step 8: Calculate the error and test for the stopping condition. The stopping condition may be number of epochs or to a certain extent weight change.

Application of Neural Network:

Forecasting:

Load forecasting is very essential to the operation of electricity companies. It enhances the energy-efficient and reliable operation of a power system. The inputs used for the neural network are the previous hour load, previous day load, previous week load, day of the week, and hour of the day. The neural network used has 3 layers: an input, a hidden, and an output layer. The input layer has 5 neurons, the number of hidden layer neurons can be varied for the different performance of the network, while the output layer has a single neuron.

Many methods have been used for load forecasting in the past. These include statistical methods such as regression and similar-day approach, fuzzy logic, expert systems, support vector machines, econometric models, end-use models, etc. A supervised artificial neural network has also been used for the same. The neural network is trained on input data as well as the associated target values. The trained network can then make predictions based on the relationships learned during training.

Data Compression:

The transport of data across communication paths is an expensive process. Data compression provides an option for reducing the number of characters or bits in transmission. It has become increasingly important to most computer networks, as the volume of data traffic has begun to exceed their capacity for transmission. Artificial Neural Network (ANN) based techniques provide other means for the compression of data at the transmitting side and decompression at the receiving side. The security of the data can be obtained along the communication path as it is not in its original form on the communication line. Artificial Neural Networks have been applied to many problems, and have demonstrated their superiority over classical methods when dealing with noisy or incomplete data. One such application is for data compression. Neural networks seem to be well suited to this particular function, as they have an ability to preprocess input patterns to produce simpler patterns with fewer components. This compressed information (stored in a hidden layer) preserves the full information obtained from the external environment. The compressed features may then exit the network into the external environment in their original uncompressed form.

Image Compression:

Neural networks can accept a vast array of input at once, and process it quickly, they are useful in image compression.

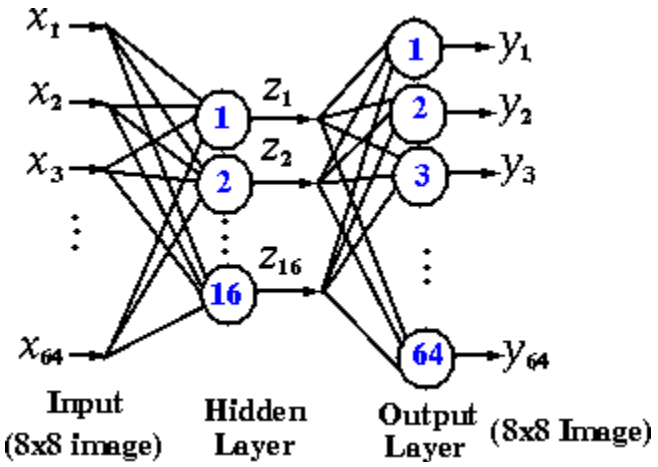


Figure 2.9: Bottleneck-type Neural Net Architecture for Image Compression

Here is a neural net architecture suitable for solving the image compression problem. This type of structure is referred to as a bottleneck type network, and consists of an input layer and an output layer of equal sizes, with an intermediate layer of smaller size in-between. The ratio of the size of the input layer to the size of the intermediate layer is - of course - the compression ratio.

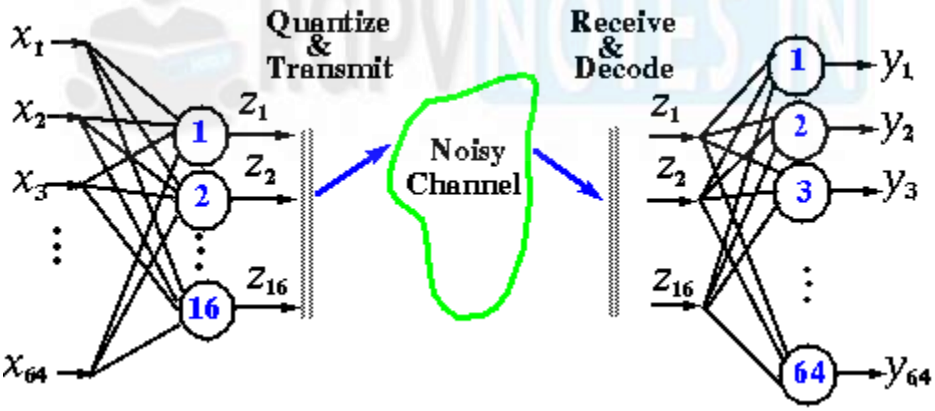


Figure 2.10: Bottleneck architecture for image compression over a network or over time

This is the same image compression scheme, but implemented over a network. The transmitter encodes and then transmits the output of the hidden layer, and the receiver receives and decodes the 16 hidden outputs to generate 64 outputs.

Pixels, which consist of 8 bits, are fed into each input node, and the hope is that the same pixels will be outputted after the compression has occurred. That is, we want the network to perform the identity function.

Actually, even though the bottleneck takes us from 64 nodes down to 16 nodes, no real compression has occurred. The 64 original inputs are 8-bit pixel values. The outputs of the hidden layer are, however, decimal values between -1 and 1. These decimal values can require a possibly infinite number of bits.

Subject Name: Soft Computing **Subject Code: IT 701**

Subject Notes

Syllabus:

Unsupervised learning: Introduction, Fixed weight competitive nets, Kohonen SOM, Counter Propagation networks, (Theory, Architecture, Flow Chart, Training Algorithm and applications). Introduction to Convolutional neural networks (CNN) and Recurrent neural networks (RNN).

Unit-3

Course Objectives

1. The objective of this course is to understand Unsupervised learning and its type.
2. To help student to understand Convolutional neural network and Recurrent neural network.

Unsupervised Learning: Introduction

Unsupervised learning is where you only have input data (X) and no corresponding output variables.

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

In ANNs following unsupervised learning, the input vectors of similar type are grouped without the use of training data to specify how a member of each group looks or to which group a number belongs. In the training process, the network receives the input patterns and organizes these patterns to form clusters. When a new input pattern is applied, the neural network gives an output response indicating the class to which the input pattern belongs.If for an input, a pattern class cannot be found then a new class is generated.

These are called unsupervised learning because unlike supervised learning above there is no correct answer and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

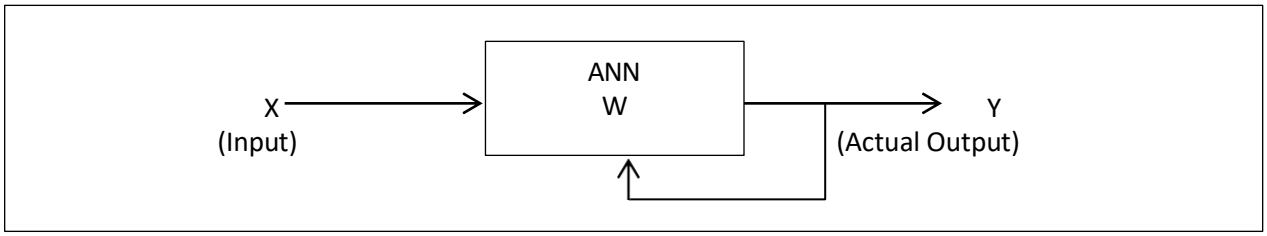


Figure 3.1: Unsupervised Learning

From the working of unsupervised learning it is clear that there is no feedback from the environment to inform what the outputs should be or whether the outputs are correct. In this case the network must itself discover patterns, regularities, features or categories from the input data and relations for the input data over the output.

Fixed weight competitive nets

Many neural nets use the idea of competition among neurons to enhance the contrast in activations of the neurons. In the most extreme situation, often called Winner-Take-All, only the neuron with the largest activation is allowed to remain "on." Typically. The neural implementation of this competition is not specified (and in computer simulations, the same effect can be achieved by a simple. non-neural sorting process). **A Hamming network** is a fixed-weight competitive ANN where the lower network feeds into a competitive network called maxnet.

Hamming Network

In most of the neural networks using unsupervised learning, it is essential to compute the distance and perform comparisons. This kind of network is Hamming network, where for every given input vectors, it would be clustered into different groups. Following are some important features of Hamming Networks −

1. Lippmann started working on Hamming networks in 1987.
2. It is a single layer network.
3. The inputs can be either binary {0, 1} of bipolar {-1, 1}.
4. The weights of the net are calculated by the exemplar vectors.
5. It is a fixed weight network which means the weights would remain the same even during training.

Max Net

This is also a fixed weight network, which serves as a subnet for selecting the node having the highest input. All the nodes are fully interconnected and there exists symmetrical weights in all these weighted interconnections.

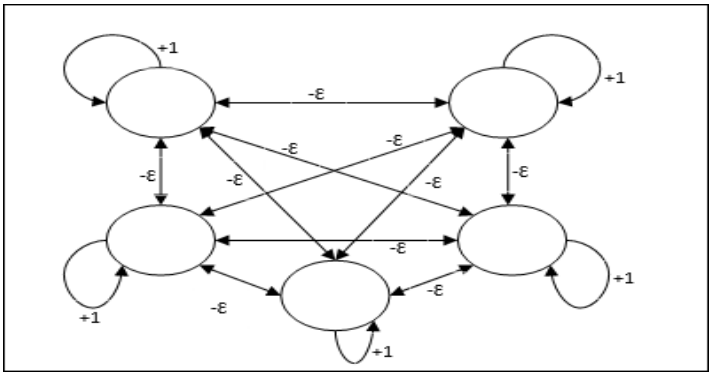


Figure 3.2: Architecture of Max Net

It uses the mechanism which is an iterative process and each node receives inhibitory inputs from all other nodes through connections. The single node whose value is maximum would be active or winner and the activations of all other nodes would be inactive. Max Net uses identity activation function with:

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

The task of this net is accomplished by the self-excitation weight of +1 and mutual inhibition magnitude, which is set like [0 < ε < 1/m] where **"m"** is the total number of the nodes.

Competitive Learning in ANN

It is concerned with unsupervised training in which the output nodes try to compete with each other to represent the input pattern. To understand this learning rule we will have to understand competitive net which is explained as follows:

Basic Concept of Competitive Network

This network is just like a single layer feed-forward network having feedback connection between the outputs. The connections between the outputs are inhibitory type, which is shown by dotted lines, which means the competitors never support themselves.

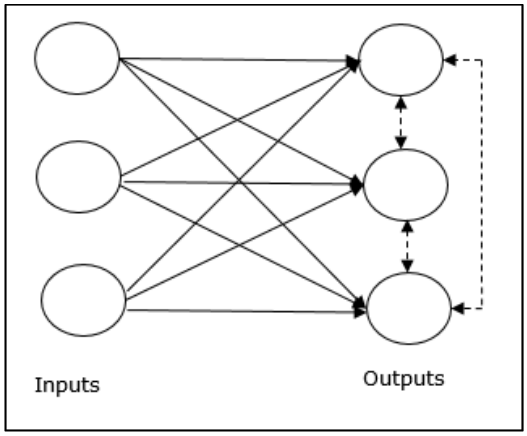


Figure 3.3: Single layer feed-forward network

Basic Concept of Competitive Learning Rule

As said earlier, there would be competition among the output nodes so the main concept isduring training, the output unit that has the highest activation to a given input pattern, will be declared the winner. This rule is also called Winner-takes-all because only the winning neuron is updated and the rest of the neurons are left unchanged.

Mathematical Formulation

Following are the three important factors for mathematical formulation of this learning rule –

- Condition to be a winner

  Suppose if a neuron y_k wants to be the winner, then there would be the following condition

$$y_k = \begin{cases} 1 & if\ v_k > v_j\ for\ all\ \ j,\ j \neq k \\ 0 & otherwise \end{cases}$$

  It means that if any neuron, say, y_k wants to win, then its induced local field the output of the summation unit say v_k, must be the largest among all the other neurons in the network.

- Condition of the sum total of weight

  Another constraint over the competitive learning rule is the sum total of weights to a particular output neuron is going to be 1. For example, if we consider neuron **k** then

$$\sum_k w_{kj} = 1 \quad for\ all\ k$$

- Change of weight for the winner

  If a neuron does not respond to the input pattern, then no learning takes place in that neuron. However, if a particular neuron wins, then the corresponding weights are adjusted as follows –

$$\Delta w_{kj} = \begin{cases} -\alpha(x_j - w_{kj}), & if\ neuron\ k\ wins \\ 0 & if\ neuron\ k\ losses \end{cases}$$

  Here α is the learning rate.

  This clearly shows that we are favoring the winning neuron by adjusting its weight and if a neuron is lost, then we need not bother to re-adjust its weight.

Kohonen Self-Organizing Map(Theory, Architecture, Flow Chart, Training Algorithm):

The Self-Organizing Map is one of the most popular neural network models. It belongs to the category of competitive learning networks. The Self-Organizing Map is based on unsupervised learning, which means that no human intervention is needed during the learning and that little needs to be known about the characteristics of the input data. The SOM can be used to detect features inherent to the problem and thus has also been called SOFM, the Self-Organizing Feature Map.

The Self-Organizing Map was developed by professor Kohonen. The SOM has been proven useful in many applications. The SOM algorithm is based on unsupervised, competitive learning. It provides a topology preserving mapping from the high dimensional space to map units. Map units, or neurons, usually form a two-dimensional lattice and thus the mapping is a mapping from high dimensional space onto a plane. The property of topology preserving means that the mapping preserves the relative distance between the points. Points that are near each other in the input space are mapped to nearby map units in the SOM. The SOM can thus serve as a cluster analyzing tool of high-dimensional data. Also, the SOM has the capability to generalize. Generalization capability means that the network can recognize or characterize inputs it has never encountered before. A new input is assimilated with the map unit it is mapped to.

Architecture of Kohonen SOM:

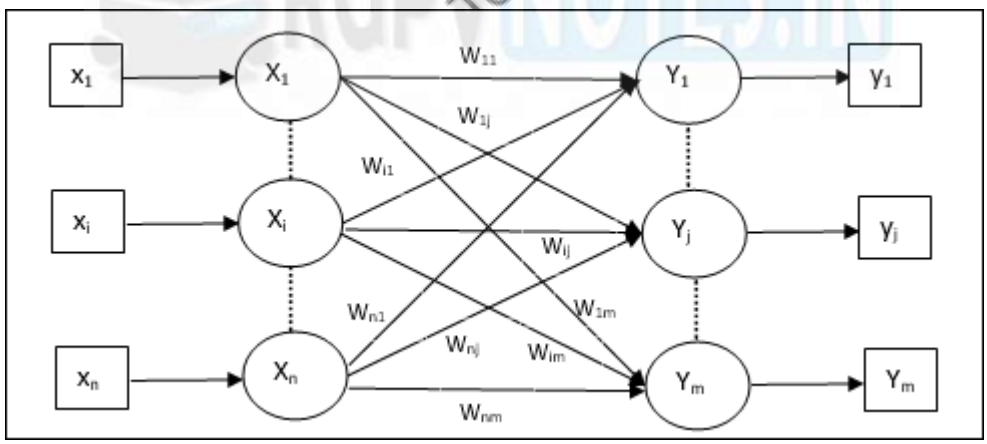


Figure 3.4: Architecture of Kohonen SOM

The architecture consists of two layers: input layer and output layer (cluster). There are “n” units in the input layer and “m” units in the output layer. Basically, here the winner unit is identified by using either dot product or Euclidean distance method and the weight updation using Kohonen learning rules is performed over the winning cluster unit.

Flowchart:

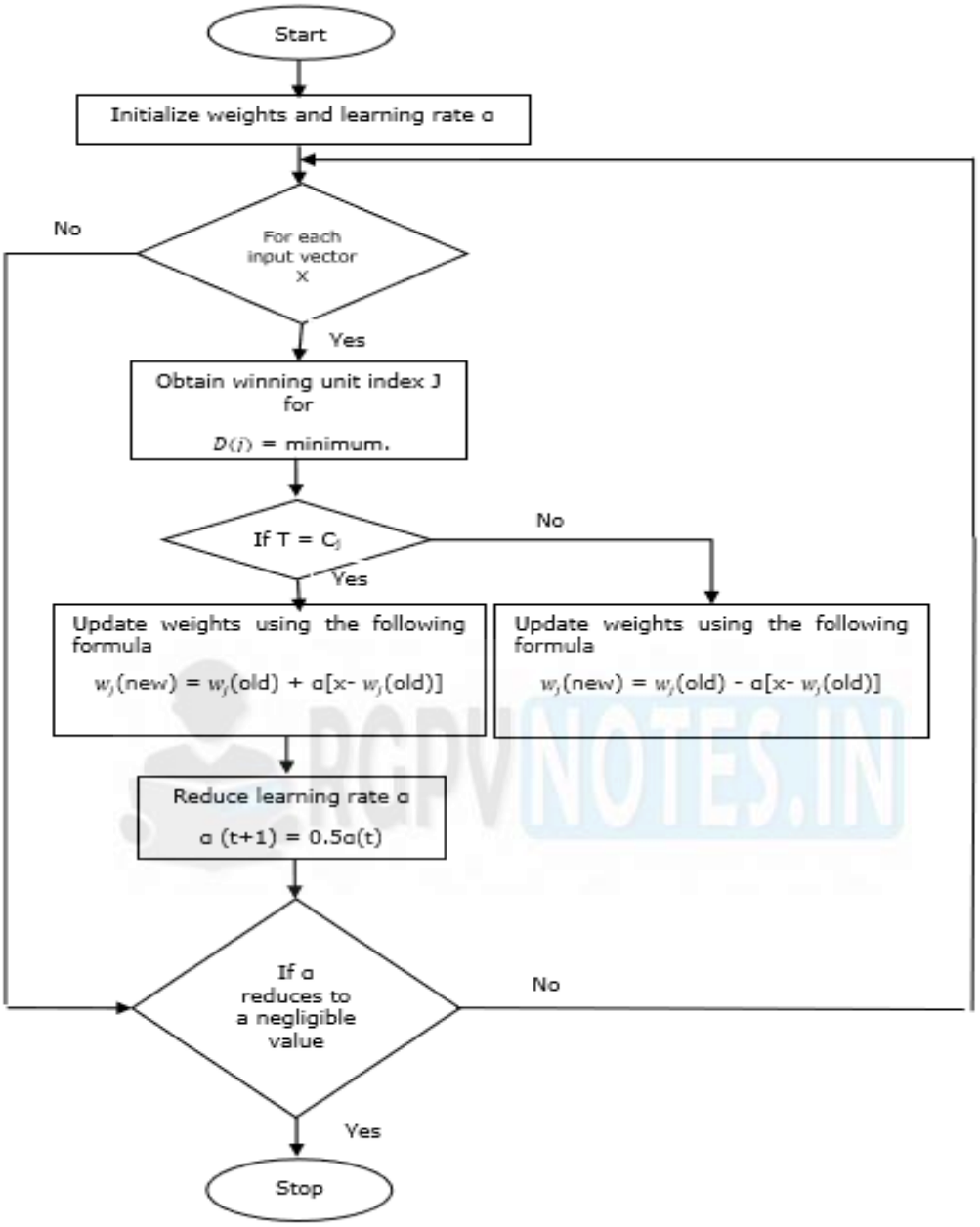


Figure 3.5: Flowchart of Kohonen SOM

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 α

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**

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

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

$$w_j(new) = w_j(old) + \alpha[x - w_j(old)]$$

if **T ≠ C**$_j$ then

$$w_j(new) = w_j(old) - \alpha[x - w_j(old)]$$

Step 8 − Reduce the learning rate α.

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

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

Counter Propagation Network:

Counter propagation networks were proposed by Hecht Nielsen in 1987. They are multilayer networks based on the combinations of the input, output and clustering layers. The application of counter propagation nets are data compression, function approximation and pattern association. The counterpropagation network is basically constructed from an instar-outstar model. This model is a three layer neural network that performs input-output data mapping producing an output vector y in response to an input vector x, on the basis of competitive learning.

A counterpropagation net is an approximation of its training input vector pairs by adaptively constructing a look-up-table. By this method, several data points can be compressed to a more manageablenumber of look-up-table entries. The accuracy of the function approximation and data

compression is basedon the number of entries in the look-up-table, which equals the number of units in the cluster layer of the net.

There are two stages involved in the training process of a counterpropagation net. The input vectors areclustered in the first stage. Originally, tr is assumed that there is no topology included in the counter propagation network. However, on the inclusion of a linear topology,

the performance of the net can be improved. The clusters are performed using Euclidean distance method or dot product method. In the second stageof training, the weights from the cluster layer units to the output units are tuned to obtain the desired response.
There are twotypesofcounterpropagation nets: (i) Fullcounterpropagation net and (ii) forward-onlycounterpropagation net.

Flow Chart:

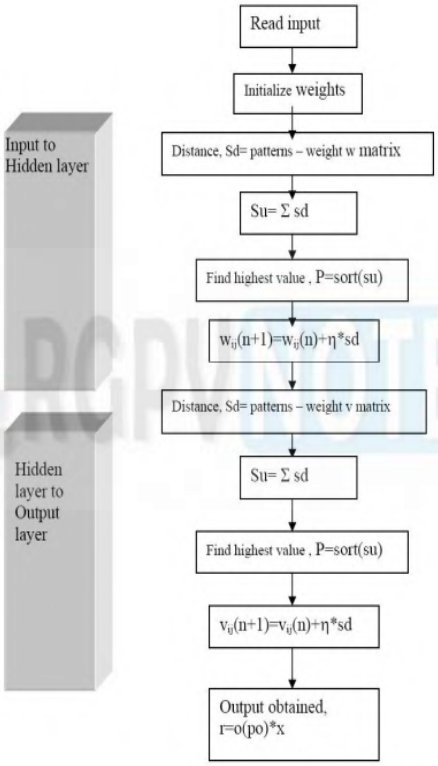


Figure 3.6 Flowchart of CPN

This figure shows three layers for CPN; an input layer that reads input patterns from the training set and forwards them to the network, a hidden layer that works in a competitive fashion and associates each input pattern with one of the hidden units and the output layer which is trained via a teaching algorithm that tries to minimize the mean square error (MSE) between the actual network output and the desired output associated with the current input vector. In some cases, a fourth layer is used to normalize the input vectors, but this normalization can be easily performed by the application before these vectors are sent to the Kohonen layer. Regarding the training process of the counter-propagation network, it can be described as a two-stage procedure; in the first stage, the process updates the weights of the synapses between the input and the Kohonen layer, while in the second stage the weights of the synapses between the Kohonen and the Grossberg layer are updated.

Applications:

The applications of counter propagation network are:

1. Datacompression specially for Image and audio
2. Function approximation
3. Pattern association

Full Counterpropagation Net:

Full counterpropagarion net (full CPN) efficiently represents a large number of vector pairs x:y by adaptivelyconstructing a look-up-table. The approximation here is x*:y*, which is based on the vector pairs x:y, possiblywith some distorted or missing elements in either vector or both vectors. The nerwork is defined to approximatea continuous function f, defined on a compact set A. The full CPN works best if the inverse function exists and is continuous.

The vectors x and y propagate through the network in a counterflow manner toyield output vectors x* and y*, which are the approximations of x andy, respectively. During competition,the winner can be determined either by Euclidean distance or by dot product method. In case of dot productmethod, the one with the largest net input is the winner. Whenever vectors are to be compared using thedot product metric, they should be normalized. Even though the normalization can be performed withoutloss of information by adding an extra component, yet to avoid the complexicy Euclidean distance methodcan be used. On the basis of this, direct comparison can be made between the full CPN and forward-onlyCPN.

For continuous function, the CPN is efficient as the back-propagation net; it is a universal continuousfunction approximator. In case of CPN, the number of hidden nodes required to achieve a particular levelof accuracy is greater than the number required by the back-propagation network. The greatest appeal ofCPN is its speed of learning. Compared to various mapping networks, ir requires only fewer steps of trainingto achieve best performance. This is common for any hybrid learning method that combines unsupervisedlearning (e.g., instar learning) and supervised learning (e.g., outsrar learning).

As already discussed, the training ofCPN occurs in two phases. In the input phase, the units in the clusterlayer and input layer are found to be active. In CPN, no topology is assumed for the cluster layer units; onlythe winning units are allowed to learn. The weight updarion learning rule on the winning cluster units is

$$V_{ij}(new) = V_{ij}(old) + \alpha [x_i - V_{ij}(old)], \quad i=1 \text{ to } n$$

$$W_{kj}(new) = W_{kj}(old) + \beta (y_k - W_{kj}(old)], \quad k=1 \text{ to } n$$

The above is standard Kohonen learning which consists of competition among the units and selection of

winner unit. The weight updation is performed for the winning unit.

Architecture:

The four major components of rhe instar-outstar model are the input layer, the instar, the competitive layerand the outstar. For each node i in the input layer, there is an input value xi. An

instar responds maximally tothe input vectors from a particular cluster. All the instars are grouped into a layer called the competitive layer.Each of the instar responds maximally to a group of input vectors in a different region of space. This layer ofinstars classifies any input vector because, for a given input, the winning instar with the strongest responseidentifies the region of space in which the input vector lies. Hence, it is necessary that the competitive layersingle outs the winning instar by setting its output ro a nonzero value and also suppressing the other outputs to zero. That is, it is a winner-take-all or a Maxnet-type network. An outstar model is found to have all thenodes in the output layer and a single node in the competitive layer. The outstar looks like the fan-out ofa node.

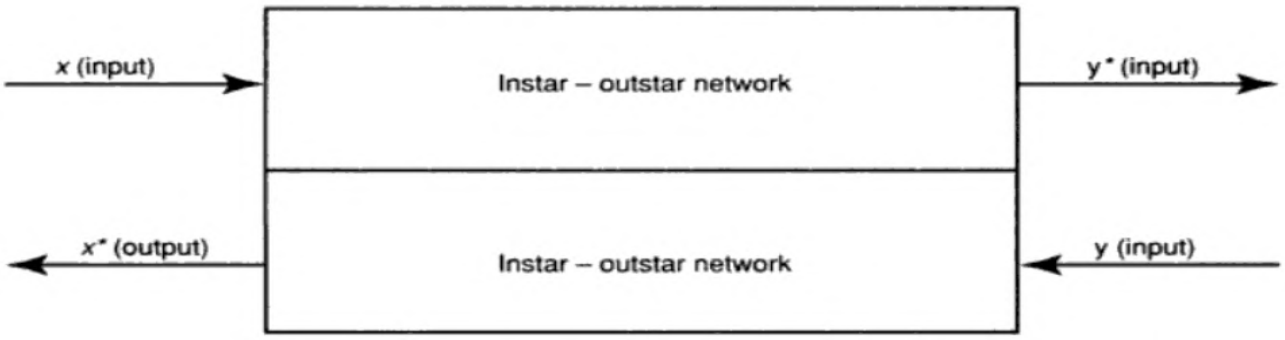


Figure 3.7: General Architecture of Full CPN

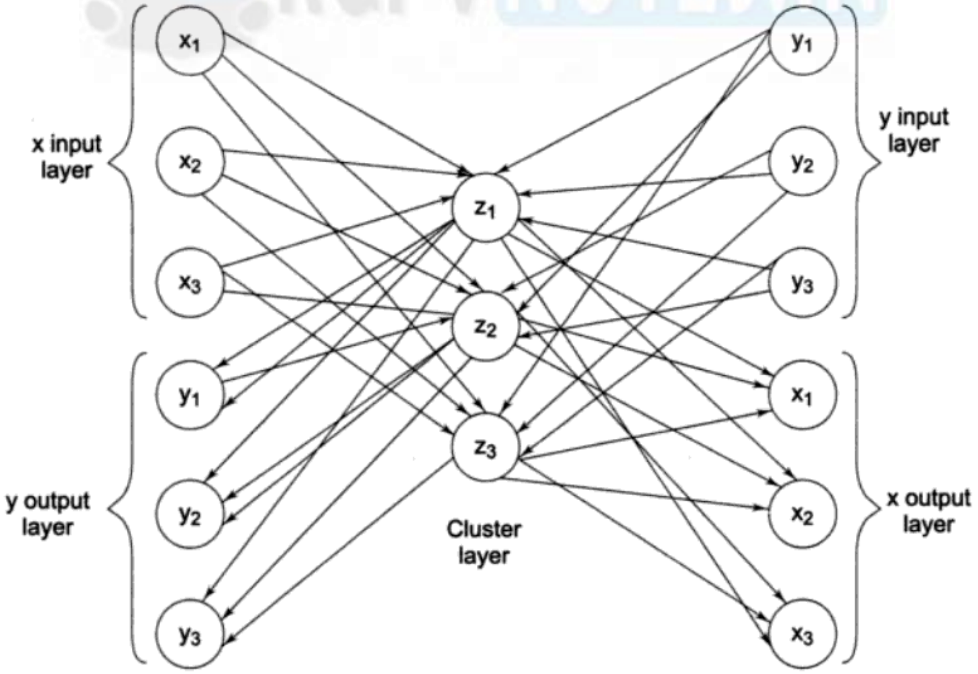


Figure 3.8: Architecture of Full CPN

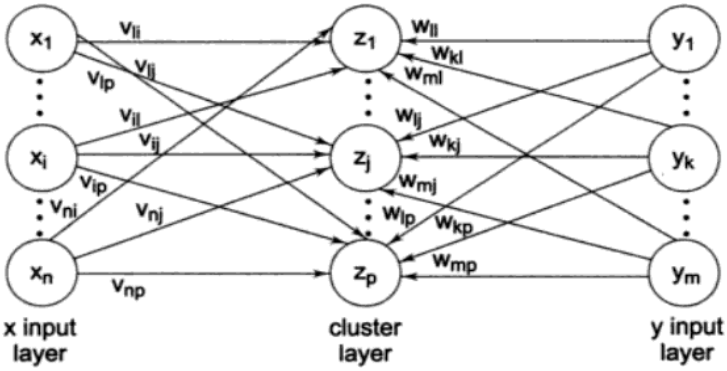


Figure 3.9: First Phase of Training of Full CPN

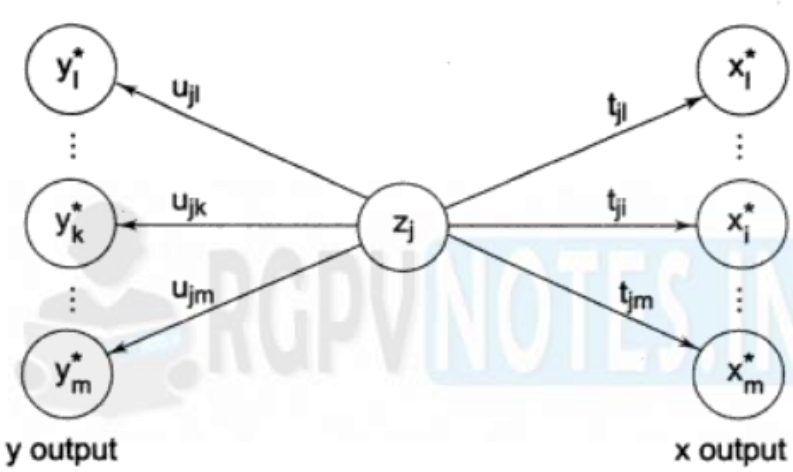


Figure 3.10: Second Phase of Training of Full CPN

Forward-Only Counterpropagation Network:

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

In case of forward-only CPN, first input vectors are presented to the input units. The cluster layer unitscompete with each other using winner-take-all policy to learn the input vector. Once entire set of trainingvecrors has been presented, there exist reduction in learning rate and the vectors are presented again, performingseveral iterations. First, the weights between the input layer and cluster layer are trained. Then the weightsbetween the cluster layer and output layer are trained. This is a specific competitive network, with targetknown. Hence, when each input vector is presented to the

input vector, its associated target vectors arepresented to the output layer. The winning cluster unit sends its signal to the output layer. Thus each ofthe output unit has a computed signal (Wjk) and the

target value {Yk}. The difference between these values iscalculated; based on this, the weights between the winning layer and output layer are updated.

Architecture:

It consists of three layers: input layer, cluster(competitive) layer and output layer. The architecture of forward-only CPN resembles the back-propagationnetwork, but in CPN there exists interconnections becween the units in the cluster layer (which are not connected). Once competition is completed in a forward-only CPN, only one unit will beactive in that layer and it sends signal to the output layer. As inputs are presented to the network, the desiredoutputs will also be presented simultaneously.

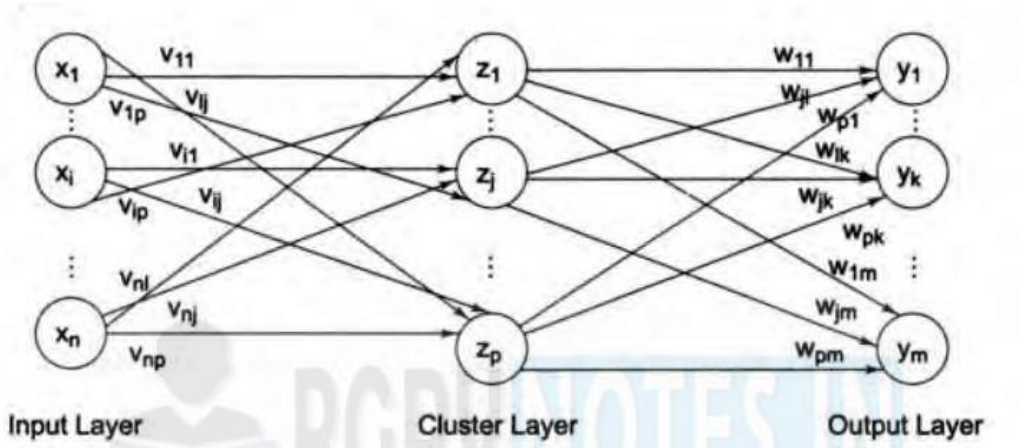


Figure 3.11: Architecture of Forward Only CPN

Introduction to Convolutional neural networks (CNN)

A convolutional neural network (CNN) is a particular implementation of a neural network used in machine learning that exclusively processes array data such as images, and is frequently used in machine learning applications targeted at medical images. For image classification we use Convolution Neural Network.CNN is a neural network with a special structure that was designed as a model of a human vision system (HVS). Thus, CNNs are most suitable for solving problems of computer vision, such as object recognition and classification of images and video data. They have also been used successfully for speech recognition and text translation.

Convolution Neural Networks or covnets are neural networks that share their parameters.It can be represented as a cuboid having its length, width (dimension of the image) and height (as image generally have red, green, and blue channels).Now imagine taking a small patch of this image and running a small neural network on it, with say, k outputs and represent them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different width, height, and depth. Instead of just R, G and B channels now we have more channels but lesser width and height, this operation is called Convolution. If patch size is same as that of the image it

will be a regular neural network. Because of this small patch, we have fewer weights.Now let's talk about a bit of mathematics which is involved in the whole convolution process:

- Convolution layers consist of a set of learnable filters (patch in the above image). Every filter has small width and height and the same depth as that of input volume (3 if the input layer is image input).
- For example, if we have to run convolution on an image with dimension 34x34x3. Possible size of filters can be a * a * 3, where 'a' can be 3, 5, 7, etc but small as compared to image dimension.
- During forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have value 2 or 3 or even 4 for high dimensional images) and compute the dot product between the weights of filters and patch from input volume.
- As we slide our filters we will get a 2-D output for each filter and we'll stack them together and as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

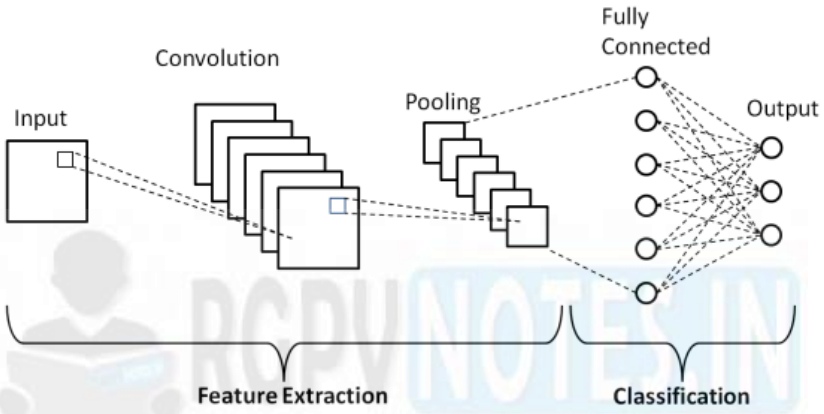


Figure 3.12 Basic convolutional neural network (CNN) architecture

Layers used to build Convolutional neural networks

A Convolutional neural networks is a sequence of layers, and every layer transforms one volume to another through differentiable function.

Types of layers:

Take an example by running a Convolutional neural network on of image of dimension 32 x 32 x 3.

1. **Input Layer:** This layer holds the raw input of image with width 32, height 32 and depth 3.
2. **Convolution Layer:** This layer computes the output volume by computing dot product between all filters and image patch. Suppose we use total 12 filters for this layer we'll get output volume of dimension 32 x 32 x 12.
3. **Activation Function Layer:** This layer will apply element wise activation function to the output of convolution layer. Some common activation functions are RELU: max(0, x), Sigmoid: 1/(1+e^-x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimension 32 x 32 x 12.
4. **Pool Layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents from overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

5. **Fully-Connected Layer:** This layer is regular neural network layer which takes input from the previous layer and computes the class scores and outputs the 1-D array of size equal to the number of classes.

Training process of the Convolution Network:

Step1: We initialize all filters and parameters / weights with random values
Step2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.

- Lets say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]
- Since weights are randomly assigned for the first training example, output probabilities are also random.

Step3: Calculate the total error at the output layer (summation over all 4 classes)

Total Error = ∑ ½ (target probability – output probability) ²

Step4: Use Backpropagation to calculate the *gradients* of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error.

- The weights are adjusted in proportion to their contribution to the total error.
- When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].
- This means that the network has *learnt* to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.
- Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process - only the values of the filter matrix and connection weights get updated.

Step5: Repeat steps 2-4 with all images in the training set.

Recurrent Neural Network

Recurrent Neural Network(RNN) is a type of Neural Network where the **output from previous step are fed as input to the current step**. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is **Hidden state**, which remembers some information about a sequence.RNN have a **"memory"** which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs

or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

How Does Recurrent Neural Network work?

Just like traditional Artificial Neural Networks, RNN consists of nodes with three distinct layers representing different stages of the operation.

- The nodes represent the "Neurons" of the network.
- The neurons are spread over the temporal scale (i.e., sequence) separated into three layers.

The layers are:

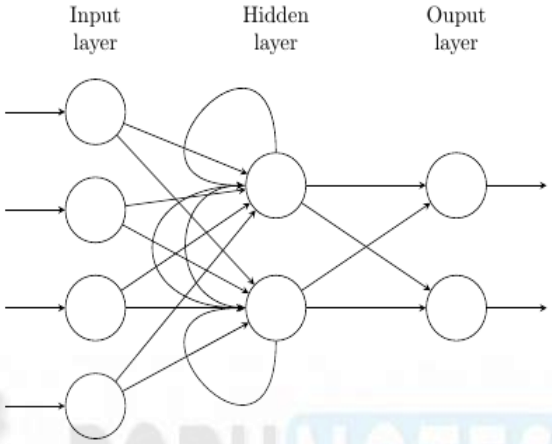


Figure 3.13: Layers of Recurrent Neural Network

1. Input layer represents information to be processed;
2. A hidden layer represents the algorithms at work;
3. Output layer shows the result of the operation;

Hidden layer contains a temporal loop that enables the algorithm not only to produce an output but to feed it back to itself.

This means the neurons have a feature that can be compared to short-term memory. The presence of the sequence makes them to "remember" the state (i.e., context) of the previous neuron and pass that information to themselves in the "future" to further analyze data.

Overall, the RNN neural network operation can be one of the three types:

1. One input to multiple outputs - as in image recognition, image described with words;
2. Several contributions to one output - as in sentiment analysis, where the text is interpreted as positive or negative;
3. Many to many - as in machine translation, where the word of the text is translated according to the context they represent as a whole;

The key algorithms behind RNN are:

1. Backpropagation Through Time to classify sequential input- linking one-time step to the next
2. Vanishing/Exploding gradients - to preserve the accuracy of the results
3. Long Short-Term Memory Units - to recognize the sequences in the data

Subject Name: Soft Computing **Subject Code: IT 701**

Fuzzy Set: Introduction, Basic Definition and Terminology, Properties and Set-theoretic Operations, Fuzzy Relations, Membership Functions and their assignment, Fuzzy rules and fuzzy Reasoning, Fuzzy if-then Rules, Fuzzy Inference Systems. Application of Fuzzy logic in solving engineering problems.

Course Objectives

1) The objective of this course is to understand terminology and operations in Fuzzy set.
2) To understand application of fuzzy logic to solve engineering problem also fuzzy rule and reasoning.

Fuzzy Set: Introduction

The word "fuzzy" means "vagueness". Fuzziness occurs when the boundary of a piece of information is not clear-cut. Fuzzy sets have been introduced by Lotfi A. Zadeh (1965) as an extension of the classical notion of set.

Human thinking and reasoning frequently involve fuzzy information originating from inherently inexact human concepts. Human can give satisfactory answers, which are probably true. However, our systems are unable to answer many questions. The reason is most systems are designed based upon classical set theory and two-valued logic which is unable to cope with unreliable and incomplete information and give expert opinions.

Fuzzy set theory is an extension of classical set theory where elements have degree of membership.

Basic Definition

A set is any well-defined collection of objects. An object in a set is called an element or member of that set.

Sets are defined by a simple statement describing whether a particular element having a certain property belongs to that particular set.

Classical set theory enumerates all its elements using

A = {a1, a2, a3, a4, …….an}

If the elements ai (i=1,2,3,4,…n) of a set A are subset of universal set X, then set A can be represented for all elements x ∈ X by its characteristics function.

Terminology

A set A is well defined by a function called characteristics function. This function defined on the universal space X, assumes:

A value of 1 for those elements x that belongs to set A and

A value of 0 for those elements x that do not belong to set A

The notation used to express these mathematically are

A: X -> [0,1]

A(x) = 1, x is a member of A

A(x) = 0, x is not a member of A

Properties

Properties on sets play an important role for obtaining the solution. Following are the different properties of classical sets –

Commutative Property

Having two sets A and B, this property states −

$A \cup B = B \cup A$

$A \cap B = B \cap A$

Associative Property

Having three sets A, B and C, this property states −

$$A \cup (B \cup C) = (A \cup B) \cup C$$
$$A \cap (B \cap C) = (A \cap B) \cap C$$

Distributive Property

Having three sets A, B and C, this property states −

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

Idempotency Property

For any set A, this property states −

$$A \cup A = A$$
$$A \cap A = A$$

Identity Property

For set A and universal set X, this property states −

$$A \cup \phi = A$$
$$A \cap X = A$$
$$A \cap \phi = \phi$$
$$A \cup X = X$$

Transitive Property

Having three sets A, B and C, the property states −

If $A \subseteq B \subseteq C$, then $A \subseteq C$

Involution Property

For any set A, this property states −

$$\overline{\overline{A}} = A$$

De Morgan's Law

It is a very important law and supports in proving tautologies and contradiction. This law states −

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

Set-theoretic Operations

Set Operations include Set Union, Set Intersection, Set Difference, Complement of Set, and Cartesian product.

Union

The union of sets A and B (denoted by A ∪ BA ∪ B) is the set of elements which are in A, in B, or in both A and B. Hence, A ∪ B = {x|x∈ A OR x ∈ B}.

Example − If A = {10,11,12,13} and B = {13,14,15}, then A ∪ B = {10,11,12,13,14,15} – The common element occurs only once.

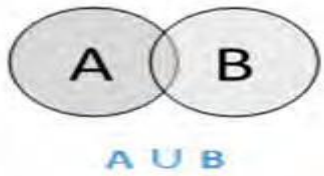


Figure 4.1: Union

Intersection

The intersection of sets A and B (denoted by A ∩ B) is the set of elements which are in both A and B. Hence, A ∩ B = {x|x∈ A AND x ∈ B}.

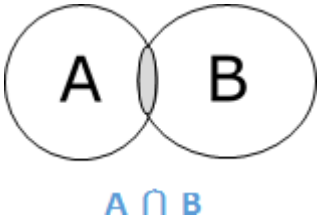


Figure 4.2: Intersection

Difference

The set difference of sets A and B (denoted by A–B) is the set of elements which are only in A but not in B. Hence, A − B = {x|x∈ A AND x ∉ B}.

Example − If A = {10,11,12,13} and B = {13,14,15}, then (A − B) = {10,11,12} and (B − A) = {14,15}. Here, we can see (A − B) ≠ (B − A)

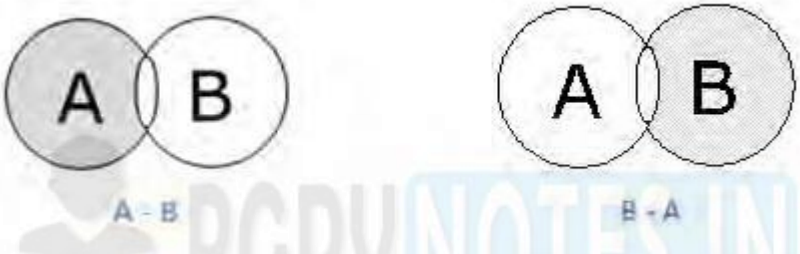


Figure 4.3: Difference

Complement of a Set

The complement of a set A (denoted by A') is the set of elements which are not in set A. Hence, A' = {x|x∉ A}. More specifically, A' = (U−A) where U is a universal set which contains all objects.

Example − If A = {x|x belongs to set of add integers} then A' = {y|y does not belong to set of odd integers}

Cartesian Product / Cross Product

The Cartesian product of n number of sets A1,A2,…An denoted as A1 × A2...× An can be defined as all possible ordered pairs (x1,x2,…xn) where x1 ∈ A1,x2 ∈ A2,…xn∈ An

Example − If we take two sets A = {a,b} and B = {1,2},

The Cartesian product of A and B is written as − A × B = {(a,1),(a,2),(b,1),(b,2)}

And, the Cartesian product of B and A is written as − B × A = {(1,a),(1,b),(2,a),(2,b)}

Fuzzy Relations

A fuzzy relation R is a 2D MF:

$$R = \{((x, y), \mu_R(x, y))|(x, y) \in X \times Y\}$$

Examples:

- x is close to y (x and y are numbers)
- x depends on y (x and y are events)
- x and y look alike (x, and y are persons or objects)
- if x is large, then y is small (x is an observed reading and Y is a corresponding action)

Max-Min Composition

The max-min composition of two fuzzy relations R1 (defined on X and Y) and R2 (defined on Y and Z) is

$$\mu_{R_1 \circ R_2}(x, z) = \vee_y [\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z)]$$

Properties:

Associativity:

$$R \circ (S \circ T) = (R \circ S) \circ T$$

Distributivity over union

$$R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$$

Week distributivity over intersection

$$R \circ (S \cap T) \subseteq (R \circ S) \cap (R \circ T)$$

Monotonicity

$$S \subseteq T \Rightarrow (R \circ S) \subseteq (R \circ T)$$

Membership functions and assignment

Definition:A membership function for a fuzzy set A on the universe of discourse X is defined as μ_A: X → [0, 1], where each element of X is mapped to a value between 0 and 1. This value, called membership value or degree of membership, quantifies the grade of membership of the element in X to the fuzzy set A.

Membership functions allow us to graphically represent a fuzzy set. The *x* axis represents the universe of discourse, whereas the *y* axis represents the degrees of membership in the [0, 1] interval.

Simple functions are used to build membership functions. Because we are defining fuzzy concepts, using more complex functions does not add more precision.

Some of the Fuzzy member functions are as follows:

1. Triangular MFs

A triangular MF is specified by three parameters {a, b, c} as follows:

$$\mu_A(x) = \begin{cases} 0, & x \le a \\ \frac{x-a}{m-a}, & a < x \le m \\ \frac{b-x}{b-m}, & m < x < b \\ 0, & x \ge b \end{cases}$$

The parameters {a, b, c} (with a < b < c) determine the x coordinates of the three corners of the underlying triangular MF.

2. Trapezoidal MFs

A trapezoidal MF is specified by four parameters {a, b, c, d} as follows:

$$\text{trapezoid}(x; a, b, c, d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right).$$

The parameters {a, b, c, d} (with a < b <= c < d) determine the x coordinates of the four corners of the underlying trapezoidal MF.

3. Gaussian MFs

A Gaussian MF is specified by tow parameters:

$$\text{gaussian}(x; c, \sigma) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}.$$

A Gaussian MF is determined complete by c and σ; c represents the MFs center and σ determines the MFs width.

Fuzzy Rules & Fuzzy Reasoning

Extension Principle

A is a fuzzy set on X

$$A = \mu_A(x_1) / x_1 + \mu_A(x_2) / x_2 + \cdots + \mu_A(x_n) / x_n$$

The image of A under f() is a fuzzy set B:

$$B = \mu_B(x_1) / y_1 + \mu_B(x_2) / y_2 + \square + \mu_B(x_n) / y_n$$

Where $Y_i = f(x_i)$, i=1 to n

If f() is a many to one mapping then

$$\mu_B(y) = \max_{x = f^{-1}(y)} \mu_A(x)$$

Fuzzy Reasoning

Following are the different modes of approximate reasoning −

Categorical Reasoning

In this mode of approximate reasoning, the antecedents, containing no fuzzy quantifiers and fuzzy probabilities, are assumed to be in canonical form.

Qualitative Reasoning

In this mode of approximate reasoning, the antecedents and consequents have fuzzy linguistic variables; the input-output relationship of a system is expressed as a collection of fuzzy IF-THEN rules. This reasoning is mainly used in control system analysis.

Syllogistic Reasoning

In this mode of approximation reasoning, antecedents with fuzzy quantifiers are related to inference rules.

This is expressed as −

x = S1A's are B's

y = S2C's are D's

---

z = S3E's are F's

Here A,B,C,D,E,F are fuzzy predicates.

- S1 and S2 are given fuzzy quantifiers.

- S3 is the fuzzy quantifier which has to be decided.

Dispositional Reasoning
In this mode of approximation reasoning, the antecedents are dispositions that may contain the fuzzy quantifier "usually". The quantifier usually links together the dispositional and syllogistic reasoning; hence it pays an important role.

For example, the projection rule of inference in dispositional reasoning can be given as follows –
Usually ((L, M) is R) ⇒ usually (L is [R ↓ L])
Here [R ↓ L] is the projection of fuzzy relation R on L

Fuzzy If-Then Rules
It is a known fact that a human being is always comfortable making conversations in natural language. The representation of human knowledge can be done with the help of following natural language expression −
IF antecedent THEN consequent
The expression as stated above is referred to as the Fuzzy IF-THEN rule base.

General format: if x is A then y is B
Examples:

- If pressure is high, then volume is small
- If the road is slippery, then driving is dangerous
- Is a tomato is red, then it is ripe
- If the speed is high, then apply the break a little.

Interpretations of Fuzzy IF-THEN Rules
Fuzzy IF-THEN Rules can be interpreted in the following four forms −

Assignment Statements
These kinds of statements use "=" (equal to sign) for the purpose of assignment. They are of the following form −
a = hello
climate = summer

Conditional Statements
These kinds of statements use the "IF-THEN" rule base form for the purpose of condition. They are of the following form −
IF temperature is high THEN Climate is hot
IF food is fresh THEN eat.

Unconditional Statements
They are of the following form −
GOTO 10
turn the Fan off

Fuzzy Inference System
Fuzzy Inference System is the key unit of a fuzzy logic system having decision making as its primary work. It

uses the “IF…THEN” rules along with connectors “OR” or “AND” for drawing essential decision rules.

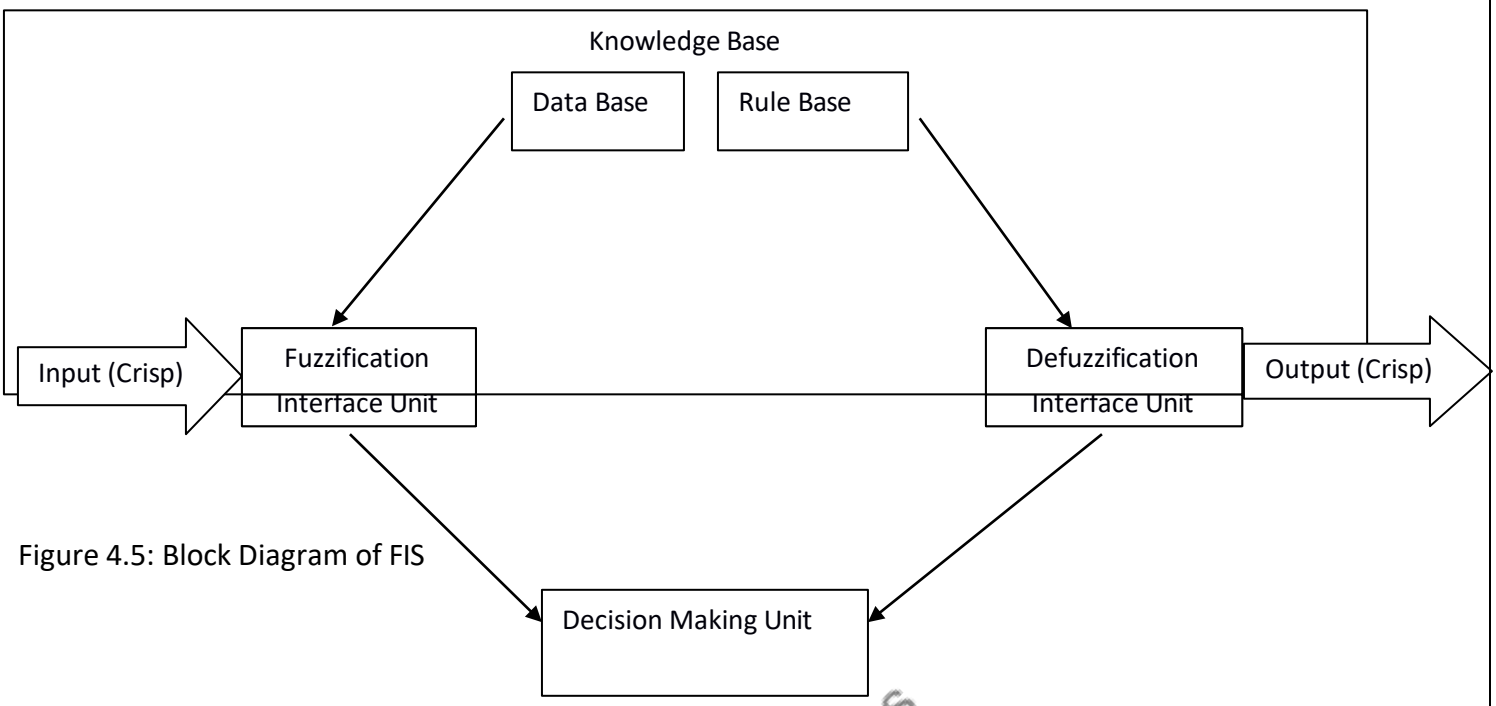


Figure 4.5: Block Diagram of FIS

Characteristics of Fuzzy Inference System

- The output from FIS is always a fuzzy set irrespective of its input which can be fuzzy or crisp.
- It is necessary to have fuzzy output when it is used as a controller.
- A defuzzification unit would be there with FIS to convert fuzzy variables into crisp variables

Functional Blocks of FIS

The following five functional blocks will help you understand the construction of FIS −

Rule Base − It contains fuzzy IF-THEN rules.

Database − It defines the membership functions of fuzzy sets used in fuzzy rules.

Decision-making Unit − It performs operation on rules.

Fuzzification Interface Unit − It converts the crisp quantities into fuzzy quantities.

Defuzzification Interface Unit − It converts the fuzzy quantities into crisp quantities. Following is a block diagram of fuzzy interference system.

Working of FIS

The working of the FIS consists of the following steps −

- A fuzzification unit supports the application of numerous fuzzification methods and converts the crisp input into fuzzy input.
- A knowledge base - collection of rule base and database are formed upon the conversion of crisp input into fuzzy input.
- The defuzzification unit fuzzy input is finally converted into crisp output.

Applications of Fuzzy Logic

Following are the applications of Fuzzy Logic in various engineering domains:

Aerospace

In aerospace, fuzzy logic is used in the following areas −

- Altitude control of spacecraft

- Satellite altitude control

- Flow and mixture regulation in aircraft deicing vehicles

Automotive

In automotive, fuzzy logic is used in the following areas −

- Trainable fuzzy systems for idle speed control
- Shift scheduling method for automatic transmission
- Intelligent highway systems
- Traffic control
- Improving efficiency of automatic transmissions

Business

In business, fuzzy logic is used in the following areas −

- Decision-making support systems
- Personnel evaluation in a large company

Defense

In defense, fuzzy logic is used in the following areas −

- Underwater target recognition
- Automatic target recognition of thermal infrared images
- Naval decision support aids
- Control of a hypervelocity interceptor
- Fuzzy set modeling of NATO decision making

Electronics

In electronics, fuzzy logic is used in the following areas −

- Control of automatic exposure in video cameras
- Humidity in a clean room
- Air conditioning systems
- Washing machine timing
- Microwave ovens
- Vacuum cleaners

Finance

In the finance field, fuzzy logic is used in the following areas −

- Banknote transfer control
- Fund management
- Stock market predictions

Industrial Sector

In industrial, fuzzy logic is used in following areas −

- Cement kiln controls heat exchanger control
- Activated sludge wastewater treatment process control
- Water purification plant control
- Quantitative pattern analysis for industrial quality assurance
- Control of constraint satisfaction problems in structural design
- Control of water purification plants

Manufacturing

In the manufacturing industry, fuzzy logic is used in following areas −

- Optimization of cheese production
- Optimization of milk production

Marine

In the marine field, fuzzy logic is used in the following areas −

- Autopilot for ships
- Optimal route selection
- Control of autonomous underwater vehicles
- Ship steering

Medical

In the medical field, fuzzy logic is used in the following areas −

- Medical diagnostic support system
- Control of arterial pressure during anesthesia
- Multivariable control of anesthesia
- Modeling of neuropathological findings in Alzheimer's patients
- Radiology diagnoses
- Fuzzy inference diagnosis of diabetes and prostate cancer

Securities

In securities, fuzzy logic is used in following areas −

- Decision systems for securities trading
- Various security appliances

Transportation

In transportation, fuzzy logic is used in the following areas −

- Automatic underground train operation
- Train schedule control
- Railway acceleration
- Braking and stopping
-

Pattern Recognition and Classification

In Pattern Recognition and Classification, fuzzy logic is used in the following areas −

- Fuzzy logic-based speech recognition
- Fuzzy logic based
- Handwriting recognition
- Fuzzy logic based facial characteristic analysis
- Command analysis
- Fuzzy image search

Psychology

In Psychology, fuzzy logic is used in following areas −

- Fuzzy logic-based analysis of human behavior
- Criminal investigation and prevention based on fuzzy logic reasoning

Subject Notes: Unit I

IT 701 Soft Computing

Syllabus:

Introduction to Neural Network: Concept, biological neural network, comparison of ANN with biological NN, evolution of artificial neural network, Basic models, Types of learning, linear separability, XOR problem, McCulloch-Pitts neuron model, Hebb rule.

Introduction to Neural Network: **Concept of Neural Network**

A neural network is a processing device, an algorithm or an actual hardware, whose design was inspired by the design and functioning of animal brains and components.

The computing world has a lot to gain from neural networks, also known as artificial neural networks or neural net.

The word neural network is referred to a network of biological neurons in the nervous system that process and transmit information. The neural network has the ability to learn by examples, which makes them very flexible and powerful.

Biological Neural Network

The human brain consists of a large number, more than a billion of neural cells that process information. Each cell works like a simple processor. The massive interaction between all cells and their parallel processing only makes the brain's abilities possible.

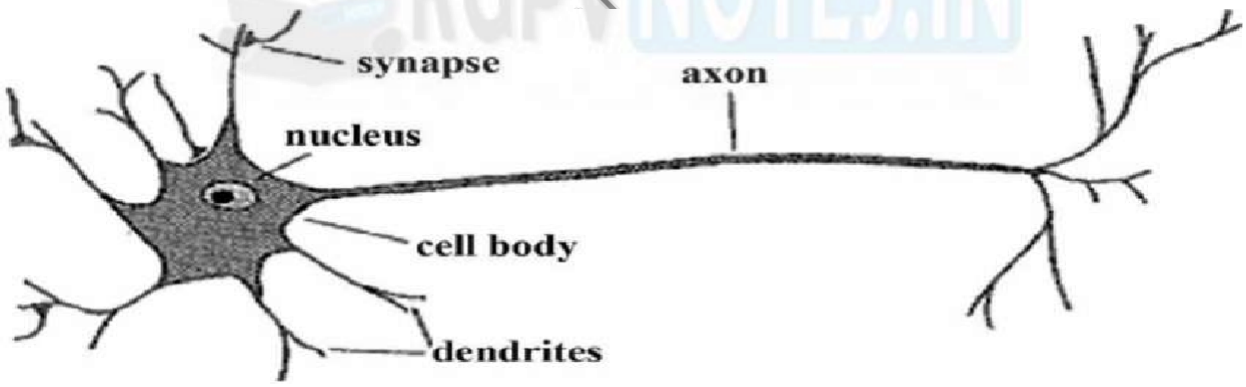


Figure 1.1: Structure of Biological Neuron

The biological neuron consists of three main parts:

Soma or Cell Body: where the cell nucleus is located

Dendrites: where the nerve is connected to the cell body

Axon: carries the impulses of the neuron

Information flow in neural cell

The input /output and the propagation of information is as give below:

- Dendrites receive activation from other neurons.
- Soma processes the incoming activations and converts them into output activations.

- Axons act as transmission lines to send activation to other neurons.
- Synapses the junctions allow signal transmission between the axons and dendrites.
- The process of transmission is by diffusion of chemicals called neuro-transmitters.

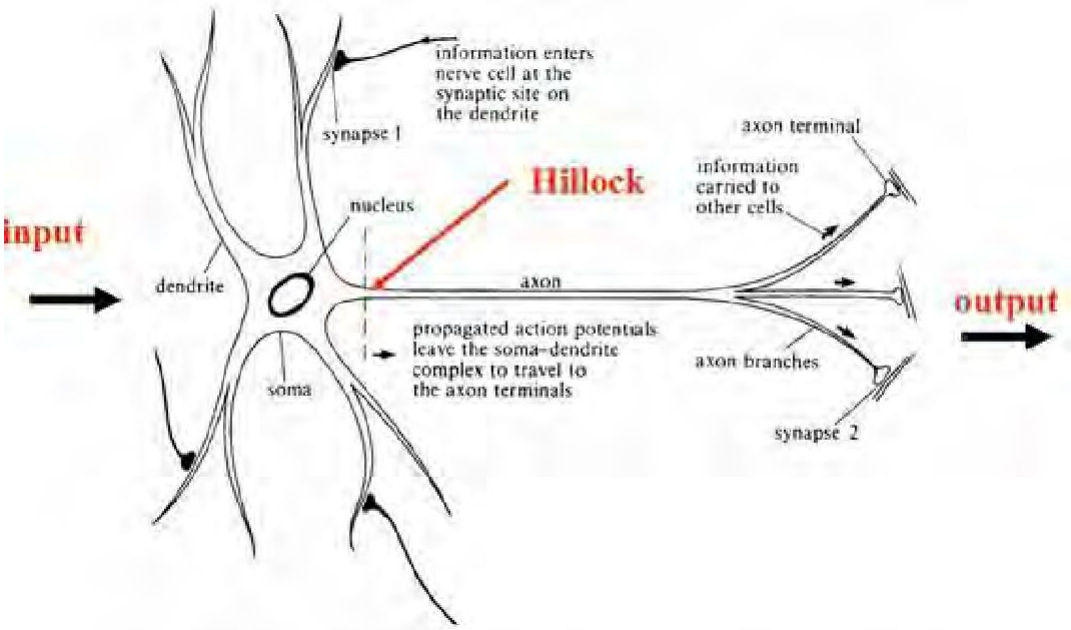


Figure 1.2: Information Flow in Neural Cell

Comparison between Biological Neuron and Artificial Neuron (Brain vs. Computer)

Particular	**Brain**	**ANN**
Speed	Few ms	Few nano second
Size & Complexity	10^{11} neurons & 10^{15} interconnections	Depends on designer
Storage Capacity	Stores information in its interconnection or in synapse. No loss of memory	Contiguous memory location. Loss of memory may happen sometimes
Tolerance	Has fault tolerance	No fault tolerance
Control Mechanism	Complicated involves chemicals in biological neuron	Simpler in ANN

Table 1.1: Brain vs ANN

Artificial Neural Network

An artificial neural network (ANN) may be defined as an information processing model that is inspired by the way biological nervous system, such as brain, process information. This model tries to replicate only the most basic functions of the brain. The key element of ANN is the novel structure of its information processing system.

An ANN is configured for a specific application, such as pattern recognition or data classification through a learning process.

Mathematical Model of Artificial Neuron

An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.

- A set of input connections brings in activations from other neurons.
- A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing / transfer / threshold function).
- An output line transmits the result to other neurons.

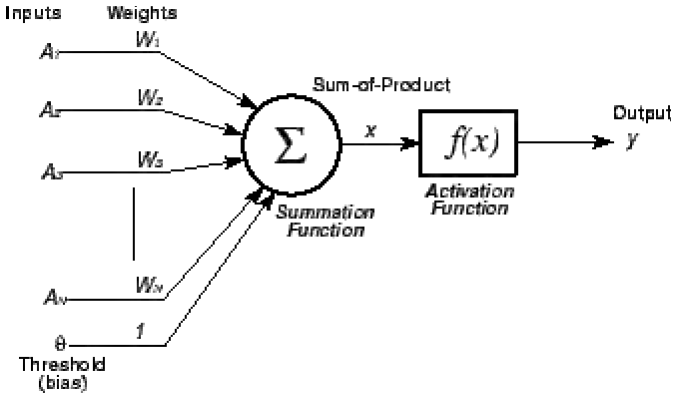


Figure 1.3: Model of Artificial Neuron

Terminology relationship between biological and artificial neurons

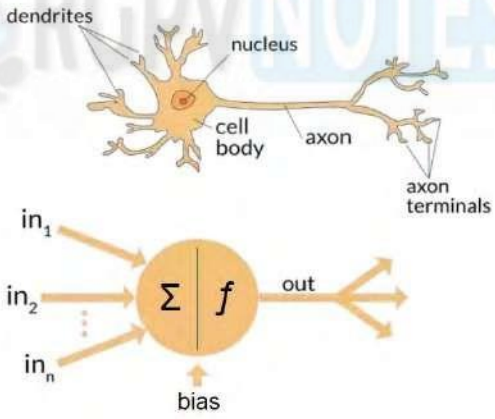


Figure 1.4: Terminology Relation

Evolution of Neural Networks

Year	Neural Network	Description
1943	McCulloch & Pitts Neuron	The arrangement of neurons in this case is a combination of logic functions. Unique feature is the concept of threshold
1949	Hebb Network	It is based upon the fact that if two neurons are found to be active simultaneously then the strength of the connection between them should be increased
1958	Perceptron	Here the weights on the connection path can be adjusted

1960	Adaline	Here the weights are adjusted to reduce the difference between the net input to the output unit and the desired output.
1972	Kohonen self-organizing feature map	The concept behind this network is that the inputs are clustered together to obtain a fired output neuron.
1982	Hopfield Network	This neural network is based on fixed weights. These nets can also act as associative memory nets.
1986	Back Propagation Network	This network is multi-layer with error being propagated backwards from the output units to be hidden units.
1988	Counter Propagation	This is similar to kohonen; here the learning occurs for all units in a particular layer, and there exists no competition among these units.
1987-90	Adaptive Resonance Theory (ART)	The ART network is designed for both binary inputs and analog valued inputs.
1988	Radial basis function	This resembles a back propagation network but the activation function used in a Gaussian function.
1988	Neo cognitron	This network is essential for character recognition.

Table 1.2: Evolution of Neural Network

McCulloch-Pitts Neuron Model

The McCulloch-Pitts model was an extremely simple artificial neuron. The inputs could be either a zero or a one. And the output was a zero or a one. And each input could be either excitatory or inhibitory. Now the whole point was to sum the inputs. If an input is one, and is excitatory in nature, it added one. If it was one, and was inhibitory, it subtracted one from the sum. This is done for all inputs, and a final sum is calculated. Now, if this final sum is less than some value (which you decide, say T), then the output is zero. Otherwise, the output is a one. Here is a graphical representation of the McCulloch-Pitts model

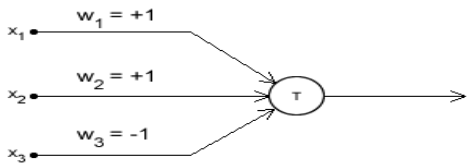


Figure 1.5: McCulloch-Pitts Neuron Model

The variables w_1, w_2 and w_3 indicate which input is excitatory, and which one is inhibitory. These are called "weights". So, in this model, if a weight is 1, it is an excitatory input. If it is -1, it is an inhibitory input.

x_1, x_2, and x_3 represent the inputs. There could be more (or less) inputs if required. And accordingly, there would be more 'w's to indicate if that particular input is excitatory or inhibitory.

Now, if you think about it, you can calculate the sum using the 'x's and 'w's... something like this:

$sum = x_1w_1 + x_2w_2 + x_3w_3 + ...$

This is what is called a 'weighted sum'.

Now that the sum has been calculated, we check if sum < T or not. If it is, then the output is made zero. Otherwise, it is made a one.

Learning

The main property of ANN is its capability to learn. Learning or training is a process by means of which a neural network adapts itself to a stimulus by making proper parameter adjustments, resulting in the production of desired response. Broadly there are two kinds of learning in ANN:

- Parameter Learning: It updates the connecting weights in a neural set.
- Structure Learning: It focuses on the change in network structure (which includes the number of processing elements as well as their connection types).

The above two types of learning can be performed simultaneously or separately. Apart from these two types, the learning in ANN can be generally classified into following categories:

- Supervised Learning
- Unsupervised Learning

Supervised Learning

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

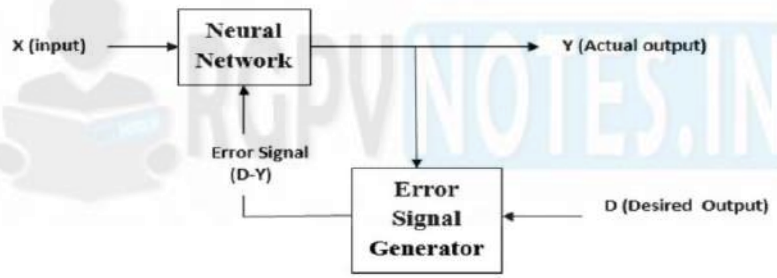


Figure 1.6: Supervised Learning

It is called supervised learning because the process of algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers; the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

During training, the input vector is presented to the network, which results in an output vector. This output vector is the actual output vector. Then the actual output vector is compared with the desired (target) output vector. If there exists a difference between the two output vectors then an error signal is generated by the network. This error is used for adjustment of weights until the actual output matches the desired (target) output.

In this type of learning, a supervisor or teacher is required for error minimization. Hence, the network trained by this method is said to be using supervised training methodology. In supervised learning it is assumed that the correct “target” output values are known for each input pattern.

Unsupervised Learning

Unsupervised learning is where you only have input data (X) and no corresponding output variables.

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

In ANNs following unsupervised learning, the input vectors of similar type are grouped without the use of training data to specify how a member of each group looks or to which group a number belongs. In the training process, the network receives the input patterns and organizes these patterns to form clusters. When a new input pattern is applied, the neural network gives an output response indicating the class to which the input pattern belongs. If for an input, a pattern class cannot be found then a new class is generated.

These are called unsupervised learning because unlike supervised learning above there is no correct answer and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

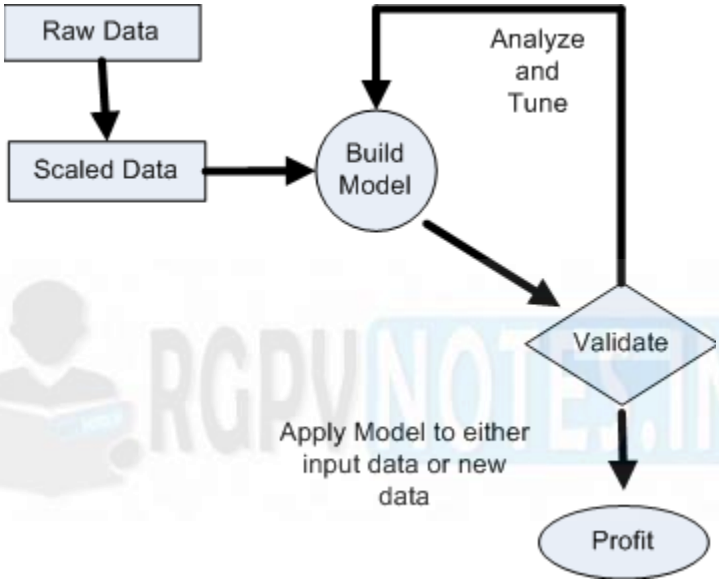


Figure 1.7: Unsupervised Learning

From the working of unsupervised learning it is clear that there is no feedback from the environment to inform what the outputs should be or whether the outputs are correct. In this case the network must itself discover patterns, regularities, features or categories from the input data and relations for the input data over the output.

Linear separability and XOR Problem

Linear separability is a powerful technique which is used to learn complicated concepts that are considerably more complicated than just hyperplane separation.

Let *f* be the XOR function which expects two binary inputs and generates a binary output .Let us try to represent the XOR function by means of an SLP with two input neurons i_1, i_2 and one output neuron:

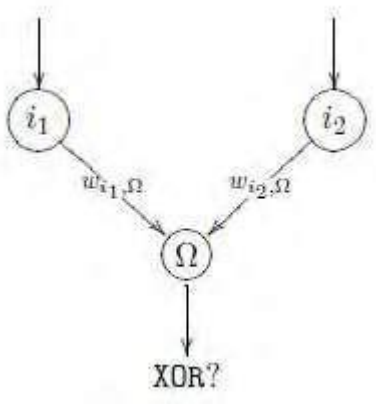


Figure 1.8: Single layer perceptron

Here we use the weighted sum as propagation function, a binary activation function with the threshold value and the identity as output function. Depending on i_1 and i_2 the output becomes the value 1 if the following holds:

$$\text{net}_\Omega = o_{i_1} w_{i_1,\Omega} + o_{i_2} w_{i_2,\Omega} \geq \Theta_\Omega \quad \text{------ (1)}$$

We assume a positive weight w_{i2}, Ω the inequality is then equivalent to:

$$o_{i_1} \geq \frac{1}{w_{i_1,\Omega}} (\Theta_\Omega - o_{i_2} w_{i_2,\Omega}) \quad \text{-------- (2)}$$

With a constant threshold value, the right part of in equation 2 is a straight line through a coordinate system defined by the possible outputs o_{i1} und o_{i2} of the input neurons i_1 and i_2.

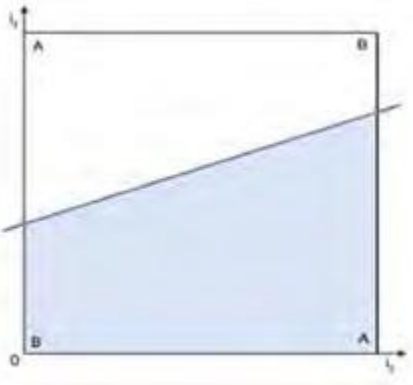


Figure 1.9: Linear separation of n=2 inputs of the input neuron i1 and i2 by 1-D line, A and B show the corner belonging to sets of XOR function that are to be separated

For a positive $w_{i2,}\Omega$ the output neuron fires for input combinations lying above the generated straight line. For a negative $w_{i2,}\Omega$ it would fire for all input combinations lying below the

straight line. Note that only the four corners of the unit square are possible inputs because the XOR function only knows binary inputs.

In order to solve the XOR problem, we have to turn and move the straight line so that input set A = {(0, 0), (1, 1)} is separated from input set B = {(0, 1), (1, 0)} this is, obviously, impossible. Generally, the input parameters of n man input neurons can be represented in an n dimensional cube which is separated by an SLP through an (n−1)-dimensional hyper plane. Only sets that can be separated by such a hyper plane, i.e. which are ***linearly separable***, can be classified by an SLP.

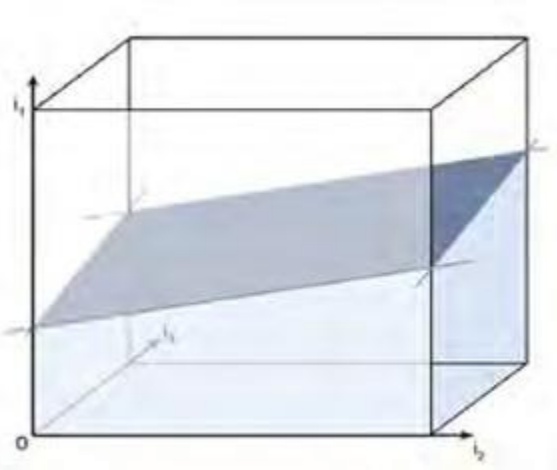


Figure 1.10: Linear separation of n=3 inputs of the input neuron i1, i2 and i3 by 2-D plane

Unfortunately, it seems that the percentage of the linearly separable problems rapidly decreases with increasing, which limits the functionality of the SLP. Additionally, tests for linear separability are difficult. Thus, for more difficult tasks with more inputs we need something more powerful than SLP. The XOR problem itself is one of these tasks, since a perceptron that is supposed to represent the XOR function already needs a hidden layer.

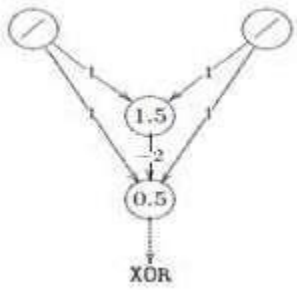


Figure 1.11: Neural network realizing the XOR Function

Activation Function

Let us assume a person is performing some task. To make the task more efficient and to obtain exact output, some force or activation may be given. This activation helps in achieving the exact

output. In the similar way, the activation function is applied over the net input to calculate the output of ANN.
It's just a thing (node) that you add to the output end of any neural network. It is also known as Transfer Function. It can also be attached in between two Neural Networks.

The information processing of a processing element can be viewed as consisting of two major parts: input and output. An integration function (f) is associated with the input of a processing element. This function serves to combine activation, information or evidence from an external source or other processing elements into a net input to the processing element.

Types of Activation Function:

- Sigmoid or Logistic
- Tanh — Hyperbolic tangent
- ReLu -Rectified linear units

Sigmoid Activation function:
It is an activation function of form f(x) = 1 / 1 + exp(-x). Its Range is between 0 and 1. It is a S — shaped curve. It is easy to understand and apply but it has major reasons which have made it fall out of popularity -

- Vanishing gradient problem
- Secondly, its output isn't zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.
- Sigmoids saturate and kill gradients.
- Sigmoids have slow convergence.

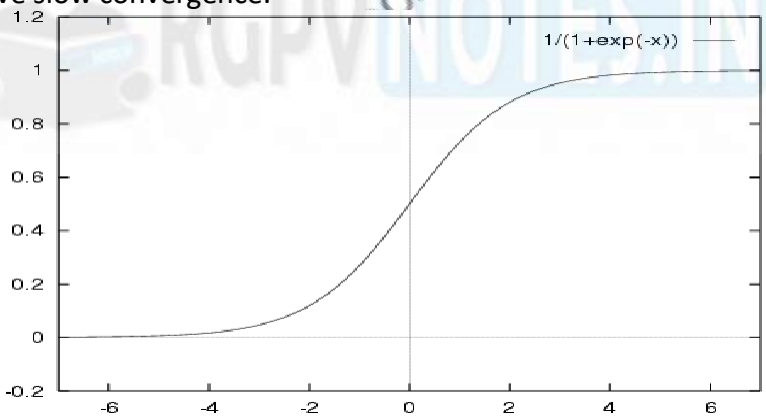


Figure 1.12: Sigmoid Activation Function

Hyperbolic Tangent Function- Tanh:
Its mathematical formula is f(x) = 1 — exp (-2x) / 1 + exp (-2x). Now its output is zero centered because its range in between -1 to 1 i.e -1 < output < 1. Hence optimization is easier in this method hence in practice it is always preferred over Sigmoid function. But still it suffers from Vanishing gradient problem.

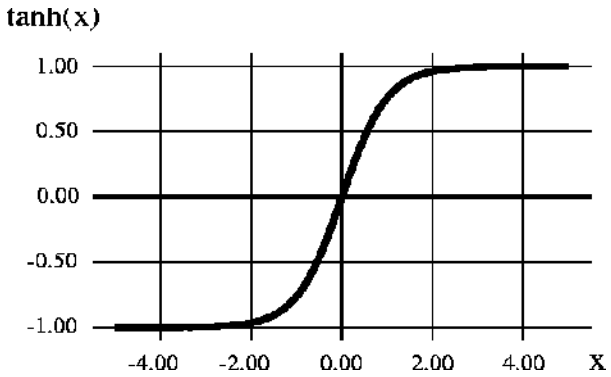


Figure 1.13: Hyperbolic Tangent Function

ReLu- Rectified Linear Units:

It has become very popular in the past couple of years. It was recently proved that it had 6 times improvement in convergence from Tanh function. It's just R(x) = max(0,x) i.e. if x < 0 , R(x) = 0 and if x >= 0 , R(x) = x. Hence as seeing the mathematical form of this function we can see that it is very simple and efficient. It avoids and rectifies vanishing gradient problem. Almost all deep learning Models use ReLu nowadays.

Models of ANN

Feed Forward Network

A feed forward neural network is an artificial neural network wherein connections between the units do not form a cycle. As such, it is different from recurrent neural networks.

The feed forward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes.

Single-Layer Feed Froward Network

The Single Layer Feed-forward Network consists of a single layer of weights, where the inputs are directly connected to the outputs, via a series of weights. The synaptic links carrying weights connect every input to every output, but no other way. The sum of the products of the weights and the inputs is calculated in each neuron node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1).

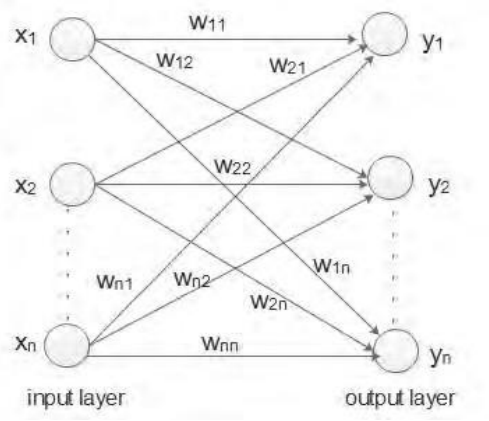


Figure 1.14: Single Layer Feed Forward Network

Multi-Layer Feed Froward Network

The name suggests, it consists of multiple layers. The architecture of this class of network, besides having the input and the output layers, also have one or more intermediary layers called hidden layers. The computational units of hidden layer are known as hidden neuron.

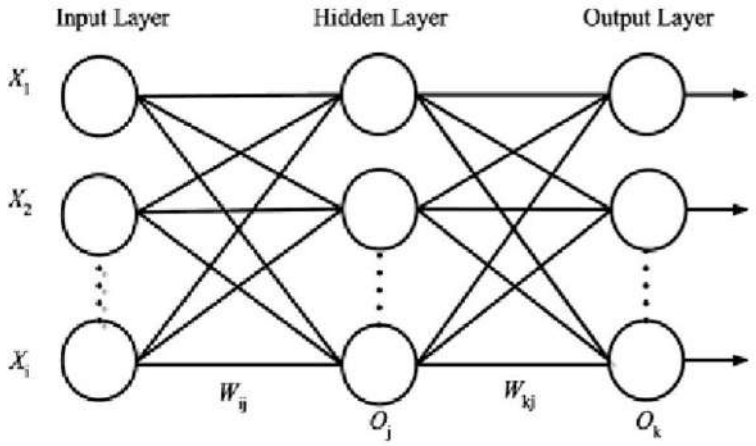


Figure 1.15: Multi-Layer Feed Forward Network

Learning Rule

Basically, learning means to do and adapt the change in itself as and when there is a change in environment. ANN is a complex system or more precisely we can say that it is a complex adaptive system, which can change its internal structure based on the information passing through it.

Learning rule or Learning process is a method or a mathematical logic. It improves the Artificial Neural Network's performance and applies this rule over the network. Thus learning rules updates the weights and bias levels of a network when a network simulates in a specific data environment.

Applying learning rule is an iterative process. It helps a neural network to learn from the existing conditions and improve its performance.

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.

$$\Delta w_{ji}(t)=\alpha x_i(t).y_j(t)$$

Here, $\Delta w_{ji}(t)$= Increment by which the weight of connection increases at time step t
α = The positive and constant learning rate
$x_i(t)$= The input value from pre-synaptic neuron at time step t
$y_i(t)$= The output of pre-synaptic neuron at same time step t

Delta Learning Rule (Widrow-Hoff Rule)

It is introduced by Bernard Widrow and Marcian Hoff, also called Least Mean Square (LMS) method, to minimize the error over all training patterns. It is kind of supervised learning algorithm with having continuous activation function.

Basic Concept − The base of this rule is gradient-descent approach, which continues forever. Delta rule updates the synaptic weights so as to minimize the net input to the output unit and the target value.

Mathematical Formulation − To update the synaptic weights, delta rule is given by

$$\Delta w_i=\alpha.x_i.e_j$$

Here Δw_i = weight change for i^{th} pattern;
α = The positive and constant learning rate;
x_i = The input value from pre-synaptic neuron;
$e_j = (t-y_{in})$, the difference between the desired/target output and the actual output y_{in}
The above delta rule is for a single output unit only.

The updating of weight can be done in the following two cases −

Case-I − when $t \neq y$, then
$w(new)=w(old)+\Delta w$

Case-II − when t = y, then
No change in weight

Subject Name: Soft Computing **Subject Code: IT 701**

Syllabus: Unit V

Genetic Algorithm: Introduction to GA, Simple Genetic Algorithm, terminology and operators of GA (individual, gene, fitness, population, data structure, encoding, selection, crossover, mutation, convergence criteria). Reasons for working of GA and Schema theorem, GA optimization problems like TSP (Travelling salesman problem), Network design routing, Introduction to Ant Colony optimization (ACO) and Particle swarm optimization (PSO).

Course Objectives:

1) The objective of this course is to understand genetic algorithm and its operators.
2) To understand genetic algorithm optimization problems

Introduction to 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.

What are Genetic Algorithms?

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as Evolutionary Computation.

In GAs, we have a pool or a population of possible solutions to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is in line with the Darwinian Theory of "Survival of the Fittest".

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

Terminology & Operators of GA

Individual: An individual is a single solution. An individual groups together two forms of solutions as given below:

1) The chromosome which is the raw "genetic" information (genotype) that the GA deals.
2) The phenotype which is the expressive of the chromosome in the terms of the model.

Genes: Genes are the basic "instructions" for building a GA. A chromosome is a sequence of genes. Genes may describe a possible solution to a problem, without actually being the solution. A gene is a bit string of arbitrary lengths. The bit string is a binary representation of number of intervals from a lower bound. A gene is the GA's representation of a single factor value for a control factor, where control factor must have an upper bound and a lower bound.

Fitness: The fitness of an individual in a GA is the value of an objective function for its phenotype. For calculating fitness, the chromosome has to be first decoded and the objective function has to be evaluated. The fitness not only indicates how good the solution is, but also corresponds to how close the chromosome is to the optimal one.

In case of multi criterion optimization, the fitness function is definitely more difficult to determine.

Population: A population is a collection of individuals. A population consist of a number of individuals being tested, the phenotype parameters defining the individuals and some information about the search space. The two important aspects of population used in GA are:

1. The initial population generation
2. The population size.

For each and every problem, the population size will depend on the complexity of the problem. It is often a random initialization of population.

Data Structure: The basic data structure of a GA is as follows −
We start with an initial population (which may be generated at random or seeded by other heuristics), select parents from this population for mating. Apply crossover and mutation operators on the parents to generate new off-springs. And finally these off-springs replace the existing individuals in the population and the process repeats. In this way genetic algorithms actually try to mimic the human evolution to some extent.
A generalized pseudo-code for a GA is explained in the following program –

```
GA()
  initialize population
  find fitness of population
  while (termination criteria is reached) do
    parent selection
    crossover with probability pc
    mutation with probability pm
    decode and fitness calculation
    survivor selection
    find best
return best
```


Structure of basic genetic algorithm:

1) Start: generate a random population initial. This population can be seen like a collection of chromosomes, as the individuals are reduced to the representation of his notable characteristics for the problem (chromosome); Structure and Operation of a Basic Genetic Algorithm.
2) Fitness: Evaluate each chromosome, by the function of fitness;
3) New population: produce a new population (or generate, or descendants), by execution of the following steps, so many times, those that the new individuals pretended;
a. Selection: select two chromosomes for crossing, respecting that while more fitness greater his probability of selection.
b. Crossing: The chromosomes of the parents have to cross somehow.
c. Mutation: consider, with low probability, the application of mutation in some position of the descendant's chromosomes.
d. To accept: to accept the descendant and place it in the new population.
4) Replacement: To substitute the old population by the new population, generated in the step 3.
5) Test: a. To test the condition of the algorithm; b. If satisfied, finish, producing how «better solution» (not confusing like optimum solution) the common population; of the contrary, continue.
6) Goto 2

Encoding: Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problems.
Types of encoding

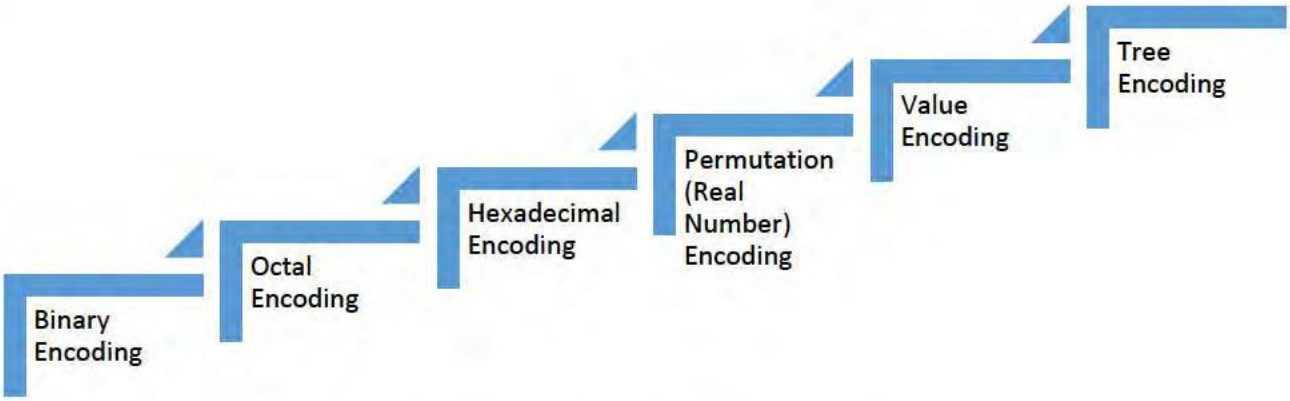


Figure 5.1: Types of Encoding

Selection: Selection is the process of choosing two parents from the population for crossing. After deciding on an encoding, the next step is to decide how to perform selection, i.e., how to choose individuals in the population that will create offspring for the next generation and how many offspring each will create.

Various Selection methods:

Figure 5.2: Types of Selection Methods

The purpose of selection is to emphasize fitter individuals in the population in hopes that their offspring have higher fitness.

Crossover (Recombination): Crossover is the process of taking two parent solutions and producing from them a child. After the selection (reproduction) process, the population is enriched with better individuals. Reproduction makes clones of good strings but does not create new ones. Crossover operator is applied to the mating pool with the hope that is creates a better offspring.

Crossover is a recombination operator that proceeds in three steps:

a. The reproduction operator selects at random a pair of two individual strings for the mating.
b. A cross site is selected at random along the string length.
c. Finally, the position values are swapped between the two strings following the cross site.

Various Crossover Techniques:

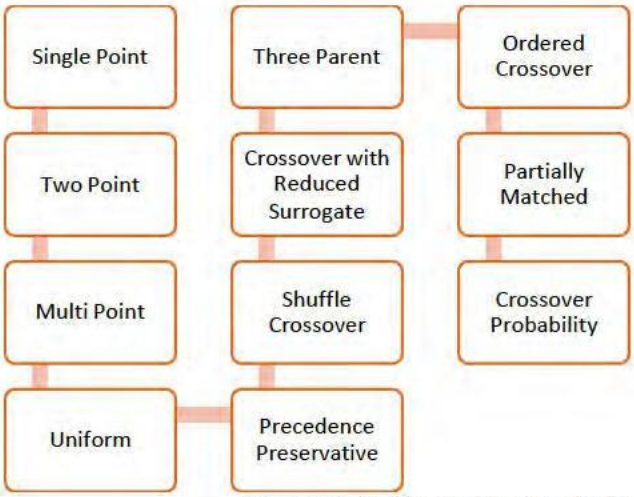


Figure 5.3: Crossover Techniques

Mutation: After crossover operation strings are subjected to mutation. Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the role of recovering the lost genetic materials as well as for randomly distributing genetic information. It is an insurance policy against the irreversible loss of genetic material. Mutation has been traditionally considered as a simple search operator. If crossover is supposed to exploit the current solution to find better ones, mutation is supposed to help for the exploration of the whole search space. Mutation is viewed as a background operator to maintain genetic diversity in the population.

Convergence criteria:

Convergence is a phenomenon in evolutionary computation. It causes evolution to halt because precisely every individual in the population is identical. Full convergence might be seen in genetic algorithms (a type of evolutionary computation) using only crossover (a way of combining individuals to make new offspring). Premature convergence is when a population has converged to a single solution, but that solution is not as high of quality as expected, i.e. the population has gotten 'stuck'. However, convergence is not necessarily a negative thing, because populations often stabilize after a time, in the sense that the best programs all have a common ancestor and their behavior is very similar (or identical) both to each other and to that of high fitness programs from the previous generations. Often the term convergence is loosely used. Convergence can be avoided with a variety of diversity-generating techniques.

The ideal convergence criterion for a genetic algorithm would be one that guaranteed that each and all of the parameters converge independently Beasley et al. (1993a); Goldberg (1989). However, this may be too demanding or may result in too many iterations, so more relaxed convergence criteria are usually employed. Here I used four convergence criteria:

1.The fitness function value must be below a given threshold value.
2.The difference between the best and the average fitness is less than a given fraction of the fitness of the average individual.
3.The difference between the best individual of the current population and the best individual so far must be very small (even zero). This means that the most-fit individual has converged even if the population itself has not.
4. The number of iterations (generations of the sample population) exceeds a given limit. This prevents the algorithm from spending too much time refining an existing solution.
The combination of these criteria is intended to guarantee that the solution is not due to a lucky guess of the random generator but to a comprehensive search of the model space.

Simple Genetic Algorithm

GA handles a population of possible solutions. Each solution is represented through s chromosome, which is just an abstract representation. Coding all the possible solutions into a chromosome is the first part, but certainly not the most straightforward one of GA. A set of reproduction operators has to be determined too. Reproduction operators are applied directly on the chromosome, and are used to perform mutations and recombination over solutions of the problem. The simple form of GA is given as:

1. Start with a randomly generated population.
2. Calculate the fitness of each chromosome in the population.
3. Repeat the following steps until n offsprings have been created:

a. Select a pair of parent chromosome from the current population
b. With probability P_c crossover the pair at a randomly chosen point to from two offspring
c. Mutate the two offspring at each locus with probability P_m

4. Replace the current population with the new population.
5. Go to step 2

General Genetic Algorithm

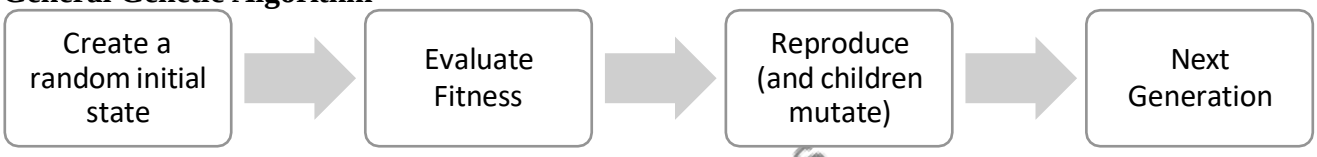


Figure 5.4: Flow of General Genetic Algorithm

Reasons for working of GA:

1. The capability of GA to be implemented as a 'universal optimizer' that could be used for optimizing any type of problem belonging to different fields.
2. Simplicity and ease of implementation.
3. Proper balance between exploration and exploitation could be achieved by setting parameters properly.
4. Logical reasoning behind the use of operators like selection, crossover and mutation.
5. Mathematical or theoretical analysis in terms of schema theory or Markov chain models for the success of GA.
6. One of the pioneer evolutionary algorithms.
7. Solving both constrained and unconstrained optimization problems that is based on natural selection.

The Schema Theorem

Holland's schema theorem, also called the fundamental theorem of genetic algorithms is an inequality that results from coarse-graining an equation for evolutionary dynamics. The Schema Theorem says that short, low-order schemata with above-average fitness increase exponentially in frequency in successive generations. The theorem was proposed by John Holland in the 1970s. It was initially widely taken to be the foundation for explanations of the power of genetic algorithms.

For example, consider binary strings of length 6. The schema 1*10*1 describes the set of all strings of length 6 with 1's at positions 1, 3 and 6 and a 0 at position 4. The * is a wildcard symbol, which means that positions 2 and 5 can have a value of either 1 or 0. The order of a schema o(H) is defined as the number of fixed positions in the template, while the defining length delta (H) is the distance between the first and last specific positions. The order of 1*10*1 is 4 and its defining length is 5. The fitness of a schema is the average fitness of all strings matching the schema. The fitness of a string is a measure of the value of the encoded problem solution, as computed by a problem-specific evaluation function. Using the established methods and genetic operators of genetic algorithms, the

schema theorem states that short, low-order schemata with above-average fitness increase exponentially in successive generations. Expressed as an equation:

$$\mathrm{E}(m(H,t+1)) \geq \frac{m(H,t)f(H)}{a_t}[1-p].$$

Here m(H,t) is the number os string belonging to schema Hat generation t, f(H) is the observed average fitness of schema H and a_t is the observed average fitness at generation t. The probability of disruption p is the probability that crossover or mutation will destroy the schema.

GA Optimization Problem using JSSP

Shop scheduling problems belong to the class of multi-stage scheduling problems, where each job consists of a set of operations. For describing these problems, we use the standard 3-parameter classification a | b | c. The parameter a indicates the machine environment, the parameter b describes job characteristics, and the parameter c givesthe optimization criterion.

In such a shop scheduling problem, a set of n jobs J_1, J_2, …..J_n has to be processed on a set of m machines M_1, M_2,…… M_m. The processing of a job J_i on a particular machine M_j is denoted as an operation and abbreviated by (I, j). Each job J_i consists of a number n_i of operations. For the deterministic scheduling problems, the processing time P_{ij} of each operation (i,j) is given in advance. Among the shop scheduling problems, there are three basic types: a flow-shop, a job shop and an open-shop. In a flow shop problem (a = F), each job has exactly m operations, and the technological route (or machine order) in which the job passes through the machines is the same for any job. Without loss of generality, we assume that the technological route for any job is given by M_1->M_2->…..->M_m. In a job shop problem (a = J), a specific technological route M_{j1}→M_{j2}→….-->M_{jni} is given for each job Ji; 1<=i<=n. Note that the number of operations per job n_i is equal to m for the classical job shop problems, but this number may be also smaller than m or larger than m (recirculation; in this case we may use the notation (I, j, k) for the k-th processing of job J_i on machine M_j). In an open shop problem (a = O), no technological routes are imposed on the jobs. Usually, it is assumed that each job has to be processed on any machine. In addition to these three major types, there also exist generalizations such as a mixed shop or a general shop.

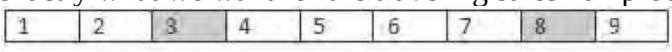
Travelling Salesman Problem using GA

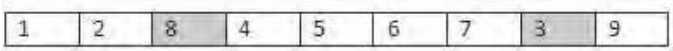
Finding a solution to the travelling salesman problem requires we set up a genetic algorithm in a specialized way. For instance, a valid solution would need to represent a route where every location is included at least once and only once. If a route contain a single location more than once, or missed a location out completely it wouldn't be valid and we would be valuable computation time calculating its distance.

To ensure the genetic algorithm does indeed meet this requirement special types of mutation and crossover methods are needed.

Firstly, the mutation method should only be capable of shuffling the route, it shouldn't ever add or remove a location from the route, and otherwise it would risk creating an invalid solution. One type of mutation method we could use is swap mutation.

With swap mutation two location in the route are selected at random then their positions are simply swapped. For example, if we apply swap mutation to the following list, [1,2,3,4,5] we might end up with, [1,2,5,4,3]. Here, positions 3 and 5 were switched creating a new list with exactly the same values, just a different order. Because swap mutation is only swapping pre-existing values, it will never create a list which has missing or duplicate values when compared to the original, and that's exactly what we want for the traveling salesman problem.



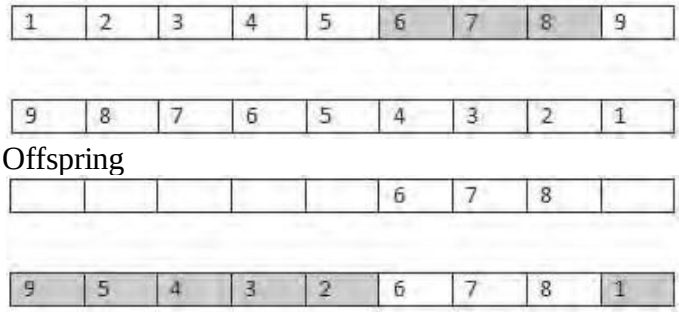


method which can

Now we've dealt with the mutation method we need to pick a crossover enforce the same constraint.

One crossover method that's able to produce a valid route is ordered crossover. In this crossover method we select a subset from the first parent, and then add that subset to the offspring. Any missing values are then adding to the offspring from the second parent in order that they are found. To make this explanation a little clearer consider the following example:

Parents



Offspring

Here a subset of the route is taken from the first parent (6,7,8) and added to the offspring's route. Next, the missing route locations are adding in order from the second parent. The first location in the second parent's route is 9 which isn't in the offspring's route so it's added in the first available position. The next position in the parent's route is 8 which is in the offspring's route so it's skipped. This process continues until the offspring has no remaining empty values. If implemented correctly the end result should be a route which contains all of the positions its parents did with no positions missing or duplicated.

A simple genetic algorithm can be defined in the following steps.

Step 1. Create an initial population of P chromosomes.
Step 2. Evaluate the fitness of each chromosome.
Step 3. Choose P/2 parents from the current population via proportional selection.
Step 4. Randomly select two parents to create offspring using crossover operator.
Step 5. Apply mutation operators for minor changes in the results.
Step 6. Repeat Steps 4 and 5 until all parents are selected and mated.
Step 7. Replace old population of chromosomes with new one.
Step 8. Evaluate the fitness of each chromosome in the new population.
Step 9. Terminate if the number of generations meets some upper bound; otherwise go to Step 3.

The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents travelling times or cost, or a similarity measure between DNA fragments.

Network Design Routing using GA

The purpose of GA is to determine an ‘‘optimal’’ or ‘‘efficient’’ route set (comprising of a pre-specified (or fixed) number of routes) for a given road network and transit demand matrix.

The proposed algorithm, the conceptual overview which follows a three step iterative process. The important features of the proposed algorithm are:

- Various reasonable route sets for the given road network and demand matrix, aredetermined using a novel stochastic procedure, IRSG. It may be pointed out that this heuristic procedure, which onlyprovides a starting point for the proposed optimization procedure.
- After the initial one-time use of IRSG, the evaluation step, and the route modification step are executed repeatedly one after the other until a route set is obtained which satisfies the basic

properties of an efficient route set maximally; i.e. until an ‘‘efficient’’ or ‘‘optimal’’ route set obtained.

- The goodness of a route set as a whole is determined in the evaluation step using the evaluation scheme, EVAL.
- Given a group of route sets and their goodness, the route sets are modified using the proposed modification procedure, MODIFY, in order to obtain a better group of route sets. The modification is done using the evolutionary principles of genetic algorithms.

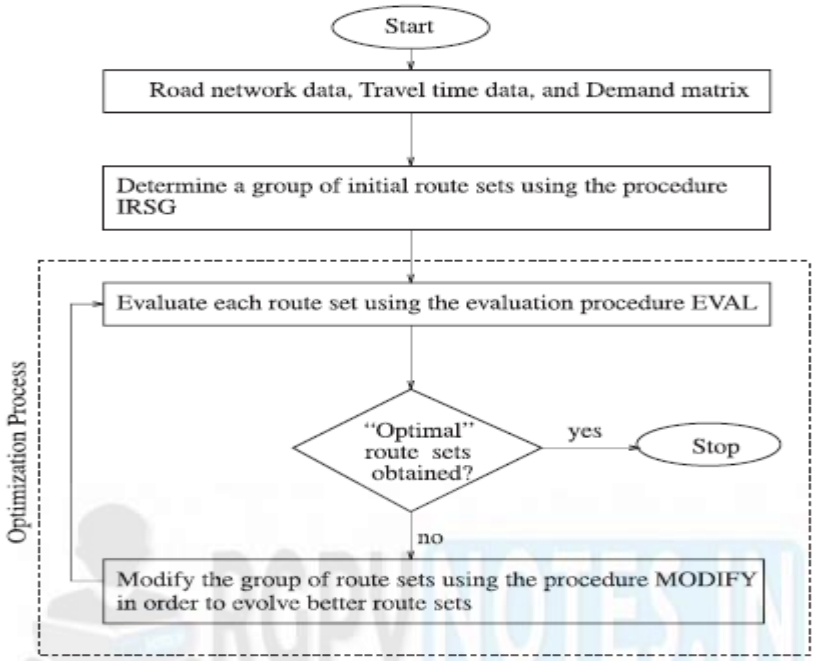


Figure 5.5: Block Diagram of Network Design Routing

Timetabling Problem Solution using GA

1. Generate a random population P
2. **while** Termination condition is not met **do**
3. **for each** individual i of P **do**
4. Generate a timetable using individual i
5. Evaluate the generated timetable using the fitness function
6. **end for**
7. Apply variation operators
8. Generate a new population P
9. End while

The main operators we use at the evolution stage are crossover and mutation. Extra operators can be easily added if necessary.

In order to calculate the fitness value of an individual, a timetable must be generated.

For this, the values in the individual (vector) are used to calculate all arrival and departure times. After the timetable is generated, it is evaluated using a slightly simplified version of the objective function.

The fitness function consists in minimizing the total weighted sum of constraints violations for the timetable.

GA implementation using MATLAB

MATLAB has a wide variety of functions useful to the genetic algorithm practitioner. Given the versatility of MATLAB's high-level language, problems can be coded in m-files in a fraction of the time that it would take to create C or FORTRAN programs for the same purpose. Couple this with MATLAB's advanced data analysis, visualization tools and special purpose application domain toolboxes and the user is presented with a uniform environment with which to explore the potential of genetic algorithms.

The Genetic Algorithm Toolbox uses MATLAB matrix functions to build a set of versatile tools for implementing a wide range of genetic algorithm methods. The Genetic Algorithm Toolbox is a collection of routines, written mostly in m-files, which implement the most important functions in genetic algorithms.

The main data structures used by MATLAB in the Genetic Algorithm toolbox are:

- Chromosomes
- Objective function values
- Fitness values

Introduction to Swarm Intelligence

Swarm Intelligence is a new subset of Artificial Intelligence (AI) designed to manage a group of connected machines. We are now entering the age of the Intelligent machines, also called the Internet of Things (IoT), where more and devices are being connected every day. Swarm intelligence is quickly emerging as a way this connectivity can be harnessed and put to good use.

Swarm intelligence (as the name suggests) comes from mimicking nature. Swarms of social insects, such as ants and bees, operate using a collective intelligence that is greater than any individual member of the swarm. Swarms are therefore highly effective problem-solving groups that can easily deal with the loss of individual members while still completing the task at hand-a capability that is very desirable for a huge number of applications. Today this concept is being applied in concert with machine learning and distributed computing systems. The result is a group of connected machines that can communicate, coordinate, learn and adapt to reach a specific goal. Check out the video below and its subsequent follow up videos to see how swarm intelligence is applied to a group of drones.

Swarm Intelligence Techniques: Ant Colony Optimization

Ant colony optimization (ACO), introduced by Dorigo in his doctoral dissertation, is a class of optimization algorithms modeled on the actions of an ant colony. ACO is a probabilistic technique useful in problems that deal with finding better paths through graphs. Artificial 'ants'—simulation agents—locate optimal solutions by moving through a parameter space representing all possible solutions. Natural ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate for better solutions.

The use of swarm intelligence in telecommunication networks has also been researched, in the form of ant-based routing. This was pioneered separately by Dorigo et al. and Hewlett Packard in the mid-1990s, with a number of variations since. Basically, this uses a probabilistic routing table rewarding/reinforcing the route successfully traversed by each "ant" (a small control packet) which flood the network. Reinforcement of the route in the forwards, reverse direction and both simultaneously has been researched: backwards reinforcement requires a symmetric network and couples the two directions together; forwards reinforcement rewards a route before the outcome is known (but then one would pay for the cinema before one knows how good the film is). As the

there are large hurdles system behaves stochastically and is therefore lacking repeatability, to commercial deployment.

Mobile media and new technologies have the potential to change the threshold for collective action due to swarm intelligence

.

The location of transmission infrastructure for wireless communication networks is an important engineering problem involving competing objectives. A minimal selection of locations (or sites) is required subject to providing adequate area coverage for users. A very different-ant inspired swarm intelligence algorithm, stochastic diffusion search (SDS), has been successfully used to provide a general model for this problem, related to circle packing and set covering. It has been shown that the SDS can be applied to identify suitable solutions even for large problem instances.

Particle Swarm Optimization

Particle swarm optimization (PSO) is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some fitness criterion after each time step. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large numbers of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.

Nanoparticles are bioengineered particles that can be injected into the body and operate as a system to do things drug treatments cannot. The primary problem with all of our current cancer treatments is most procedures target healthy cells in addition to tumors, causing a whole host of side effects. Nanoparticles by comparison, are custom designed to accumulate ONLY in tumors, while avoiding healthy tissue.

Nanoparticles can be designed to move, sense, and interact with their environment, just like robots. In medicine, we call this embodied intelligence. The challenge thus far has been figuring out how to properly "program" this embodied intelligence to ensure it produces the desired outcome.

Swarms are very effective when a group of individual elements (nanoparticles in this case) begin reacting as a group to local information. Swarm intelligence is emerging as the key to which will unlock the true potential of these tiny helpers. Researchers are now reaching out to the gaming community in an effort to crowd source the proper programming for swarm of nanoparticles.

Bee Colony Optimization

The Artificial Bee Colony (ABC) algorithm is a swarm based meta-heuristic algorithm that was introduced by Karaboga in 2005 (Karaboga, 2005) for optimizing numerical problems. It was inspired by the intelligent foraging behavior of honey bees. The algorithm is specifically based on the model proposed by Tereshko and Loengarov (2005) for the foraging behavior of honey bee colonies. The model consists of three essential components: employed and unemployed foraging bees, and food sources. The first two components, employed and unemployed foraging bees, search for rich food sources, which is the third component, close to their hive. The model also defines two leading modes of behavior which are necessary for self-organizing and collective intelligence: recruitment of foragers to rich food sources resulting in positive feedback and abandonment of poor sources by foragers causing negative feedback.

In ABC, a colony of artificial forager bees (agents) search for rich artificial food sources (good solutions for a given problem). To apply ABC, the considered optimization problem is first converted to the problem of finding the best parameter vector which minimizes an objective function. Then, the artificial bees randomly discover a population of initial solution vectors and then iteratively improve them by employing the strategies: moving towards better solutions by means of a neighbor search mechanism while abandoning poor solutions.
The general scheme of the ABC algorithm is as follows:

First Initialization Phase
REPEAT

1. Employed Bees Phase
2. Onlooker Bees Phase
3. Scout Bees Phase
4. Memorize the best solution achieved so far

UNTIL (Cycle=Maximum Cycle Number or a Maximum CPU time)