UNIT-1

Soft Computing

Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation. In effect, the role model for soft computing is the human mind.

The guiding principle of soft computing is: **"Exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost."**

The principal constituents of Soft Computing (SC) are Fuzzy Logic (FL), Neural Computing (NC), Evolutionary Computation (EC) Machine Learning (ML) and Probabilistic Reasoning (PR), chaos theory and parts of learning theory.

Importance of Soft Computing

The complementarity of FL, NC and PR has an important consequence: in many cases a problem can be solved most effectively by using FL, NC and PR in combination rather than exclusively. A striking example of a particularly effective combination is what has come to be known as "neurofuzzy systems." Such systems are becoming increasingly visible as consumer products ranging from air conditioners and washing machines to photocopiers and camcorders. Less visible but perhaps even more important are neurofuzzy systems in industrial applications. What is particularly significant is that in both consumer products and industrial systems, the employment of soft computing techniques leads to systems which have high MIQ (Machine Intelligence Quotient).

Hard Computing

Hard computing deals with precise models where accurate solutions are achieved quickly. Hard Computing is the ancient approach employed in computing that have an accurate analytical model. The outcome of hard computing approach is a warranted, settled and accurate result. It deals with binary and crisp logic that need the precise input. Hard computing isn't capable of finding the solution of real world problems' which are not well defined mathematically or if the inputs are not precise but have lot of dependency on environment.

Difference between Hard Computing and Soft Computing

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

Main Computing Paradigms of Soft Computing:

The main computing paradigms of soft computing are.

1. **Artificial Neural Network (ANN):** ANNs are inspired from biological nervous system. They have the capability of learning and adaptability.

2. Fuzzy Systems: This technique is inspired from human experience. In this approach knowledge representation is performed via Fuzzy "If-Then" rules.

3. Genetic Algorithm: Genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution.

Neural Networks

A neural network is a processing device, whose design was inspired by the design and functioning of human brains and components thereof. The neural networks have the ability to learn by example which makes them very flexible and powerful. For neural networks, there is no need to understand the internal mechanisms of the task.

"A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.

2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge."

The procedure used to perform the learning process is called a learning algorithm, the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective.

A neural network derives its computing power through, first, its massively parallel distributed structure and, second, its ability to learn and therefore generalize. Generalization refers to the neural network's production of reasonable outputs for inputs not encountered during training (learning). These two information processing capabilities make it possible for neural networks to find good approximate solutions to complex (large-scale) problems that are intractable.

An artificial neural network (ANN) may be defined as an information processing model that is inspired by the way biological nervous systems, such as the brain, process information. This model tries to replicate only the most basic functions of brain. The key element of ANN is the novel structure of its information processing system. An ANN is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems.

CHARACTERISTICS OF NEURAL NETWORK

(i) The NNs exhibit mapping capabilities, that is, they can map input patterns to their associated output patterns.

(ii) The NNs learn by examples. Thus, NN architectures can be 'trained with known examples of a problem before they are tested for their 'inference capability on unknown instances of the problem. They can, therefore, identify new objects previously untrained.

(iii) The NNs possess the capability to generalize. Thus, they can predict new outcomes from past trends.

(iv) The NNs are robust systems and are fault tolerant. They can, therefore, recall full patterns from incomplete, partial or noisy patterns.

(v) The NNs can process information in parallel, at high speed, and in a distributed manner.

HISTORY OF NEURAL NETWORK RESEARCH

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

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

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

The Human Brain

The human brain is one of the most complicated part. However, the concept of neurons as the fundamental constituent of the brain has made the study of its functioning comparatively easier. Figure 1 illustrates the physical structure of the human brain.

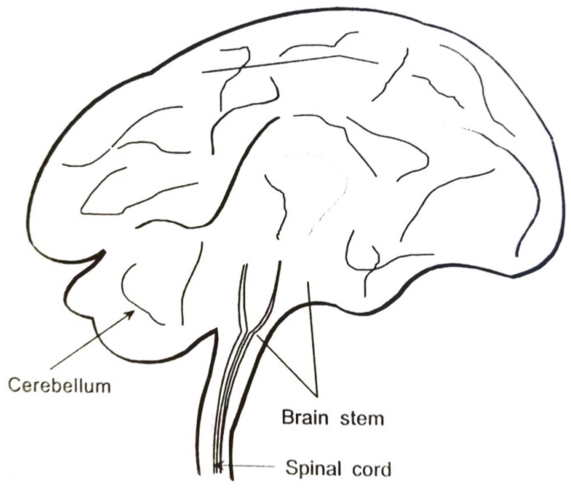


Fig. 1 Physical structure of the human brain-cross-sectional view

Brain contains about 10^{10} basic units called neurons. Each neuron in turn, is connected to about 10^4 other neurons. A neuron is a small cell that receives electro-chemical signals from its various sources and in turn responds by transmitting electrical impulses to other neurons. An average brain weighs about 1.5 kg and an average neuron has a weight of 1.5×10^{-9} gms. While some of the neurons perform input and output operations (referred to as afferent and efferent cells respectively), the remaining form a part of an interconnected network of neurons which are responsible for signal transformation and storage of information. However, despite their different activities, all neurons share common characteristics.

- A neuron is composed of a nucleus-a cell body known as soma (refer Fig. 2).
- Attached to the soma are long irregularly shaped filaments called dendrites. The dendrites behave as input channels, (i.e.) all inputs from other neurons arrive through the dendrites. Dendrites look like branches of a tree during winter.
- Another type of link attached to the soma is the Axon. Unlike the Dendritic links, the axon is electrically active and serves as an output channel. Axons, which mainly appear on output cells are non-linear threshold devices which produce a voltage pulse called Action Potential or Spike that lasts for about a millisecond.
- If the cumulative inputs received by the soma raise the internal electric potential of the cell known as Membrane Potential, then the neuron fires by propagating the action potential down the axon to excite or inhibit other neurons.
- The axon terminates in a specialized contact called synapse or synaptic junction that connects the axon with the dendritic links of another neuron. The synaptic junction, which is a very minute gap at the end of the dendritic link contains a neuro-transmitter fluid. It is this fluid which is responsible for accelerating or retarding the electric charges to the soma. Each dendritic link can have many synapses acting on it thus bringing about massive interconnectivity.

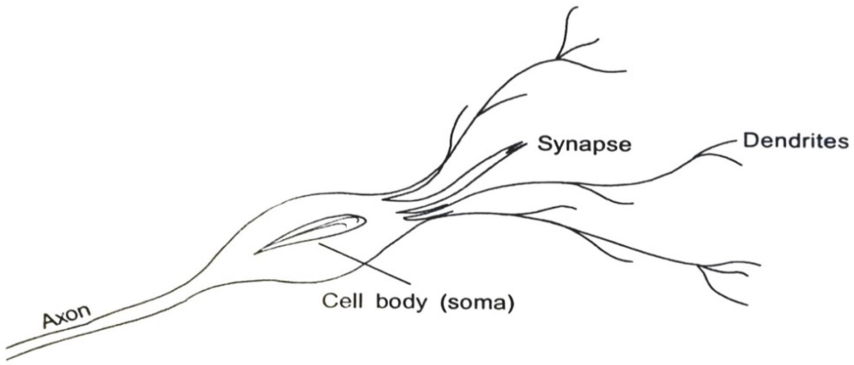


Fig. 2 Structure of a neuron

MODEL OF AN ARTIFICIAL NEURON

The human brain no doubt is a highly complex structure viewed as a massive, highly interconnected network of simple processing elements called neurons. However, the behaviour of a neuron can be captured by a simple model as shown in Fig. 3. Every component of the model bears a direct analogy to the actual constituents of a biological neuron and hence is termed as artificial neuron. It is this model which forms the basis of Artificial Neural Networks.

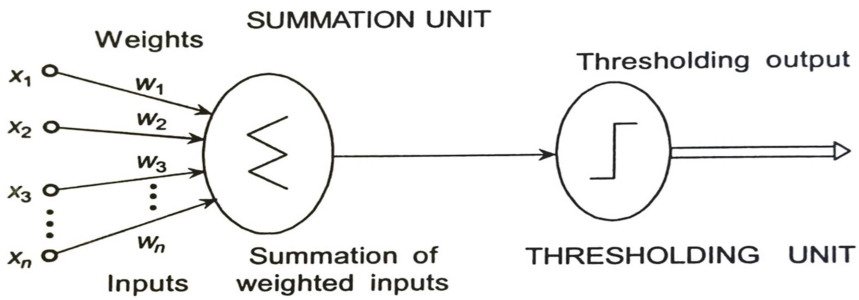


Fig. 3

- Here, x_1, x_2, x_3,….., x_n are the n inputs to the artificial neuron. w_1, w_2, ….., w_n are the weights attached to the input links.
- Recollect that a biological neuron receives all inputs through the dendrites, sums them and produces an output if the sum is greater than a threshold value. The input signals are passed on to the cell body through the synapse which may accelerate or retard an arriving signal.
- It is this acceleration or retardation of the input signals that is modeled by the weights. An effective synapse which transmits a stronger signal will have a correspondingly larger weight while a weak synapse will have smaller weights. Thus, weights here are multiplicative factors of the inputs to account for the strength of the synapse. Hence, the total input I received by the soma of the artificial neuron is

$$I = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

$$= \sum_{i=1}^{n} w_i x_i \qquad \text{.....(1.1)}$$

- To generate the final output y, the sum is passed on to an activation function ϕ, which provides the output.

i.e. $y = \phi(I)$ ..... (1.2)

Necessity of Activation Function:

- It helps to calculate exact output of artificial neural network
- It introduces non-linear properties to our network.
- A very commonly used Activation function is the Threshold function. In this, the sum is compared with a threshold value θ. If the value of I is greater than θ, then the output is 1 else it is 0.

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

..... (1.3)

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

$$\Phi(I) = \begin{cases} 1, & I \geq \theta \\ 0, & I < \theta \end{cases} \quad \text{..... (1.4)}$$

Fig. 4 illustrates the Threshold function. This is convenient in the sense that the output signal is either 1 or 0 resulting in the neuron being on or off.

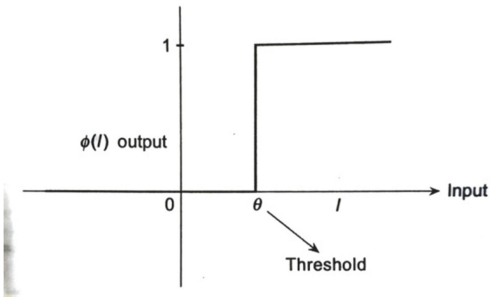


Fig. 4 Thresholding function

Binary Threshold function

The threshold activation function is

$$f(I) = \begin{cases} 1 \; if \; I \geq \theta \\ 0 \; if \; I < \theta \end{cases}$$

In binary threshold function, the value of θ is 0. So the binary threshold activation here is

$$f(I) = \begin{cases} 1 \; if \; I \geq 0 \\ 0 \; if \; I < 0 \end{cases}$$

Bipolar Threshold function

In case of bipolar, 1 is replaced by +1 and 0 is replaced by -1. +1 represents true and -1 represents false. So the bipolar threshold activation function is

$$f(I) = \begin{cases} 1 \; if \; I \geq 0 \\ -1 \; if \; I < 0 \end{cases}$$

Signum Function

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

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

…… (1.5)

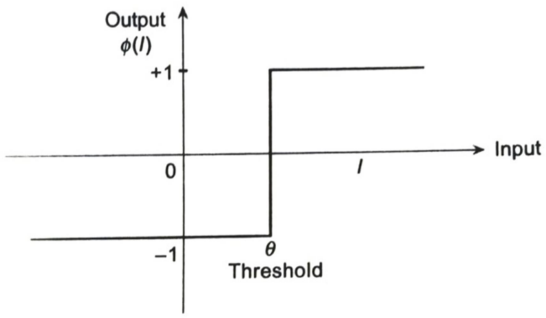


Fig. 5 Signum function

Sigmoidal funtion

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

$$\phi(I) = \frac{1}{1 + e^{-\alpha I}} \quad \textbf{…….....(1.6)}$$

Where, α is the slope parameter, which adjusts the abruptness of the function as it changes between the two asymptotic values. Sigmoidal functions are differentiable, which is an important feature of NN theory. Figure 6 illustrates the sigmoidal function.

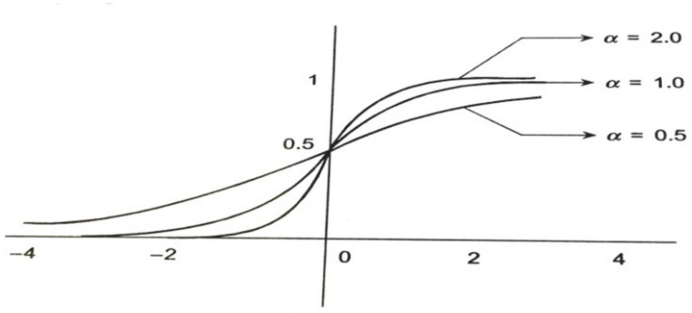


Fig. 6 Sigmoidal Function

Hyperbolic tangent function

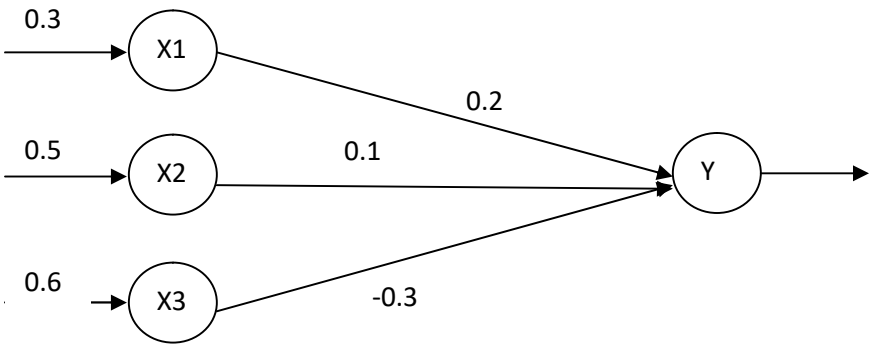
The function is given by

$$\Phi(I) = \tanh(I) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1.7)$$

and can produce both positive and negative output values.

Problems on Basic Neuron Model and Activation function:

Q1. For the network shown in Figure, calculate the net input to the output neuron.



The given neural net consists of three input neurons and one output neuron. The inputs and weights are:

$[x_1, x_2, x_3] = [0.3, 0.5, 0.6]$

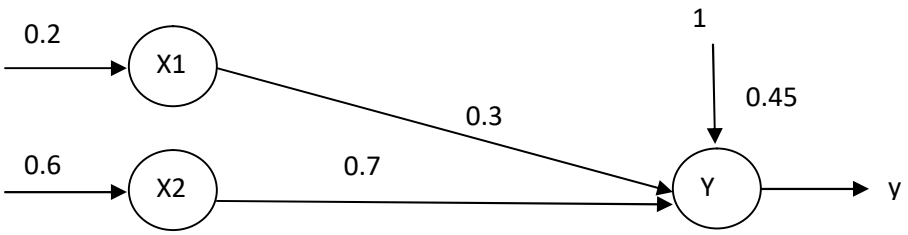
$[w_1, w_2, w_3] = [0.2, 0.1, -0.3]$

The net input Y_{in} at neuron Y can be calculated as:

$Y_{in} = w_1x_1 + w_2x_2 + w_3x_3$

$Y_{in} = 0.3*0.2 + 0.5*0.1 + 0.6*(-0.3) = -0.07$

Q2. For the network shown in Figure, calculate the net input to the output neuron.



Here inputs are $[x_1, x_2] = [0.2, 0.6]$, weights are $[w_1, w_2] = [0.3, 0.7]$ and bias is b=0.45

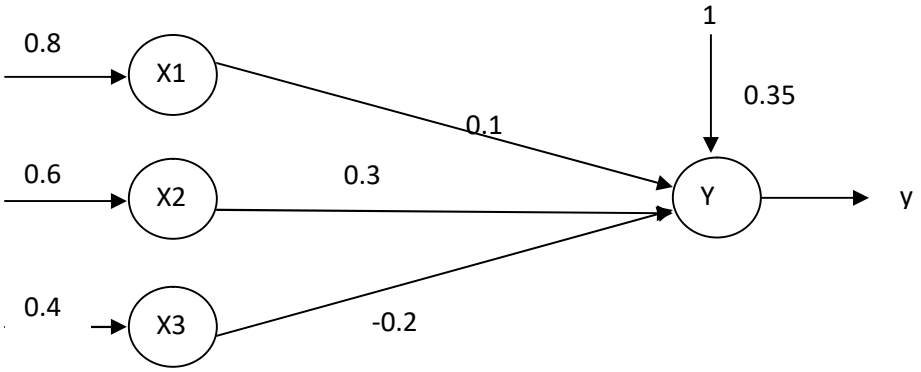
The net input Y_{in} at neuron Y can be calculated as: $Y_{in} = w_1x_1 + w_2x_2 + b$

$Y_{in} = 0.2*0.3 + 0.6*0.7 + 0.45 = 0.93$

The net input is neuron Y is 0.93.

Q3. For the network shown in Figure, calculate the output of the neuron Y using activation function as

- **(i) Binary threshold function**
- **(ii) Bipolar threshold function**
- **(iii) Binary sigmoidal function**



The given network has three input neurons with bias and one output neuron. These form a single-layer network. The inputs are given as $[x_1, x_2, x_3] = [0.8, 0.6, 0.4]$ and the weights $[w_1, w_2, w_3] = [0.1, 0.3, -0.2]$ with bias=0.35

The net input to the output neuron is

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

So here $Y_{in} = w_1x_1 + w_2x_2 + w_3x_3 + b$

$Y_{in} = 0.8*0.1 + 0.6*0.3 + 0.4*(-0.2) + 0.35 = 0.53$

(i) Using Binary Threshold function

The threshold activation function is

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

In binary threshold function, the value of θ is 0. So the binary threshold activation here is

$$f(y_{in}) = \begin{cases} 1 \; if \; y_{in} \geq 0 \\ 0 \; if \; y_{in} < 0 \end{cases}$$

Here Y_{in} is 0.53, that is greater than 0, so the output Y here is 1.

(ii) Using Bipolar Threshold function

In case of bipolar, 1 is replaced by +1 and 0 is replaced by -1. +1 represents true and -1 represents false. So the bipolar threshold activation function is

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

Here Y_{in} is 0.53 that is greater than 0, so the output Y here is + 1.

(iii) Using Binary Sigmoidal function

The binary sigmoidal activation function is:

$$Y = f(Y_{in}) = \frac{1}{1 + e^{-Y_{in}}}$$

$$Y = \frac{1}{1 + e^{-0.53}}$$

So the output Y = 0.625

NEURAL NETWORK ARCHITECTURE

Generally, an ANN structure can be represented using a directed graph. A graph G is an ordered 2-tuple (V, E) consisting of a set V of vertices and a set E of edges. When each edge is assigned an orientation, the graph is directed and is called a directed graph or a digraph. Figure 7 illustrates a digraph. Digraphs assume significance in Neural Network theory since signals in NN systems are restricted to flow in specific directions.

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

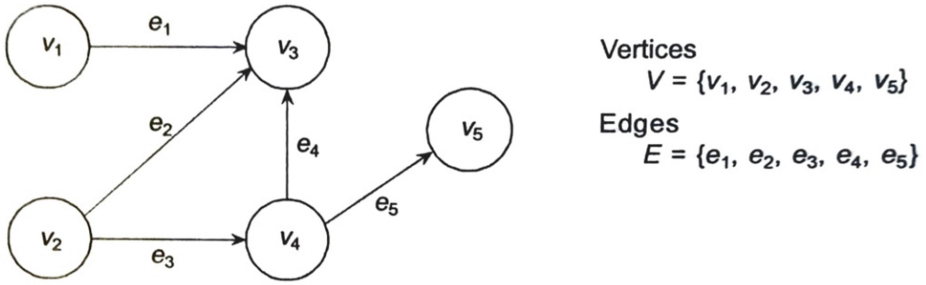


Fig. 7 An example digraph

There are several classes of NN, classified according to their learning mechanisms. However, there are three fundamentally different classes of networks based on architecture. All the three classes employ the digraph structure for their representation.

1. SINGLE LAYER FEED FORWARD NETWORK

This type or network comprises of two layers, namely the input layer and the output layer. The input layer neurons receive the input signals and the output layer neurons computes the output signals. The synaptic links carrying the weights connect every input neuron to the output neuron but not vice-versa. Such a network is said to be feed forward in type or acyclic n nature. **Despite, the two layers, the network is termed single layer since it is the output layer, alone which performs computation.** The input layer merely transmits the signals to the output layer. Hence, the name single layer feed forward network. Figure 8 illustrates an example network.

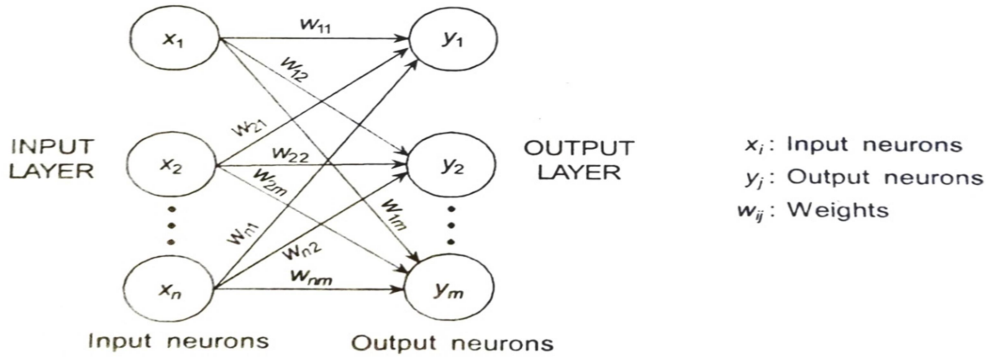


Fig. 8 Single layer feedforward network

2. MULTILAYER FEED FORWARD NETWORK

This network, as its name indicates is made up of multiple layers. Thus, architectures of this class besides possessing an input and an output layer also have one or more intermediary layers called hidden layers. The computational units of the hidden layer are known as the hidden neurons or hidden units. The hidden layer aids in performing useful intermediary computations before directing the input to the output layer. The input layer neurons are linked to the hidden layer neurons and the weights on these links are referred to as input-hidden layer weights. Again, the hidden layer neurons are linked to the output layer neurons and the corresponding weights are referred to as hidden-output layer weights. Figure 9 illustrates a multilayer feed forward network with a configuration 1-m- n.

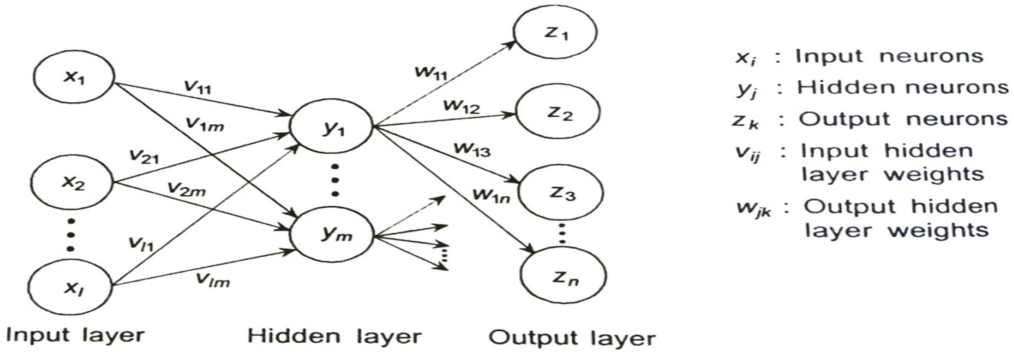


Fig. 9 A multilayer feed forward network (l- m- n configuration)

3. RECURRENT NETWORKS

These networks differ from feed forward network architectures in the sense that there is atleast one feedback loop. Thus, in these networks, for example, there could exist one layer with

feedback connections as shown in Fig. 10. There could also be neurons with self-feedback links, i.e. the output of a neuron is fed back into itself as input.

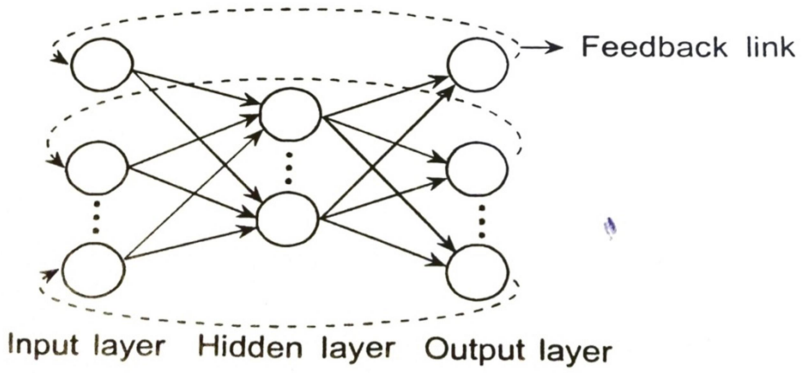


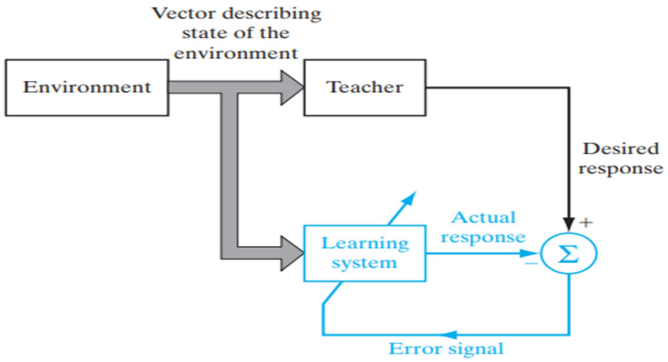
Fig. 10 A recurrent neural network

Learning Processes

We may categorize the learning processes of a neuron into following categories:

1. Supervised learning
2. Unsupervised learning
3. Reinforced learning

1. Supervised learning.



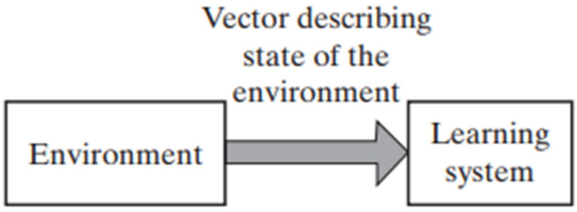
Block diagram of learning with a teacher; the part of the figure printed in red constitutes a feedback loop.

In conceptual terms, we may think of the teacher as having knowledge of the environment, with that knowledge being represented by a set of input–output examples. The environment is, however, unknown to the neural network. Suppose now that the teacher and the neural network are both exposed to a training vector (i.e., example) drawn from the same environment. By virtue of built-in knowledge, the teacher is able to provide the neural network with a desired response for that training vector. Indeed, the desired response represents the "optimum" action

to be performed by the neural network. The network parameters (weights) are adjusted under the combined influence of the training vector and the error signal. The error signal is defined as the difference between the desired response and the actual response of the network. This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network emulate the teacher; the emulation is presumed to be optimum in some statistical sense. In this way, knowledge of the environment available to the teacher is transferred to the neural network through training and stored in the form of "fixed" synaptic weights, representing long-term memory. When this condition is reached, we may then dispense with the teacher and let the neural network deal with the environment completely by itself.

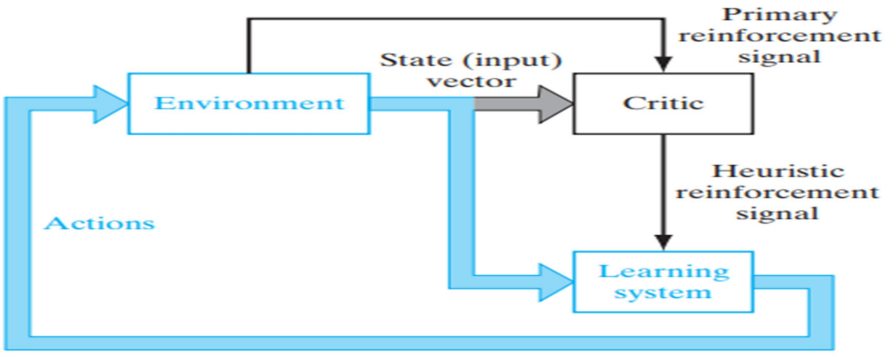
2. Unsupervised Learning

In unsupervised, or self-organized, learning, there is no external teacher or critic to oversee the learning process.



Rather, provision is made for a task-independent measure of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. For a specific task-independent measure, once the network has become tuned to the statistical regularities of the input data, the network develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically.

3. Reinforced learning



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

APPLICATIONS OF NEURAL NETWORK

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

1. **Pattern recognition (PR)/image processing**

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

2. **Optimization/constraint satisfaction**

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

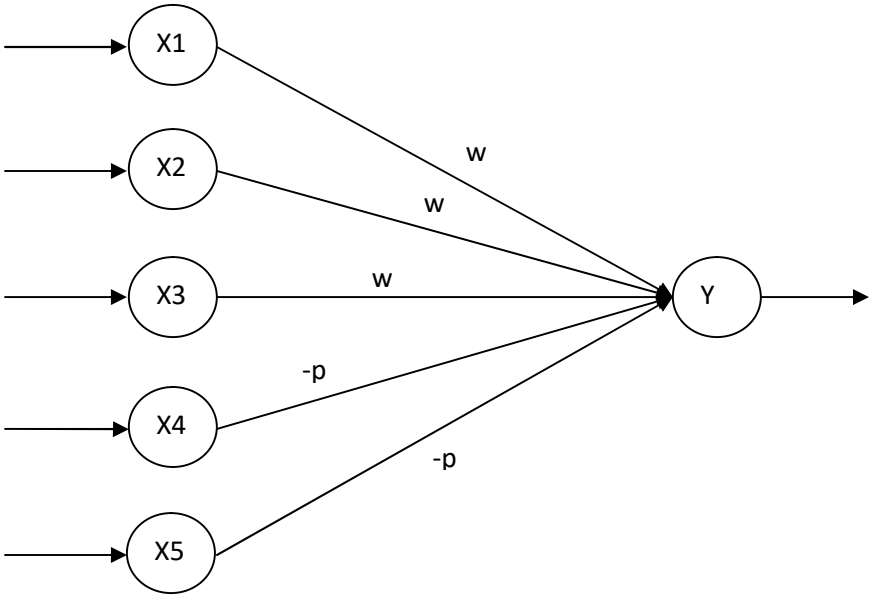
3. **Forecasting and risk assessment**

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

McCulloch-Pitts Neuron

The McCulloch-Pitts neuron was the earliest neural network discovered in 1943. It is usually called as M-P neuron. The M-P neurons are connected by directed weighted paths. It should be noted that the activation of an M-P neuron is binary, that is, at any time step the neuron may fire or may not fire. The weights associated with the communication links may be excitatory (weight is positive) or inhibitory (weight is negative).

The threshold plays a major role in M-P neuron: There is a fixed threshold for each neuron, and if the net input to the neuron is greater than the threshold then the neuron fires.



A simple M-P neuron is shown in above figure. The M-P neuron has both excitatory and inhibitory connections. It is excitatory with weight (w > 0) or inhibitory with weight –p (p < 0). In Figure, inputs from X_1 to X_3 possess excitatory weighted connections and inputs from X_4 to X_5 possess inhibitory weighted interconnections. Since the firing of the output neuron is based upon the threshold θ, the activation function here is defined as

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

Here y_{in} is the net input at neuron y. If the net input is greater than or equal to the threshold then the neuron fires else the neuron will not fire.

The M-P neuron has no particular training algorithm. An analysis has to be performed to determine the values of the weights and the threshold. Here the weights of the neuron are set along with the threshold to make the neuron perform a simple logic function.

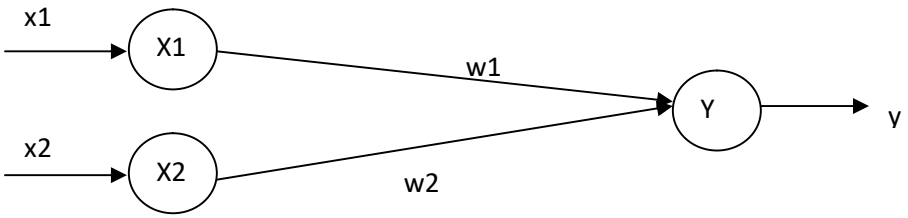
- **In M-P Neuron model, our basic assumptions are**:
  - Threshold activation function will be used
  - Weights can be selected as excitatory (+ve) or inhibitory (-ve), i.e. +1 or -1.
  - Using the given combination of weights, we try to set the threshold for solving the problem.

Q4. Implement AND function using McCulloch-Pitts neuron model.

The truth table for AND function is

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

The network architecture is:



With these assumed weights, the net input is calculated for four inputs:

Initially we assume here that $w_1=1$ and $w_2=1$

(0, 0) $Y_{in}=w_1*x_1+w_2*x_2$ $Y_{in}= 1*0 + 1*0 = 0$

(0, 1) $Y_{in}=w_1*x_1+w_2*x_2$ $Y_{in}= 1*0 + 1*1 = 1$

(1, 0) $Y_{in}=w_1*x_1+w_2*x_2$ $Y_{in}= 1*1 + 1*0 = 1$

(1, 1) $Y_{in}=w_1*x_1+w_2*x_2$ $Y_{in}= 1*1 + 1*1 = 2$

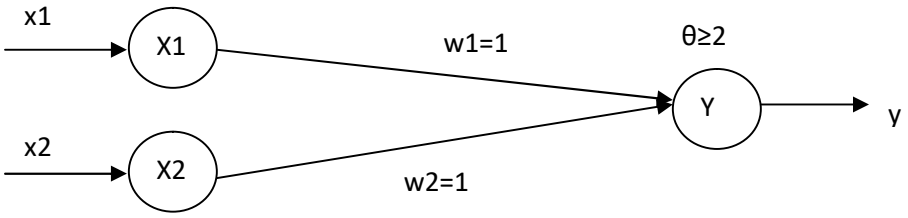
Based on this net input for all inputs, we will set the threshold so that output from neuron Y will match the desired output for AND function.

For example here in AND function, the output is high only for input (1, 1) otherwise the output is 0. **So we will adjust the threshold in such a way that this net input will be converted into desired output. Here we can set the threshold value θ = 2.**

So where the net input is greater than or equal to 2, the neuron will output 1 otherwise neuron will give the output 0.

(0, 0)	$Y_{in}=w_1*x_1+w_2*x_2$	$Y_{in}= 1*0 + 1*0 = 0$		Output 0
(0, 1)	$Y_{in}=w_1*x_1+w_2*x_2$	$Y_{in}= 1*0 + 1*1 = 1$		Output 0
(1, 0)	$Y_{in}=w_1*x_1+w_2*x_2$	$Y_{in}= 1*1 + 1*0 = 1$	θ = 2	Output 0
(1, 1)	$Y_{in}=w_1*x_1+w_2*x_2$	$Y_{in}= 1*1 + 1*1 = 2$		Output 1

After using the weights as (1, 1) and with threshold θ≥2, the neuron Y provides the output as desired by the AND function. The network for AND function is:



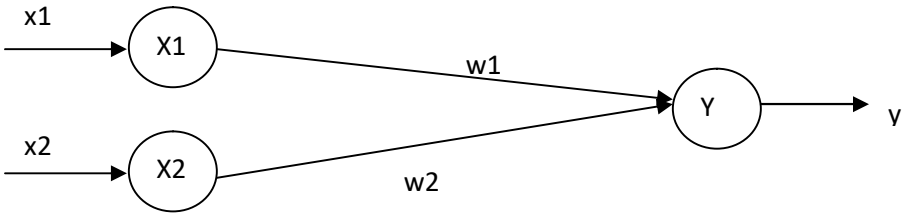
If we are able to set the threshold with these assumed weights, then we can assume another pair of weights such as (1, -1), (-1, 1) and (-1, -1).

Q5. Implement OR function using McCulloch-Pitts neuron model.

The truth table for OR function is

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

First, assume the weights be $w_1 = 1$ and $w_2 = 1$. The network architecture is:



With these assumed weights, the net input is calculated for four inputs:

Initially we assume here that $w_1=1$ and $w_2=1$

(0, 0) $Y_{in}=w_1*x_1+w_2*x_2$ $Y_{in}= 1*0 + 1*0 = 0$

(0, 1) $Y_{in}=w_1*x_1+w_2*x_2$ $Y_{in}= 1*0 + 1*1 = 1$

(1, 0) $Y_{in}=w_1*x_1+w_2*x_2$ $Y_{in}= 1*1 + 1*0 = 1$

(1, 1) $Y_{in}=w_1*x_1+w_2*x_2$ $Y_{in}= 1*1 + 1*1 = 2$

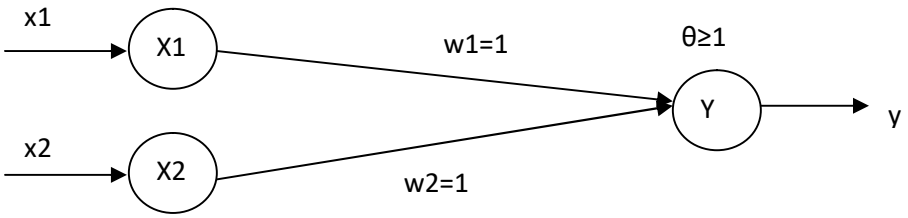
Here in OR function, the output is 1 for all pair of inputs except the pair (0, 0). **So we will adjust the threshold in such a way that this net input will be converted into desired output. Here we can set the threshold value θ = 1.**

So where the net input is greater than or equal to 1, the neuron will output 1 otherwise neuron will give the output 0.

(0,0)	$Y_{in}=w_1*x_1+w_2*x_2$	$Y_{in}= 1*0 + 1*0 = 0$		Output 0
(0,1)	$Y_{in}=w_1*x_1+w_2*x_2$	$Y_{in}= 1*0 + 1*1 = 1$	$\theta = 1$	Output 1
(1,0)	$Y_{in}=w_1*x_1+w_2*x_2$	$Y_{in}= 1*1 + 1*0 = 1$		Output 1
(1,1)	$Y_{in}=w_1*x_1+w_2*x_2$	$Y_{in}= 1*1 + 1*1 = 2$		Output 1

After using the weights as (1, 1) and with threshold $\theta \geq 1$, the neuron Y provides the output as desired by the OR function.

The network for OR function is:



Rosenblatt's Perceptron

The perceptron is the simplest form of a neural network used for the classification of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane). Basically, it consists of a single neuron with adjustable synaptic weights and bias. The algorithm used to adjust the free parameters of this neural network first appeared in a learning procedure developed by Rosenblatt (1958, 1962) for his perceptron brain model.

Perceptron convergence theorem or convergence rule states that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes. The proof of convergence of the algorithm is known as the **perceptron convergence theorem**.

Rosenblatt's perceptron is built around a nonlinear neuron, namely, the McCulloch–Pitts model of a neuron. Such a neural model consists of a linear combiner followed by a hard limiter

(performing the threshold / signum function), as depicted in Figure. The summing node of the neural model computes a linear combination of the inputs applied to its synapses, as well as incorporates an externally applied bias. The resulting sum, that is, the induced local field, is applied to a hard limiter. Accordingly, the neuron produces an output equal to 1 if the hard limiter input is positive, and 0 or -1 if it is negative (0 for threshold and -1 for signum function).

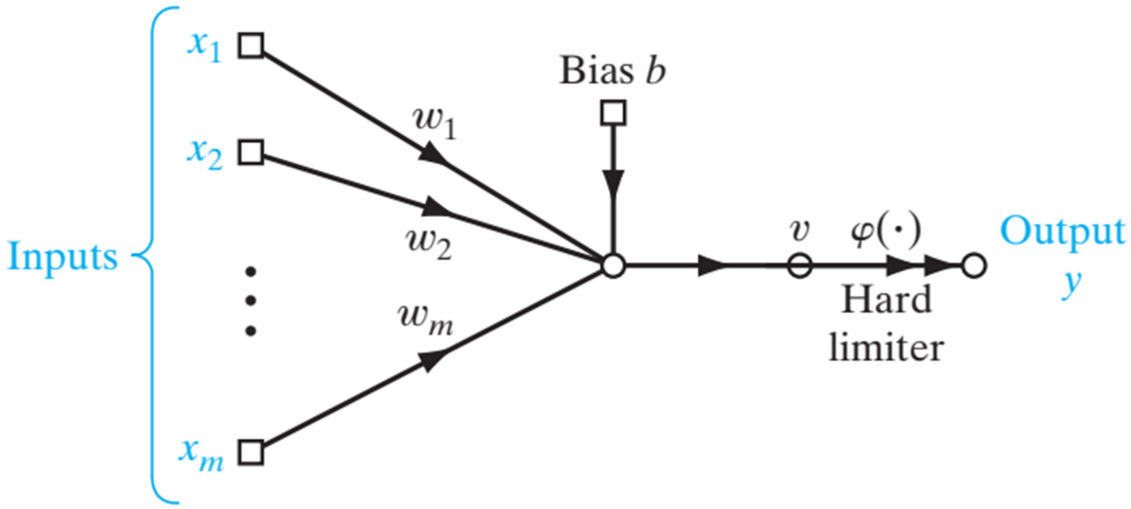


Fig. Signal flow graph of perceptron

From the model, we find that the hard limiter input, or induced local field, of the neuron is:

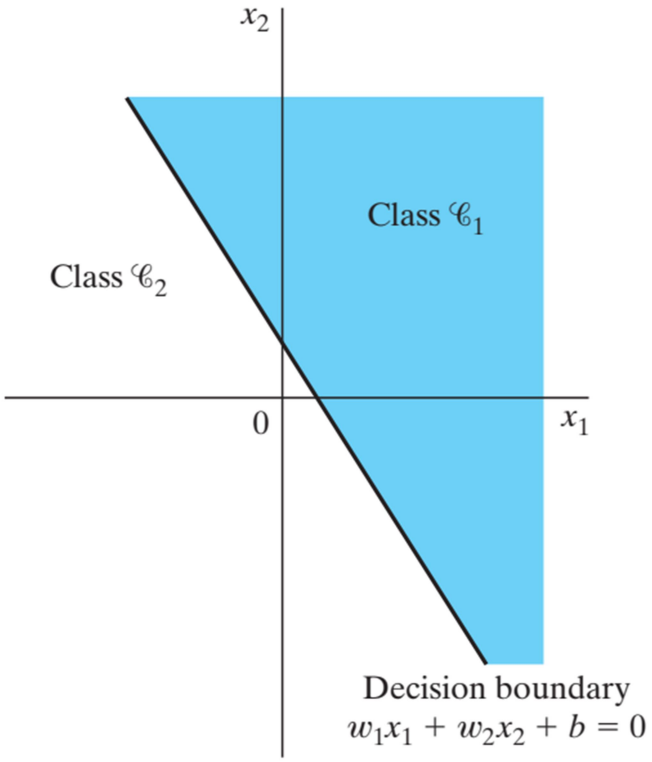
$$v = \sum_{i=1}^{m} w_i x_i + b$$

The goal of the perceptron is to correctly classify the set of externally applied stimuli x_1, $x_2, \ldots, x_m$ into one of two classes, $\mathscr{C}_1$ or $\mathscr{C}_2$. The decision rule for the classification is to assign the point represented by the inputs $x_1, x_2, \ldots, x_m$ to class $\mathscr{C}_1$ if the perceptron output y is +1 and to class $\mathscr{C}_2$ if it is −1.

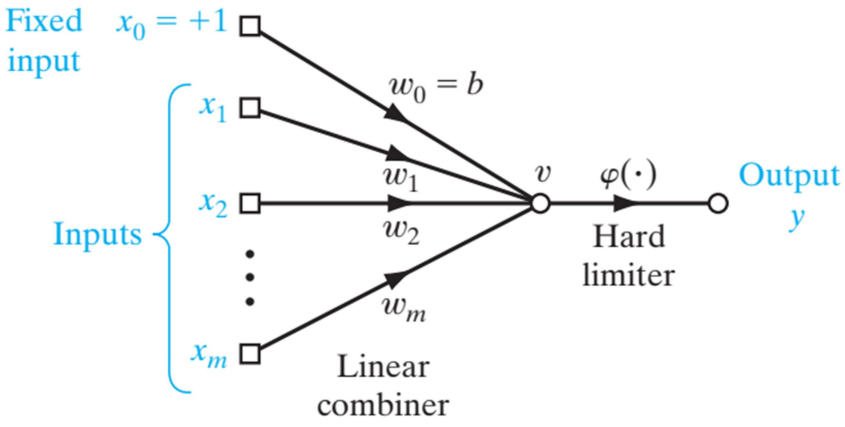
To develop insight into the behavior of a pattern classifier, it is customary to plot a map of the decision regions in the m-dimensional signal space spanned by the m input variables $x_1, x_2, \ldots, x_m$. In the simplest form of the perceptron, there are two decision regions separated by a *hyperplane*, which is defined by

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

This is illustrated in Fig. below for the case of two input variables x_1 and x_2, for which the decision boundary takes the form of a straight line. A point (x_1, x_2) that lies above the boundary line is assigned to class $\mathcal{C}_1$, and a point (x_1, x_2) that lies below the boundary line is assigned to class $\mathcal{C}_2$.



The synaptic weights $w_1, w_2, \ldots, w_m$ of the perceptron can be adapted on an iteration by-iteration basis. For the adaptation, we may use an error-correction rule known as the perceptron convergence algorithm. The equivalent signal flow graph can also be represented as:



The algorithm for adapting the weight vector of the elementary perceptron may now be formulated as follows:

1. If the nth member of the training set, $\mathbf{x}(n)$, is correctly classified by the weight vector $\mathbf{w}(n)$ computed at the nth iteration of the algorithm, no correction is made to the weight vector of the perceptron in accordance with the rule:

$$\mathbf{w}(n+1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2$$

2. Otherwise, the weight vector of the perceptron is updated in accordance with the rule

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1$$

where the *learning-rate parameter* $\eta(n)$ controls the adjustment applied to the weight vector at iteration n.

Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

$\mathbf{x}(n)$ = $(m+1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time step $n = 1, 2, \ldots$.

2. *Activation.* At time step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.

3. *Computation of Actual Response.* Compute the actual response of the perceptron:

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where sgn(·) is the signum function.

4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

Problems for practice:

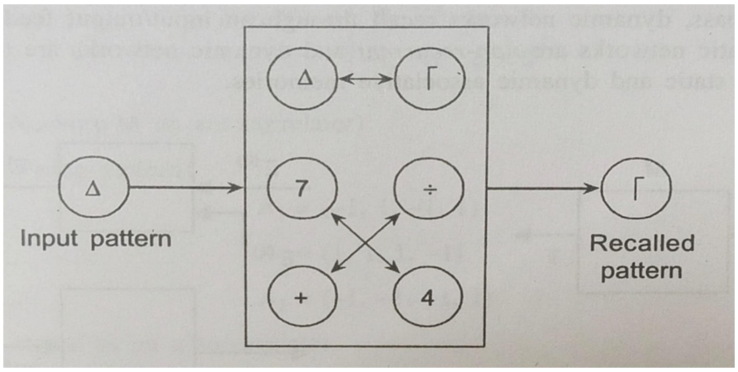
1. Learn the truth table of AND Gate using perceptron. Assume initial weights as $w_1 = 0.9$ and $w_2 = 0.9$, Assume learning rate=0.5, bias = 0.5.
2. Learn the truth table of OR Gate using perceptron. Assume initial weights as $w_1 = 0$ and $w_2 = 0$. Assume learning rate=0.5, bias = 0.5.

ASSSOCIATIVE MEMORY

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

An associate memory is a storehouse of associated patterns which are encoded in some form. When the storehouse is submitted with a pattern, the associated pattern pair is recalled or output. The input pattern could be an exact replica of the stored pattern or a distorted or partial

representation of a stored pattern. Figure shown below illustrates the working of an associative memory.



In the figure, (Δ, Γ), (7, 4), and (+, ÷) are associated pattern pairs. The associations represented using symbols are stored in the memory. When the memory is triggered for instance, with a Δ , the associated pattern is retrieved automatically.

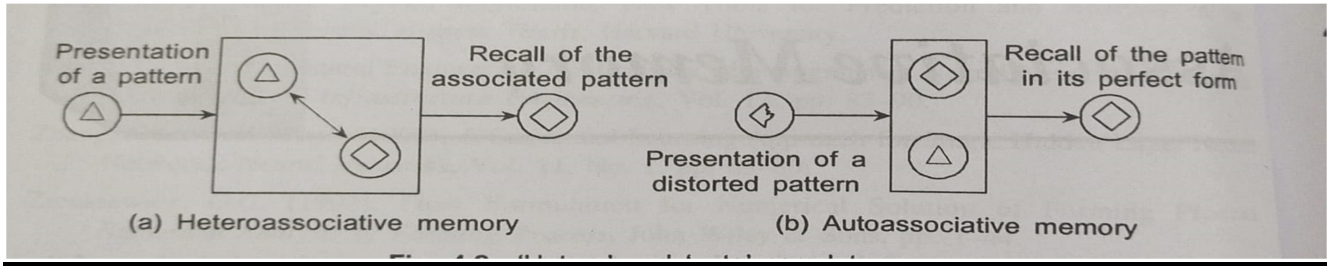
Hetero associative Memory

- If the associated pattern pairs (x. y) are different and if the model recalls a 'y' given an 'x' or vice versa, then it is termed as hetero associative memory.
- It is useful for the association of patterns
- Hetero associative correlation memories are known as hetero correlators.

Auto associative Memory

- If x and y refer to the same kind of pattern, then the model is termed as auto associative memory.
- Auto associative memories are useful for image refinement, that is, given a distorted or a partial pattern, the whole pattern stored in its perfect form can be recalled.
- Auto associative correlation memories are known as auto correlators.

Figure shown below illustrates hetero associative and auto associative memories.



Additional Questions

1. Differentiate between recurrent neural network and multilayer neural network

Recurrent neural network	Multilayer neural network
• Data and calculations flow in backward direction, from the output to the input layer.	Data and calculations flow in a single direction, from input data to the outputs.
• It contains feedback links.	It does not contain feedback links.
• It is used for text data, speech data.	It is used for image data, time series data.

2. Differentiate between Artificial Neural Network (ANN) and Biological Neural Network (BNN).

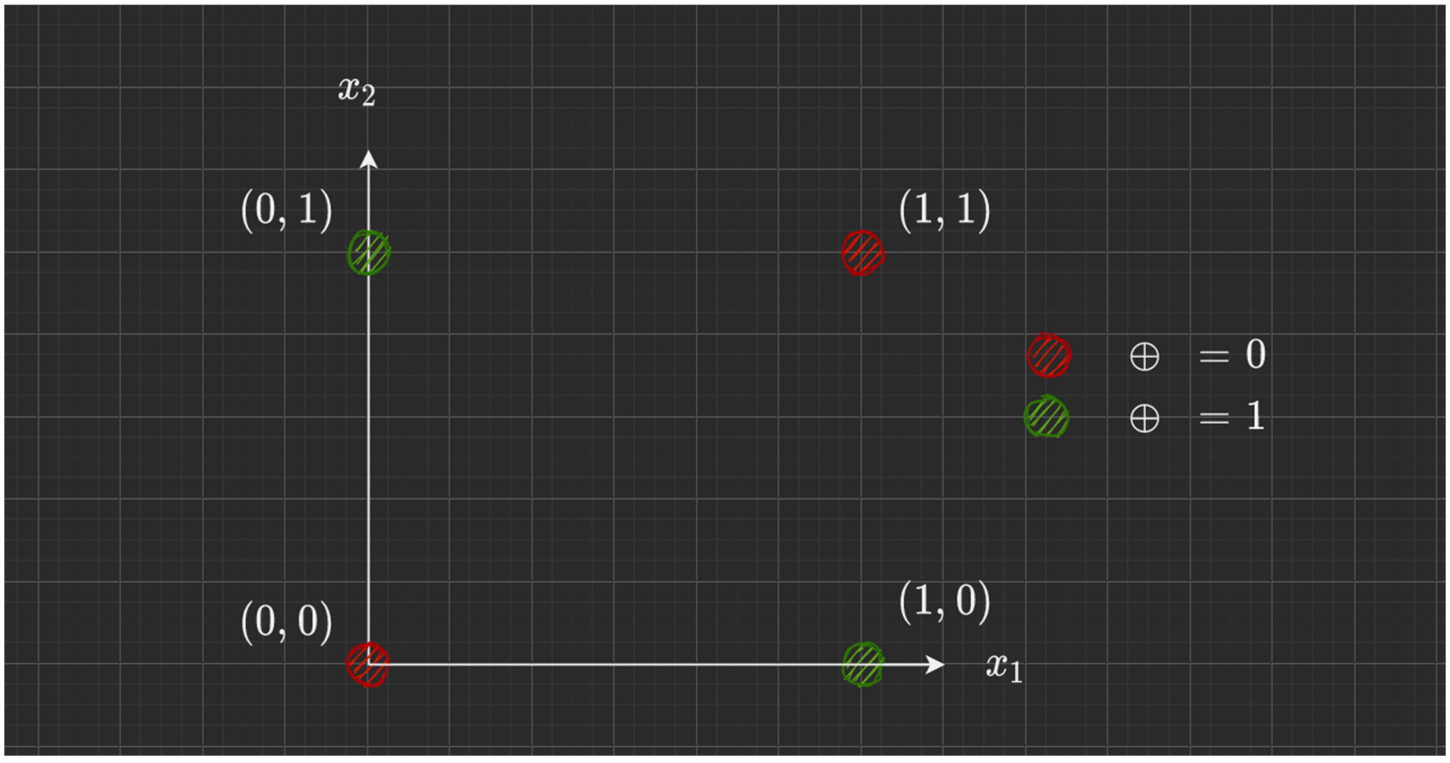
ANN	BNN
Processing speed is fast as compared to Biological Neural Network.	They are slow in processing information.
Allocation for Storage to a new process is strictly irreplaceable as the old location is saved for the previous process.	Allocation for storage to a new process is easy as it is added just by adjusting the interconnection strengths.
Processes operate in sequential mode.	The process can operate in massive parallel operations.
If any information gets corrupted in the memory it cannot be retrieved.	Information is distributed into the network throughout into sub-nodes, even if it gets corrupted it can be retrieved.
The activities are continuously monitored by a control unit.	There is no control unit to monitor the information being processed into the network.

3. Differentiate between Supervised and Unsupervised Learning.

SUPERVISED	UNSUPERVISED
• Uses Known and Labeled Data as input	Uses Unknown Data as input
• Very Complex	Less Computational Complexity
• Uses off-line analysis	Uses Real Time Analysis of Data
• Accurate and Reliable Results	Moderate Accurate and Reliable Results

4. Can single layer perceptron learn XOR Logic?

The classes in XOR are not linearly separable. It will not be possible for you to draw a straight line to separate the points (0,0),(1,1) from the points (0,1),(1,0). Single layer perceptron can only learn linearly separable patterns.



Therefore, a single layer perceptron does not have the ability to implement XOR. This gave rise to the need for and the invention of multilayer networks and perceptron.