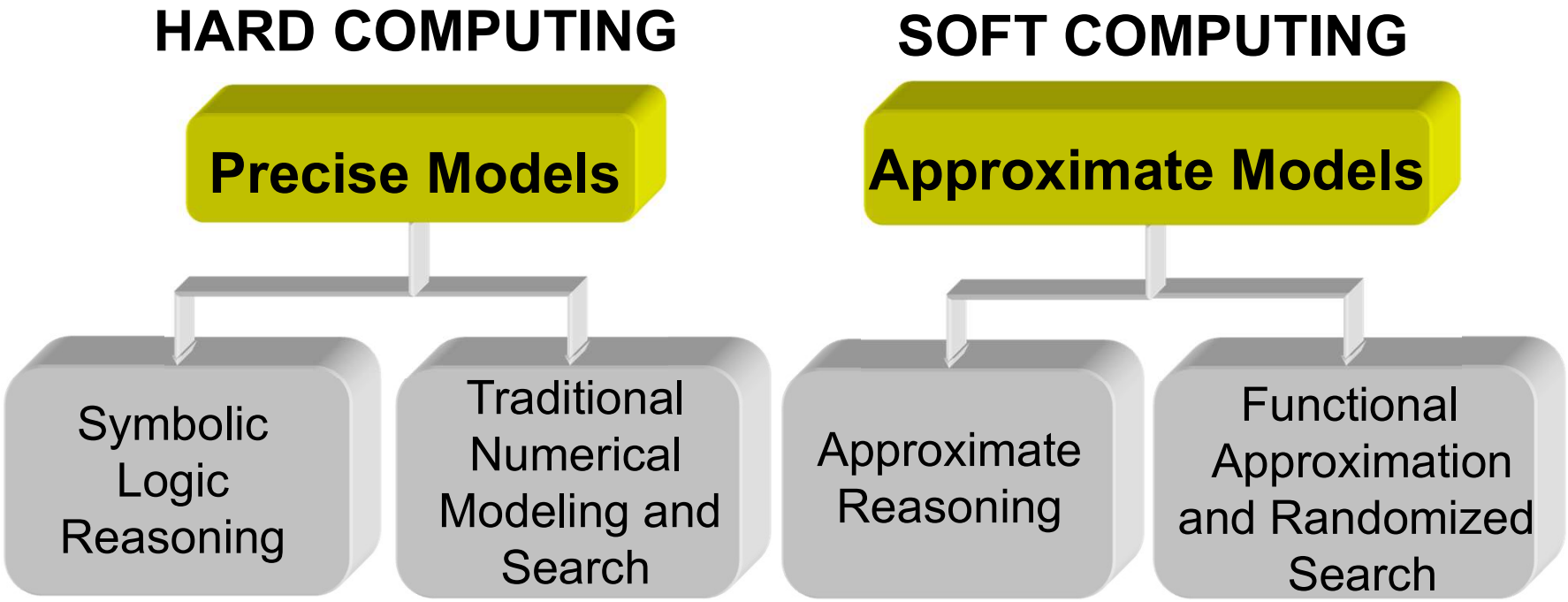
Application of Soft Computing

Unit 1

PROBLEM SOLVING TECHNIQUES



Soft Computing

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

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.

The main computing paradigm of soft computing are:

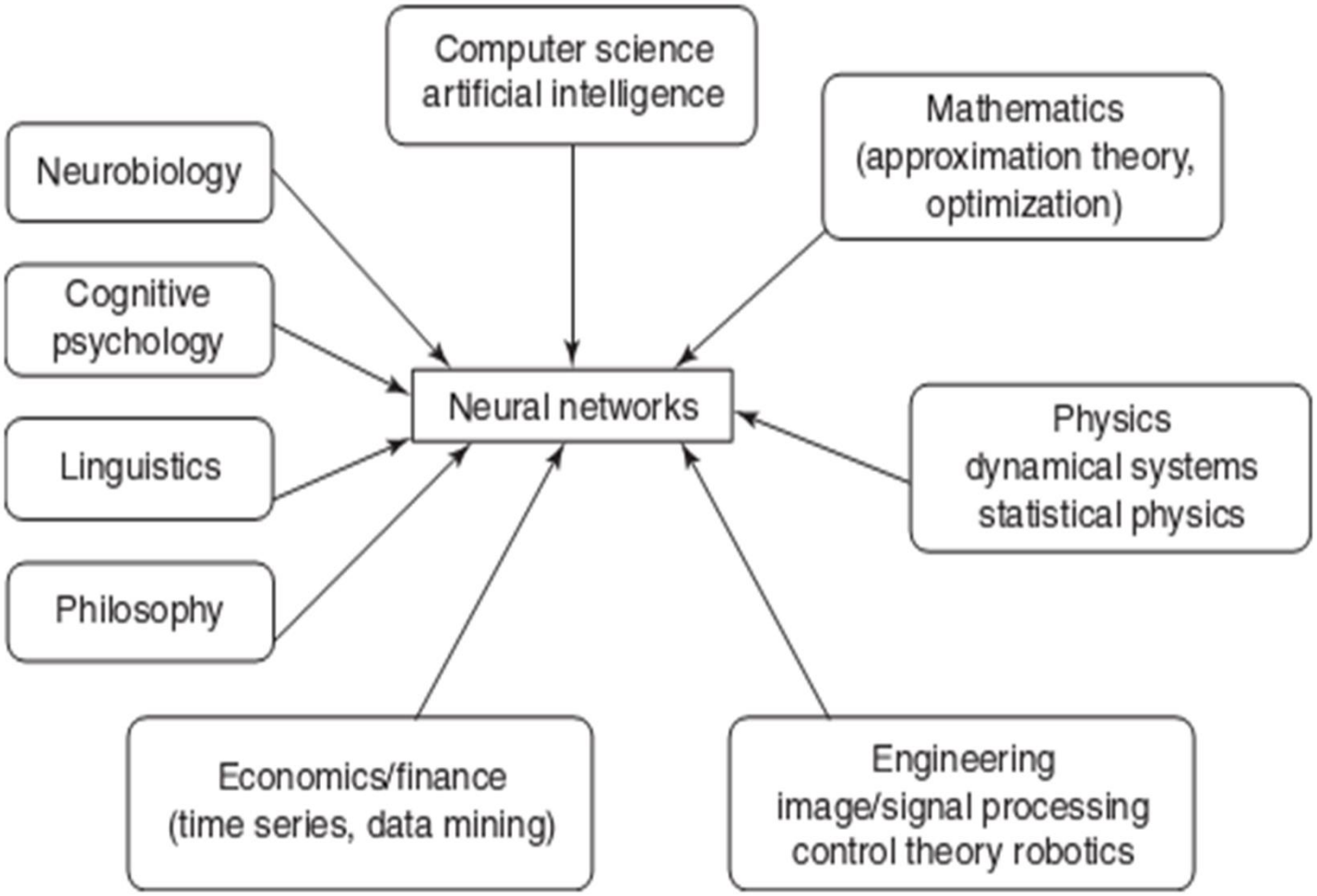
- Neural Networks
- Fuzzy systems
- Genetic Algorithms

Neural network for learning and adaptivity.

Fuzzy set are for knowledge representation via fuzzy If – Then rules.

Genetic algorithm for evolutionary computation.

MULTIDISCIPLINARY VIEW OF NEURAL NETWORKS



FUZZY LOGIC

- **Origins: Multivalued Logic for treatment of imprecision and vagueness**
  - 1930s: Post, Kleene, and Lukasiewicz attempted to represent *undetermined*, *unknown,* and other possible intermediate truth-values.
  - 1937: Max Black suggested the use of a *consistency profile* to represent vague (ambiguous) concepts.
  - 1965: Zadeh proposed a complete theory of fuzzy sets (and its isomorphic fuzzy logic), to represent and manipulate ill-defined concepts.

Fuzzy logic gives us a language (with syntax and local semantics) in which we can translate our qualitative domain knowledge.

GENETIC ALGORITHM

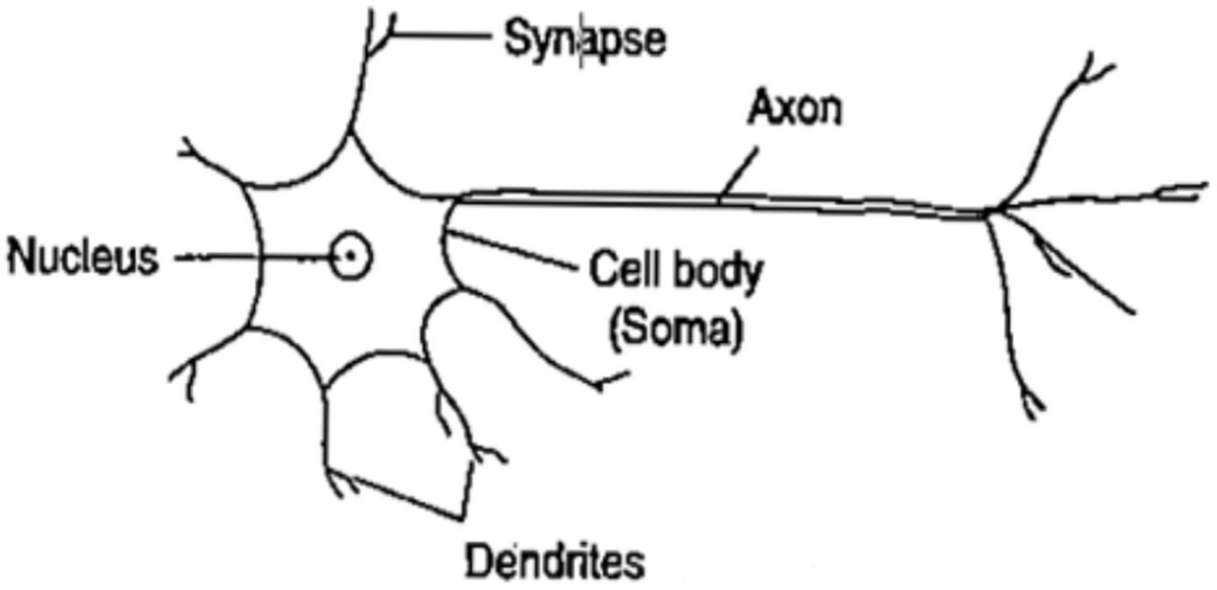
EVOLUTIONARY PROCESS

APPLICATIONS OF SOFT COMPUTING

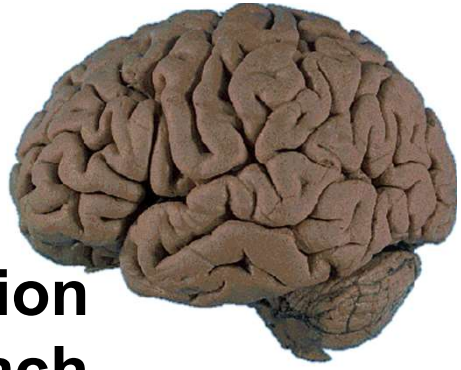
- **Handwriting Recognition**
- **Image Processing and Data Compression**
- **Automotive Systems and Manufacturing**
- **Soft Computing to Architecture**
- **Decision-support Systems**
- **Soft Computing to Power Systems**
- **Neuro Fuzzy systems**
- **Fuzzy Logic Control**
- **Machine Learning Applications**
- **Speech and Vision Recognition Systems**
- **Process Control and So On**

ANN: Inspired from Biological Neural Network

Biological Neurons



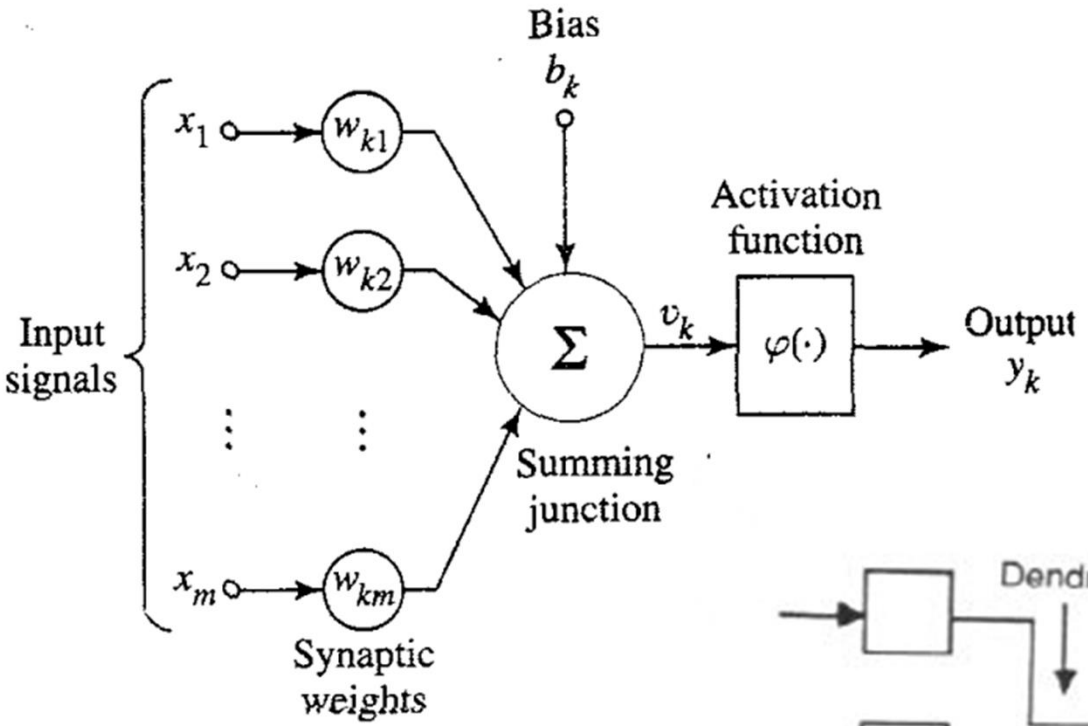
BRAIN COMPUTATION

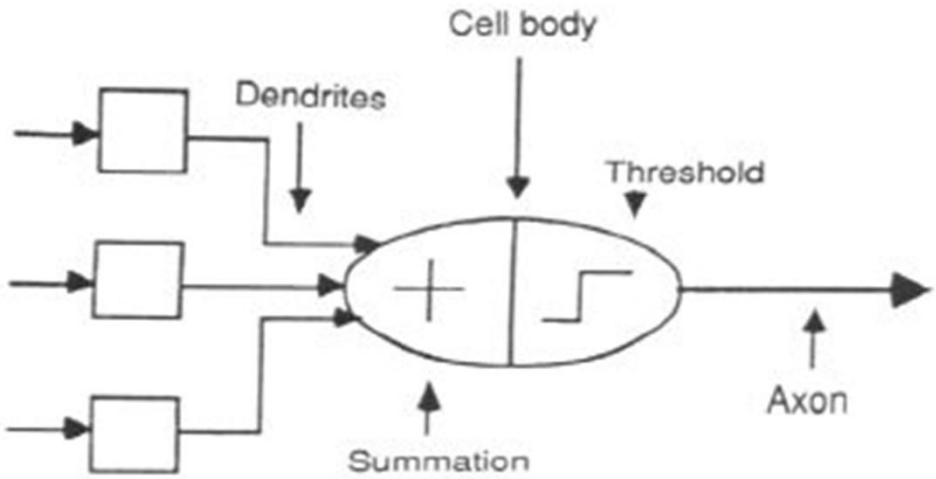


The human brain contains about 10 billion nerve cells, or neurons. On average, each neuron is connected to other neurons through approximately 10,000 synapses.

	processing elements	element size	energy use	processing speed	style of computation	fault tolerant	learns	intelligent, conscious
	10^{14} synapses	10^{-6} m	30 W	100 Hz	parallel, distributed	yes	yes	usually
	10^{8} transistors	10^{-6} m	30 W (CPU)	10^{9} Hz	serial, centralized	no	a little	not (yet)

Model of an ANN





ASSOCIATION OF BIOLOGICAL NET WITH ARTIFICIAL NET

In mathematical terms, we may describe a neuron k by writing the following pair of equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

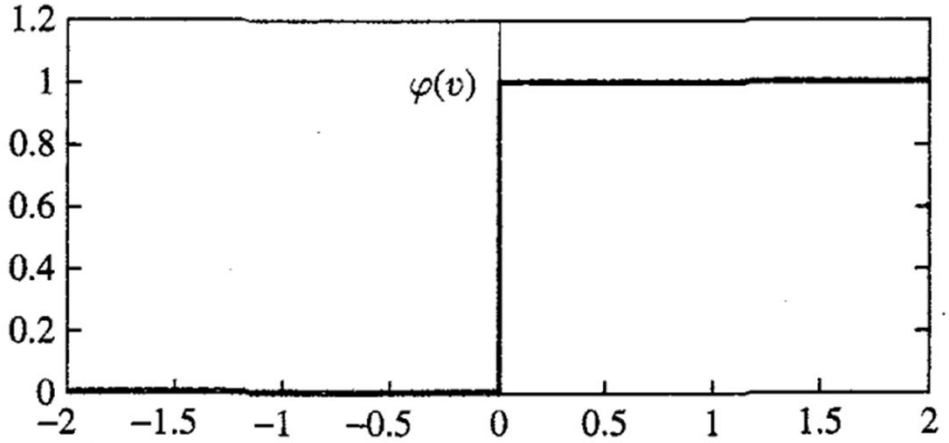
$$y_k = \varphi(u_k + b_k)$$

$$v_k = \sum_{j=0}^{m} w_{kj} x_j$$

$$y_k = \varphi(v_k)$$

Types of Activation Function

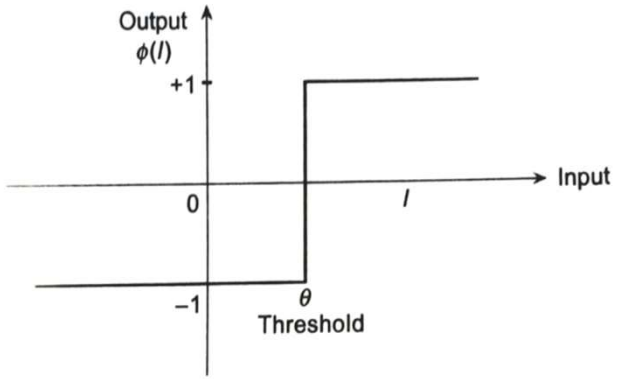
1. *Threshold Function*



$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

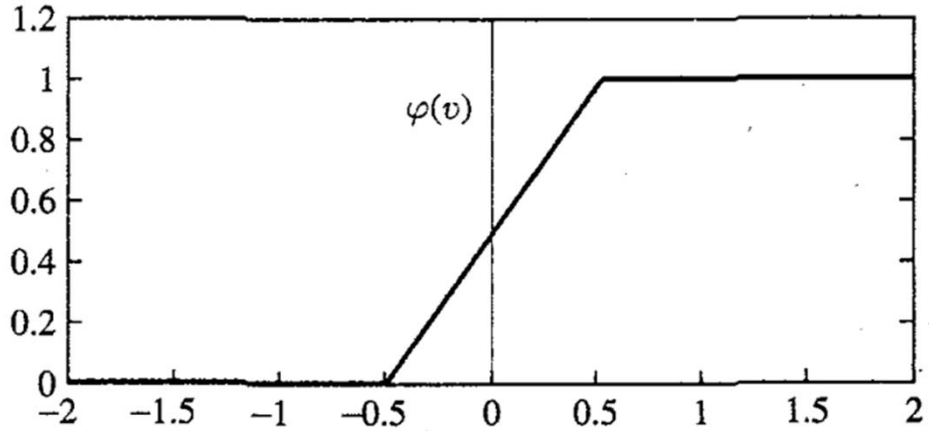
$$y_k = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases}$$

signum function



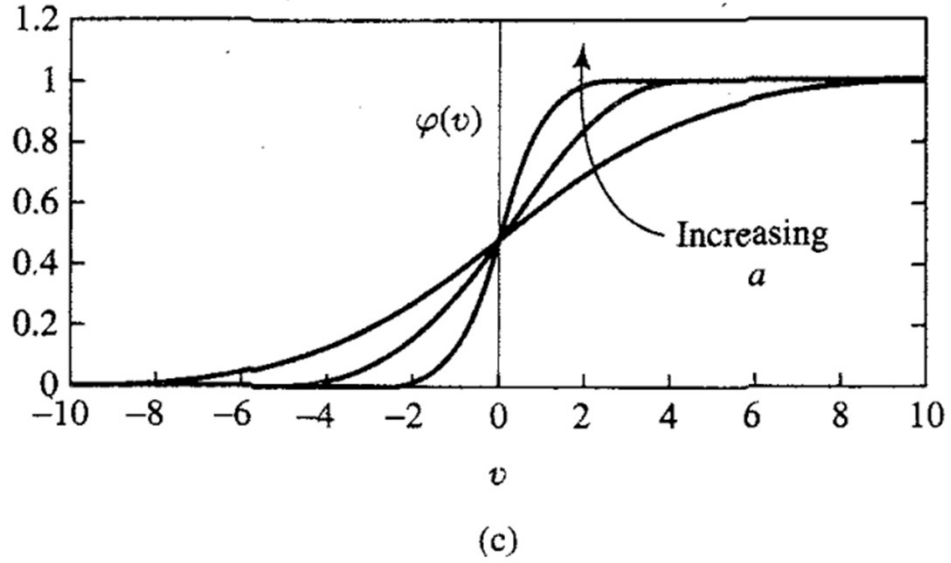
$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases}$$

Piecewise-Linear Function



$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2} \\ v, & +\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases}$$

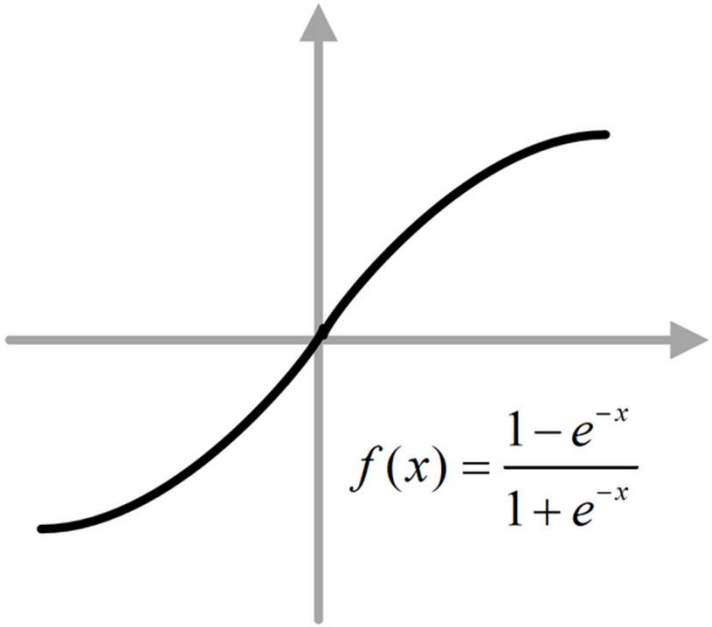
Sigmoïd Function



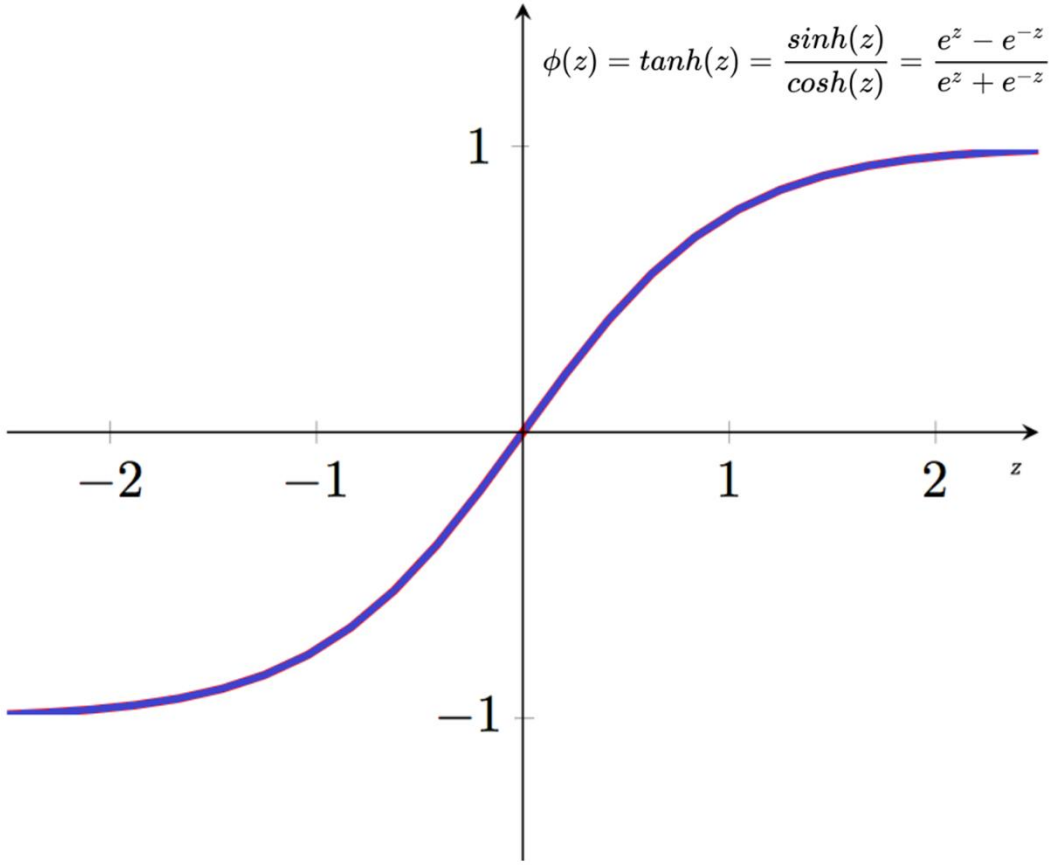
(c)

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

where a is the *slope parameter* of the sigmoid function.



Bipolar Sigmoid

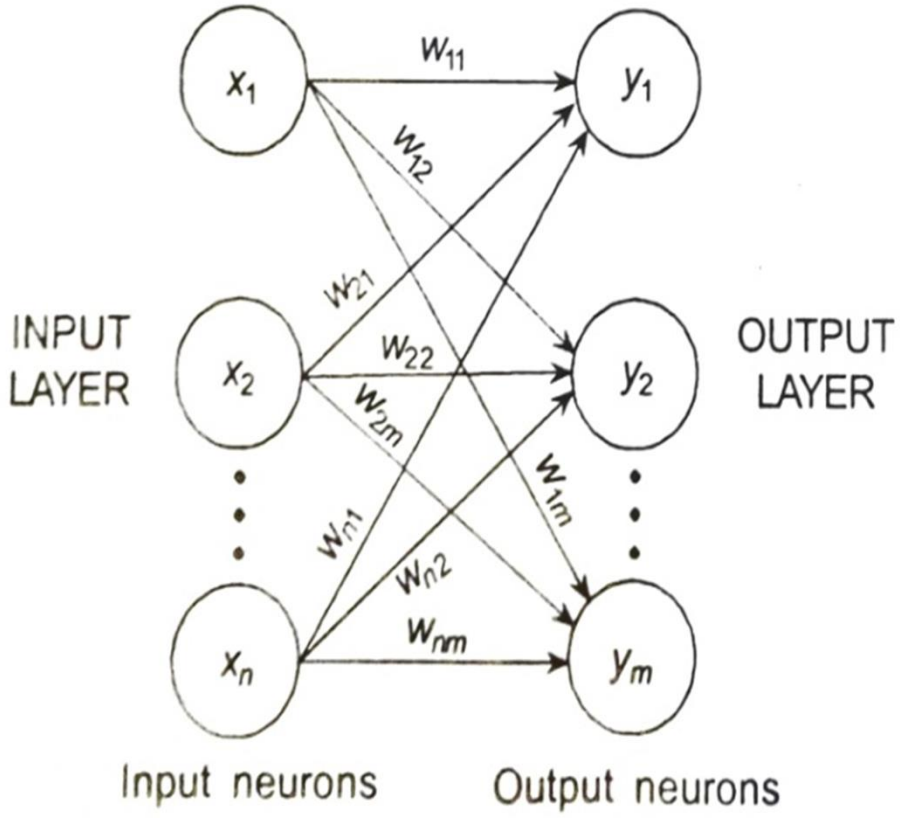


hyperbolic tangent function

$$\varphi(v) = \tanh(v)$$

NETWORK ARCHITECTURES

1. Single-Layer Feedforward Networks

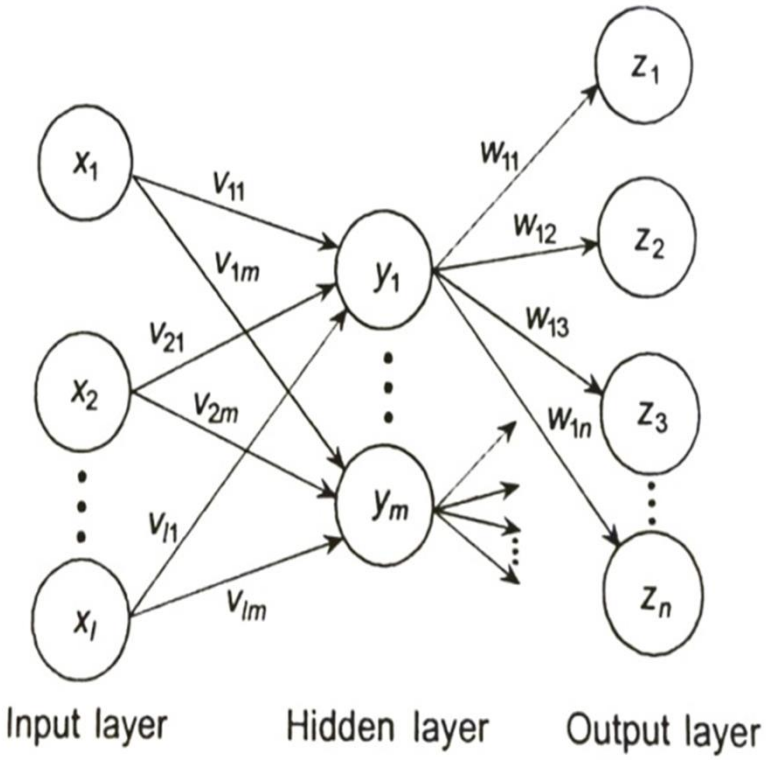


x_i: Input neurons

y_j: Output neurons

w_{ij}: Weights

MULTILAYER FEED FORWARD NETWORK



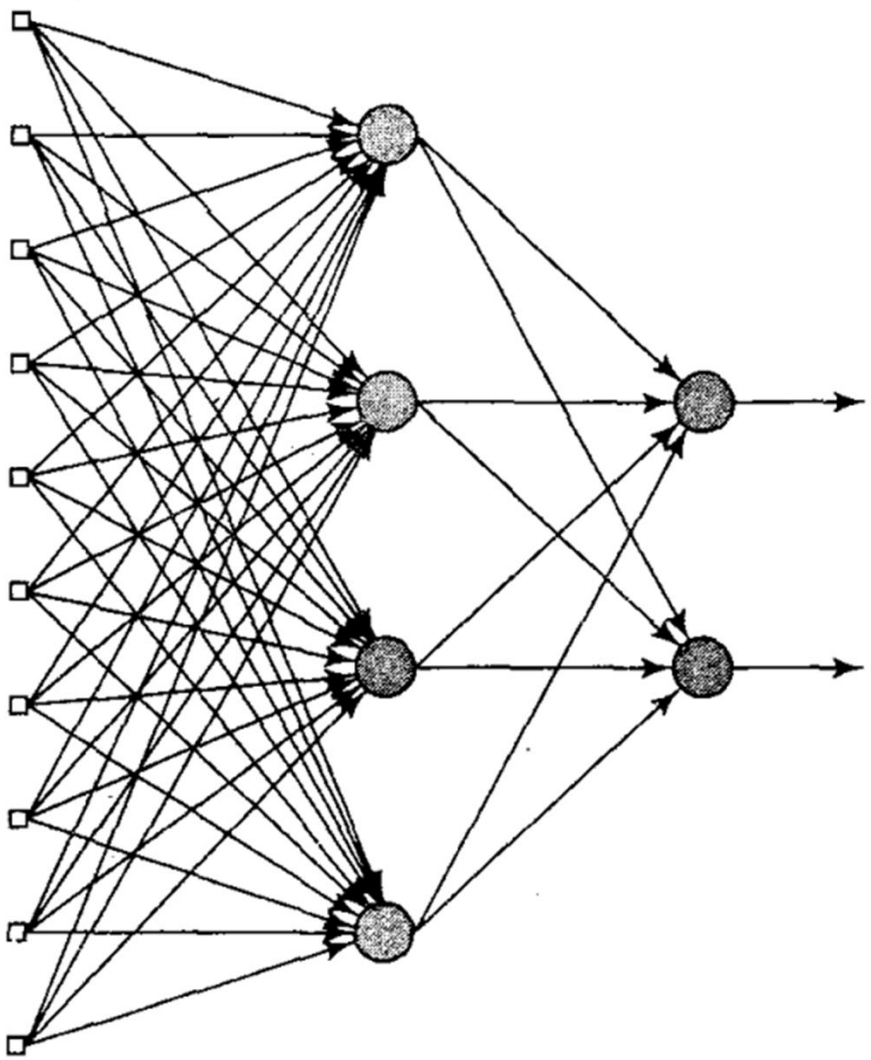
x_i : Input neurons

y_j : Hidden neurons

z_k : Output neurons

v_{ij} : Input hidden layer weights

w_{jk} : Output hidden layer weights



Input layer of source nodes

Layer of hidden neurons

Layer of output neurons

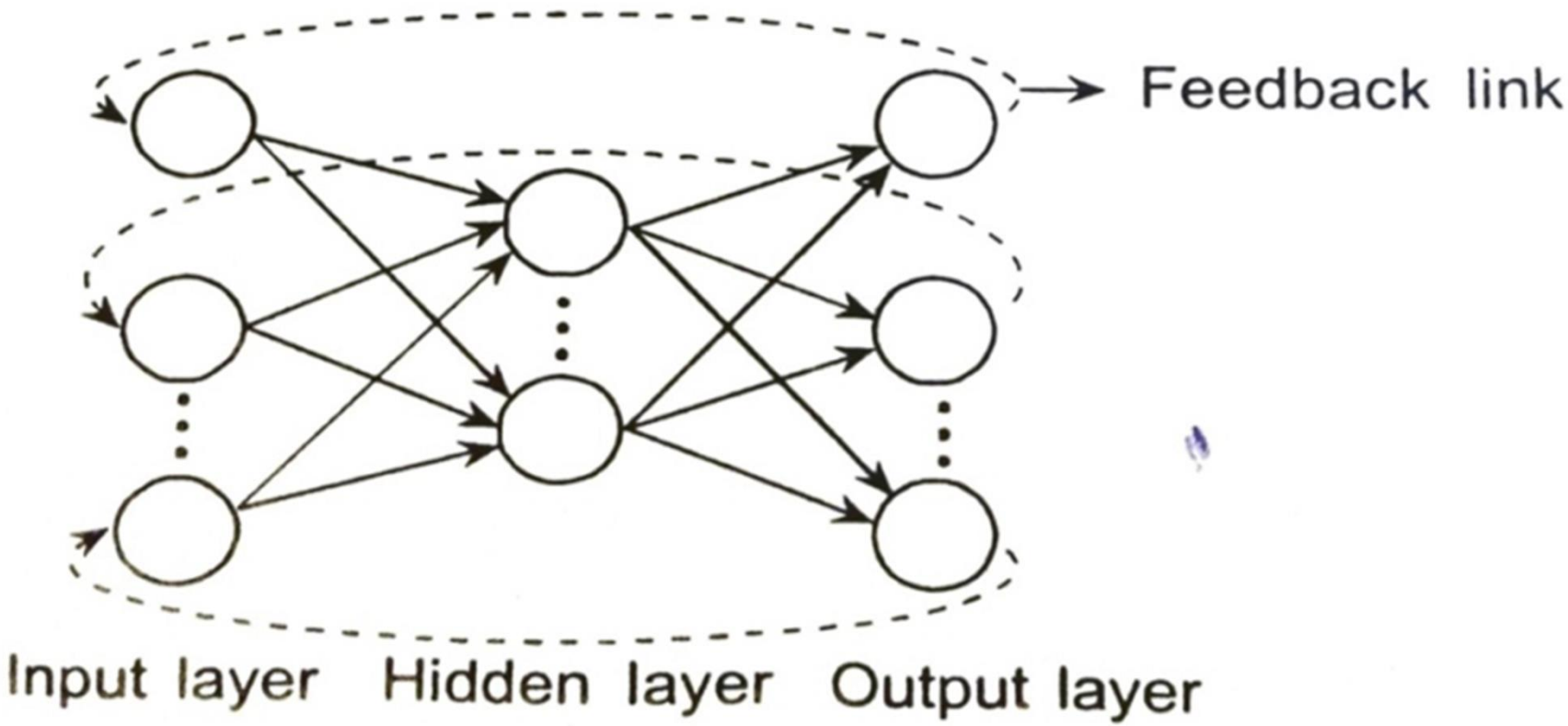
LAYER PROPERTIES

- **Input Layer:** Each input unit may be designated by an attribute value possessed by the instance.

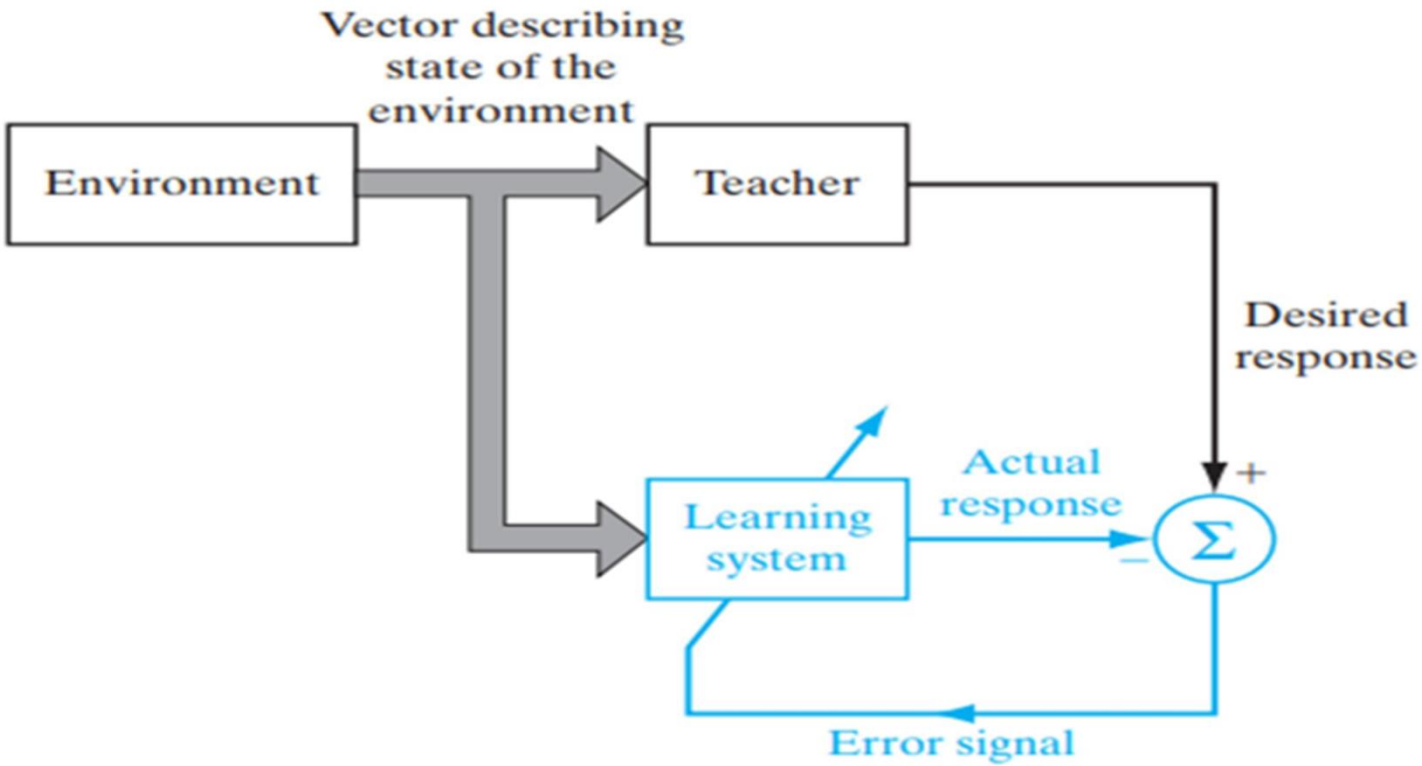
- **Hidden Layer:** Not directly observable, provides nonlinearities for the network.

- **Output Layer:** Encodes possible values.

FEEDBACK OR RECURRENT NEURAL NETWORK

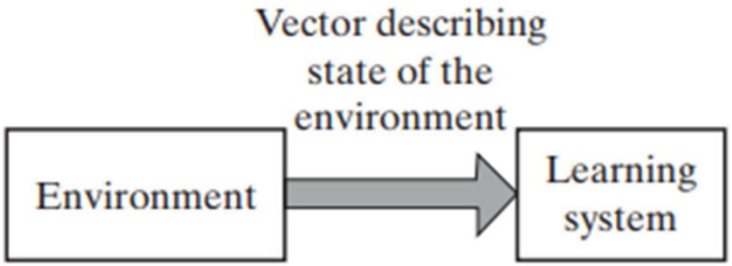


1. Supervised learning.



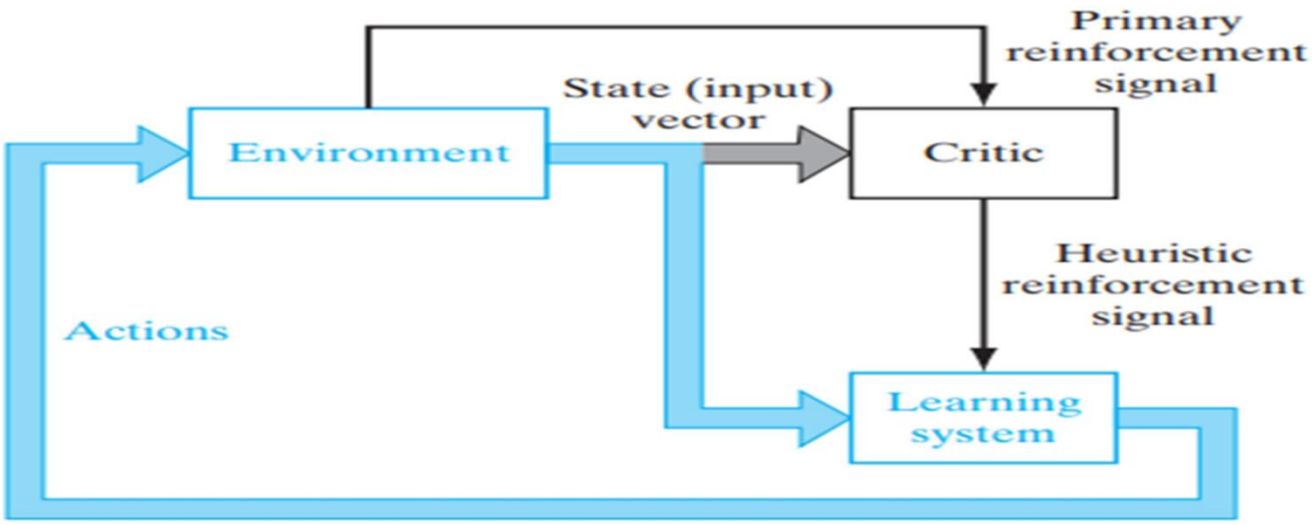
2. Unsupervised Learning

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



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**.**

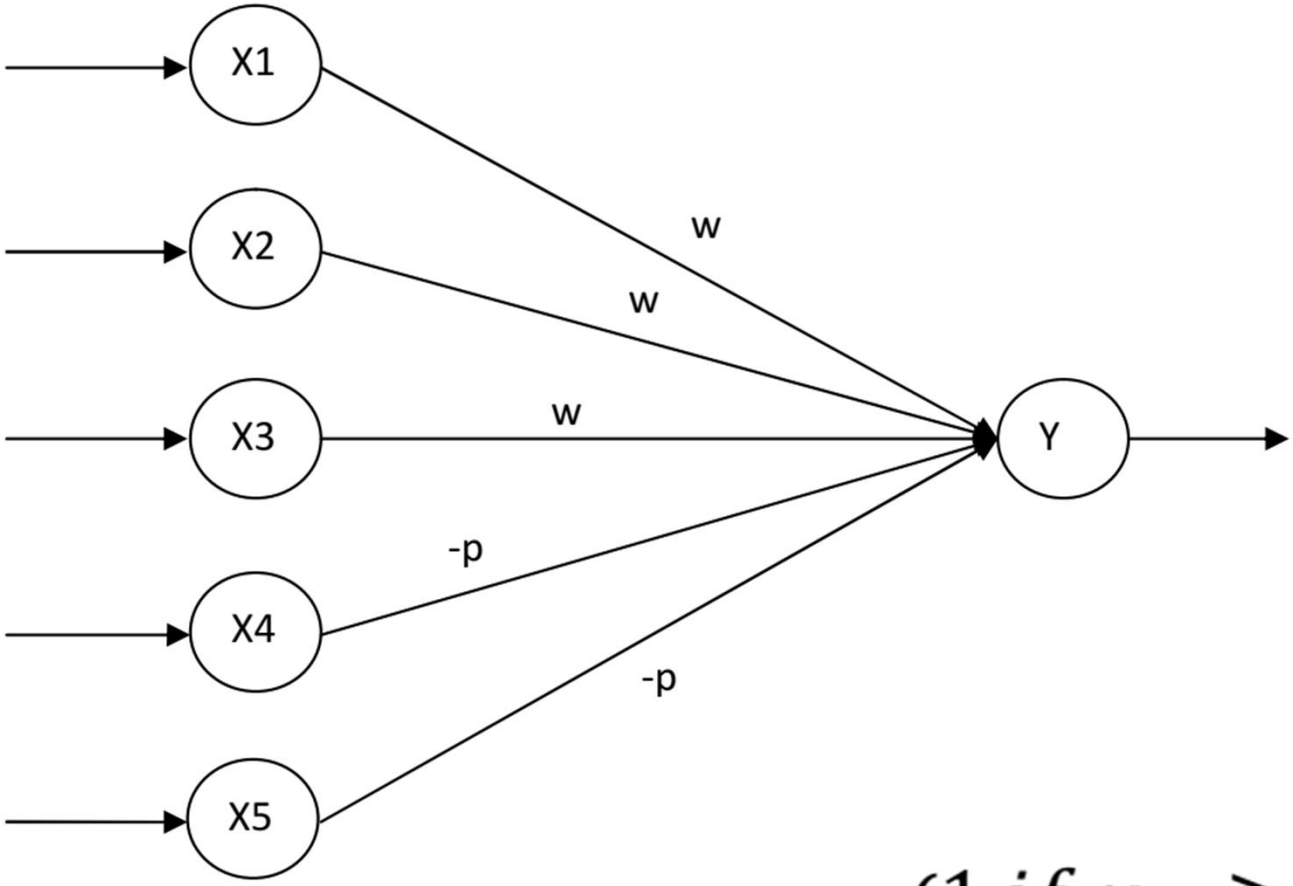
3. **Forecasting and risk assessment**

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
if the net input to the neuron is greater than the threshold then the neuron fir



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

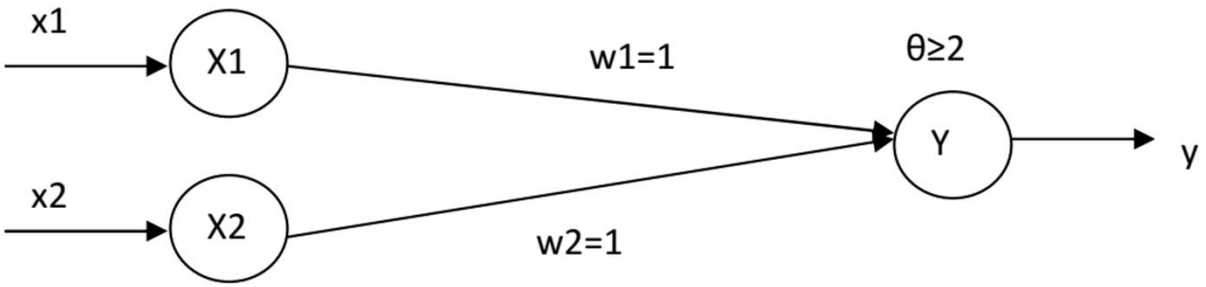
The M-P neuron has no particular training algorithm. An analysis has to be performed m determine the values of the weights and the threshold. Here the weights of the neuron are set along with the threshold to make the neuron perform a simple logic function. The M-P neurons are used as building blocks on which we can model any function or phenomenon, which can be represented as a logic function.

In M-P Neuron model, our basic assumptions are:

- Here we do only analysis, Threshold will work as activation function
- Weights can be excitatory (+ve) or inhibitory (-ve)
- Possible combinations are (both positive) (1 positive 1 negative) (both negative)
- We will take the value of weights 1
- So possible weights are (+1, +1) (+1, -1) (-1, +1) (-1, -1)
- We will use one from these combinations and try to set the threshold for solving the problem.

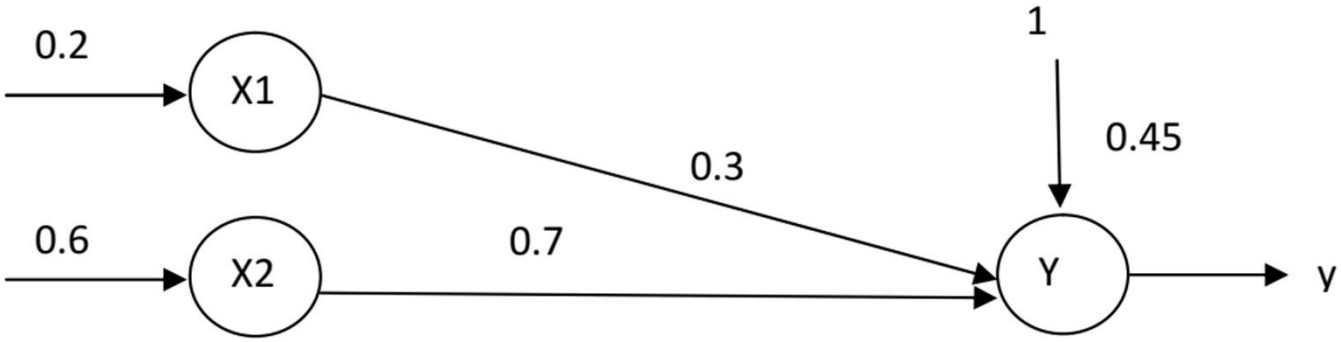
Implement AND function using McCulloch-Pitts neuron model.

The truth table for AND function is



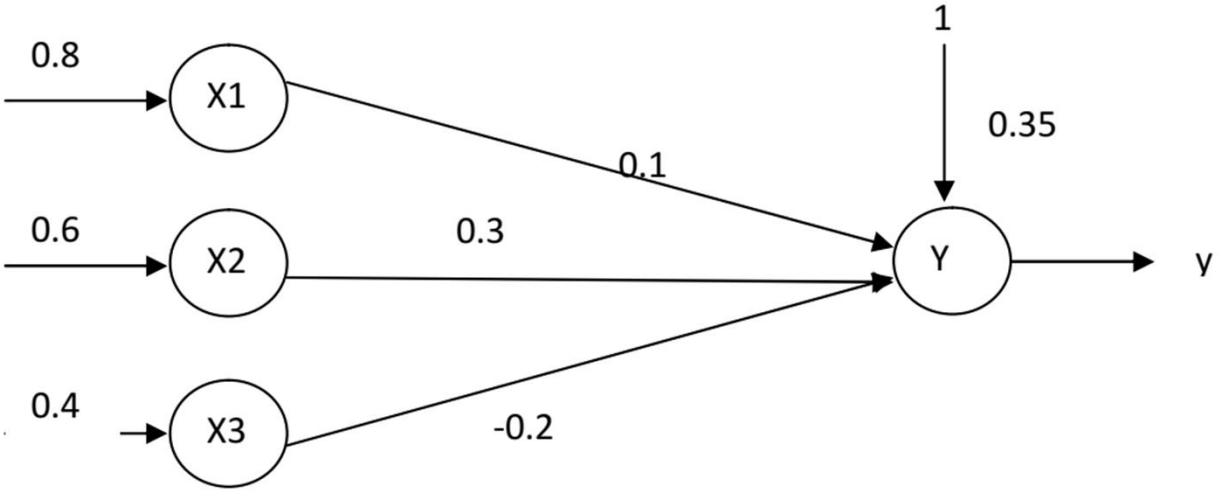
X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

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



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

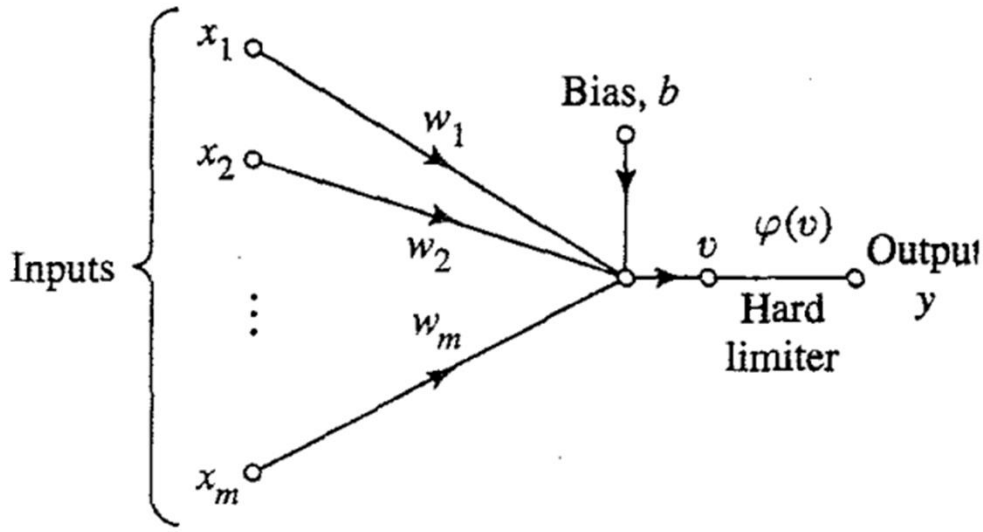


Perceptron

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.[1] Indeed, Rosenblatt proved 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*. The perceptron built around a *single neuron* is limited to performing pattern classification with only two classes (hypotheses).

Signal-flow graph of the perceptron

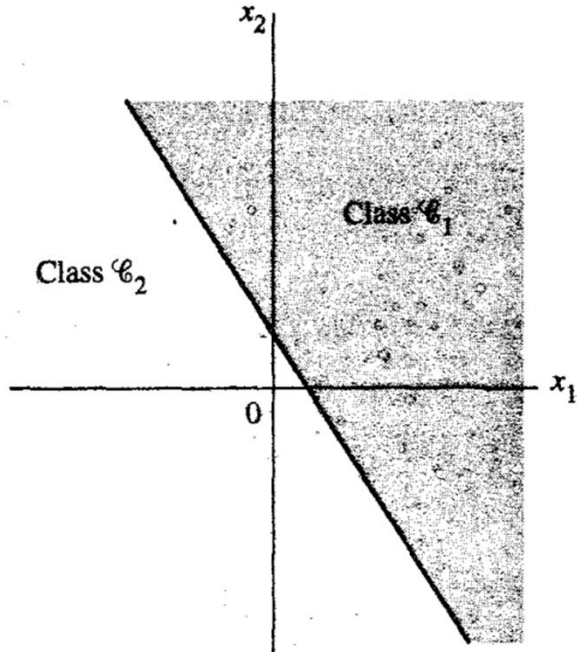
$$v = \sum_{i=1}^{m} w_i x_i + b$$



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.

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.

For the perceptron to function properly, the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ must be *linearly separable*. This, in turn, means that the patterns to be classified must be sufficiently separated from each other to ensure that the decision surface consists of a hyperplane.



Suppose then that the input variables of the perceptron originate from two linearly separable classes. Let $\mathscr{X}_1$ be the subset of training vectors $\mathbf{x}_1(1), \mathbf{x}_1(2), \ldots$ that belong to class $\mathscr{C}_1$, and let $\mathscr{X}_2$ be the subset of training vectors $\mathbf{x}_2(1), \mathbf{x}_2(2), \ldots$ that belong to class $\mathscr{C}_2$. The union of $\mathscr{X}_1$ and $\mathscr{X}_2$ is the complete training set $\mathscr{X}$. Given the sets of vectors $\mathscr{X}_1$ and $\mathscr{X}_2$ to train the classifier, the training process involves the adjustment of the weight vector $\mathbf{w}$ in such a way that the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ are linearly separable. That is, there exists a weight vector $\mathbf{w}$ such that we may state

$$\mathbf{w}^T\mathbf{x} > 0 \text{ for every input vector } \mathbf{x} \text{ belonging to class } \mathscr{C}_1$$

$$\mathbf{w}^T\mathbf{x} \leq 0 \text{ for every input vector } \mathbf{x} \text{ belonging to class } \mathscr{C}_2$$

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

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \end{aligned} \tag{3.55}$$

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

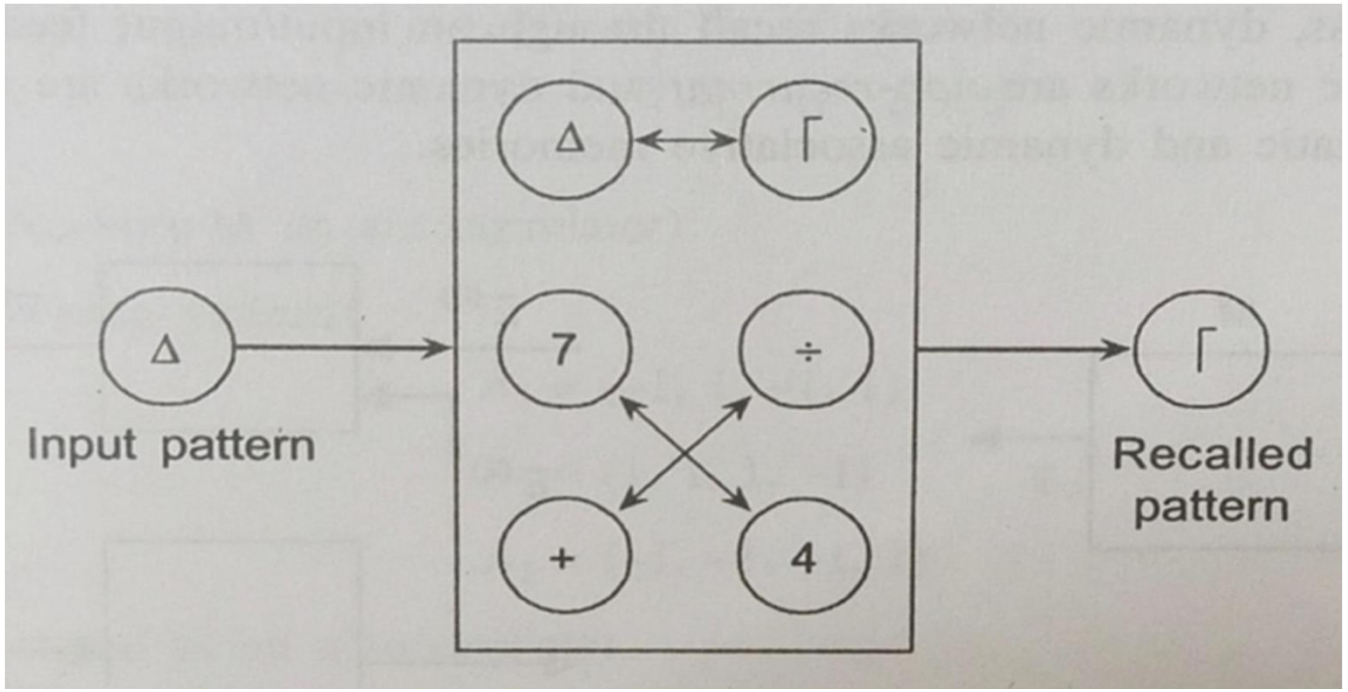
Suppose that we are going to work on AND Gate problem using perceptron. The gate returns true value if and only if both inputs are true.
We are going to set weights randomly. Let's say that $w_1 = 0.9$ and $w_2 = 0.9$. Learning rate=0.5, bias = 0.5

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

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 feedforward or recurrent network architecture depending on its association capability, exhibits Hebbian learning.

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

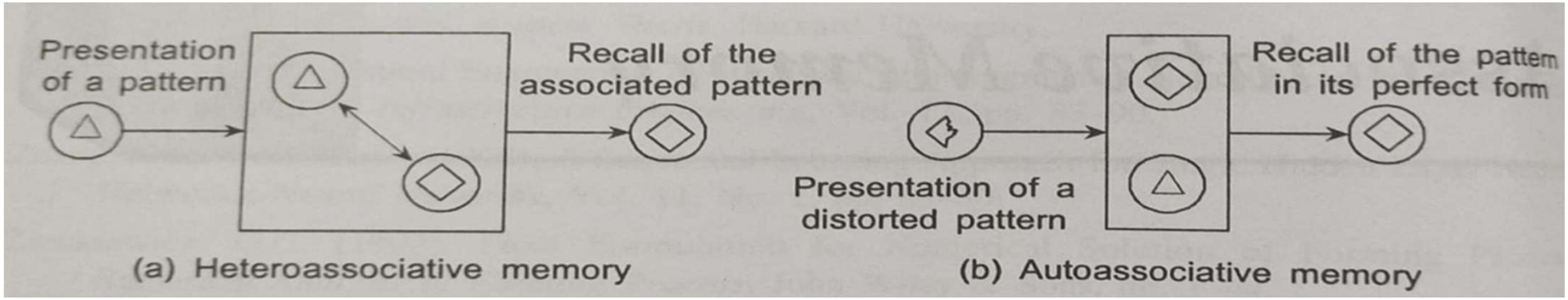


Heteroassociative Memory

- 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 heteroassociative memory.
- It is useful for the association of patterns
- Heteroassociative correlation memories are known as heterocorrelators.

Autoassociative Memory

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



HEBB NETWORK

Donald Hebb stated in 1949 that in the brain, the learning is performed by the change in the synaptic gap. Hebb explained it:

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

HEBB LEARNING

- The weights between neurons whose activities are positively correlated are increased:

$$\frac{dw_{ij}}{dt} \sim \text{correlation}(x_i, x_j)$$

- Associative memory is produced automatically

The Hebb rule can be used for pattern association, pattern categorization, pattern classification and over a range of other areas.

Hebbian Learning Algorithm

According to Hebb's rule, the weights are found to increase proportionately to the product of input and output. It means that in a Hebb network if two neurons are interconnected then the weights associated with these neurons can be increased by changes in the synaptic strength.

This network is suitable for bipolar data. The Hebbian learning rule is generally applied to logic gates.

The weights are updated as:

$$W(new) = w(old) + x*y$$

Training Algorithm For Hebbian Learning Rule

The training steps of the algorithm are as follows:

1. Initially, the weights are set to zero, i.e. w =0 for all inputs i =1 to n and n is the total number of input neurons.

2. Let s be the output. The activation function for inputs is generally set as an identity function (linear function of slope 1).

3. The activation function for output is also set to y= t.

4. The weight adjustments and bias are adjusted to:

$$W(n) = w(o) + x*y$$

$$B(n) = b(old) + y$$

The change is weights is $\Delta w = x*y$

$$w(n) = w(o) + \Delta w$$

•The steps 2 to 4 are repeated for each input vector and output.

Example Of Hebbian Learning Rule

Let us implement logical AND function with bipolar inputs using Hebbian Learning.
X1 and X2 are inputs, b is the bias taken as 1, the target value is the output of logical AND operation over inputs.

Input	Input	Bias	Target
X1	**X2**	**b**	**y**
1 (High)	1 (High)	1	1 (High)
1 (High)	-1 (Low)	1	-1 (Low)
-1 (Low)	1 (High)	1	-1 (Low)
-1 (Low)	-1 (Low)	1	-1 (Low)

#1) Initially, the weights are set to zero and bias is also set as zero.
$w_1=w_2=b=0$
#2) First input vector is taken as [x1 x2 b] = [1 1 1] and target value is 1.
The new weights will be:

W (new) = w (old) +x*y
W1 (n) = w1 (o) + x1*y = 0 + 1*1 = 1
W2 (n) = w2 (o) + x2*y = 0 + 1*1 = 1
B (n) =b (o) + y = 0+1 =1
The change in weights is: Δw1= x1 *y =1 Δw2= x2 *y =1Δb= y =1

#3) The above weights are the final new weights. When the second input is passed, these become the initial weights.

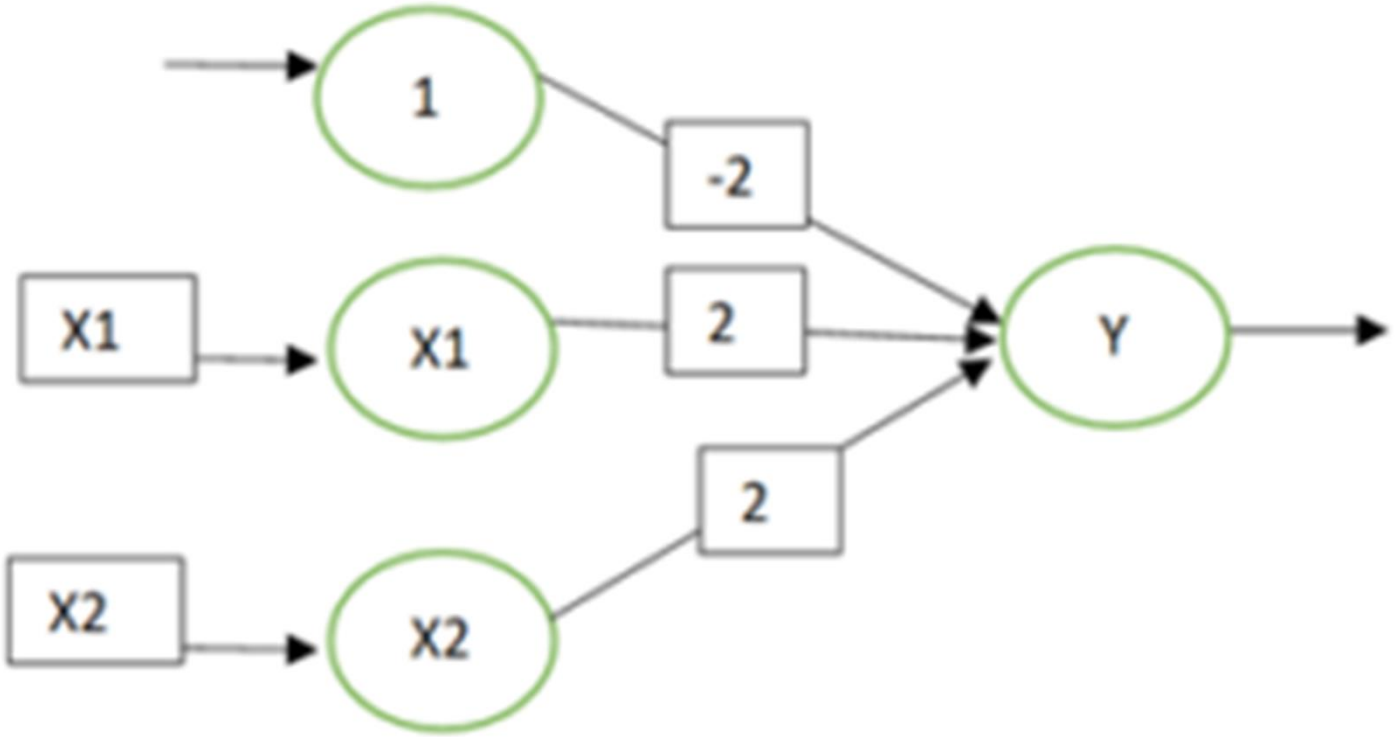
#4) Take the second input = [1 -1 1]. The target is -1.

The weights vector is = [w1 w2 b] = [1 1 1]
The change is weights is: Δw1= x1 *y =-1 Δw2= x2 *y =1 Δb= y =-1
The new weights will be w1 (n) = w1 + Δw1 => 1 + (-1) = 0
W2 (n) = w2 + Δw2 => 1 + (1) = 2
B (n) = b + Δb => 1 + (-1) = 0

#5) Similarly, the other inputs and weights are calculated.

Inputs		Bias	Target Output	Weight Changes		Bias Changes	New Weights		
X1	X2	b	y	Δw1	Δw2	Δb	W1	W2	b
1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	0	2	0
-1	1	1	-1	1	-1	-1	1	1	-1
-1	-1	1	-1	1	1	-1	2	2	-2

Hebb Net for AND Function



FEW APPLICATION AREAS OF NEURAL NETWORKS

- Aerospace
- Automotive
- Banking
- Credit Card Activity Checking
- Defense
- Electronics
- Entertainment
- Financial
- Industrial
- Insurance
- Insurance
- Manufacturing
- Medical
- Oil and Gas
- Robotics
- Speech
- Securities
- Telecommunications
- Transportation