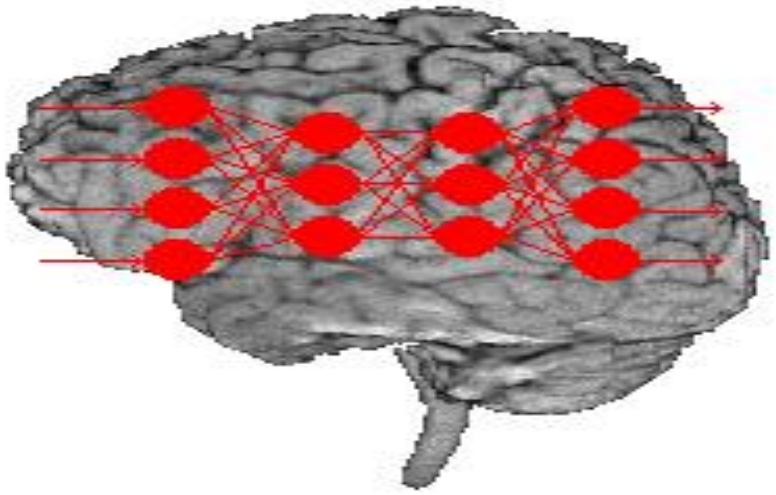
AMITY UNIVERSITY

Amity School of Engineering
& Technology

INTRODUCTION TO ARTIFICIAL NEURAL NETWORK



DEFINITION OF NEURAL NETWORKS

According to the DARPA Neural Network Study (1988, AFCEA International Press, p. 60):

• ... a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

According to Haykin (1994), p. 2:

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

- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the knowledge.

BRAIN COMPUTATION

The **human brain** contains approximately 100 billion nerve cells, or neurons. On average, each neuron is connected to other neurons through approximately 1000 to 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)

Artificial Neural Network

- An artificial neural network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections.

Artificial Neural Network

A set of major aspects of a parallel distributed model include:

- a set of processing units (cells).
- a state of activation for every unit, which equivalent to the output of the unit.
- connections between the units. Generally each connection is defined by a weight.
- a propagation rule, which determines the effective input of a unit from its external inputs.
- an activation function, which determines the new level of activation based on the effective input and the current activation.
- an external input for each unit.
- a method for information gathering (the learning rule).
- an environment within which the system must operate, providing input signals and if necessary error signals.

Why Artificial Neural Networks?

There are two basic reasons why we are interested in building artificial neural networks (ANNs):

- **Technical viewpoint:** Some problems such as character recognition or the prediction of future states of a system require massively parallel and adaptive processing.

- **Biological viewpoint:** ANNs can be used to replicate and simulate components of the human (or animal) brain, thereby giving us insight into natural information processing.

Artificial Neural Networks

- The "building blocks" of neural networks are the **neurons**.
  In technical systems, we also refer to them as **units** or **nodes**.

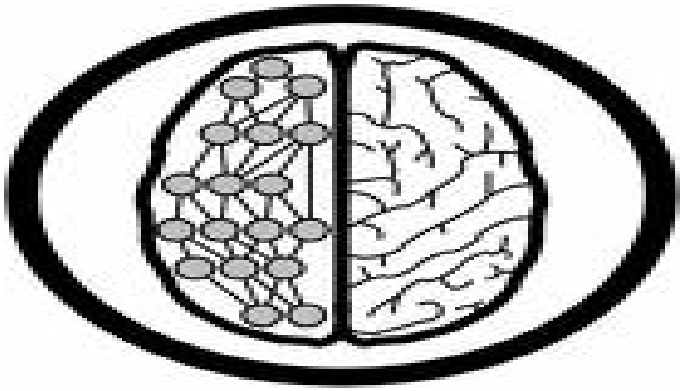
- Basically, each neuron
  - receives **input** from many other neurons.
  - changes its internal state (**activation**) based on the current input.
  - sends **one output signal** to many other neurons, possibly including its input neurons (recurrent network).

Artificial Neural Networks

- Information is transmitted as a series of electric impulses, so-called **spikes**.
- The **frequency** and **phase** of these spikes encodes the information.
- In biological systems, one neuron can be connected to as many as **10,000** other neurons.
- Usually, a neuron receives its information from other neurons in a confined area, its so-called **receptive field**.

How do ANNs work?

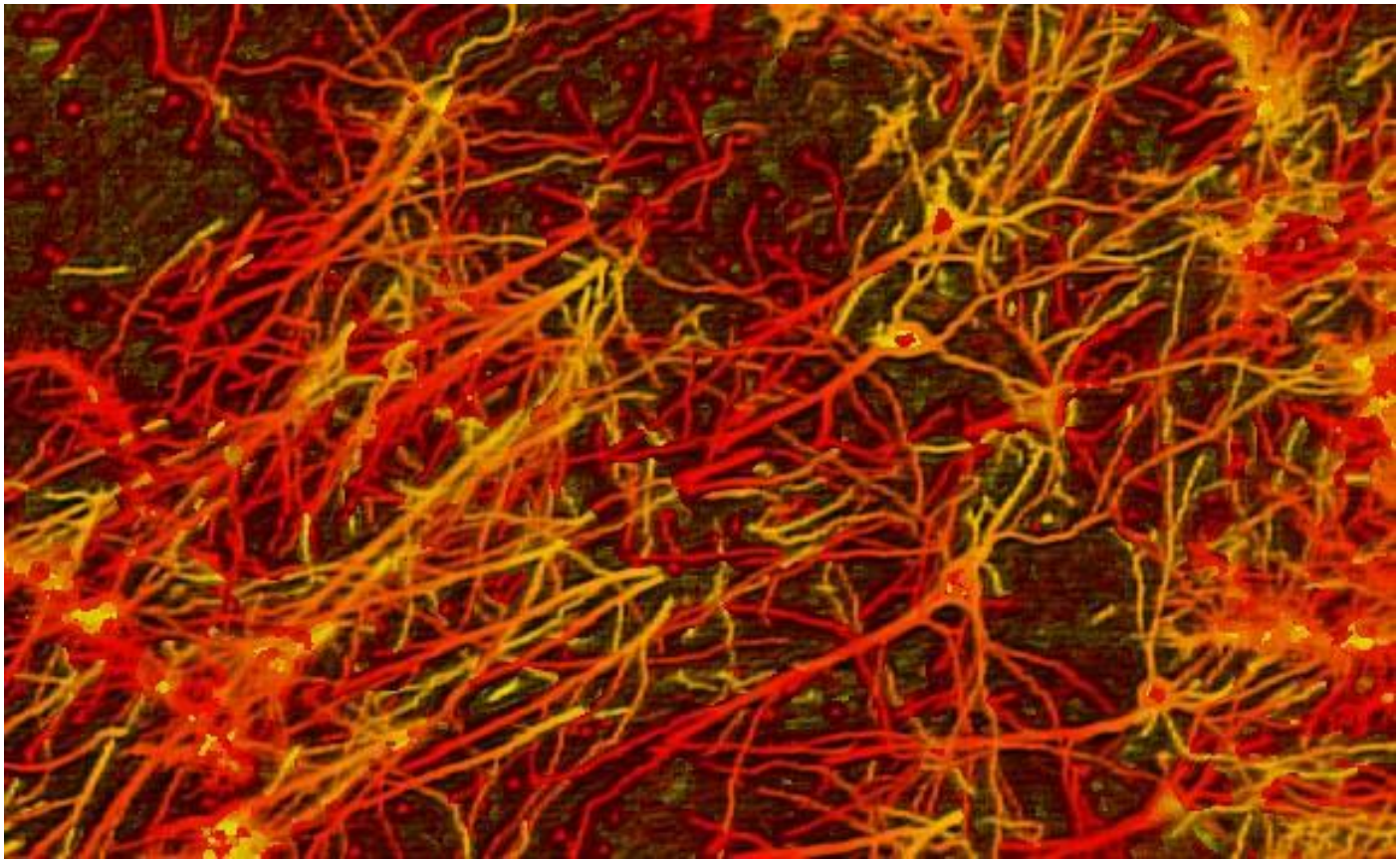
- An artificial neural network (ANN) is either a hardware implementation or a computer program which strives to simulate the information processing capabilities of its biological exemplar. ANNs are typically composed of a great number of interconnected artificial neurons. The artificial neurons are simplified models of their biological counterparts.
- ANN is a technique for solving problems by constructing software that works like our brains.



How do our brains work?

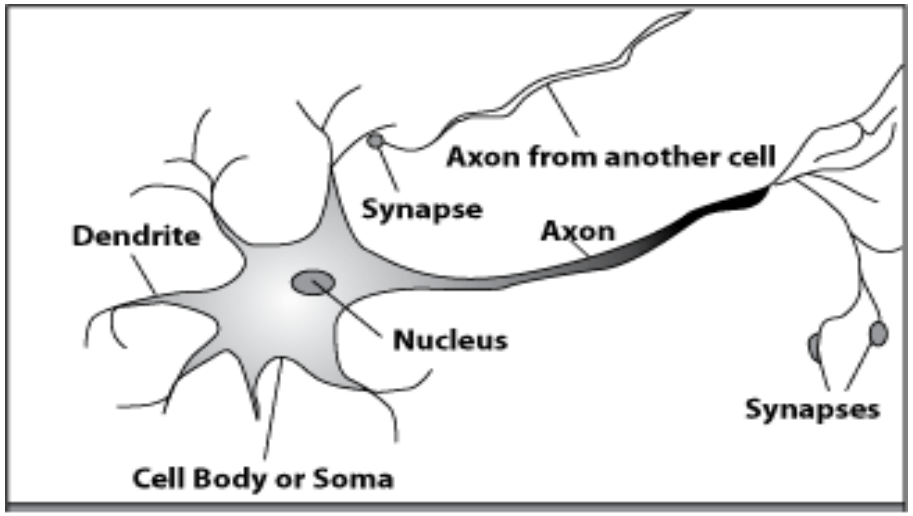
- The Brain is A massively parallel information processing system.
- Our brains are a huge network of processing elements. A typical brain contains a network of 10 billion neurons.

INTERCONNECTIONS IN BRAIN



How do our brains work?

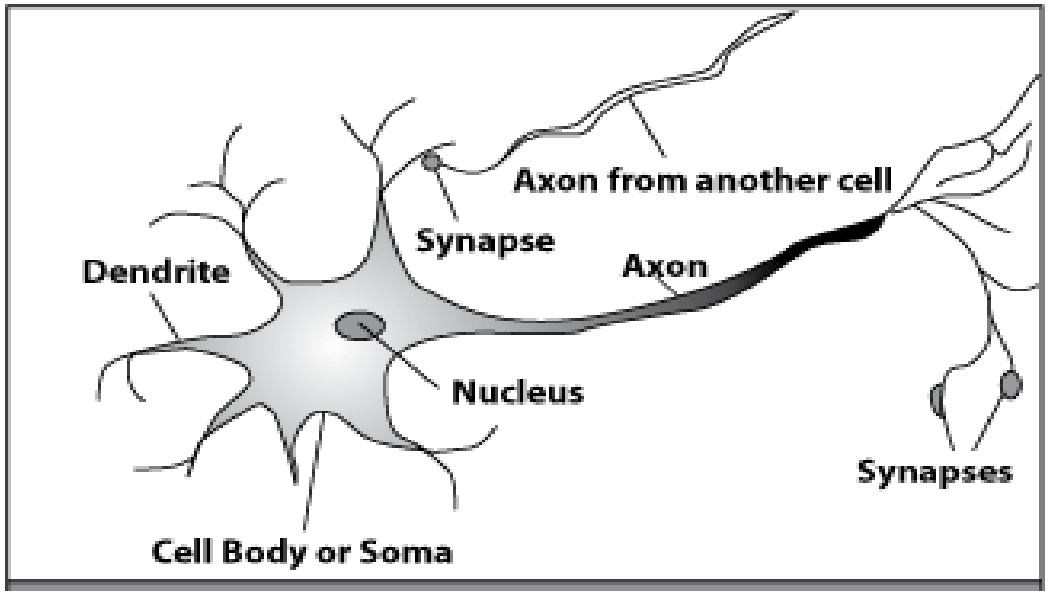
- A processing element



  - Dendrites: Input
  - Cell body: Processor
    - Synaptic: Link
    - Axon: Output

How do our brains work?

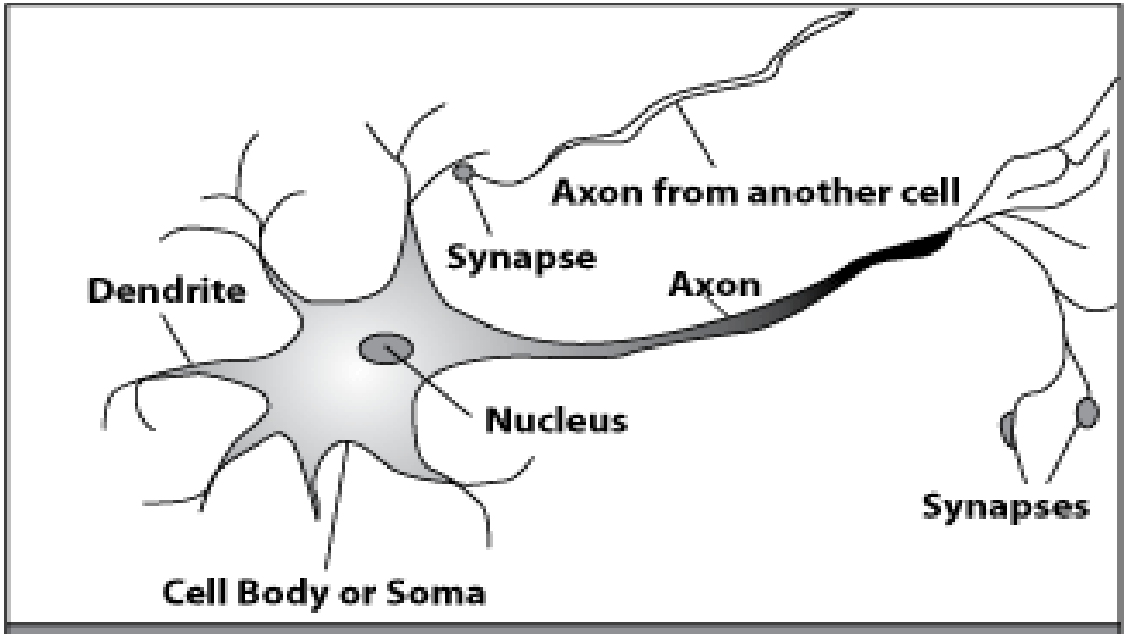
- A processing element



- A neuron is connected to other neurons through about 10,000 synapses

How do our brains work?

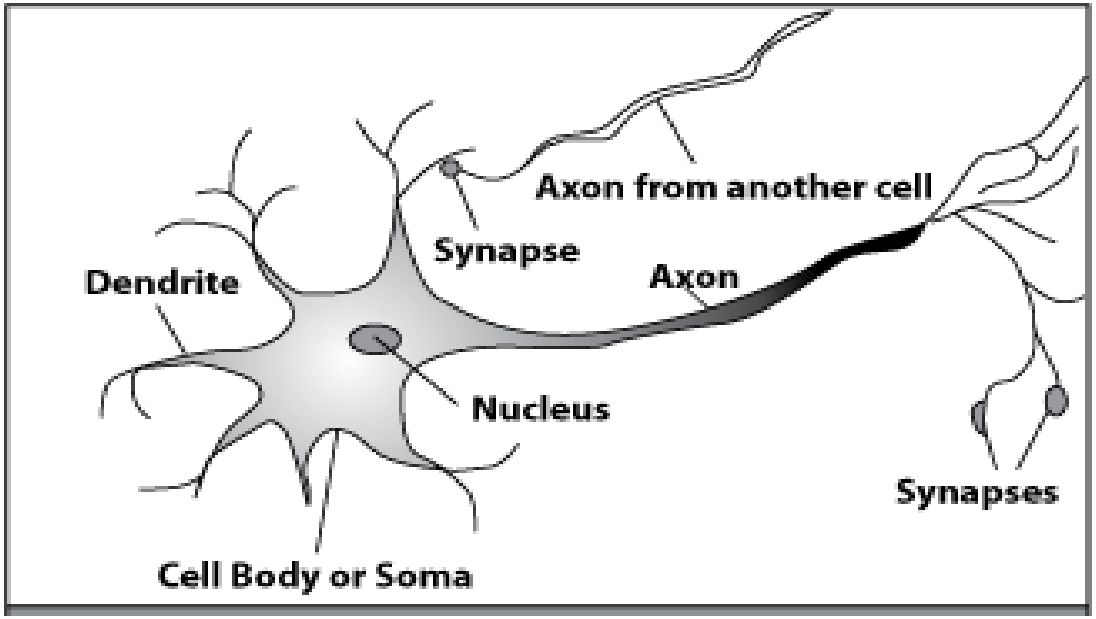
- A processing element



- A neuron receives input from other neurons. Inputs are combined.

How do our brains work?

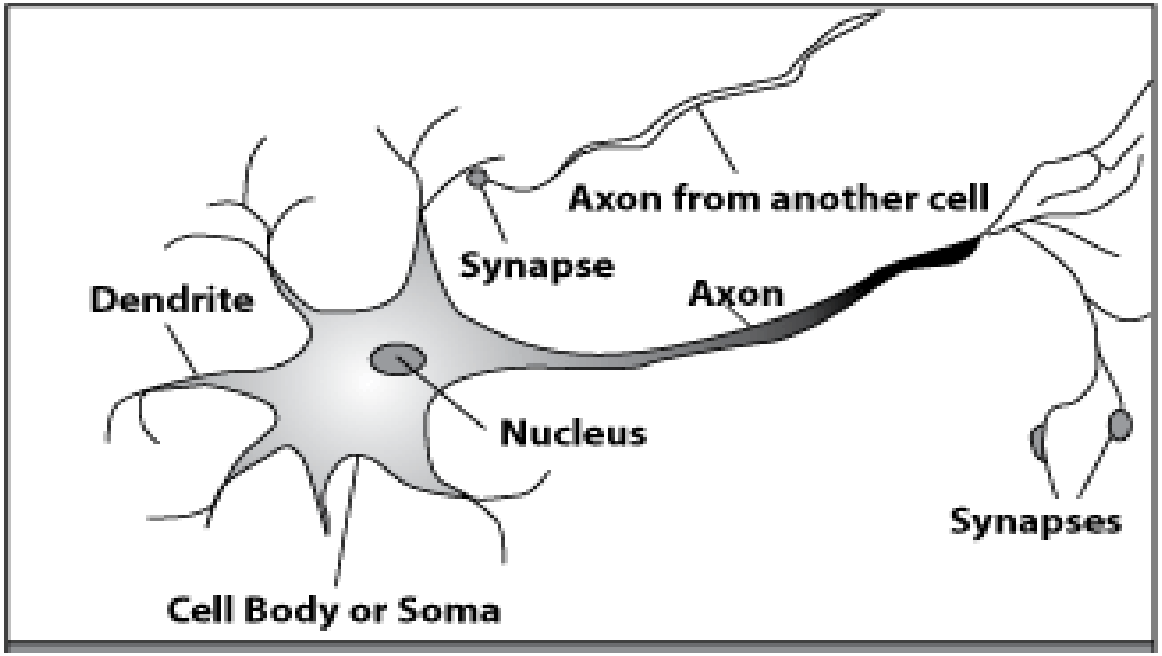
- A processing element



- Once input exceeds a critical level, the neuron discharges a spike - an electrical pulse that travels from the body, down the axon, to the next neuron(s)

How do our brains work?

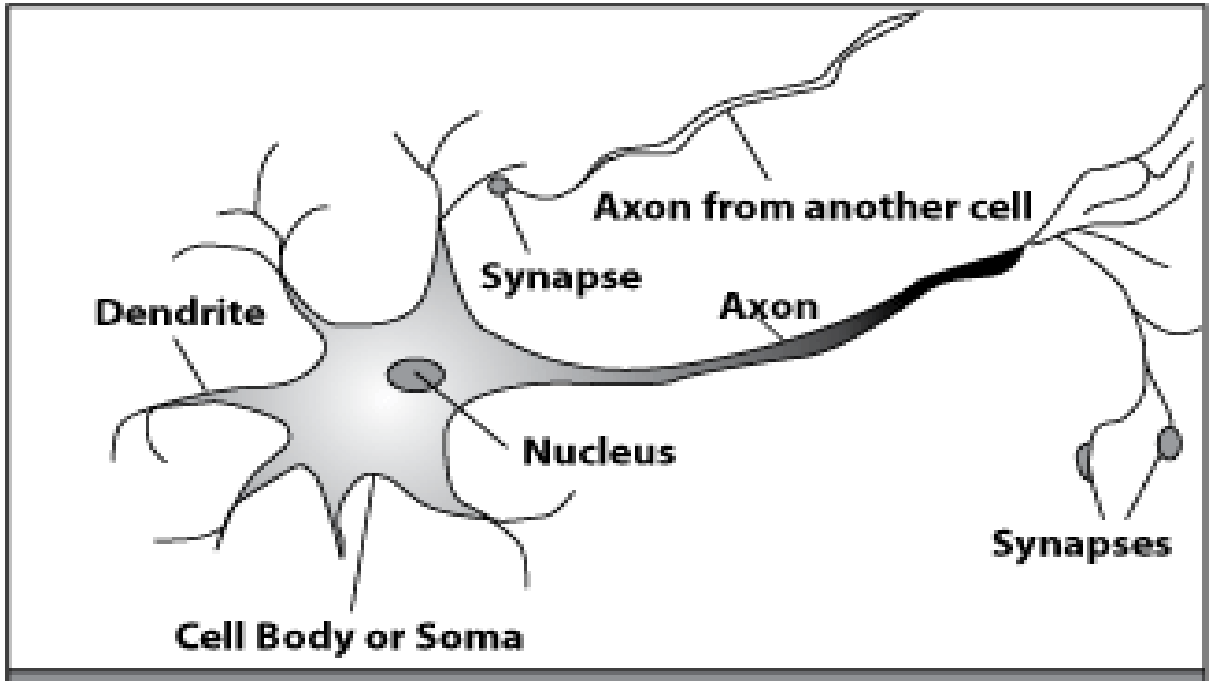
- A processing element



- The axon endings almost touch the dendrites or cell body of the next neuron.

How do our brains work?

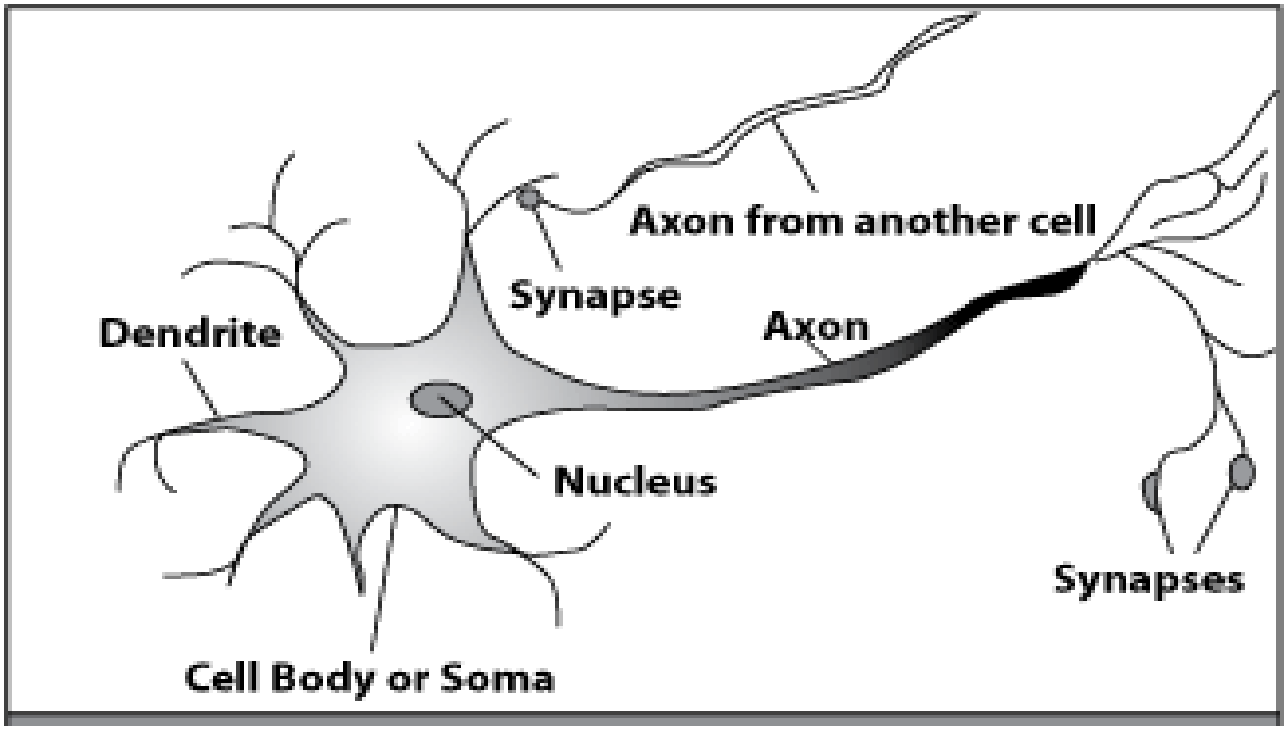
- A processing element



Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters.

How do our brains work?

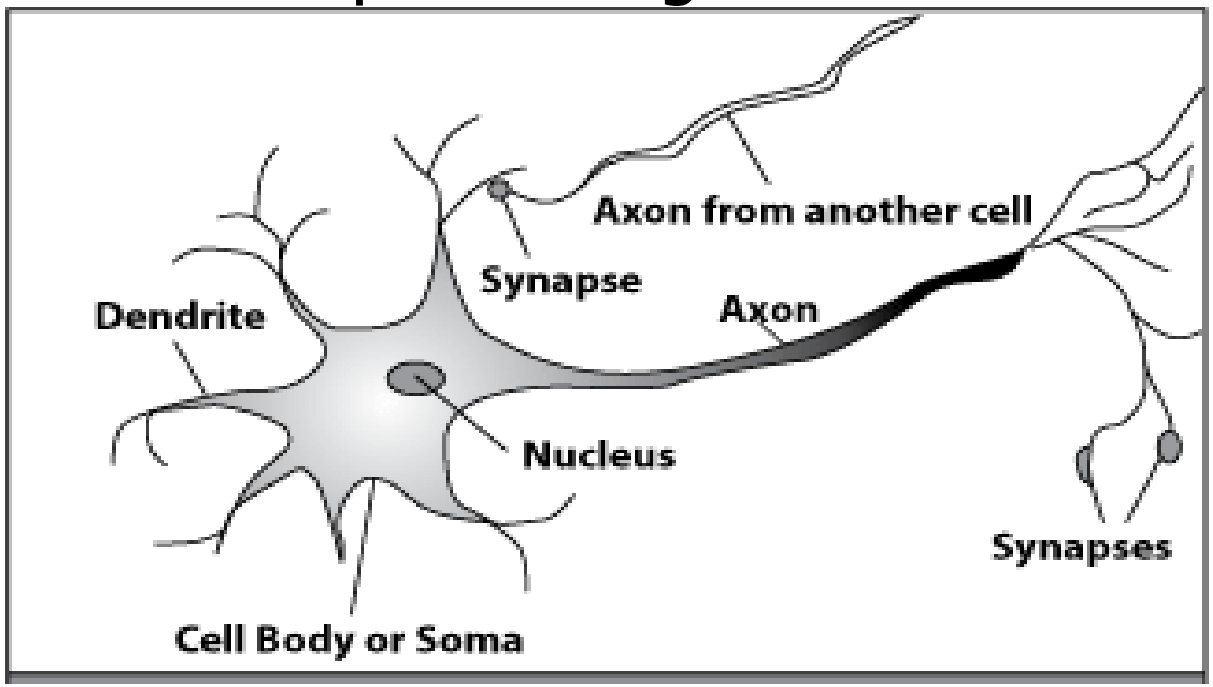
- A processing element



- Neurotransmitters are chemicals which are released from the first neuron and which bind to the Second.

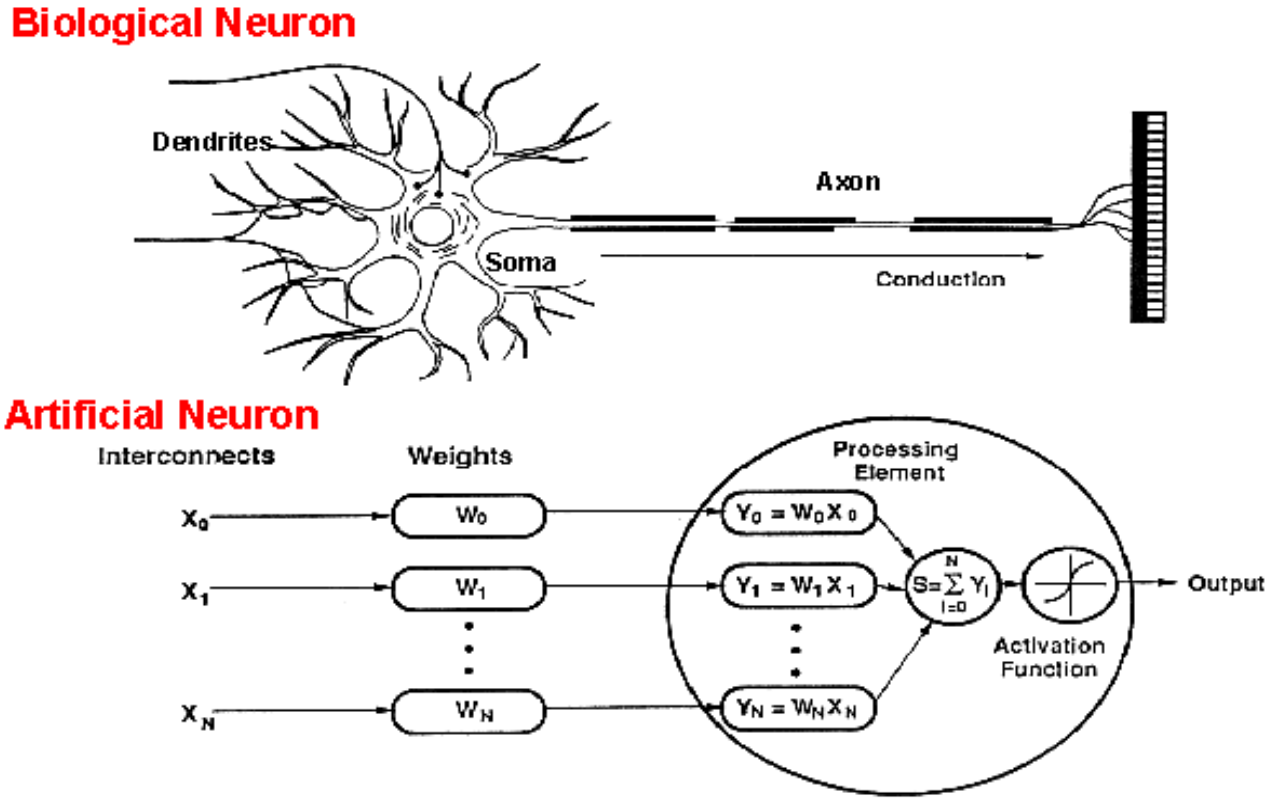
How do our brains work?

- A processing element



- This link is called a synapse. The strength of the signal that reaches the next neuron depends on factors such as the amount of neurotransmitter available.

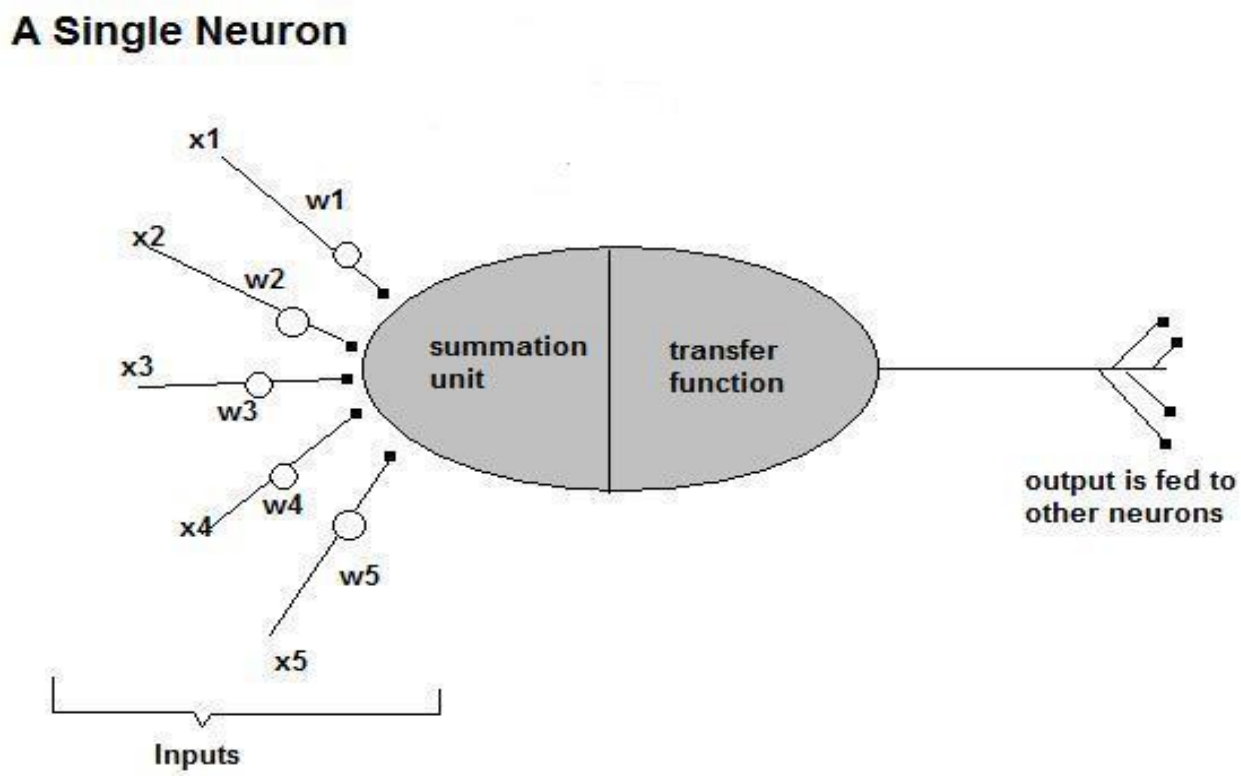
How do ANNs work?



- An artificial neuron is an imitation of a human neuron

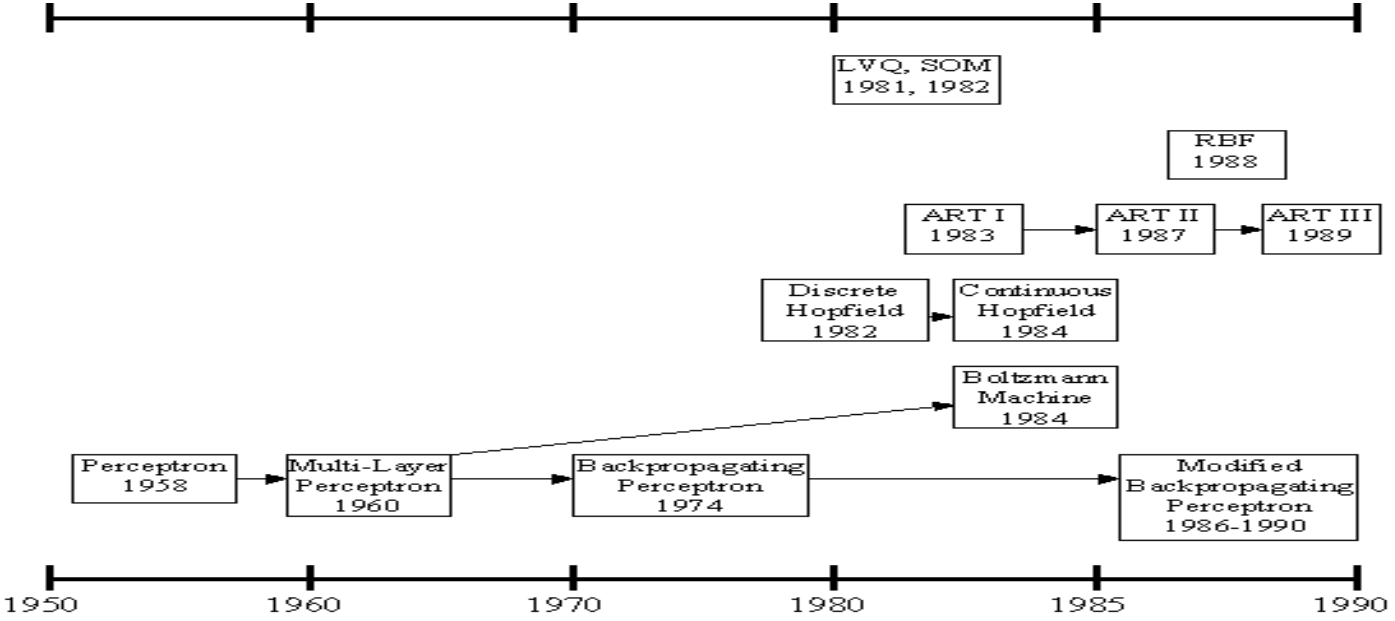
How do ANNs work?

- Now, let us have a look at the model of an artificial neuron.



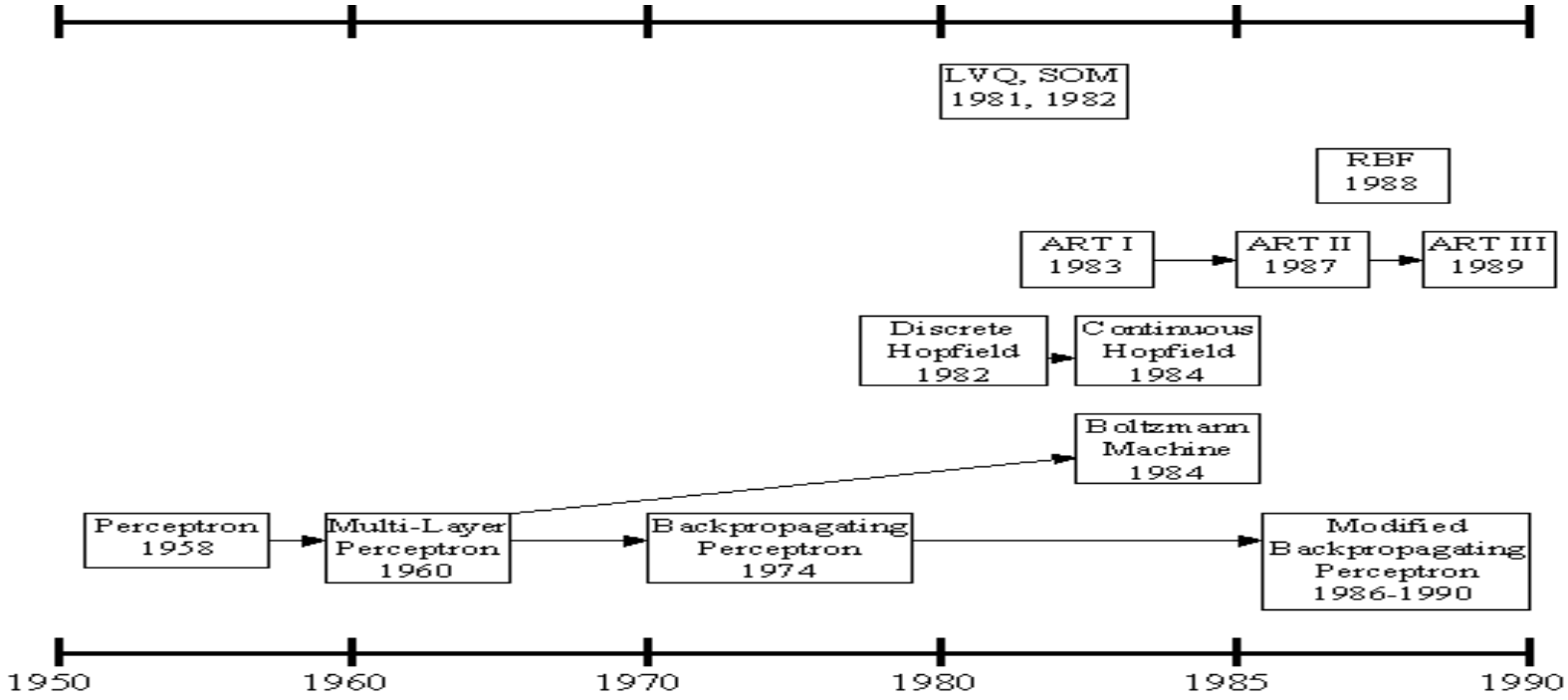
History of the Artificial Neural Networks

- History of the ANNs stems from the 1940s, the decade of the first Electronic computer.
- However, the first important step took place in 1957 when Rosenblatt introduced the first concrete neural model, the perceptron. Rosenblatt also took part in constructing the first successful neurocomputer, the Mark I Perceptron. After this, the development of ANNs has proceeded as described in Figure.



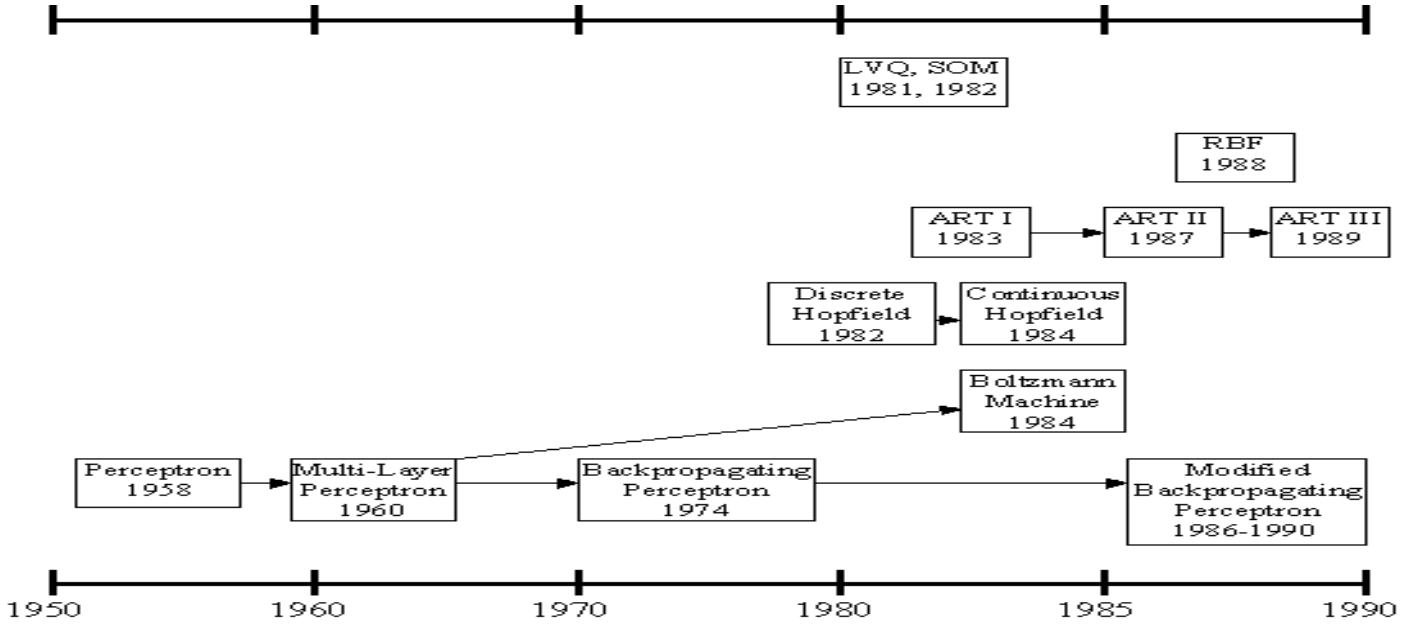
History of the Artificial Neural Networks

- Rosenblatt's original perceptron model contained only one layer. From this, a multi-layered model was derived in 1960. At first, the use of the multi-layer perceptron (MLP) was complicated by the lack of a appropriate learning algorithm.
- In 1974, Werbos came to introduce a so-called backpropagation algorithm for the three-layered perceptron network.



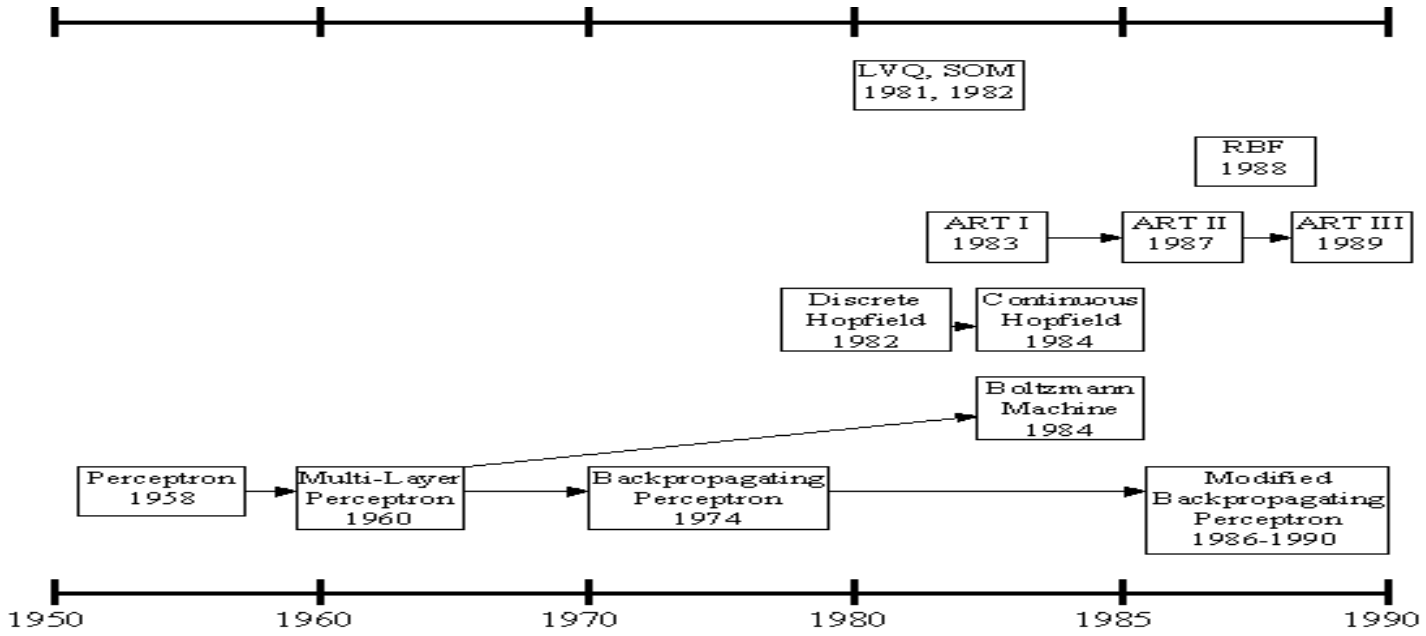
History of the Artificial Neural Networks

- in 1986, The application area of the MLP networks remained rather limited until the breakthrough when a general back propagation algorithm for a multi-layered perceptron was introduced by Rummelhart and Mclelland.
- in 1982, Hopfield brought out his idea of a neural network. Unlike the neurons in MLP, the Hopfield network consists of only one layer whose neurons are fully connected with each other.



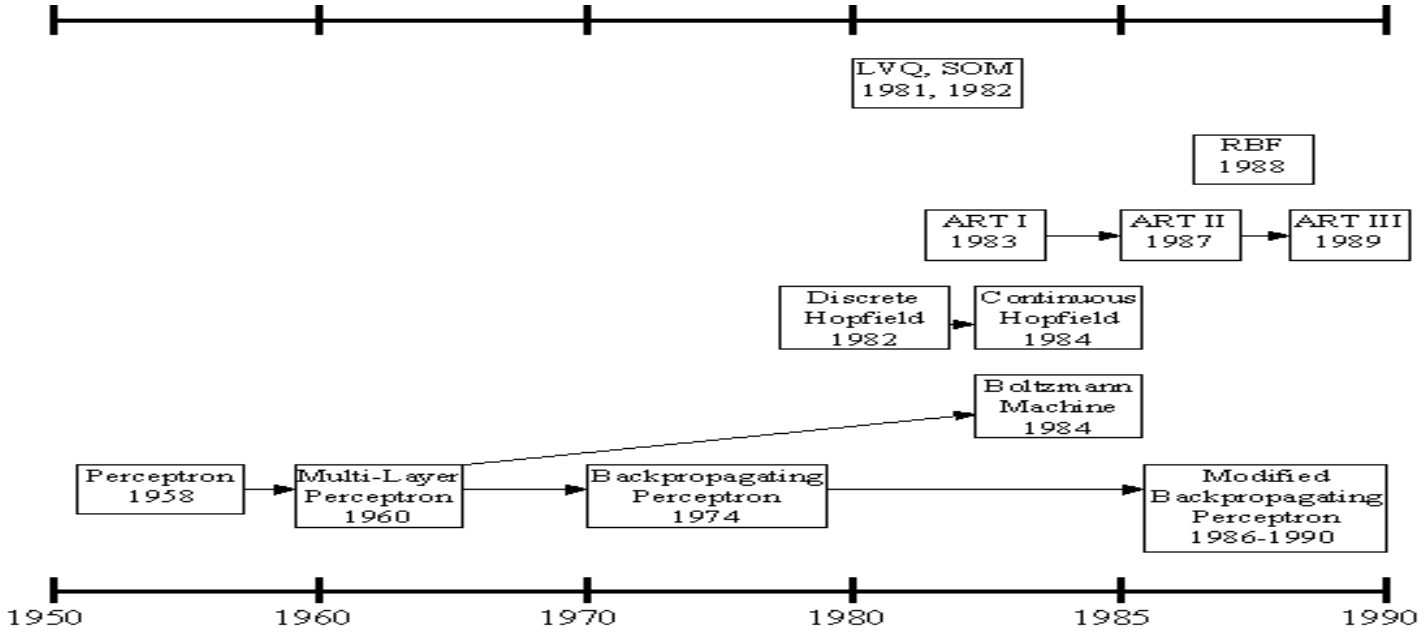
History of the Artificial Neural Networks

- Since then, new versions of the Hopfield network have been developed. The Boltzmann machine has been influenced by both the Hopfield network and the MLP.



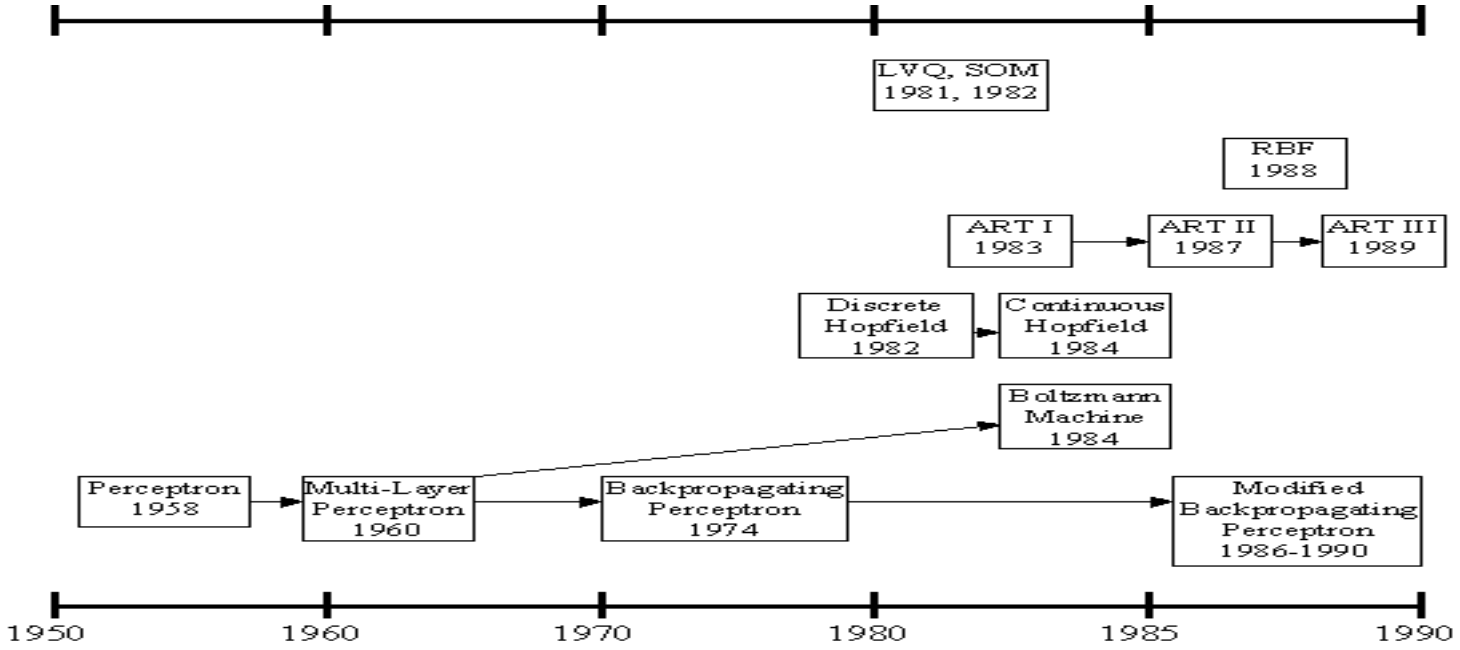
History of the Artificial Neural Networks

- In 1988, Radial Basis Function (RBF) networks were first introduced by Broomhead & Lowe. Although the basic idea of RBF was developed 30 years ago under the name method of potential function, the work by Broomhead & Lowe opened a new frontier in the neural network community.



History of the Artificial Neural Networks

- In 1982, A totally unique kind of network model is the Self-Organizing Map (SOM) introduced by Kohonen. SOM is a certain kind of topological map which organizes itself based on the input patterns that it is trained with. The SOM originated from the LVQ (Learning Vector Quantization) network the underlying idea of which was also Kohonen's in 1972.



History of Artificial Neural Networks

Since then, research on artificial neural networks has remained active, leading to many new network types, as well as hybrid algorithms and hardware for neural information processing.

ARTIFICIAL NEURAL NET

- Information-processing system.
- Neurons process the information.
- The signals are transmitted by means of connection links.
- The links possess an associated weight.
- The output signal is obtained by applying activations to the net input.

MOTIVATION FOR NEURAL NET

- Scientists are challenged to use machines more effectively for tasks currently solved by humans.
- Symbolic rules don't reflect processes actually used by humans.
- Traditional computing excels in many areas, but not in others.

The major areas being:

- Massive parallelism
- Distributed representation and computation
- Learning ability
- Generalization ability
- Adaptivity
- Inherent contextual information processing
- Fault tolerance
- Low energy consumption.

PROCESSING OF AN ARTIFICIAL NET

The neuron is the basic information processing unit of a NN. It consists of:

1. A set of links, describing the neuron inputs, with weights W_1, W_2, …, W_m.

2. An adder function (linear combiner) for computing the weighted sum of the inputs (real numbers):

$$\mathbf{u} = \sum_{j=1}^{m} \mathbf{W}_j \mathbf{X}_j$$

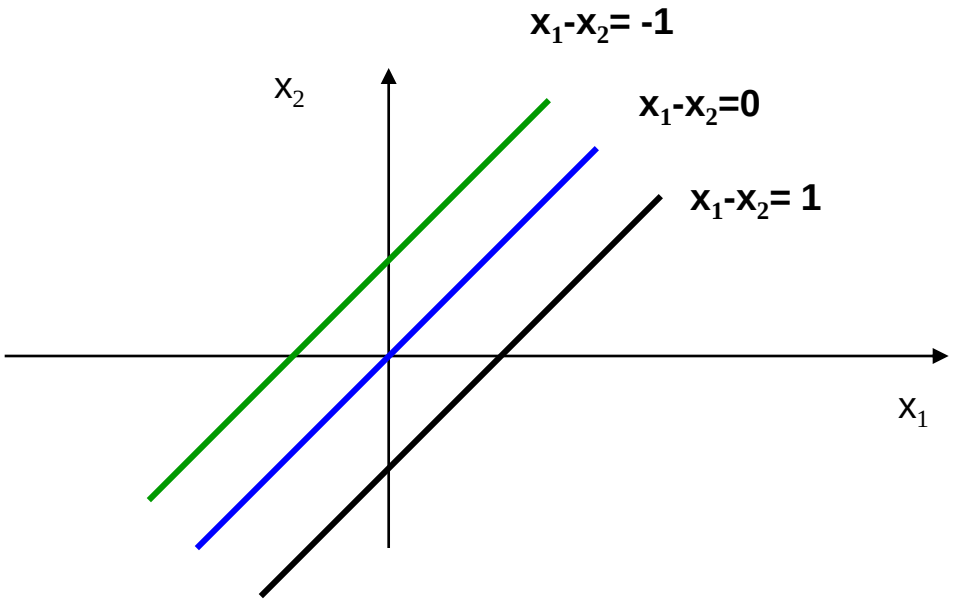
3. Activation function for limiting the amplitude of the neuron output.

$$\mathrm{y} = \varphi(\mathrm{u} + b)$$

BIAS OF AN ARTIFICIAL NEURON

The bias value is added to the weighted sum $\sum w_i x_i$ so that we can transform it from the origin.

$$Y_{in} = \sum w_i x_i + b, \text{ where } b \text{ is the bias}$$



MULTI LAYER ARTIFICIAL NEURAL NET

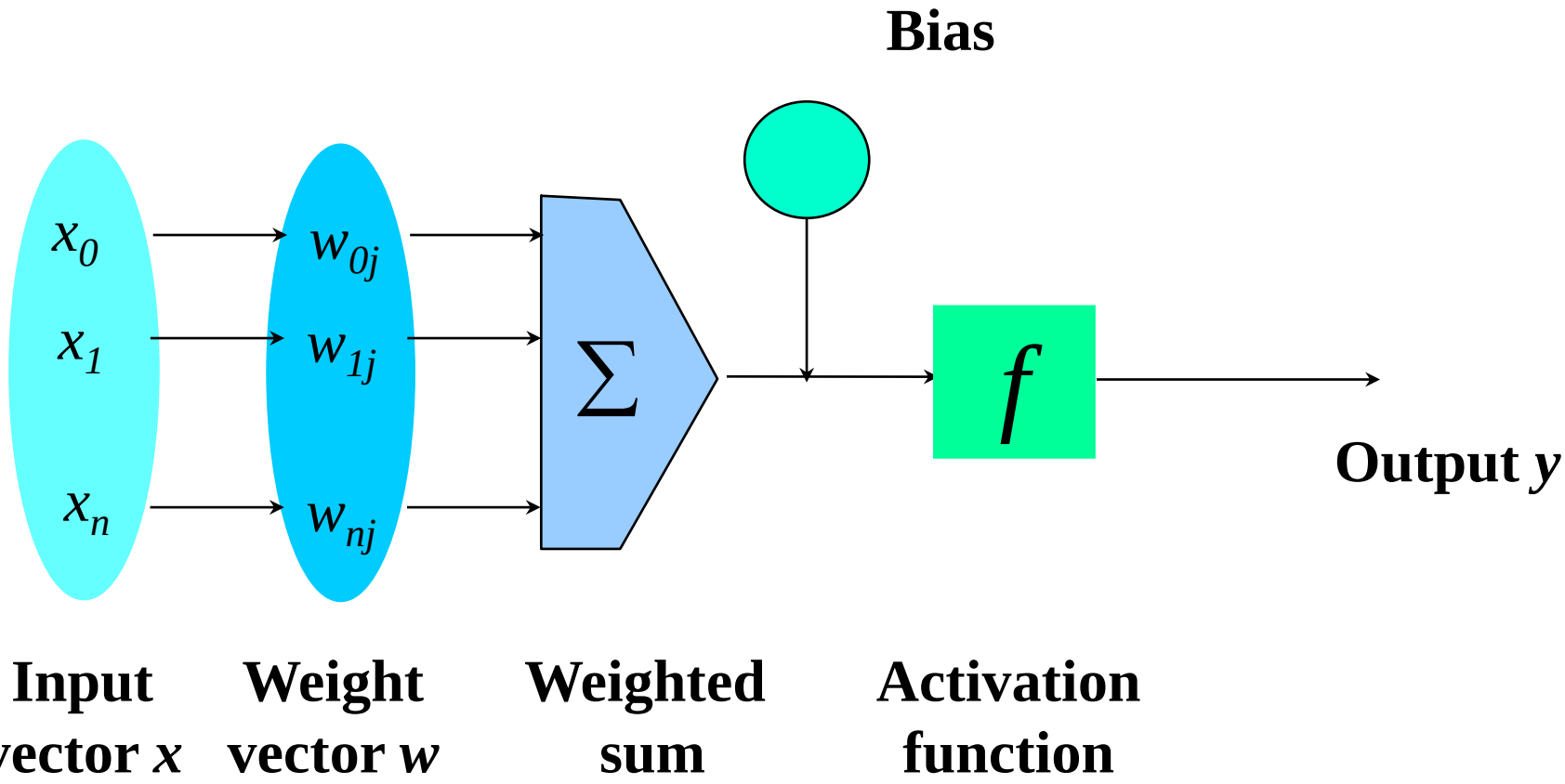
INPUT: records without class attribute with normalized attributes values.

INPUT VECTOR: $X = \{ x_1, x_2, \ldots, x_n\}$ where n is the number of (non-class) attributes.

INPUT LAYER: there are as many nodes as non-class attributes, i.e. as the length of the input vector.

HIDDEN LAYER: the number of nodes in the hidden layer and the number of hidden layers depends on implementation.

OPERATION OF A NEURAL NET



WEIGHT AND BIAS UPDATION

Per Sample Updating

- updating weights and biases after the presentation of each sample.

Per Training Set Updating (Epoch or Iteration)

- weight and bias increments could be accumulated in variables and the weights and biases updated after all the samples of the training set have been presented.

STOPPING CONDITION

- All change in weights (Δwij) in the previous epoch are below some threshold, or
- The percentage of samples misclassified in the previous epoch is below some threshold, or
- A pre-specified number of epochs has expired.
- In practice, several hundreds of thousands of epochs may be required before the weights will converge.

BUILDING BLOCKS OF ARTIFICIAL NEURAL NET

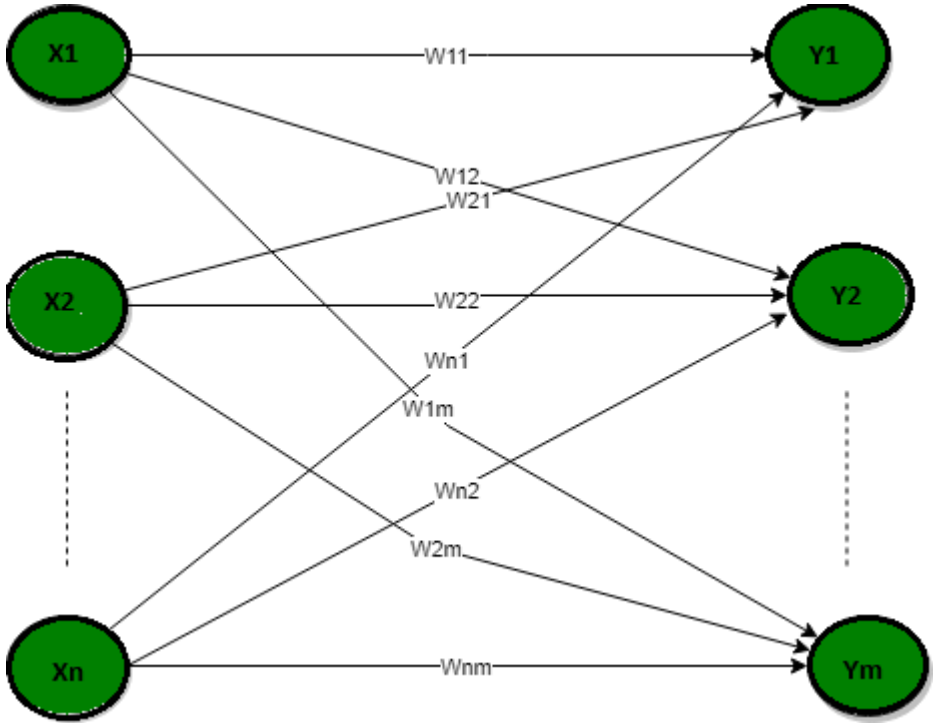
- Network Architecture (Connection between Neurons)
- Setting the Weights (Training)
- Activation Function

Network Architecture

- Single layer feed-forward network
- Multilayer feed-forward network
- Single node with its own feedback
- Single layer recurrent network
- Multilayer recurrent network

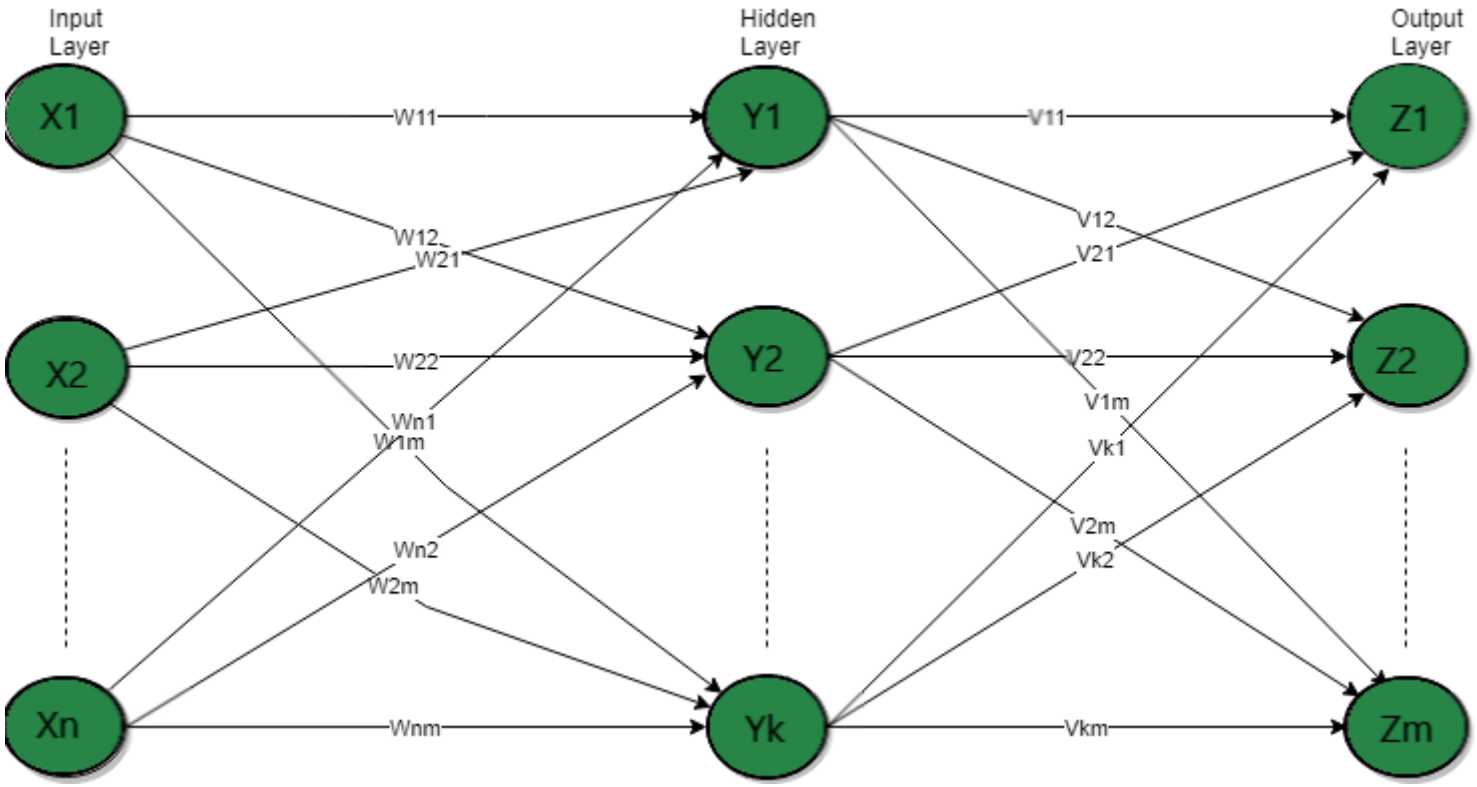
Network Architecture

- **Single layer feed-forward network**



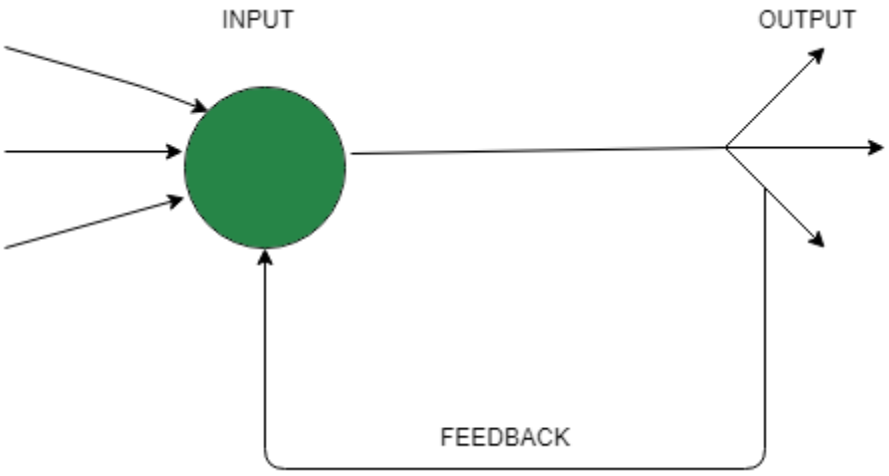
Network Architecture

- **Multilayer feed-forward network**



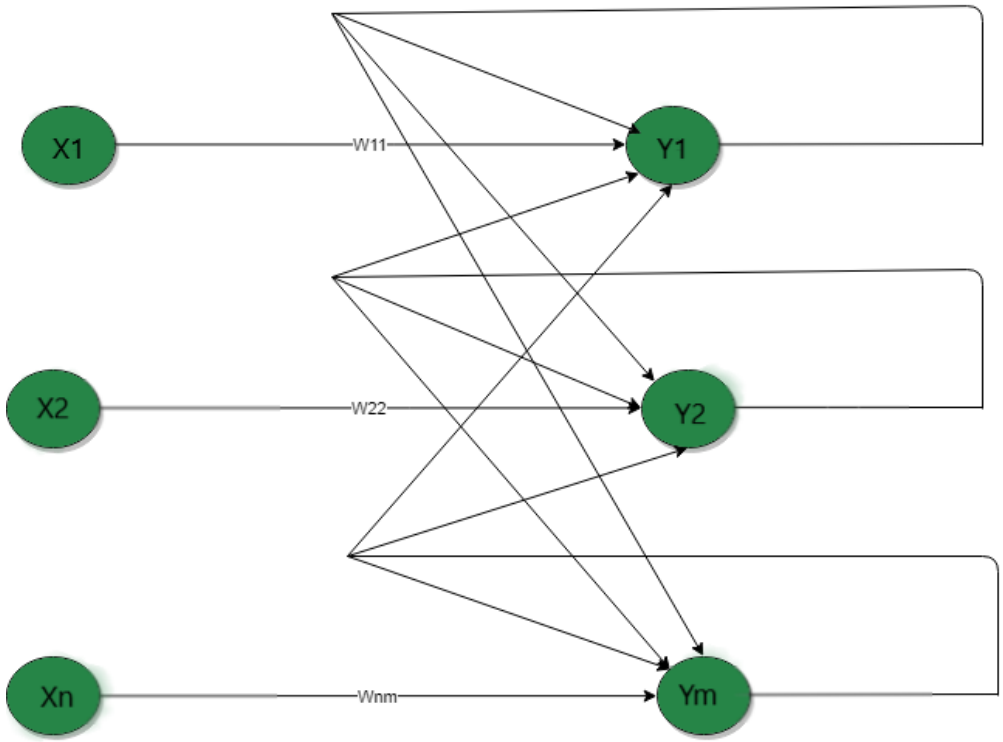
Network Architecture

- **Single node with its own feedback**



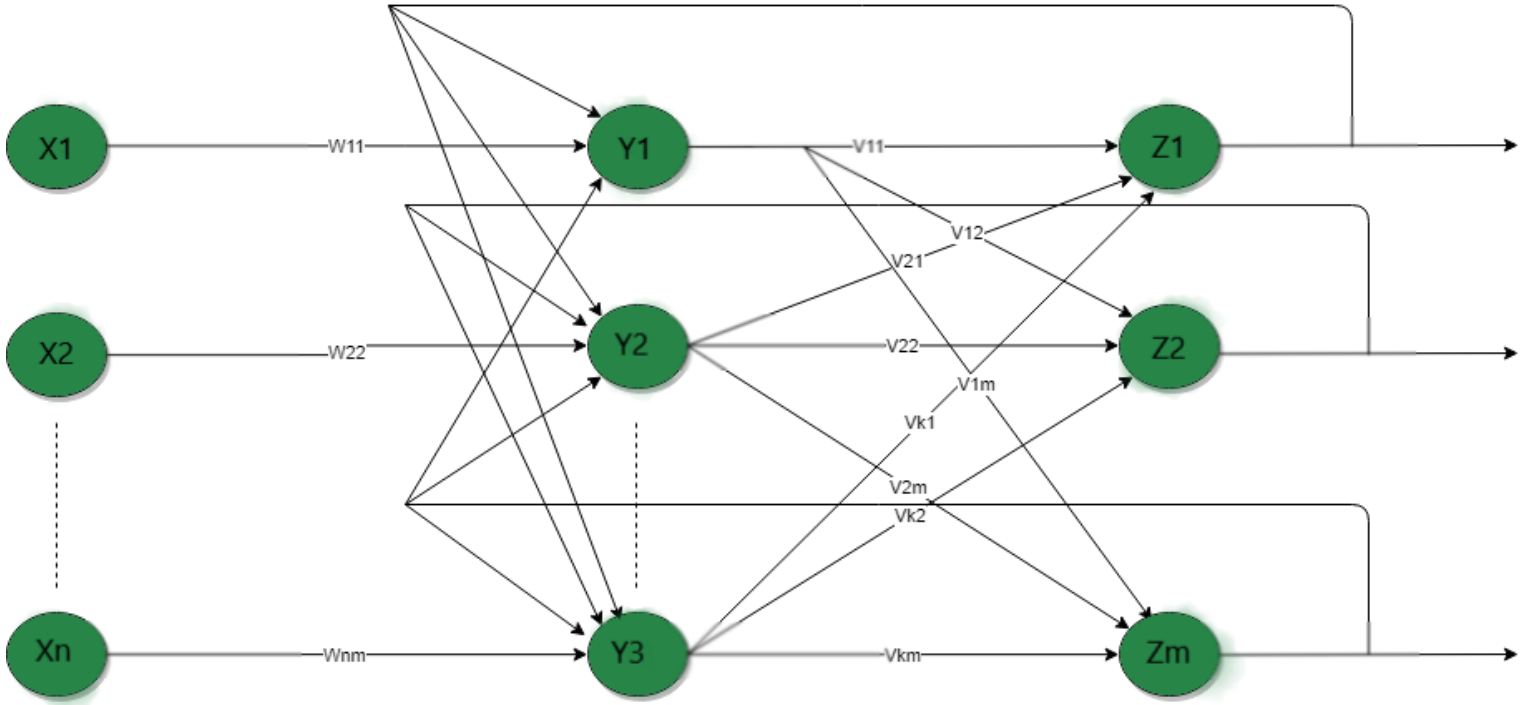
Network Architecture

- **Single layer recurrent network**



Network Architecture

- **Multilayer recurrent network**

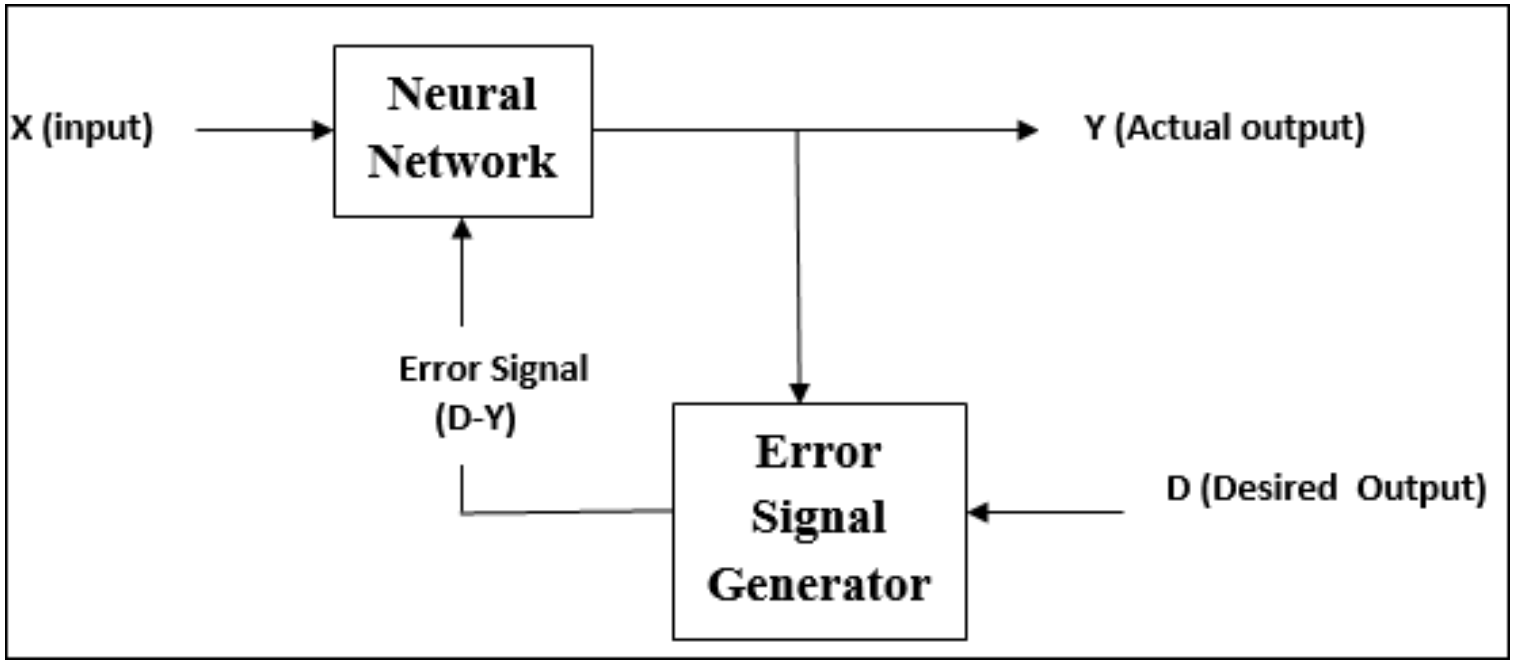


LAYER PROPERTIES

- **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.

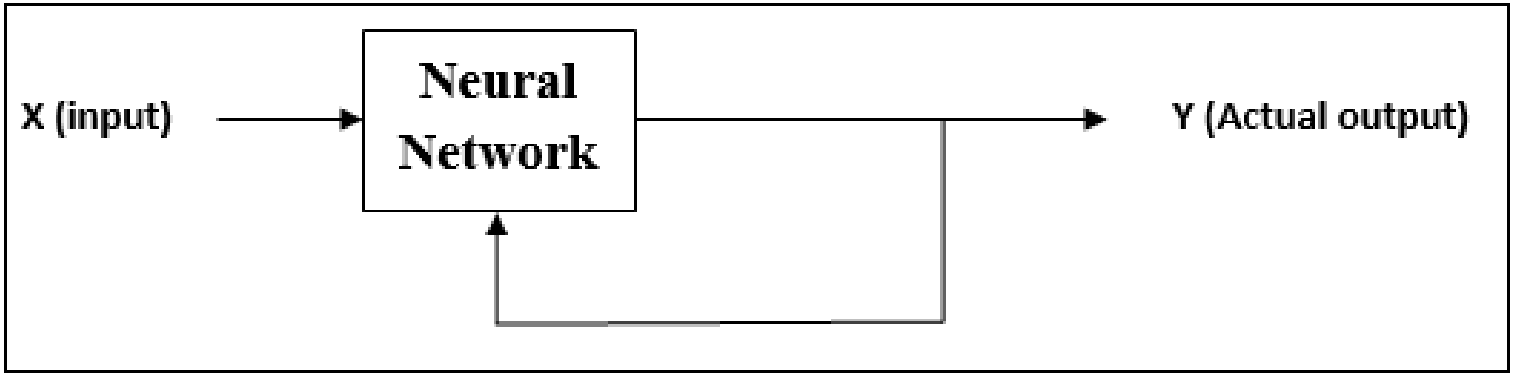
TRAINING PROCESS

- **Supervised Training** - Providing the network with a series of sample inputs and comparing the output with the expected responses.



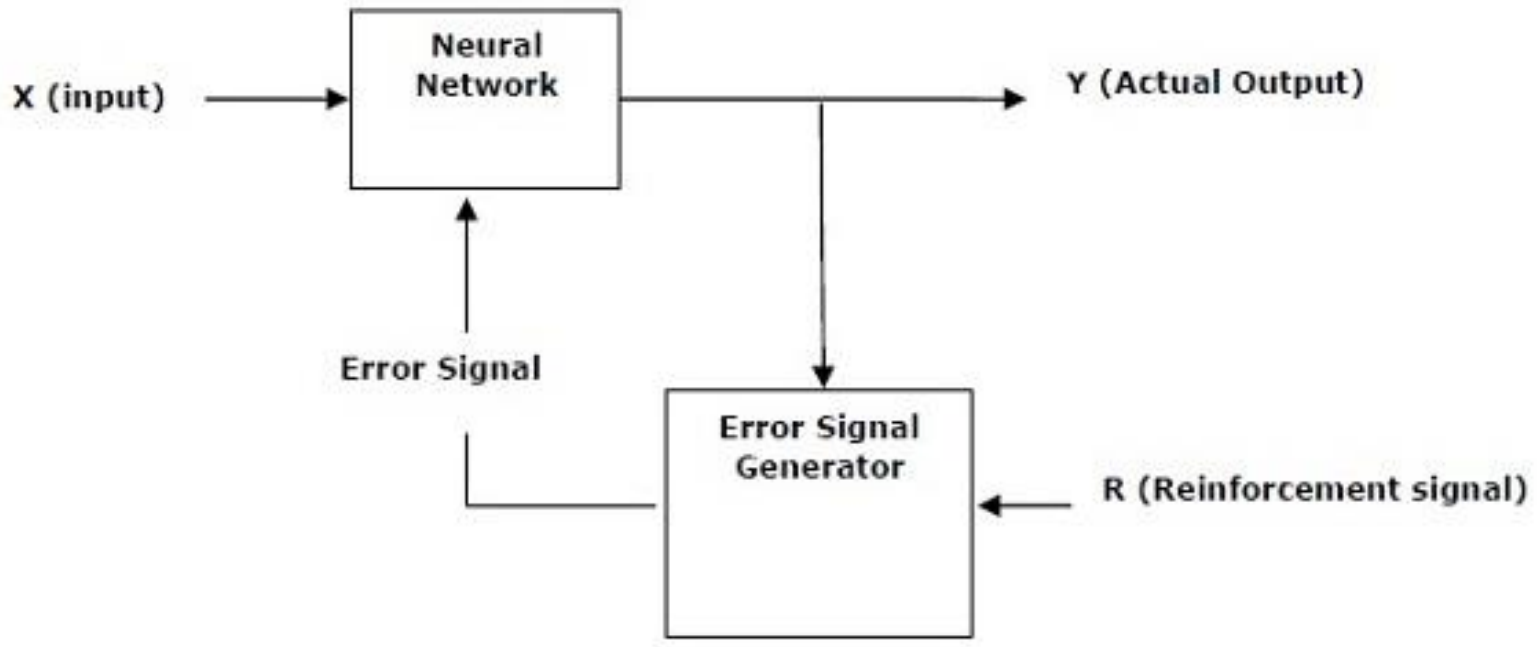
TRAINING PROCESS

- **Unsupervised Training** - Most similar input vector is assigned to the same output unit.



TRAINING PROCESS

- **Reinforcement Training** - Right answer is not provided but indication of whether 'right' or 'wrong' is provided.



ACTIVATION FUNCTION

- **ACTIVATION LEVEL – DISCRETE OR CONTINUOUS**

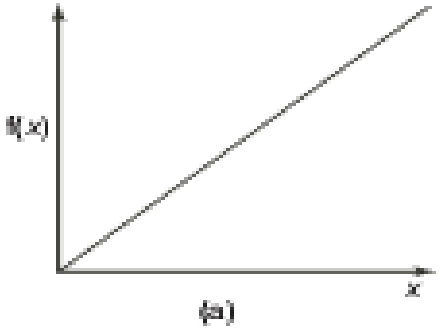
- **HARD LIMIT FUCNTION (DISCRETE)**
  - Binary Activation function
  - Bipolar activation function
  - Identity function

- **SIGMOIDAL ACTIVATION FUNCTION (CONTINUOUS)**
  - Binary Sigmoidal activation function
  - Bipolar Sigmoidal activation function

ACTIVATION FUNCTION

1. ***Identity function:*** It is a linear function and can be defined as

$$f(x) = x \quad \text{for all } x$$

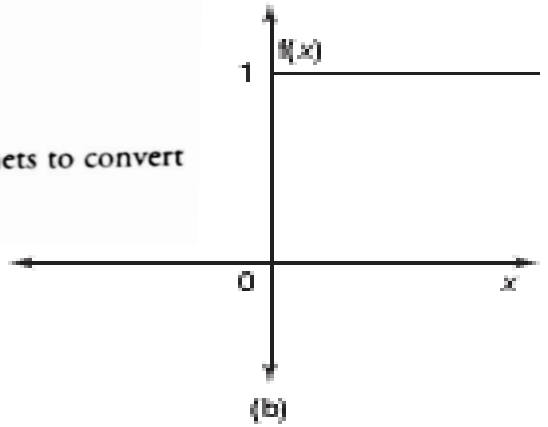


(a)

2. *Binary step function:* This function can be defined as

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

where θ represents the threshold value. This function is most widely used in single-layer nets to convert the net input to an output that is a binary (1 or 0).



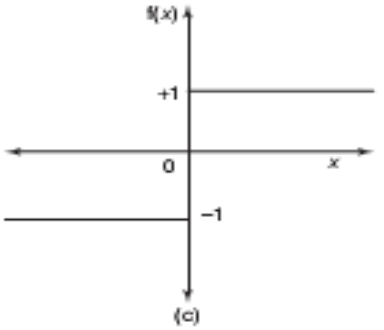
(b)

ACTIVATION FUNCTION

3. *Bipolar step function*: This function can be defined as

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

where θ represents the threshold value. This function is also used in single-layer nets to convert the net input to an output that is bipolar (+1 or −1).



(c)

4. *Sigmoidal functions*: The sigmoidal functions are widely used in back-propagation nets because of the relationship between the value of the functions at a point and the value of the derivative at that point which reduces the computational burden during training.
Sigmoidal functions are of two types:

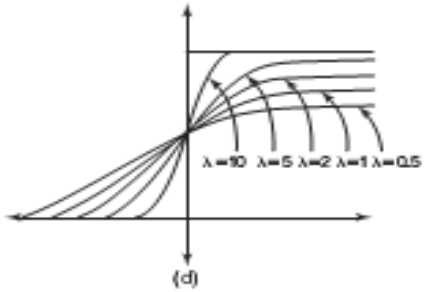
- *Binary sigmoid function*: It is also termed as logistic sigmoid function or unipolar sigmoid functic It can be defined as

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

where λ is the steepness parameter. The derivative of this function is

$$f'(x) = \lambda f(x)[1 - f(x)]$$

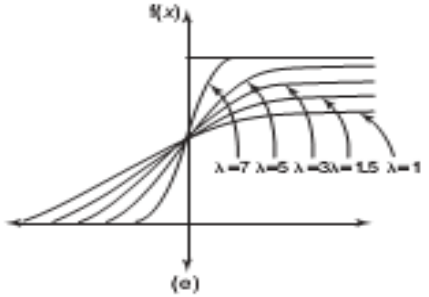
Here the range of the sigmoid function is from 0 to 1.



(d)

- *Bipolar sigmoid function*: This function is defined as

$$f(x) = \frac{2}{1 + e^{-\lambda x}} - 1 = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$$

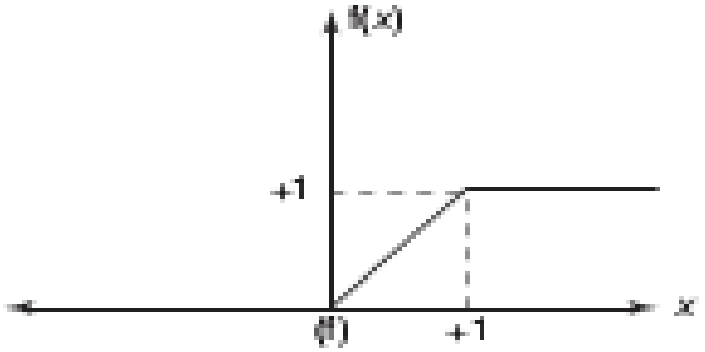


(e)

ACTIVATION FUNCTION

(F) Ramp

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



CONSTRUCTING ANN

- Determine the network properties:
  - Network topology
  - Types of connectivity
  - Order of connections
  - Weight range
- Determine the node properties:
  - Activation range
- Determine the system dynamics
  - Weight initialization scheme
  - Activation – calculating formula
  - Learning rule

PROBLEM SOLVING

- Select a suitable NN model based on the nature of the problem.
- Construct a NN according to the characteristics of the application domain.
- Train the neural network with the learning procedure of the selected model.
- Use the trained network for making inference or solving problems.

NEURAL NETWORKS

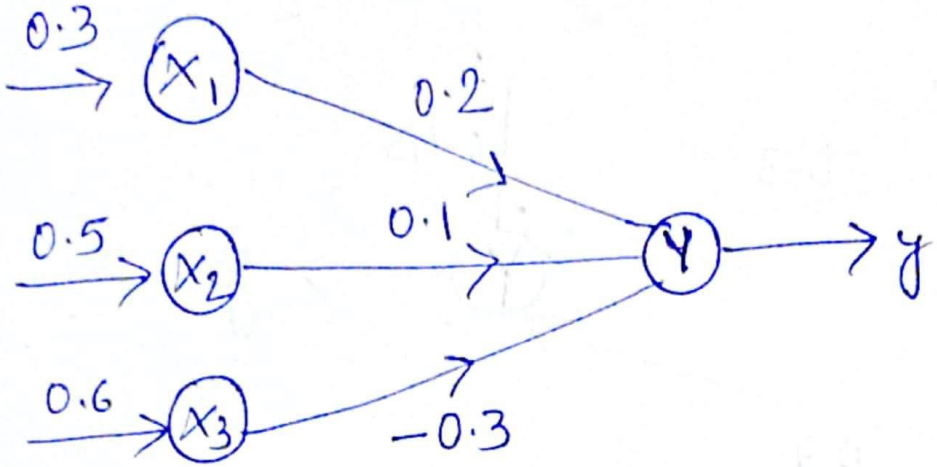
- **Neural Network** learns by adjusting the weights so as to be able to correctly classify the training data and hence, after testing phase, to classify unknown data.
- **Neural Network** needs long time for training.
- **Neural Network** has a high tolerance to noisy and incomplete data.

McCULLOCH–PITTS NEURON

- Neurons are sparsely and randomly connected
- Firing state is binary (1 = firing, 0 = not firing)
- All but one neuron are excitatory (tend to increase voltage of other cells)
  - One inhibitory neuron connects to all other neurons
  - It functions to regulate network activity (prevent too many firings)

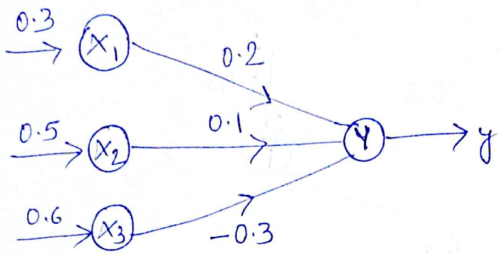
McCULLOCH–PITTS NEURON

Q:- For the network shown in figure, calculate the net input to the output neuron.



McCULLOCH–PITTS NEURON

Q:- For the network shown in figure, calculate the net input to the output neuron.



Soln:- In the given neural network, there are three input neurons and one output neuron.

The input 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 i/p can be calculated as,

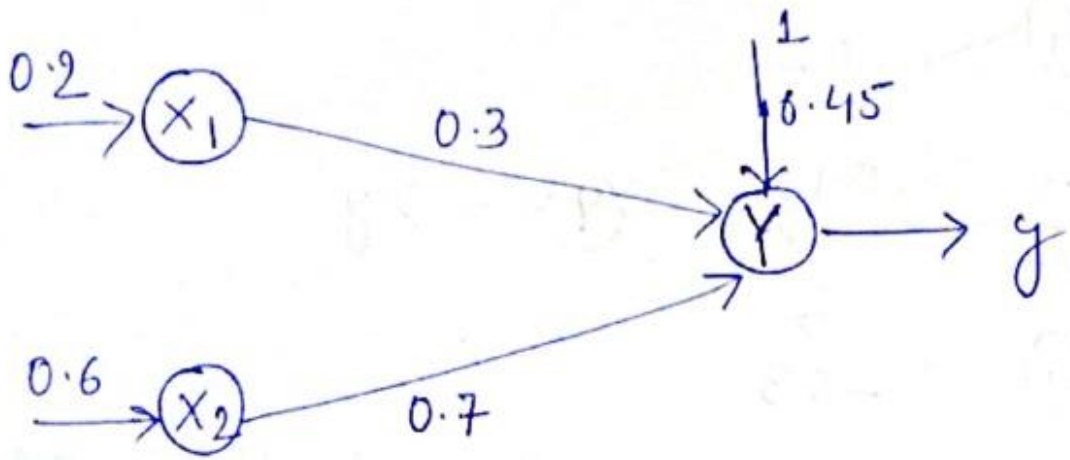
$$y_{in} = x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$= 0.3 \times 0.2 + 0.5 \times 0.1 + 0.6 \times (-0.3)$$

$$= 0.06 + 0.05 - 0.18 = -0.07$$

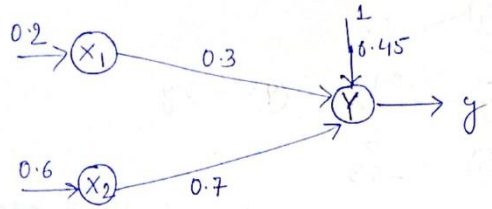
McCULLOCH–PITTS NEURON

Q6: Calculate the net input for the network shown in figure with bias included in the network.



McCULLOCH–PITTS NEURON

Q: Calculate the net input for the network shown in figure with bias included in the network.



Sol^n: The given net consists of two input neurons, a bias and an output neuron.

The inputs are

$$[x_1 \; x_2] = [0.2 \quad 0.6]$$

The weights are,

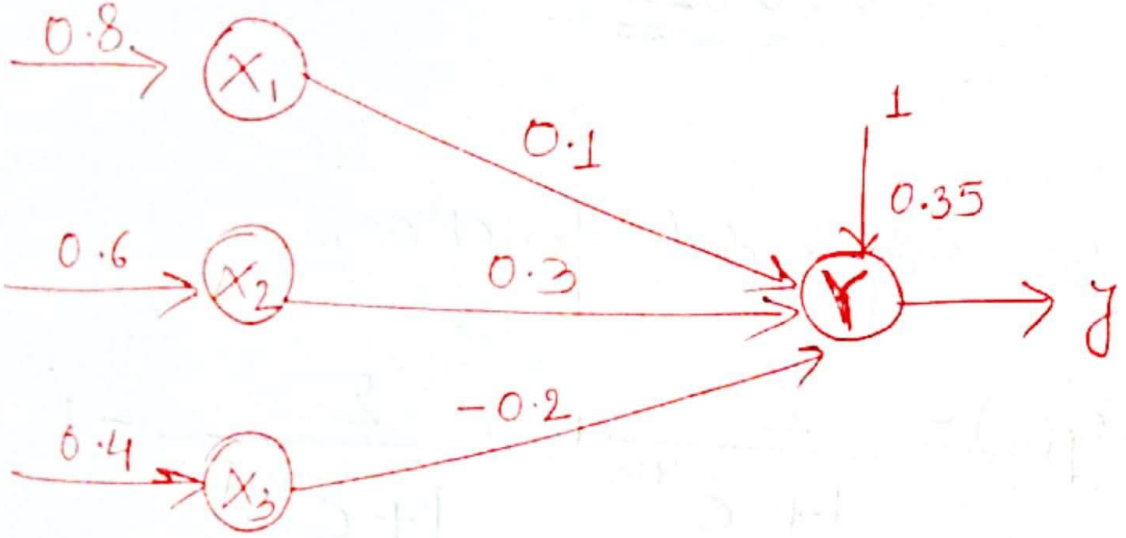
$$[w_1 \quad w_2] = [0.3 \quad 0.7]$$

Since, the bias is included $b = 0.45$ and bias input x_0 is equal to 1, the net input is calculated as follows:-

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= 0.45 + 0.2 * 0.3 + 0.6 * 0.7$$
$$= 0.45 + 0.06 + 0.42$$
$$= 0.93$$

$\therefore$ $y_{in} = 0.93$ is the net input.

Q: Obtain the output of the neuron Y for the network shown in figure using activation functions as (i) Binary sigmoidal and (ii) Bipolar sigmoidal.



McCULLOCH–PITTS NEURON

Soln:

$$[x_1 \quad x_2 \quad x_3] = [0.8 \quad 0.6 \quad 0.4]$$

$$[w_1 \quad w_2 \quad w_3] = [0.1 \quad 0.3 \quad -0.2]$$

with $b = 0.35$ (i/p = 1)

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

$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$= 0.35 + 0.8 \times 0.1 + 0.6 \times 0.3 + 0.4 \times -0.2$$

$$= 0.35 + 0.08 + 0.18 - 0.08$$

$$= 0.53$$

ii) For binary sigmoidal function

$$y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-0.53}}$$

$$= \underline{0.625}$$

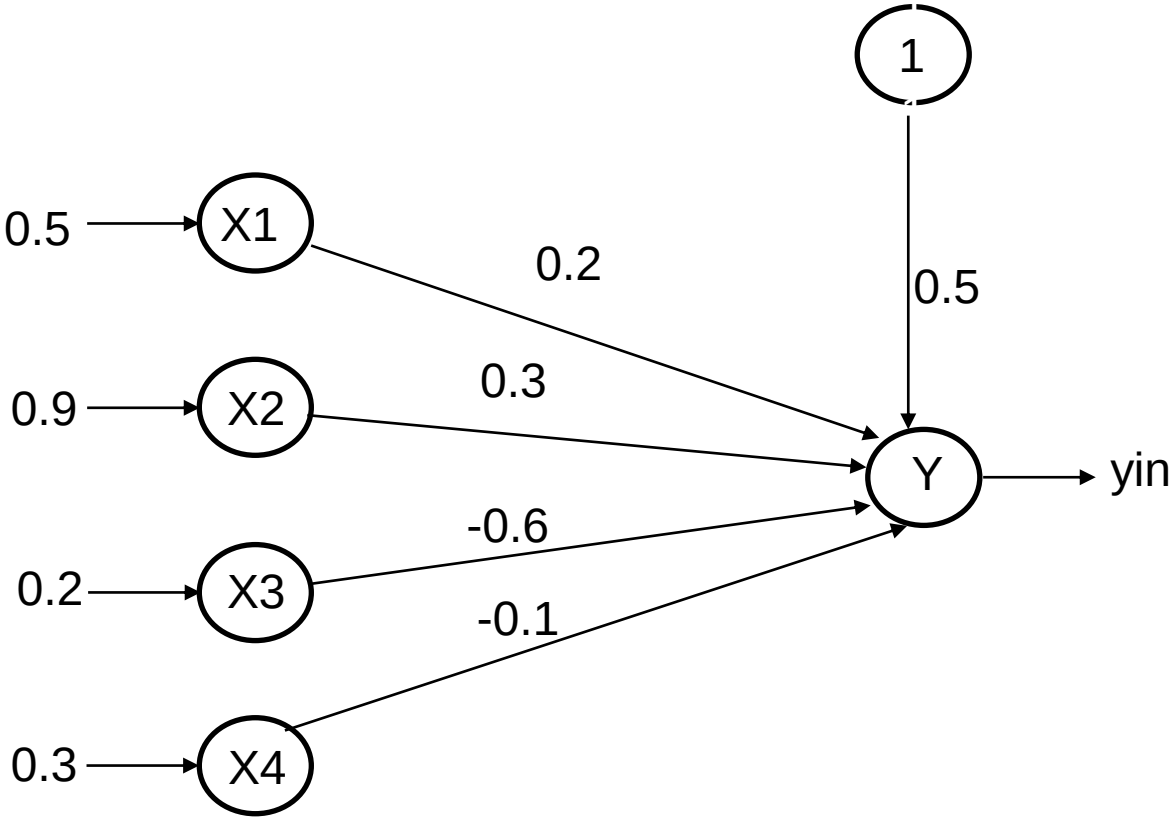
iii) For bipolar sigmoidal function,

$$y = f(y_{in}) = \frac{2}{1+e^{-y_{in}}} - 1 = \frac{2}{1+e^{-0.53}} - 1$$

$$= \underline{0.259}$$

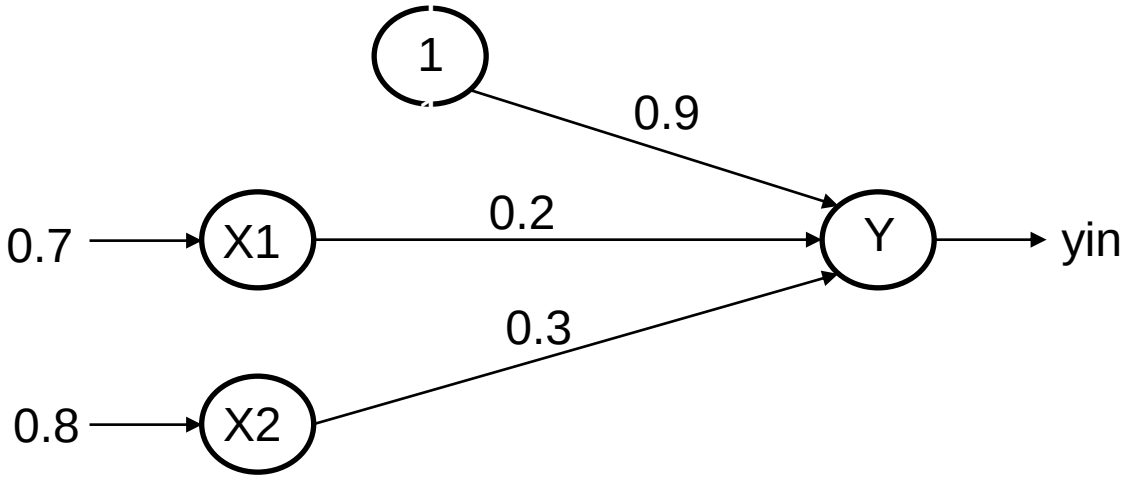
McCULLOCH–PITTS NEURON

Q: Calculate the output of neuron Y for the network shown in Figure.

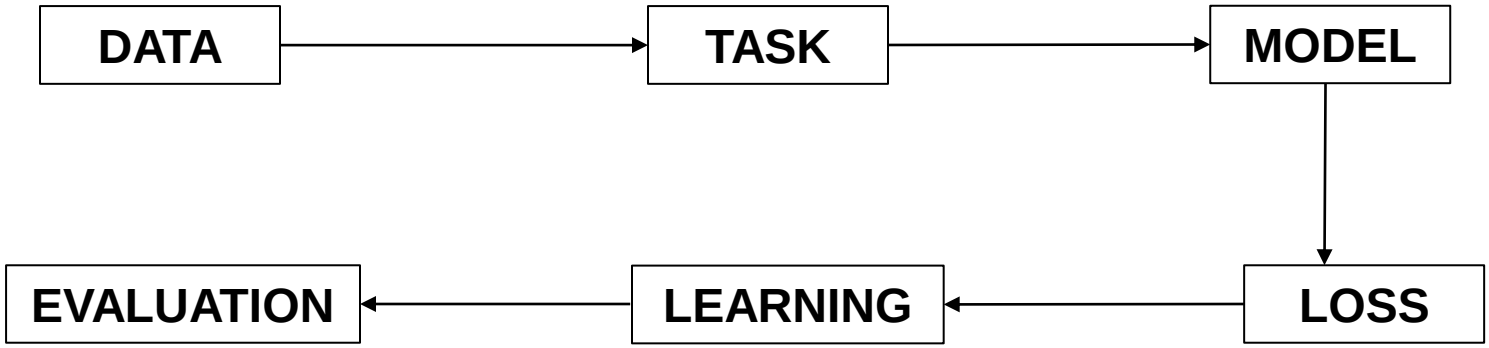


McCULLOCH–PITTS NEURON

Q: Calculate the output of neuron Y for the network shown in Figure. Use binary and bipolar sigmoidal activation function.

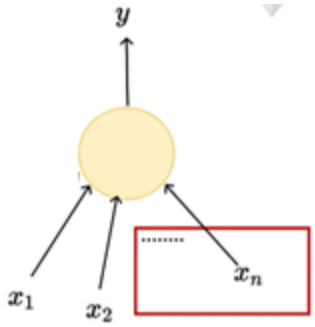


WORKING OF NEURAL NETWORKS



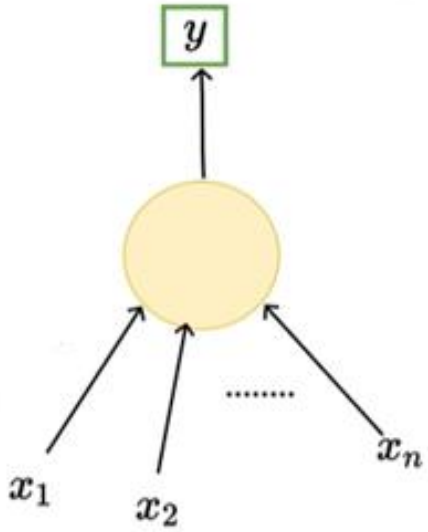
Data and Task

									
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (<160g)	1	0	1	0	0	0	1	0	0
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0
Price > 20k	0	1	1	0	0	0	1	1	1



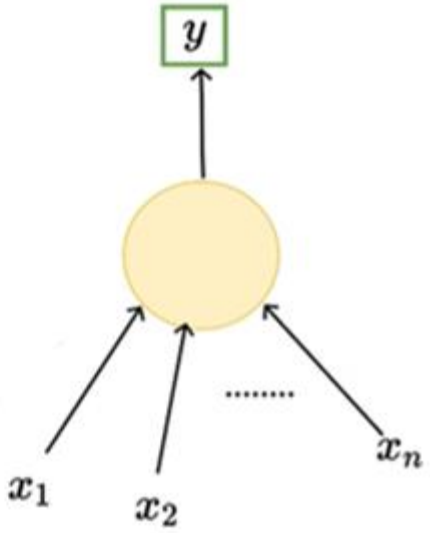
Data and Task

									
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (<160g)	1	0	1	0	0	0	1	0	0
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0
Price > 20k	0	1	1	0	0	0	1	1	1
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**



Model

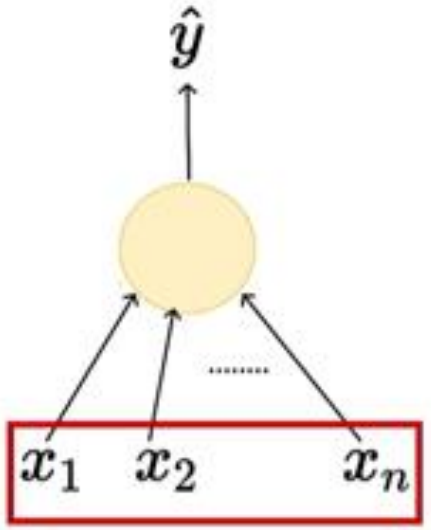
									
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (<160g)	1	0	1	0	0	0	1	0	0
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0
Price > 20k	0	1	1	0	0	0	1	1	1
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**



$$\hat{y} = \sum_{i=1}^{n} x_i \geq \text{Threshold}$$

Loss Function

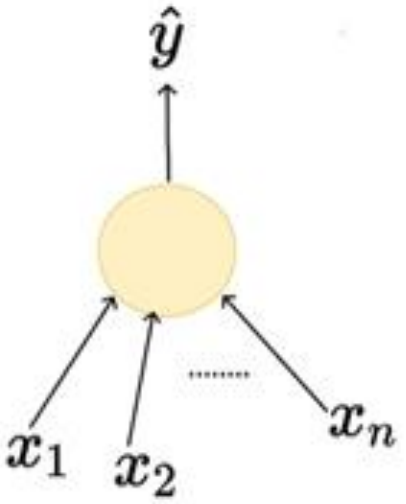
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (<160g)	1	0	1	0	0	0	1	0	0
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0
Price > 20k	0	1	1	0	0	0	1	1	1
Like (y)	1	0	1	0	1	1	0	1	0



$$\hat{y} = \sum_{i=1}^{n} x_i \geq \text{Threshold}$$

Loss Function

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (<160g)	1	0	1	0	0	0	1	0	0
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0
Price > 20k	0	1	1	0	0	0	1	1	1
Like (y)	1	0	1	0	1	1	0	1	0
Prediction $\hat{y}$	0	0	1	0	0	1	1	1	0

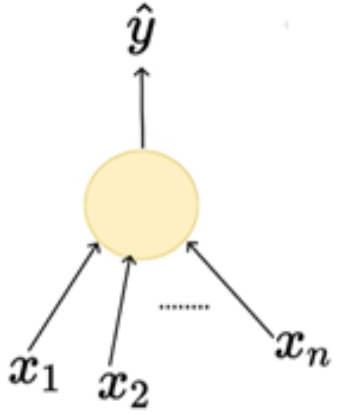


$$\hat{y} = \sum_{i=1}^{n} x_i \geq \text{Threshold}$$

$$loss = \sum_i y_i - \hat{y}_i$$

Loss Function

										
Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	0	**1**	**0**	1	1	**0**	**1**	0	0
prediction $\hat{y}$	1	0	**0**	**1**	1	1	**1**	**0**	0	0
loss	0	0	1	-1	0	0	-1	1	0	0

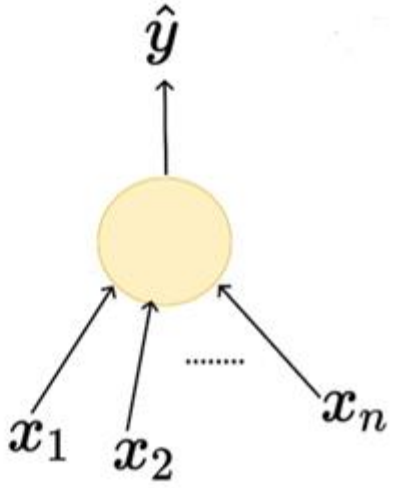


$\hat{y} = \sum_{i=1}^{n} x_i \geq$ Threshold

$$loss = \sum_i y_i - \hat{y}_i$$

Loss Function

Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	0	**1**	**0**	1	1	**0**	**1**	0	0
prediction $\hat{y}$	1	0	**0**	**1**	1	1	**1**	**0**	0	0
loss	0	0	1	-1	0	0	-1	1	0	0



$$\hat{y} = \sum_{i=1}^{n} x_i \geq \text{Threshold}$$

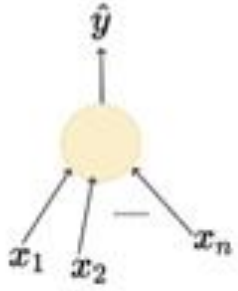
$$loss = \sum_i (y_i - \hat{y}_i)^2$$

Learning Algorithm

Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	1	1	0	0	1	1	1	0	0
prediction	?	?	?	?	?	?	?	?	?	?

$$\hat{y} = \sum_{i=1}^{n} x_i \geq \text{Th}$$

$$loss = \sum_i (y_i - \hat{y}_i)^2$$



Learning Algorithm

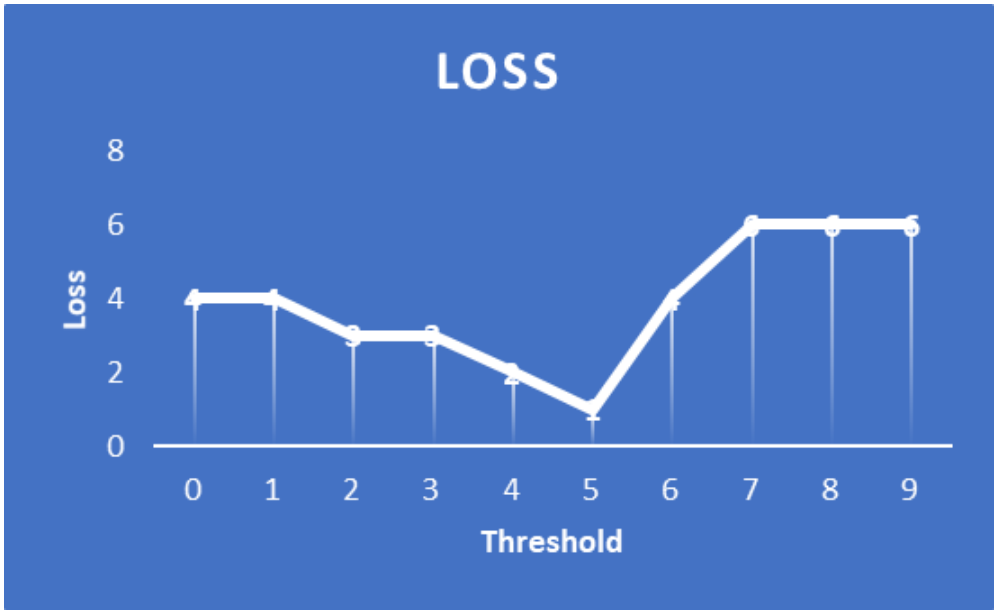
Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	1	1	0	0	1	1	1	0	0
prediction	?	?	?	?	?	?	?	?	?	?

Brute Force Approach

	Phone 1	Phone2	Phone3	Phone4	Phone5	Phone6	Phone7	Phone8	Phone9	Phone10	
Launch	0	1	1	0	0	1	0	1	1	0	
Weight	1	0	1	0	0	0	1	0	0	1	
Screen Size	1	0	1	0	1	0	1	0	1	0	
Dual sim	1	1	0	0	0	1	0	1	0	0	
Internal Memor	1	1	1	1	1	1	1	1	1	0	
NFC	0	1	1	0	1	0	1	1	1	0	
Radio	1	0	0	1	1	1	0	0	0	0	
Battery	0	0	0	1	0	1	0	1	0	0	
Price	0	1	1	0	0	0	1	1	1	0	Loss
Actual o/p	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	
pred(th=0)	1	1	1	1	1	1	1	1	1	1	4
pred(th=1)	1	1	1	1	1	1	1	1	1	1	4
pred(th=2)	1	1	1	1	1	1	1	1	1	0	3
pred(th=3)	1	1	1	1	1	1	1	1	1	0	3
pred(th=4)	1	1	1	0	1	1	1	1	1	0	2
pred(th=5)	1	1	1	0	0	1	1	1	1	0	1
pred(th=6)	0	0	1	0	0	0	0	1	0	0	4
pred(th=7)	0	0	0	0	0	0	0	0	0	0	6
pred(th=8)	0	0	0	0	0	0	0	0	0	0	6
pred(th=9)	0	0	0	0	0	0	0	0	0	0	6

Brute Force Approach

Threshold	loss
0	4
1	4
2	3
3	3
4	2
5	1
6	4
7	6
8	6
9	6

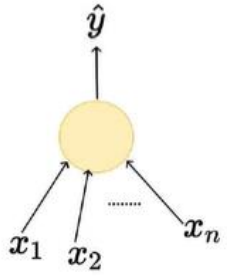


Evaluation

Training data

	Phone 1	Phone2	Phone3	Phone4	Phone5	Phone6	Phone7	Phone8	Phone9	Phone10
Launch	0	1	1	0	0	1	0	1	1	0
Weight	1	0	1	0	0	0	1	0	0	1
Screen Size	1	0	1	0	1	0	1	0	1	0
Dual sim	1	1	0	0	0	1	0	1	0	0
Internal Memory	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery	0	0	0	1	0	1	0	1	0	0
Price	0	1	1	0	0	0	1	1	1	0
Actual o/p	1	1	1	0	0	1	1	1	0	0
pred(th=5)	1	1	1	0	0	1	1	1	1	0

$$loss = \sum_i (y_i - \hat{y}_i)^2$$



Evaluation

Training data

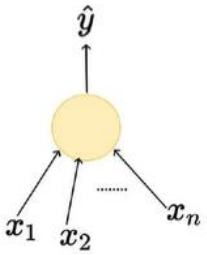
Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	1	1	0	0	1	1	1	0	0
predicted	1	1	0	1	1	1	1	0	0	0

Test Data

	phone11	phone12	phone13	phone14
Launch (within 6 months) x_1	1	0	0	1
Weight (<160g) x_2	0	1	1	1
Screen Size (< 5.9in) x_3	0	1	1	1
Dual sim x_4	0	1	0	0
Internal mem(>= 64gb, 4gb ram) x_5	1	0	0	0
NFC x_6	0	0	1	0
Radio x_7	1	1	1	0
Battery (>= 3500mAh) x_8	1	1	1	0
Price? (> 20k) x_9	0	0	1	0
Liked (y)	0	1	0	0
Prediction $\hat{y}$	0	1	1	0

Threshold = 5

$$loss = \sum_i (y_i - \hat{y}_i)^2$$



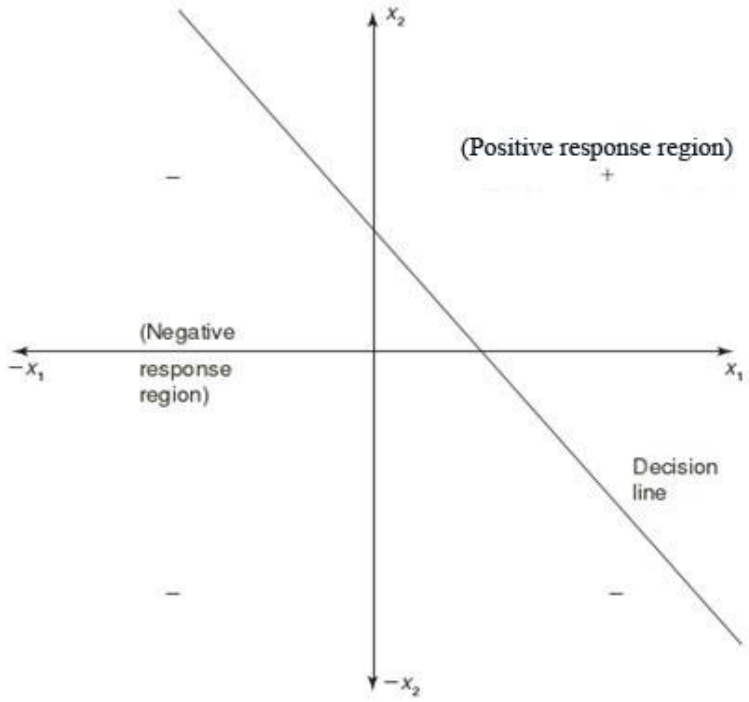
$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Accuracy = ¾ = 75%

LINEAR SEPARABILITY

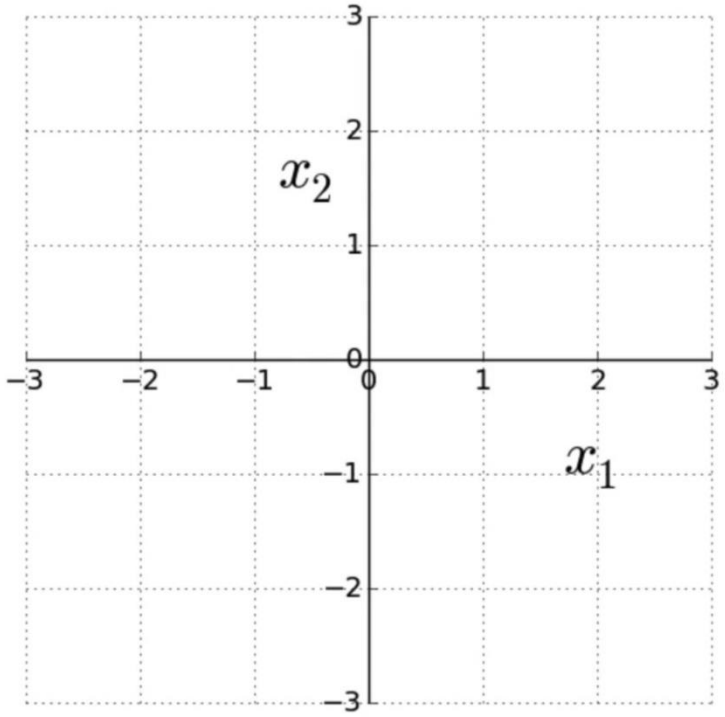
- Linear separability is the concept wherein the separation of the input space into regions is based on whether the network response is positive or negative.

- Consider a network having positive response in the first quadrant and negative response in all other quadrants (AND function) with either binary or bipolar data, then the decision line is drawn separating the positive response region from the negative response region.



LINEAR SEPARABILITY

Geometric Interpretation



LIN

Generally, the net input calculated to the output unit is given as –

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

Consider, a single layer network as shown in fig.

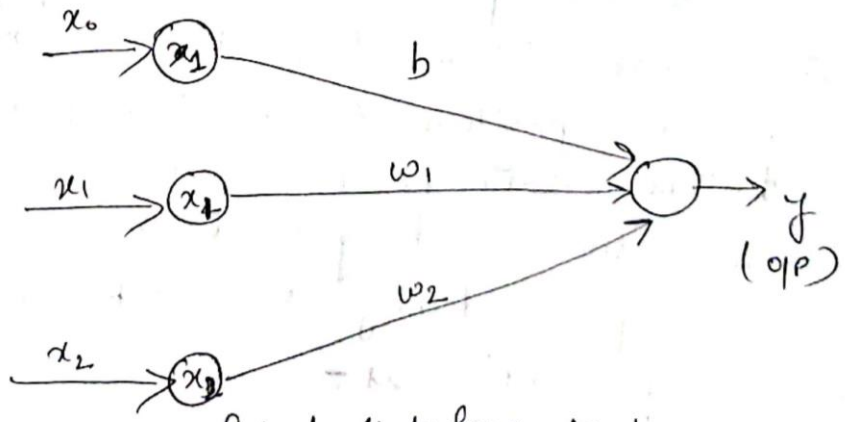


Fig. A single layer network

The net input for the network shown in fig is given as -

$$y_{in} = b + x_1 w_1 + x_2 w_2$$

The separating line for which the boundary lies between the values x_1 and x_2, so that the net gives a positive response on one side and negative response on the other side, is given as,

$$b + x_1 w_1 + x_2 w_2 = 0$$

$x_2 w_2 = -x_1 w_1 - b$

$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$

If weight w_2 is not equal to 0 then we get,

$$\boxed{x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}}$$

Thus, the requirement for the positive response of the net is

$$b + x_1 w_1 + x_2 w_2 > 0$$

Net input received $> \theta$ (threshold)

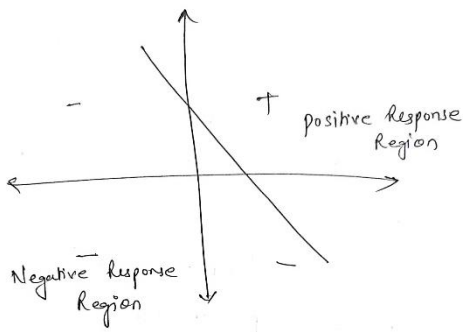
$$y_{in} > \theta$$

$$x_1 w_1 + x_2 w_2 > \theta$$

The separating line equation will then be,

$$x_1 w_1 + x_2 w_2 = \theta$$

$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2} \quad (\text{with } w_2 \neq 0)$$



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."

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

Hebb Rule is more suited for bipolar data.

HEBB NETWORK

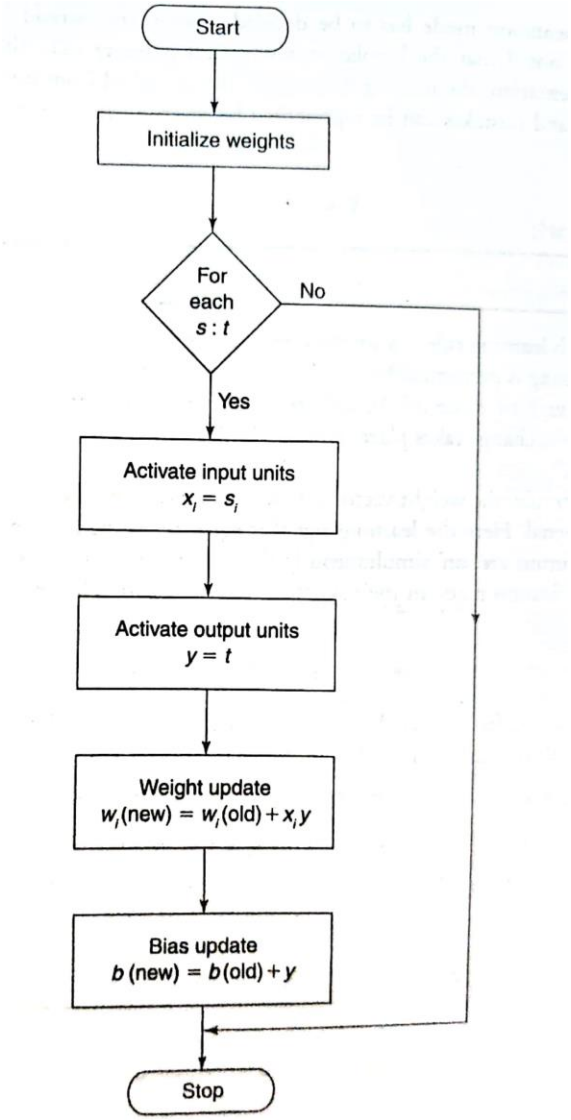
Donald Hebb stated in 1949 that in the brain, the learning is performed by the change in the synaptic gap. Hebb explained it:

"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."

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

- Hebb Rule is more suited for bipolar data.
- The Hebb rule can be used for pattern association, pattern categorization, pattern classification and over a range of other areas.

HEBB LEARNING



HEBB LEARNING

Step 0: First initialize the weights. Basically in this network they may be set to zero, i.e., $w_i = 0$ for $i = 1$ to n where "n" may be the total number of input neurons.

Step 1: Steps 2–4 have to be performed for each input training vector and target output pair, $s : t$.

Step 2: Input units activations are set. Generally, the activation function of input layer is identity function: $x_i = s_i$ for $i = 1$ to n.

Step 3: Output units activations are set: $y = t$.

Step 4: Weight adjustments and bias adjustments are performed:

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

The above five steps complete the algorithmic process. In Step 4, the weight updation formula can also be given in vector form as

$$w\,(\text{new}) = w\,(\text{old}) + xy$$

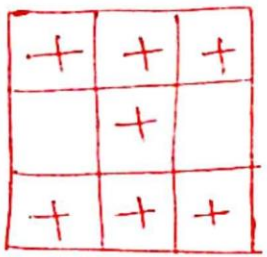
Here the change in weight can be expressed as

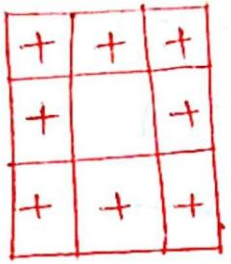
$$\Delta w = xy$$

As a result,

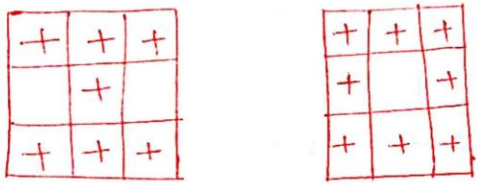
$$w\,(\text{new}) = w\,(\text{old}) + \Delta w$$

Q: Using the Hebb rule, find the weights required to perform the following classifications of the given input patterns shown in fig. The '+' symbol represents the value "1" and the empty square indicates "-1". Consider "I" belongs to the member of class and "O" doesn't belong to the member of class.





Q: Using the Hebb rule, find the weights required to perform the following classifications of the given input patterns shown in fig. The '+' symbol represents the value "1" and the empty square indicates "-1". Consider "I" belongs to the member of class and "O" doesn't belong to the member of class.



Sol^n: The training input patterns for the given network are indicated in following table:-

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
I	1	1	1	-1	1	-1	1	1	1	1	1
O	1	1	1	1	-1	1	1	1	1	1	-1

Here, a single layer network with nine input neurons, one bias and one o/p neuron is formed. We will set the initial weights and bias to zero i.e $w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = w_7 = w_8 = w_9 = b = 0$

Pattern	Inputs									Bias	Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
I	1	1	1	-1	1	-1	1	1	1	1	1

Case 1: Presenting first input pattern (I), we calculate change in weights:

$\Delta w_i = x_i y \quad i = 1 \text{ to } 9$

$\Delta w_1 = x_1 y = 1 \times 1 = 1$

$\Delta w_2 = x_2 y = 1 \times 1 = 1$

$\Delta w_3 = x_3 y = 1 \times 1 = 1$

$\Delta w_4 = x_4 y = -1 \times 1 = -1$

$\Delta w_5 = x_5 y = 1 \times 1 = 1$

$\Delta w_6 = x_6 y = -1 \times 1 = -1$

$\Delta w_7 = x_7 y = 1 \times 1 = 1$

$\Delta w_8 = x_8 y = 1 \times 1 = 1$

$\Delta w_9 = x_9 y = 1 \times 1 = 1$

$\Delta b = y = 1$

We now calculate the new weight using the formula,

$$w_i(new) = w_i(old) + \Delta w_i$$

By using this formula, we will get,

$w_1(new) = w_1(old) + \Delta w_1 = 0 + 1 = 1$

$w_2(new) = w_2(old) + \Delta w_2 = 0 + 1 = 1$

$w_3(new) = w_3(old) + \Delta w_3 = 0 + 1 = 1$

$w_4(new) = w_4(old) + \Delta w_4 = 0 + (-1) = -1$

$w_5(new) = \Delta w_5(old) + \Delta w_5 = 0+1 = 1$

$w_6(new) = w_6(old) + \Delta w_6 = 0+(-1) = -1$

$w_7(new) = w_7(old) + \Delta w_7 = 0+1 = 1$

$w_8(new) = w_8(old) + \Delta w_8 = 0+1 = 1$

$w_9(new) = w_9(old) + \Delta w_9 = 0+1 = 1$

$b(new) = b_0(old) + \Delta b = 0+1 = 1$

The weight after presenting the first i/p pattern are

$w(new) = [1 \quad 1 \quad 1 \quad -1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 1]$

Case 2: Now, we will present second i/p pattern (0).

The initial weights used here are the final weights obtained after presenting the first i/p pattern. Here, the weights are calculated as shown below:-

$t = -1$

initial weights $= [1 \quad 1 \quad 1 \quad -1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 1]$

$w_i(new) = w_i(old) + \Delta w_i$

$(\Delta w_i = x_{ij})$

Initial weights = [1 1 1 −1 1 −1 1 1 1 1]

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
I	1	1	1	−1	1	−1	1	1	1	1	1
0	1	1	1	1	−1	1	1	1	1	1	−1

$w_1(new) = w_1(old) + x_1 y = 1 \times 1 = 0$

$w_2(new) = w_2(old) + x_2 y = 1 + 1 \times -1 = 0$

$w_3(new) = w_3(old) + x_3 y = 1 + 1 \times -1 = 0$

$w_4(new) = w_4(old) + x_4 y = -1 + 1 \times -1 = -2$

$w_5(new) = w_5(old) + x_5 y = 1 + -1 \times -1 = 2$

$w_6(new) = w_6(old) + x_6 y = -1 + 1 \times -1 = -2$

$w_7(new) = w_7(old) + x_7 y = 1 + 1 \times -1 = 0$

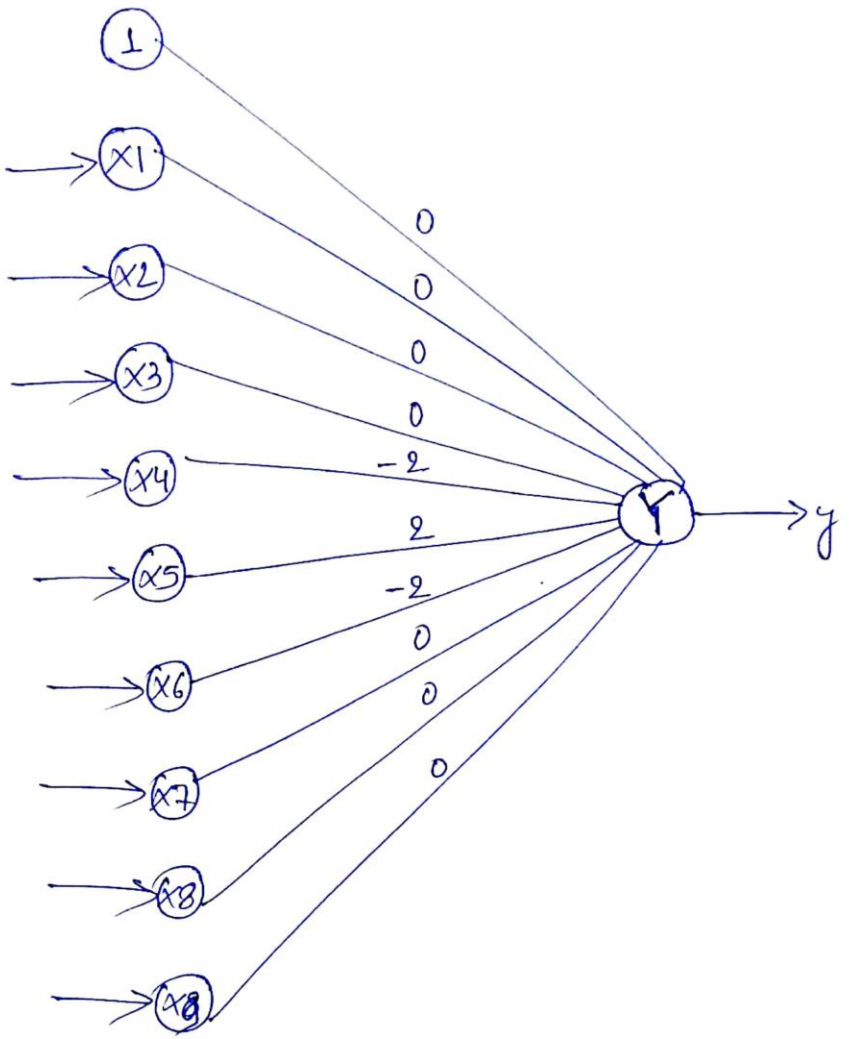
$w_8(new) = w_8(old) + x_8 y = 1 + 1 \times -1 = 0$

$w_9(new) = w_9(old) + x_9 y = 1 + 1 \times -1 = 0$

$b(new) = b(old) + y = 1 + 1 \times -1 = 0$

The final weight after presenting the second input pattern are given as,

W(new) = [0 0 0 −2 2 −2 0 0 0 0]



Hebb net for the given data matrix

Q°. Design a Hebb network to implement logical AND function (Bipolar inputs).

Q:- Design a Hebb network to implement logical AND function (Bipolar inputs).

Sol^{n}:- The training data for AND function,

x_1	x_2	y	b
1	1	1	1
1	-1	-1	1
-1	1	-1	1
-1	-1	-1	1

Initially, the weights and bias are set to 0, i.e,

$$w_1 = w_2 = b = 0.$$

First Input :-

$$[x_1 \quad x_2 \quad b] = [1 \quad 1 \quad 1]$$

target $y = 1$

Setting the initial weights as old weights and applying the Hebb rule we get,

$$\boxed{w_i(new) = w_i(old) + x_i y}$$

$$w_1(new) = w_1(old) + x_1 y = 0 + 1 \times 1 = 1$$
$$w_2(new) = w_2(old) + x_2 y = 0 + 1 \times 1 = 1$$
$$b(new) = b(old) + y = 0 + 1 = 1$$

The weight calculated above are the final weights that are obtained after presenting the first input. These weights are used as initial weights when the second input is presented.

The change in weight here is calculated as, $\boxed{\Delta w_i = x_i y}$

$$\Delta w_1 = 1$$
$$\Delta w_2 = 1$$
$$\Delta b = 1$$

Second Input :- $[x_1 \quad x_2 \quad b] = [1 \quad -1 \quad 1]$

target $y = -1$

The weight change here is,

$$\Delta w_i = x_i y$$

$\Delta w_1 = -1$

$\Delta w_2 = 1$

$\Delta b = -1$

The new weights are,

$w_1(new) = w_1(old) + \Delta w_1 = 1 + (-1) = 0$

$w_2(new) = w_2(old) + \Delta w_2 = 1 + 1 = 2$

$b(new) = b(old) + \Delta b = 1 - 1 = 0$

* Third Input : $[x_1 \; x_2 \; b] = [-1 \; 1 \; 1]$

target $[y] = -1$

The weight change here is,

$$\Delta w_i = x_i y$$

$\Delta w_1 = +1 \quad \Delta w_2 = -1 \quad \Delta b = -1$

The new weights are,

$w_1(new) = w_1(old) + \Delta w_1 = 0 + 1 = 1$

$w_2(new) = w_2(old) + \Delta w_2 = 2 + (-1) = 1$

$b(new) = b(old) + \Delta b = 0 + (-1) = -1$

* Fourth Input :- $[x_1 \; x_2 \; b] = [-1 \; -1 \; 1]$

target $[y] = [-1]$

weight change :- $\Delta w_1 = 1 \quad \Delta w_2 = 1 \quad \Delta b = -1$

New weights :- $w_1(new) = 2$

$w_2(new) = 2$

$b(new) = -2$

Inputs				Weight change			weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0)	w_2 (0)	b (0)
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

Inputs			Output	Weight change			weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0)	w_2 (0)	b (0)
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

The separating line equation is given as,

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}$$

For all inputs, use the final weight obtained for each input to obtain the separating line.

For 1st input [1 1 1], the separating line is given by,

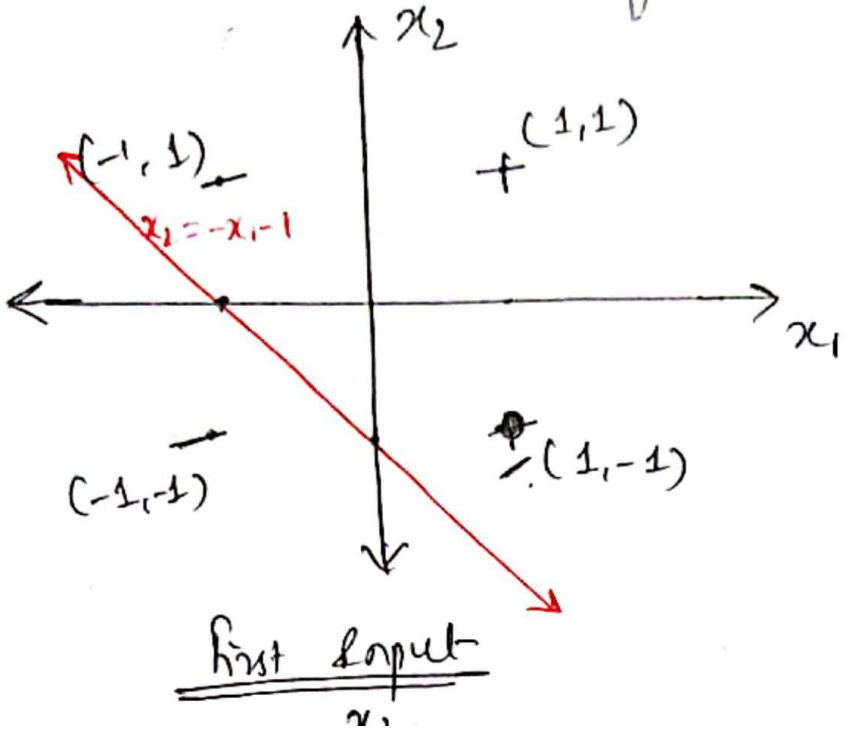
$$x_2 = \frac{-1}{1} * x_1 - \frac{1}{1} = -x_1 - 1$$

2nd input, $x_2 = \frac{-0}{2} x_1 - 0 \Rightarrow x_2 = 0$

3rd input, $x_2 = \frac{-1}{1} x_1 - \frac{(-1)}{1} = -x_1 + 1$

4th input $x_2 = \frac{-2}{2} x_1 - \frac{(-2)}{2} = -x_1 + 1$

The Graph for each of these separating lines is obtained as follows:—



$0 = -x_1 - 1 \Rightarrow$

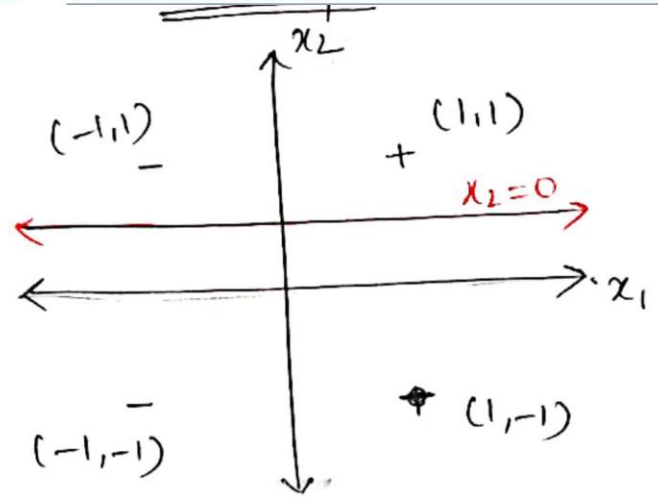
$x_2 = -x_1 - 1$ (0, -1)

$x_1 = 0 \Rightarrow x_2 = -1$ (-1, 0)

$x_2 = 0 \Rightarrow x_1 = -1$

$x_2 = -x_1 - 1$

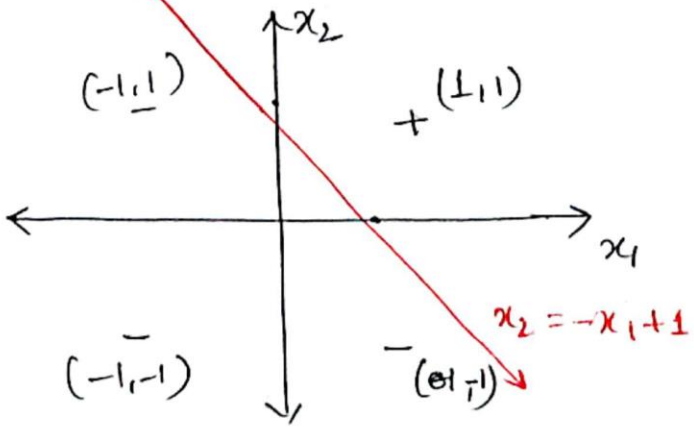
Here, the decision boundary separates only 1st and 4th input, not all negative responses are separated from positive response.



$x_2 = 0$

Here, decision boundary only separates (1,1) from (1,-1) & (-1,-1) not from (-1,1).

Second Input



Third & fourth Input

$x_2 = 0 \Rightarrow x_1 = 1$ (1,0)

$x_1 = 0 \Rightarrow x_2 = 1$ (0,1)

$x_2 = -x_1 + 1$

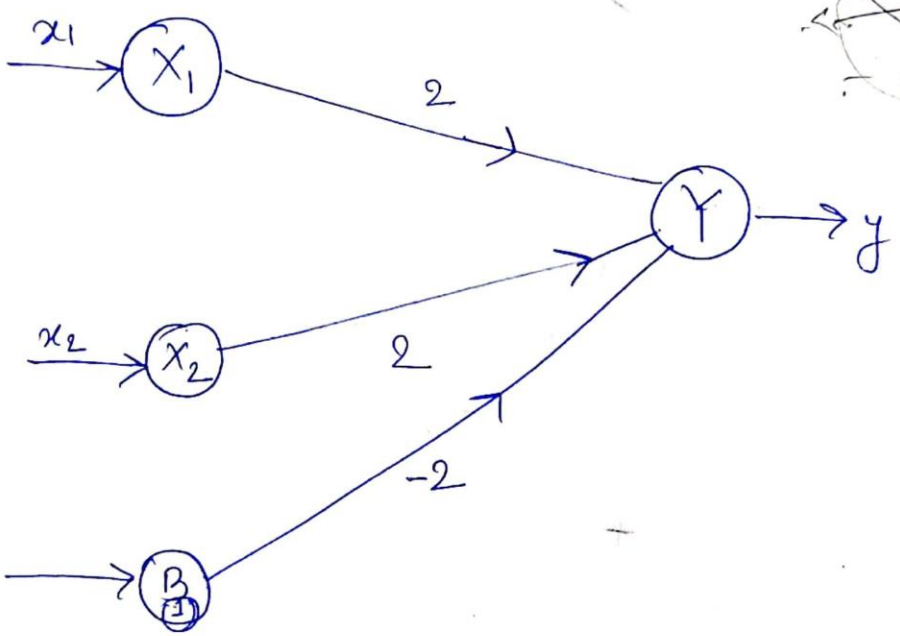
Decision boundary line obtained here separates the positive response region from all negative response region.

By, overviewing the above graphs we can conclude that,

weights obtained from this are final weights and are given as,

$$w_1 = 2 \quad w_2 = 2 \quad \& \quad b = -2$$

The network can be represented as,



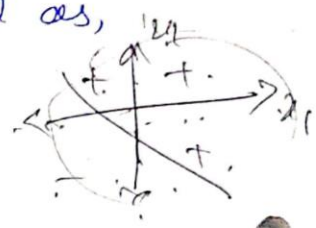
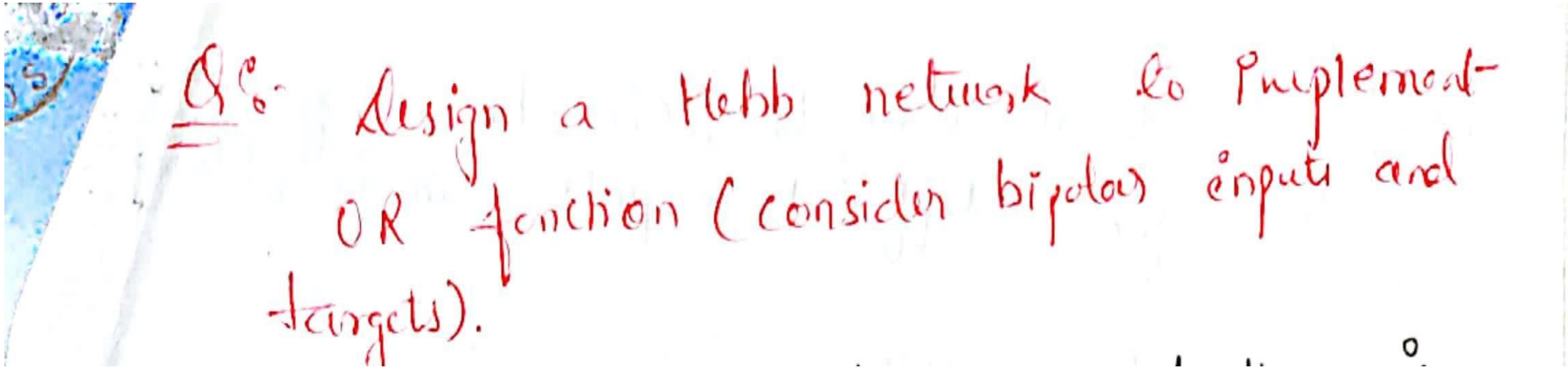


Fig: Hebb Net for AND function


Q°. Design a Hebb network to Implement
OR function (consider bipolar inputs and
targets).

Q:- Design a Hebb network to implement OR function (consider bipolar inputs and targets).

Sol^n:- The training pair for the OR function is given as,

Inputs			Targets
x_1	x_2	b	y
1	1	1	1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

Initially, the weights & bias are set to 0.

i.e $w_1 = 0 ; w_2 = 0 ; b = 0$

Hebb Training:-

Inputs				weight changes			weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
1	1	1	1	1	1	1	1	1	1
1	-1	1	1	1	-1	1	2	0	2
-1	1	1	1	-1	1	1	1	1	3
-1	-1	1	-1	1	+1	-1	2	2	2

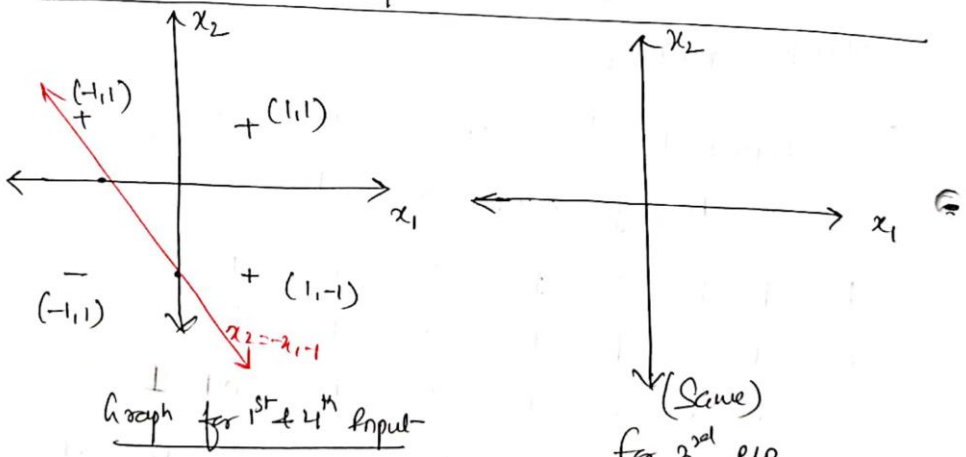
$\Delta w_i = x_i y$

$w_i (new) = w_i (old) + \Delta w_i$

Using, the final weights, the boundary line equation for different inputs can be obtained-

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}$$

Input	Boundary line Equation
[1 1 1]	$x_2 = -x_1 - 1$
[1 -1 1]	$x_2 = \infty$
[-1 1 1]	$x_2 = -x_1 - 3$
[-1 -1 1]	$x_2 = -x_1 - 1$



Graph for 1st & 4th Input-

for 3rd I/P

Decision Boundary line separates the I/P pattern (1,1), (-1,1) & (1,-1) from (-1,-1) i.e positive response is separated by negative response.

Hence, the weights obtained from this are the final weights, and are given as,

$$w_1 = 2, \quad w_2 = 2, \quad b = 2$$

The network can be represented as -

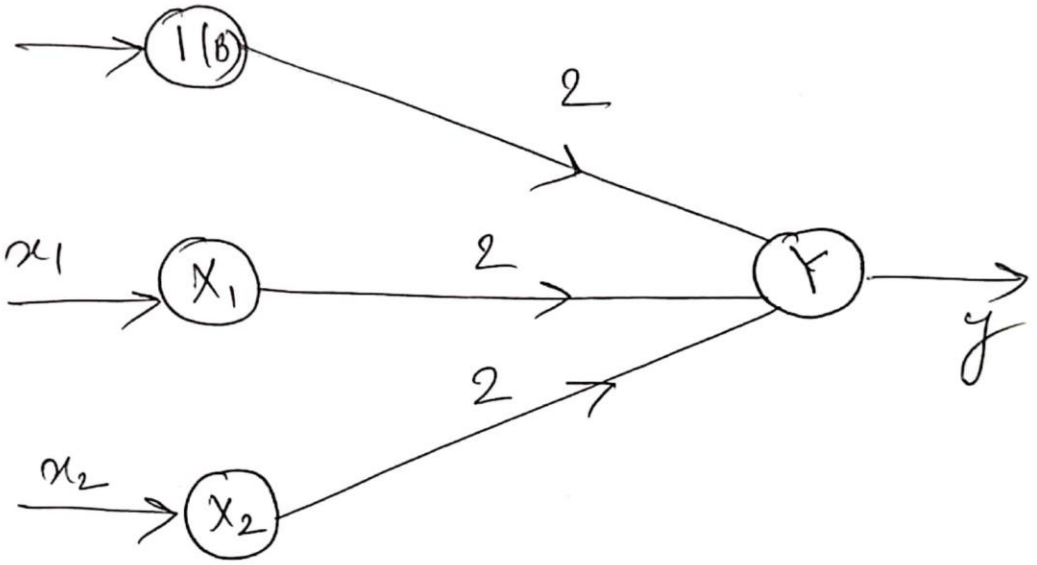


Fig:- Hebb Net for OR function

FEW APPLICATIONS 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