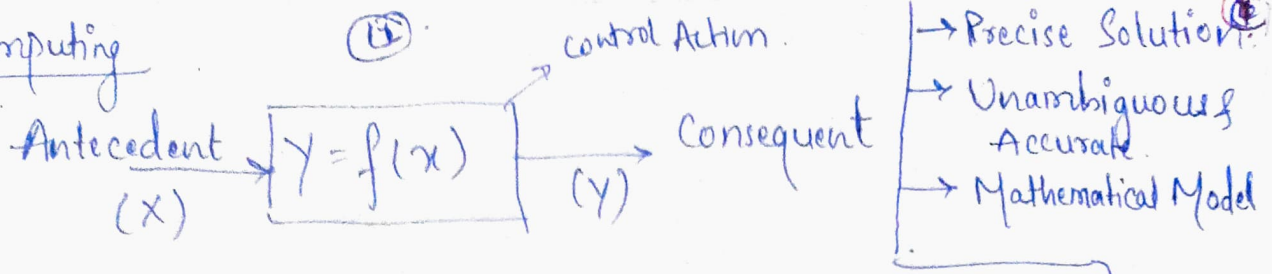
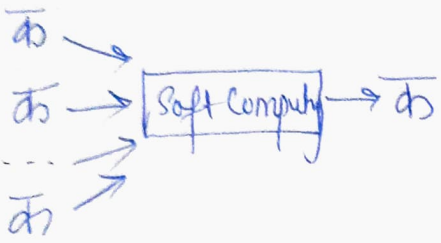
Computing

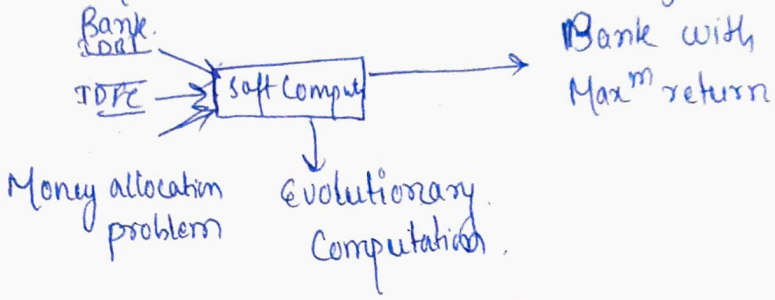


f : formula / Method / Algorithm / Mapping function

Soft-Computing :

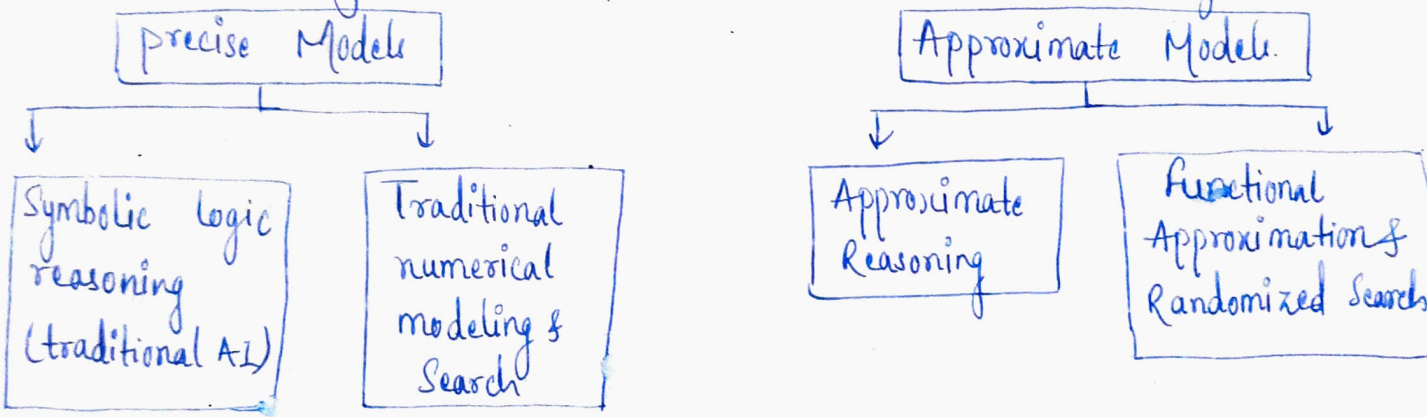


Example : Animal character Recognition.



Hard Computing → Deals with precise models.

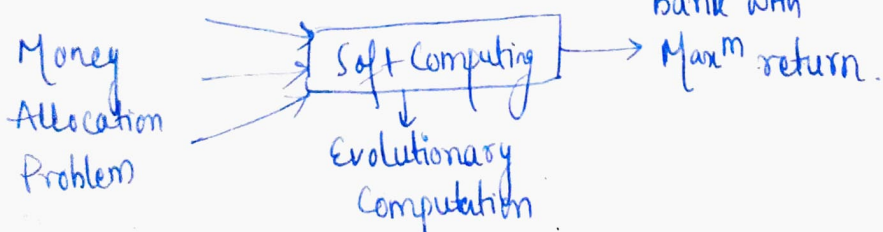
Soft Computing



* Example of Soft Computing :

① Animal character Recognition

② Banking



SC = FL + NN + GA → for evolutionary Computation

FL → for knowledge rep^n (like 0 & 1)

NN → for learning & Adaptation (brain like human)

Goals of Soft Computing:

1. Its main aim is to exploit the tolerance for

(i). Approximation - here the model features are similar to the real ones, but not the same.

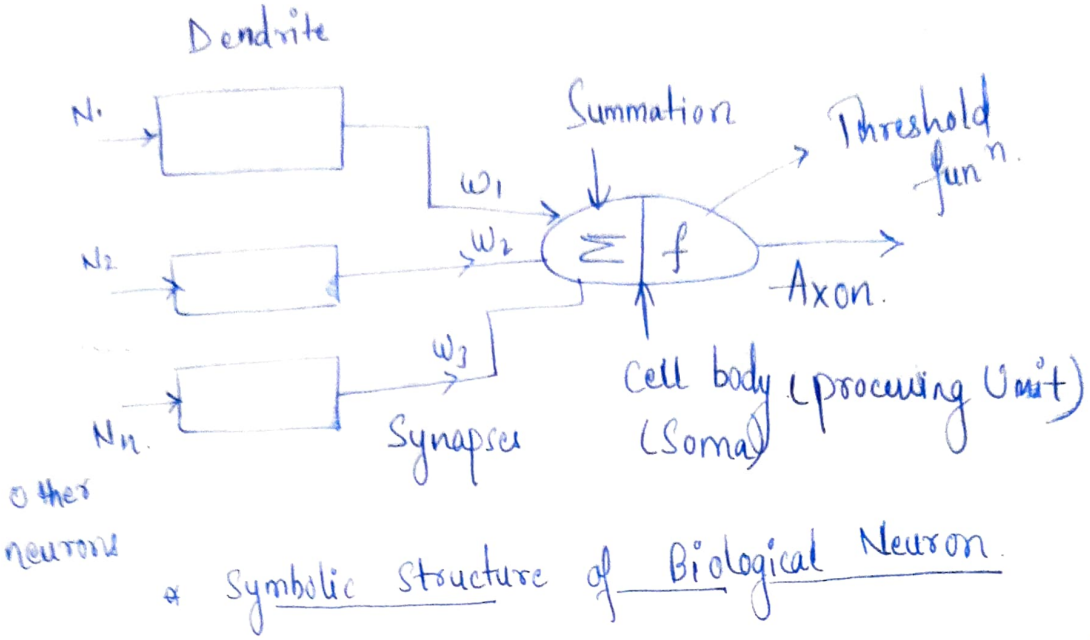
(ii). Uncertainty - here we are not sure that the features of the Model are the same as that of the entity or belief.

(iii). Imprecision - here the model features (quantities) are not the same as that of the real ones but close to them.

Important Questions:

1. What is Computing?
2. What are Hard and Soft Computing?
3. Difference Between Hard and Soft Computing.

2. Biological Neurons and its Structure



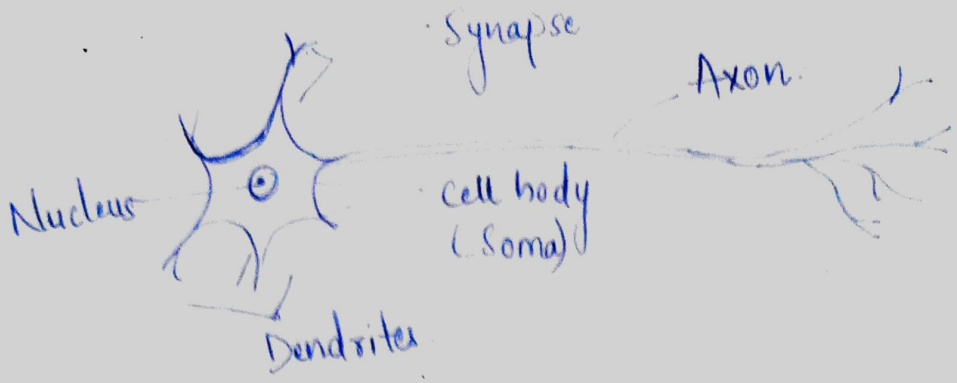
* Symbolic Structure of Biological Neuron.

1. Dendrite - where the nerve is connected to the cell body. (input)
   - It receives signals from other neurons.

2. Soma (Cell body) - where the cell nucleus is located.
   - two major function: (i) Summation - Sum up all the incoming input signals.
   (ii). Threshold function - threshold value is fixed if the sum cross the threshold value then it generates other signals.

3. Synapse - It is the Connection Between the axon and other neuron dendrites.
   - interConnection between One neuron to other neuron.
   - It is through Synapse that the neurons introduces its signal to other nearby neurons.

4. Axon - which carries the impulses of the neurons.
   - It is a single, long Connection extending from the cell body and carrying signals from the neurons.
   (basically generates the Output).



• Schematic Diagram of a biological neuron.

process.

(i). Electric impulses are passed between the Synapse and the dendrites

↔ This type of signal transmission involves a chemical process in which specific transmitter substances are released from the sending side of the junction.

→ Due to this there is increase or decrease in the electric potential inside the body of receiving cell.

→ If the electric potential reaches a threshold then the receiving cell fires and a pulse of fixed strength and duration is sent out through the axon to the synaptic junctions of the other cells.

→ After firing, A cell has to wait for a period of time called the refractory period before it can fire again.

→ here, two terms are introduced one is inhibitory Synapse - if they let passing impulses hinder the firing of the receiving cell. or Excitatory Synapse - if they let passing impulses cause the firing of receiving cell.

Synapse : (Weighted connections) - to calculate the amount of signals.

- The amount of signal transmitted depends upon the strength (synaptic weight) of the connections.
- Connection can be have increasing or decreasing strength. It can be Inhibitory which is decreasing strength or Excitatory which is for increasing strength in the nature.
- If we increase the weight of signals the strength will be increase means excitatory Synapse and if reduces the weights of connection then inhibitory Synapse.

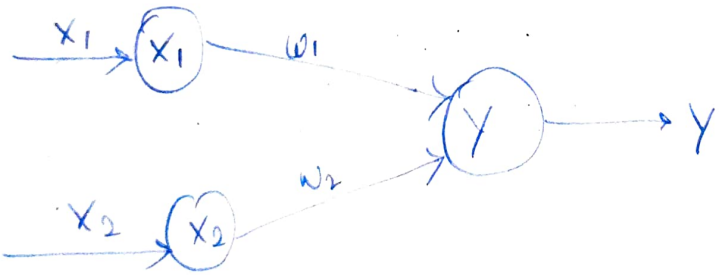
Artificial Neural Network :

→ It is an efficient information processing System which resembles in characteristics with a biological neural network.

→ It poses large number of highly interconnected processing elements called node or Units or neurons, which usually operate in parallel and are configured in regural architecture.

→ Each neuron is connected with the other by a connection link. Each connection link is associated with weights which contain information about the input signal.

→ This information is used by the neuron net to solve a particular problem.



• Architecture of a Simple Artificial neuron net.

Each neuron has an internal state of its own.

→ This internal state is called the activation or activity level of neuron, which is the function of the inputs the neuron receives.

→ The activation signal of a neuron is transmitted to other neurons.

Calculation:

→ Consider a set of neurons – X_1 & X_2 – input neurons. transmit the signal to another neuron Y.

Here, X_1 & X_2 = input neurons, – which transmit signals.
Y = Output Neuron – which receives signals.

X_1 & X_2 are connected to Output neuron Y over a weighted interconnection links (w_1 & w_2).

So, net input has to be calculated in this way –

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

→ Here, x_1 & x_2 are the activations of input neurons X_1 & X_2 i.e the Output of input signals.

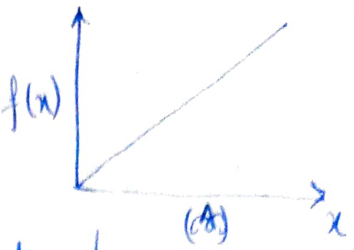
The Output y of the neuron Y can be obtained by applying activations over the net input. i.e. function of net input:

$$y = f(y_{in})$$

Output = function (net input calculated)

→ The function to be applied over the net input is called activation function.

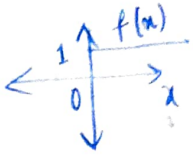
Activation function:



1. Identity function:

→ It is a linear function. The Output here remains the same as input. The input layer Uses the identity activation function.

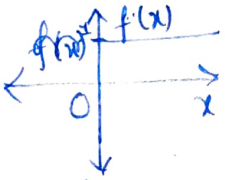
defined as $f(x) = x$ for all x



2. Binary step function:

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

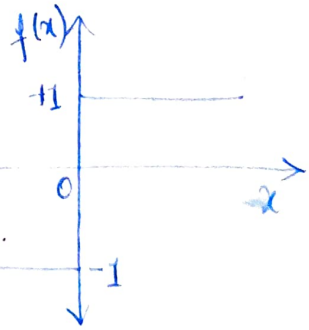
← Threshold Value.



→ This function is Used in Single-layer nets to Convert the net input to an Output that is binary (1 or 0).

3. Bipolar step function:

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



→ This function is also Used in Single-layer nets to Convert the net input to an Output that is bipolar (+1 or −1)

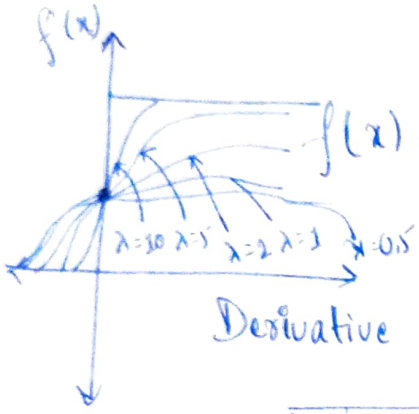
4. Sigmoidal function:

→ The Sigmoidal function are widely Used in back-propagation nets because of the relationship b/w the

Value of the function at a point and the value of derivative at that point, which reduces the Computational burden during training.

→ Two types:

1. Binary Sigmoid function: Also termed as Logistic Sigmoid function or Unipolar Sigmoid function.



$$f(x) = \frac{1}{1+e^{-\lambda x}}$$, λ = Steepness parameters.

Derivative of this function is

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

→ Here range of Sigmoid function is from 0 to 1.

2. Bipolar Sigmoid function:

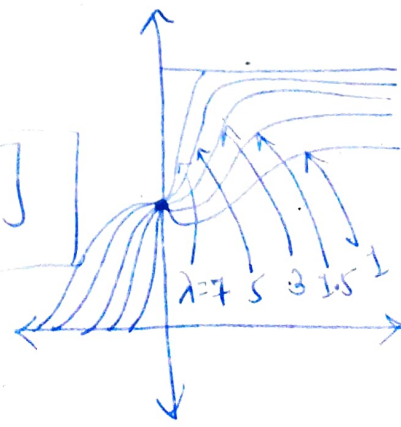
$$f(x) = \frac{2}{1+e^{-\lambda x}} - 1 = \frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}$$

λ = Steepness parameter

Sigmoid function range is between -1 and +1.

Derivative of this function.

$$f'(x) = \frac{\lambda}{2}[1+f(x)][1-f(x)]$$



→ The Bipolar Sigmoid function is closely related to hyperbolic tangent function,

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

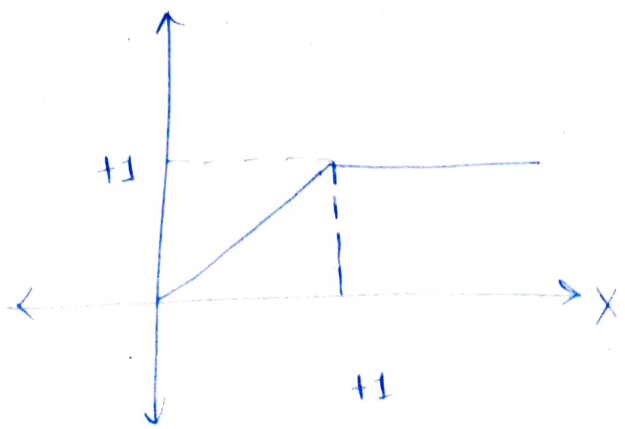
The derivative of Hyperbolic tangent function is

$$h'(x) = [1 + h(x)][1 - h(x)]$$

→ If network Uses a binary data, it is better to Convert it to bipolar form and Use bipolar Sigmoidal activation function function or hyperbolic tangent function.

5. Ramp function:

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



• Graphical Representation of Ramp function

4. Important Terminologies of ANN's

(a). Weights: The weights contain information about the input signal.

→ The weight can be represented in terms of matrix.

→ The weight matrix can also be called connection matrix.

→ To form a mathematical notation, it is assumed that there are "n" processing elements in an ANN and each processing element has exactly "m" adaptive weights.

Then weight matrix is

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_n^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & & & \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix}$$

where,

$w_i = [w_{i1}, w_{i2}, \ldots w_{im}]^T$, $i = 1, 2, \ldots n$ is weight vector of processing element. "i" (Source

w_{ij} = weight from processing element "i" (Source node) to processing element "j" (destination node).

(b) Bias :

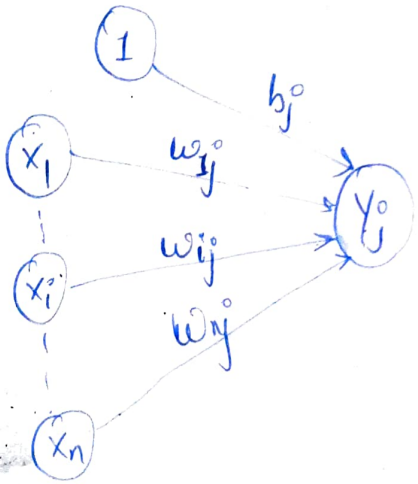
→ Bias included in the network has its impact in calculating the net input.

→ The bias is included by adding a component $x_0 = 1$ to the input vector X. Thus, input vector becomes —

$$X = (1, x_1, \ldots, x_i, \ldots, x_n)$$

→ The bias is considered like another weight, that is $w_{0j} = b_j$.

Example: Simple net with bias :



net input to the Output neuron Y_j is calculated as

$$Y_{inj} = \sum_{i=0}^{n} x_i w_{ij} = x_0 w_{0j} + x_1 w_{1j} + x_2 w_{2j} + \ldots + x_n w_{nj}$$

$$= w_{0j} + \sum_{i=1}^{n} x_i w_{ij}$$

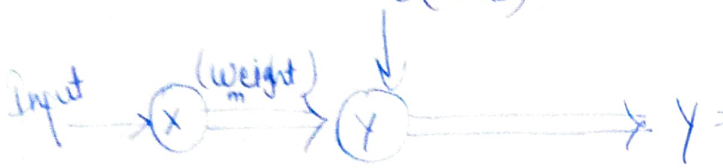
$$y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

(ii) when activation function is applied over this to calculate the output. The bias can also explained as follows

When equation of straight line,

$$y = mx + c$$

x = input
m = weight
c = bias
y = Output



- Blocke diagram for straight line.

(c) Threshold:

→ It is a set value based upon the final Output the network may be calculated.

→ The threshold value is used in the activation function.

→ A comparison is made b/w the calculated net input and threshold to calculate the network Output.

→ The activation function using threshold value, can be defined as.

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

θ = fixed threshold value

(d) Learning rate: ("α")

→ It is used to control the amount of weight adjustment at each step of training

→ The learning rate, ranging from 0 to 1, determines the rate of learning at each time step.

(e). Momentum factor:

→ Convergence is made faster if a momentum factor is added to the weight updation process. This is generally done in the back propagation network.

→ Momentum helps the net in reasonably large weight adjustments until the corrections are in the same general direction for several patterns.

(f). Vigilance Parameter: "ρ"

→ It is generally used in adaptive resonance theory (ART) network.

→ The vigilance parameter is used to control the degree of similarity required for patterns to be assigned to the same cluster unit.

(g). Notations:

x_i : Activation of unit X_i, input signal.

y_i : Activation of unit Y_j, $y_j = f(y_{inj})$

w_{ij} : Weight on connection from unit X_i to unit Y_j.

b_j : Bias acting on unit j.

W : Weight Matrix $W = \{w_{ij}\}$

y_{inj} : Net input to unit Y_j given by

$$Y_{inj} = b_j + \Sigma_i x_i w_{ij}$$

15 $\|x\|$ → Norm of Magnitude Vector X

θ_j → Threshold for activation of neuron Y_j.

S → Training input Vector, $S = (s_1, \ldots s_i, \ldots s_n)$

T → Training Output Vector, $T = (t_1, \ldots t_j, \ldots t_n)$.

X → Input Vector, $X = (X_1, \ldots x_i, \ldots x_n)$

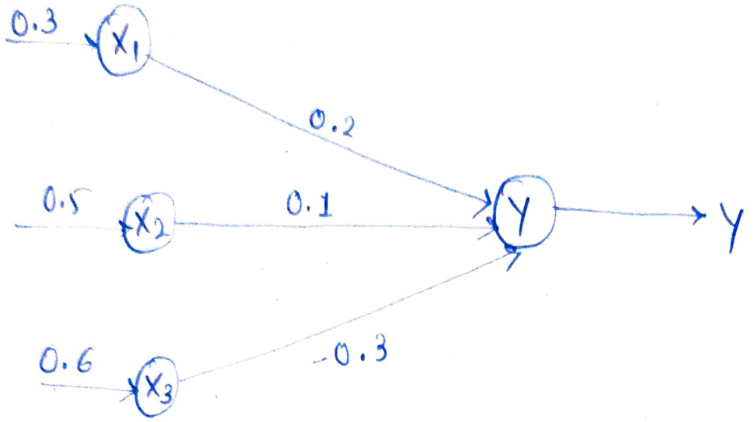
Δw_{ij} → change in Weights

$$\Delta w_{ij} = w_{ij}(\text{new}) - w_{ij}(\text{old})$$

α : Learning rate.

Solved Problem :

Q 1. Calculate the net input to the Output neuron.



Soln :

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

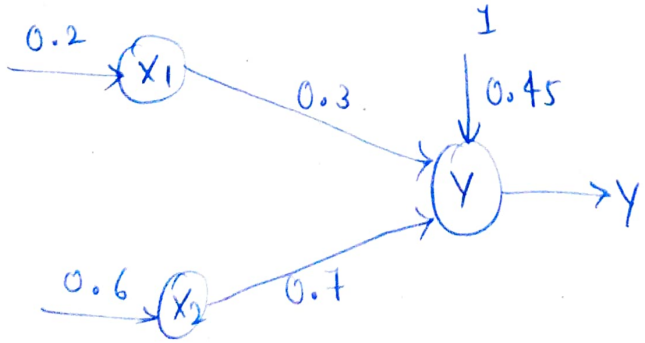
$$[w_1, w_2, w_3] = [0.2, 0.1, -0.3]$$

$$y_{in} = x_1w_1 + x_2w_2 + x_3w_3$$

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

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

Q 2. Calculate the net input for the network shown with bias included in the network.



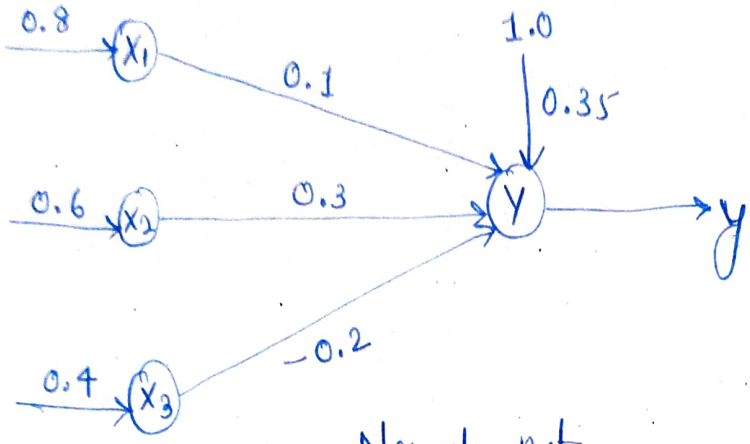
Soln :

$$y_{in} = b + x_1w_1 + x_2w_2$$

$$= 0.45 + 0.2 \times 0.3 + 0.6 \times 0.7$$

$$= 0.45 + 0.06 + 0.42 = 0.93$$

Q3. Obtain the output of the neuron Y for the network shown in fig. using activation function as : (i). binary Sigmoidal and (ii). bipolar Sigmoidal.



. Neural net.

Soln :- $[x_1, x_2, x_3] = [0.8, 0.6, 0.4]$

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

bias. $b = 0.35$ (its input is always 1).

The net input to the Output neuron is

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i \qquad [n = 3, \text{ because only 3 input neurons are given}]$$

$$= b + x_1 w_1 + x_2 w_2 + x_3 w_3$$

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

$$= 0.53$$

(i). For binary Sigmoidal activation function,

$$y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-0.53}} = 0.625$$

(ii) For bipolar Sigmoidal activation function,

$$y = f(y_{in}) = \frac{2}{1+e^{-y_{in}}} - 1 = 0.259$$

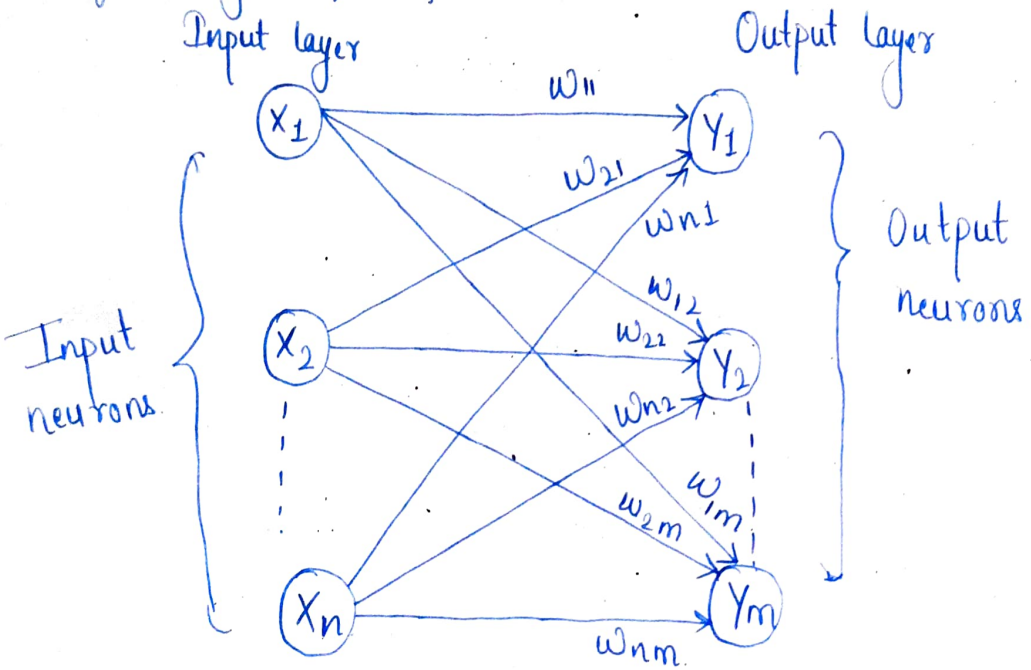
Basic Models of ANN

1. The Model's Synaptic interconnections;
2. The training or learning rules adopted for Updating and adjusting the Connection Weights;
3. Their activation functions.

* Network Architecture - The arrangement of neurons to form layers and the Connection pattern formed within & b/w layers is called network architecture. basically five types of neuron Connection architecture.

1. Connections :

(a). Single - Layer feed-forward network.



• Single - Layer feed-forward network.

→ A layer implies a stage, going stage by stage, i.e input stage and the output stage are linked with each other.

→ These linked interconnections lead to the formation of various network architectures.

→ when a layer of the processing nodes is formed, the inputs can be connected to these nodes with

various, resulting in a series of outputs, one per node. Thus, a single layer feed-forward network is formed.

- Multilayer feed-forward network.

→ It is formed by the interconnection of several layers.

→ The input layer is that which receives the input and this layer has no function except buffering the input signal.

→ The output layer generates the output of the network. Any layer that is formed between the input and output layer is called hidden layer.

→ This hidden layer is internal to the network and has no direct contact with the external environment.

→ More the number of the hidden layers, more is the complexity of the network.

→ In case of a fully connected network, every output from one layer is connected to each and every node in the next layer.

feed forward Network:

* A network is said to be feed-forward network if no neuron in the output layer is an input to a node in the same layer or in the preceding layer. When output can be directed back as inputs to same or preceding layer nodes then it results in the formation of feedback networks.

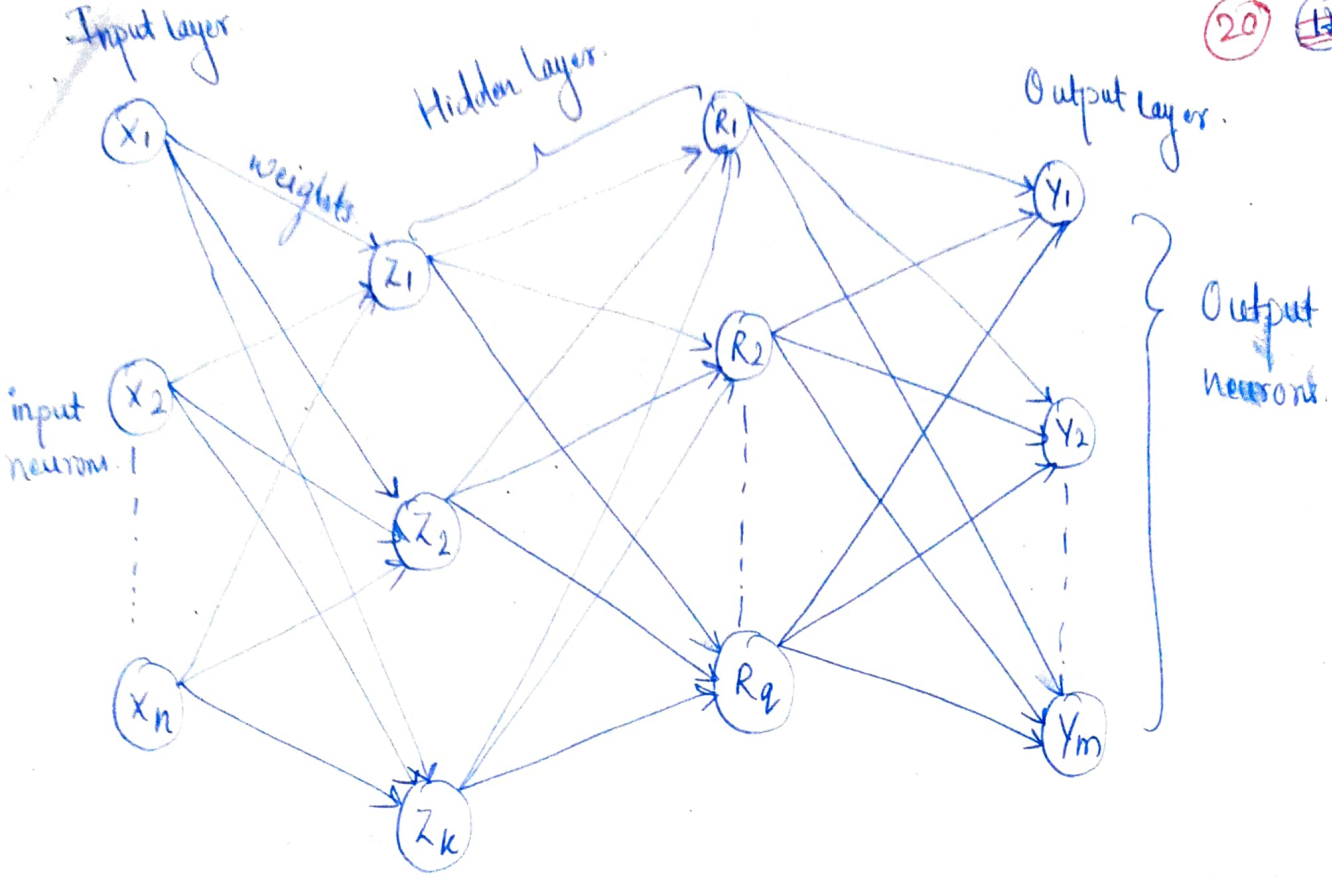
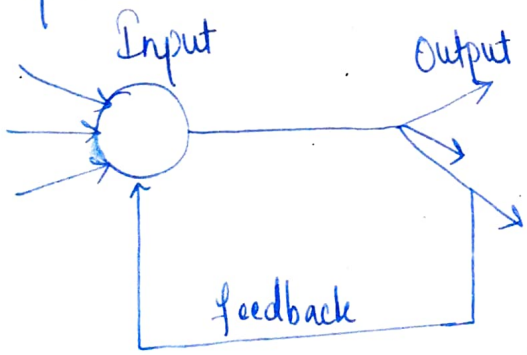


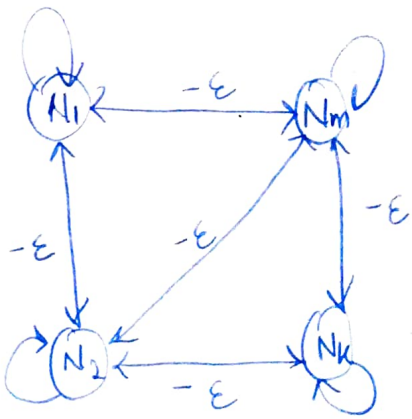
fig- Multilayer feed-forward network.

(c). Recurrent Network:

→ Recurrent networks are feedback networks with closed loop.



• Single node with own feedback.



• Competitive nets.

(i). Single-Layer recurrent network.

→ It is a feedback connection in which a processing element's output can be directed back to the processing element itself or to other processing element

or to both.

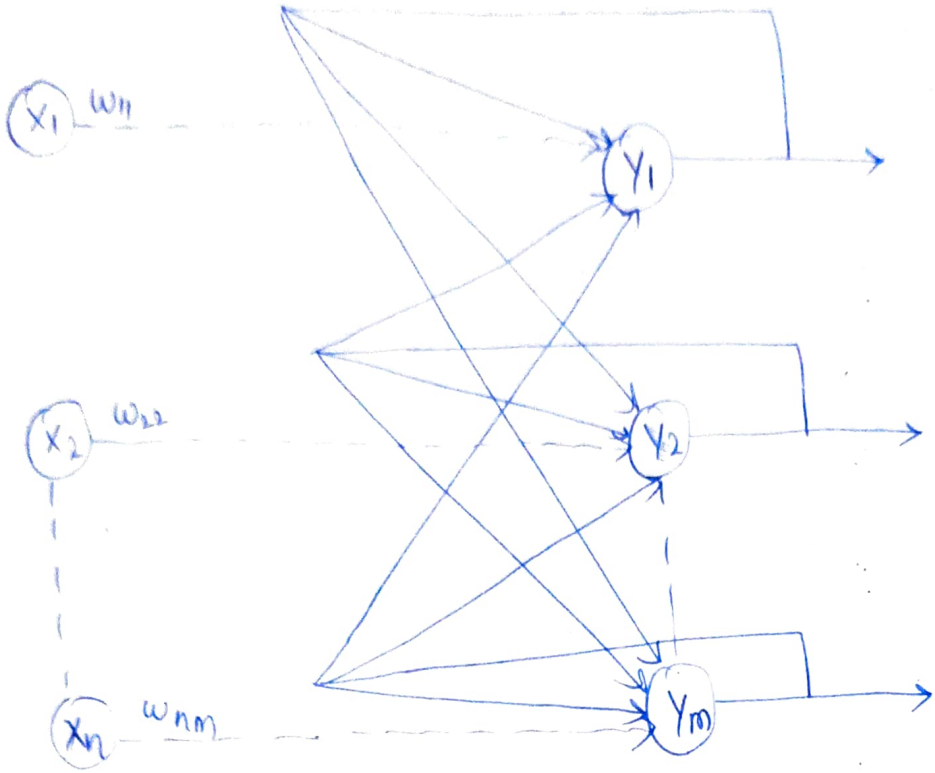
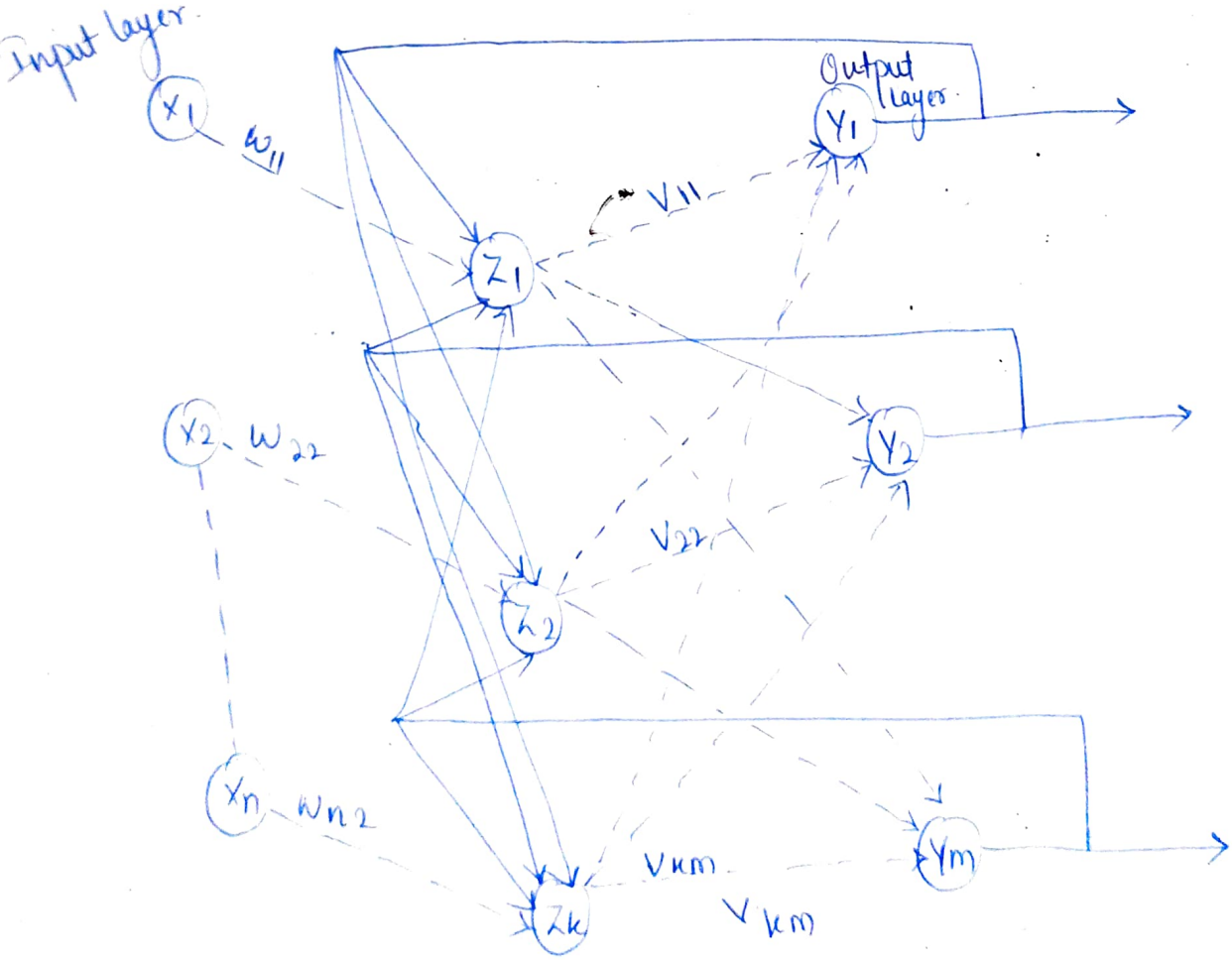


fig- Single-layer recurrent network.



- Multi layer reccurrent network.

* Lateral feedback:

→ If the feedback of the output of the processing elements is directed back as input to the processing elements in the same layer then it is called Lateral feedback.

(ii). Learning:

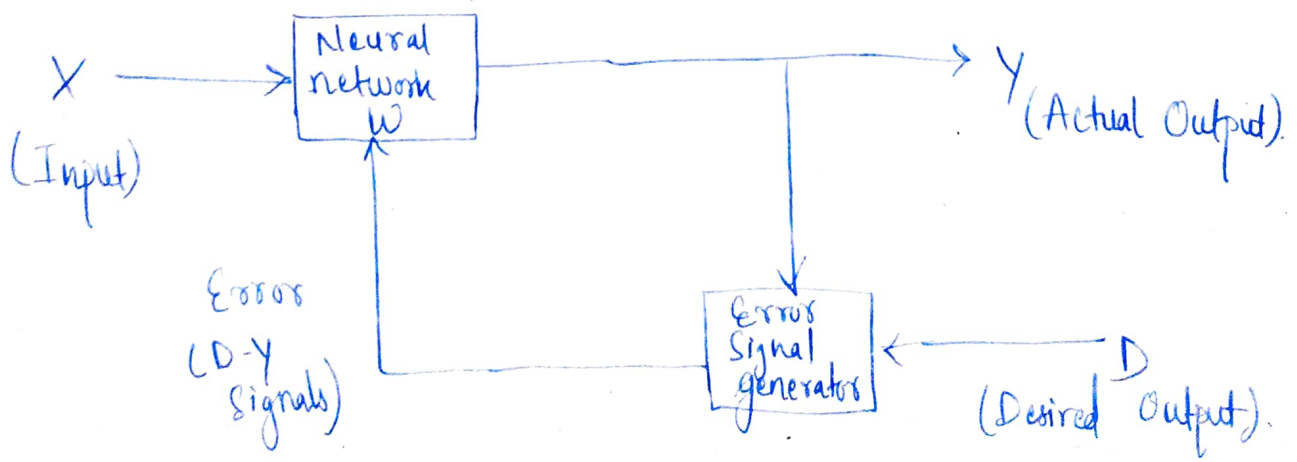
→ Learning or training is a process by means of which a neural network adapts itself to a stimulus by making proper parameter adjustments, resulting in the production of desired response.

→ Broadly, there are two kinds of learning in ANN's.

(i). Parameter Learning - It updates the connecting weights in a neural net.

(ii) Structure Learning - It focuses on the change in network structure (which includes the no. of processing elements as well as their connection types).

(a). Supervised Learning:



→ In ANN's following Supervised Learning, each input Vector requires a corresponding target Vector, which represents the desired Output. The input Vector along with the target Vector is called training pair.

→ The network here is informed precisely about what should be emitted as Output.

→ During Training, the input Vector is presented to the network, which results in an Output Vector.

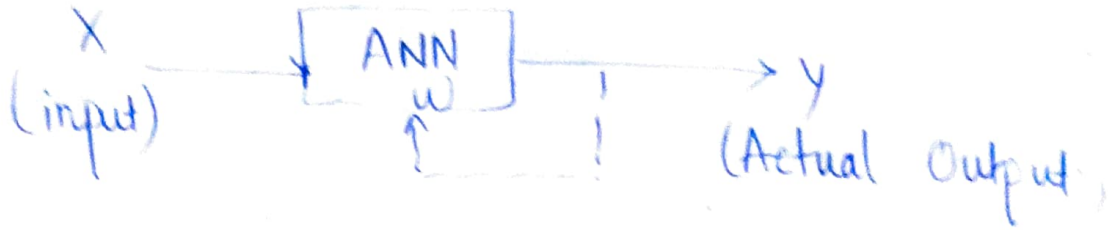
→ This Output Vector is the actual Output Vector.

→ Then actual Output Vector is Compared with the desired Output Vector.

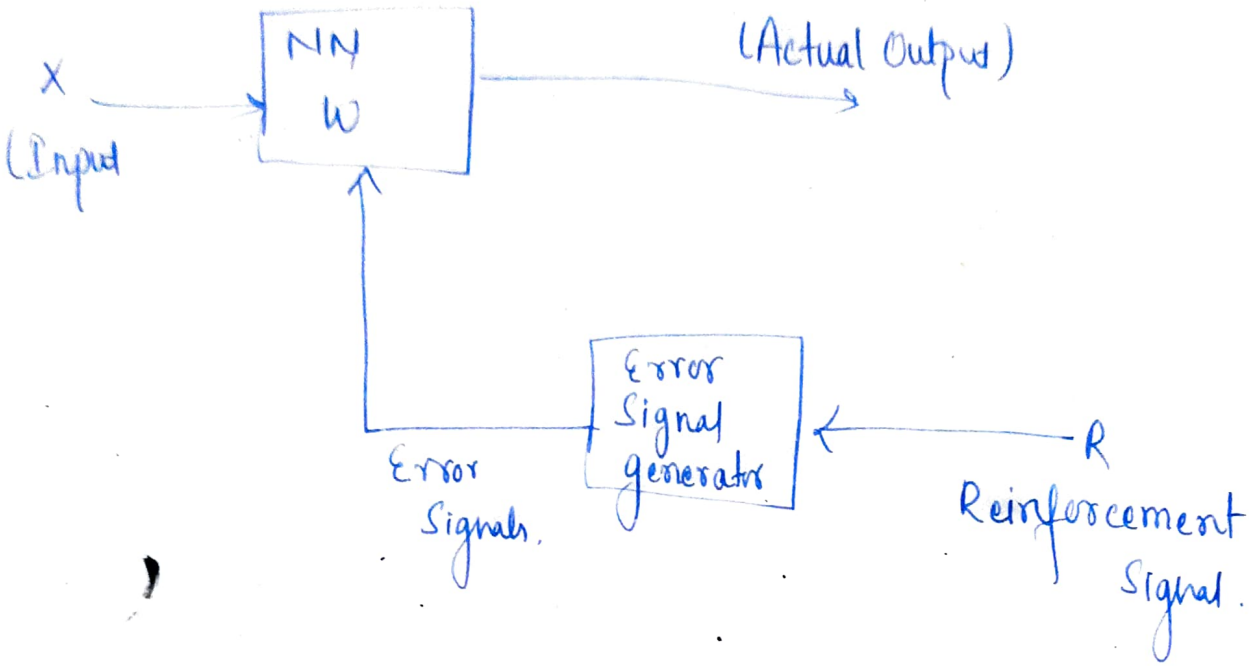
→ If there exists a difference b/w the two Output Vectors then an error Signal is generated by the network.

→ This error Signal is Used for adjustment of weights Until the actual Output matches the desired Output.

(b) Unsupervised Learning:



(c) Reinforcement Learning



Implementing Models of Artificial NN. (M-P Neuron Model)

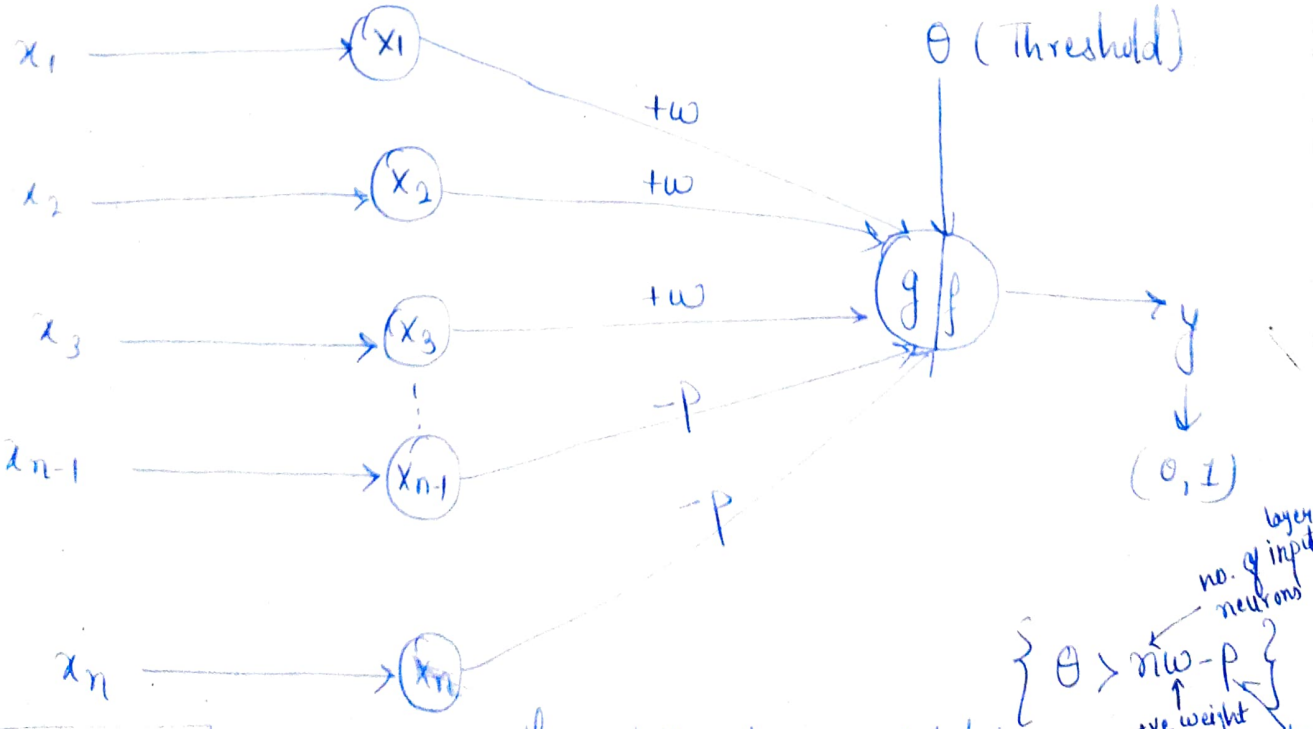
1. McCulloch Pitts Model of Neuron

→ McCulloch Pitts neural model was the earliest ANN Model, has only two types of inputs - Excitatory and Inhibitory.

→ The excitatory inputs have weights of positive magnitude and inhibitory weights have weight of negative magnitude.

→ The inputs of the McCulloch-Pitts neuron could be either 0 or 1.

→ It has a threshold function as an activation function. So, the Output Signal y_{out} is 1 if input y_{sum} or y_{in} is greater than or equal to a given threshold value, else 0.



$x \in [0, 1]$

fig - MP Neuron Model.

→ Simple McCulloch Pitts Neurons can be used to design Logical Operations. For that purpose, the connection weights need to be correctly decided along with the threshold function.

In this Model.

→ The Connected path can have positive weights (Excitatory) or negative weights (Inhibitory).

→ There will be same weights for the excitatory connection entering into particular neuron.

→ The Neuron activates (fires), if the total inputs to the neuron is ≥ to the threshold

→ Main divided into two parts in the form of (g, f). where g will be the one which collects all the inputs, perform an aggregation and based on aggregated value.

→ f will the decision part which decide or makes a desision.

Mathematically,

$$g(x_1, x_2, x_3 \dots x_{n-1}, x_n) = g(x) = \sum_{i=1}^{n} x_i$$

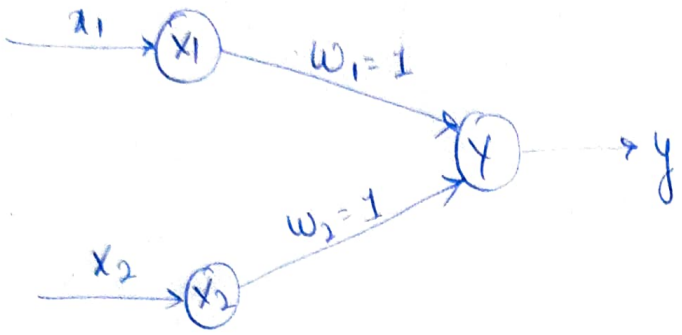
$$y = f(g(x)) = 1 \quad \text{if } g(x) \geq \theta$$
$$= 0 \quad \text{if } g(x) < \theta$$

Implement ANDNOT function Using MP Neuron.

→ Considering the truth table for ANDNOT function

→ here, $w_1 = 1$, $w_2 = 1$

x_1	x_2	Y
0	0	0
0	1	0
1	0	1
1	1	0



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

(1,1), $y_{in} = 1 \times 1 + 1 \times 1 = 2$

(1,0), $y_{in} = 1 \times 1 + 0 \times 1 = 1$

(0,1), $y_{in} = 0 \times 1 + 1 \times 1 = 1$

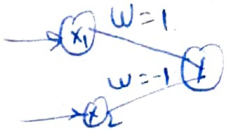
(0,0), $y_{in} = 0 \times 1 + 0 \times 1 = 0$

here, θ will be 1 or 2.

~~$2 \times 1 = 0 = 2$~~

It means it will fire at (1,1) in Case if θ will be 2.
(1,0) & (0,1) in Case if θ will 1.
but we need Only (1,0) Case to be fired. So, we modify the weights and check if it fires at (1,0) input Only.
Let, $w_1 = 1$ & $w_2 = -1$.
then check y_{in} at these two modified weights.

$w_1 = 1$ & $w_2 = -1$.



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

(1,1), $y_{in} = 1 \times 1 + 1 \times (-1) = 0$

(1,0), $y_{in} = 1 \times 1 + 0 \times -1 = 1$

(0,1), $y_{in} = 0 \times 1 + 1 \times -1 = -1$

(0,0), $y_{in} = 0 \times 1 + 0 \times -1 = 0$.

↳ From the Calculated net inputs now it is possible to fire the neuron for input (1,0) only by fixing a threshold of 1,

i.e. $\theta \geq 1$ for Y Unit

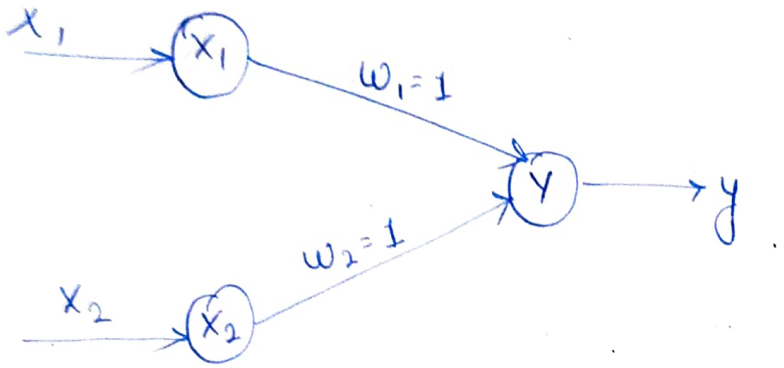
Implementing AND function Using Mcculloch Pitts Neuron

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

→ M-P Neuron has no particular training algorithm.

→ In M-P neuron, Only analysis is being performed.

→ Assume the weights be $w_1 = 1$ & $w_2 = 1$



$(1,1)$, $y_{in} = x_1w_1 + x_2w_2 = 1\times1 + 1\times1 = 2$

$(1,0)$, $y_{in} = x_1w_1 + x_2w_2 = 1\times1 + 0\times1 = 1$

$(0,1)$, $y_{in} = x_1w_1 + x_2w_2 = 0\times1 + 1\times1 = 1$

$(0,0)$, $y_{in} = x_1w_1 + x_2w_2 = 0\times1 + 0\times1 = 0$

check for value of θ

$\theta \geq nw - p$

$\geq 2\times1 - 0$

≥ 2

→ Threshold value should be set as 2. So neuron fires only with (1,1) Output

Thus, the Output of neuron Y can be written

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 2 \\ 0 & \text{if } y_{in} < 2 \end{cases}$$

Implement XOR function Using McCulloch-Pitts Neuron

→ In M-P Neuron, Only analysis is being performed.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

→ XOR function cannot be represented by Simple and Single Logic function; because it is a non-linear reciprocable logical function. It is represented as

$$y = x_1\bar{x}_2 + \bar{x}_1 x_2$$

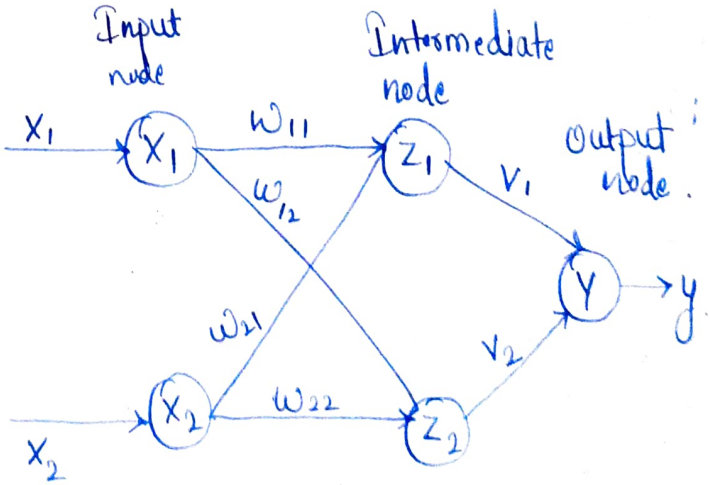
This can be explained as $y = z_1 + z_2$ where, $z_1 = x_1\bar{x}_2$ (funct

$z_2 = \bar{x}_1 x_2$ (funct

$y = z_1$ (OR) z_2 (function 3).

$y = z_1$ (OR) z_2 (fun

→ A Single-layer net is not sufficient to represent the XOR function. We need to add an intermediate layer is necessary.



→ We will take x_1, x_2 neurons as input node, z_1, z_2 as intermediate neurons, and y as Output neurons.

→ Then assign them weight. Now we will solve all three functions individually to calculate y_{in} and threshold Value.

So lets start with first function

- $Z_1 = x_1 \bar{x}_2$
- The truth table for function Z_1

x_1	x_2	Z_1
0	0	0
0	1	0
0	0	1
1	1	0

- Assume the weights are initialized to $w_{11} = w_{e21} = 1$
- Calculate the net inputs.

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

$(0,0)$, $y_{in} = 0 \times 1 + 0 \times 1 = 0$

$(0,1)$, $y_{in} = 0 \times 1 + 1 \times 1 = 1$

$(1,0)$, $y_{in} = 1 \times 1 + 0 \times 1 = 1$

$(1,1)$, $y_{in} = 1 \times 1 + 1 \times 1 = 2$

here,

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

Since, if we take 1 as threshold i.e. θ value.
then we will be have three inputs $(0,1), (1,0), (1,1)$ where the function of neurons fires and this is not okay as we need at only $(1,0)$ point when neuron fires.

⇒ Hence, it is not possible to obtain function Z_1 using these weights.

⇒ We will change weights as $w_{11} = 1$, $w_{21} = -1$

Now Calculate the net inputs,

$(0,0)$, $y_{in} = 0 \times 1 + 0 \times -1 = 0$

$(0,1)$, $y_{in} = 0 \times 1 + 1 \times -1 = -1$

$(1,0)$, $y_{in} = 1 \times 1 + 0 \times -1 = 1$

$(1,1)$, $y_{in} = 1 \times 1 + 1 \times -1 = 0$

- If $\theta = 1$ then neuron will fire

Hence, $w_{11} = 1$; $w_{21} = -1$

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

If we take $\theta = 1$ then only at $(1,0)$ it will fire. So as we need here for function 1.

Second function = $Z_2 = x_1 x_2$

Truth table for x_2

x_1	x_2	Z_2
0	0	0
0	1	1
1	0	0
1	1	0

Assume the Weights

$w_{12} = w_{22} = 1$

Calculate the net input

(0,0), $y_{in} = 0 \times 1 + 0 \times 1 = 0$

(0,1), $y_{in} = 0 \times 1 + 1 \times 1 = 1$

(1,0), $y_{in} = 1 \times 1 + 0 \times 1 = 1$

(1,1), $y_{in} = 1 \times 1 + 1 \times 1 = 2$

⇒ Hence, it is not possible to obtain function Z_2 using these weights.

→ If $\theta = 1$, then it fires here at (0,1).

→ Hence weight will be

$w_{12} = -1$, $w_{22} = 1$.

here Upgradation in Weight Same as function 1.

$w_{12} = -1$, $w_{22} = 1$.

Calculate the net input

(0,0), $Z_{2in} = 0 \times -1 + 0 \times 1 = 0$

(0,1), $Z_{2in} = 0 \times -1 + 1 \times 1 = 1$

(1,0), $Z_{2in} = 1 \times -1 + 0 \times 1 = -1$

(1,1), $Z_{2in} = 1 \times -1 + 1 \times 1 = 0$

(3) Now for third function

$y = z_1$ (OR) z_2.

- Truth table for function y

$y_{in} = z_1 v_1 + z_2 v_2$

- Assume the weights are initialized to $v_1 = v_2 = 1$
- Calculate the net inputs.

$(0,0)$, $y_{in} = 0\times1 + 0\times1 = 0$

$(0,1)$, $y_{in} = 0\times1 + 1\times1 = 1$ ← at $\theta=1$

$(1,0)$, $y_{in} = 1\times1 + 0\times1 = 1$ ← $\theta=1$

$(1,1)$, $y_{in} = 0\times1 + 0\times1 = 0$

x_1	x_2	y	z_1	z_2
0	0	0	0	0
0	1	1 ←	0	1
1	0	1 ←	1	0
1	1	0	0	0

check. $f(y_{in}) = \begin{cases} 1, & y_{in} \geq \theta \\ 0, & y_{in} < \theta \end{cases}$

for $\theta = 0$, we are having four case $(0,0)$ & $(1,1)$ where neurons fires. So θ can't be 0.

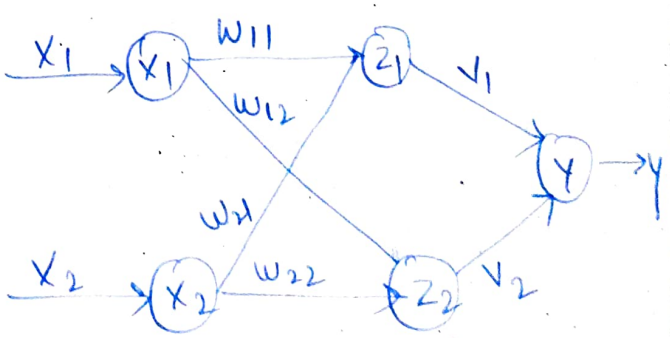
at $\theta = 1$, we have $y_{in} \geq 1$ then neurons fire at $(1,0)$ and $(0,1)$.

→ Hence $\theta = 1$ then neuron fires.

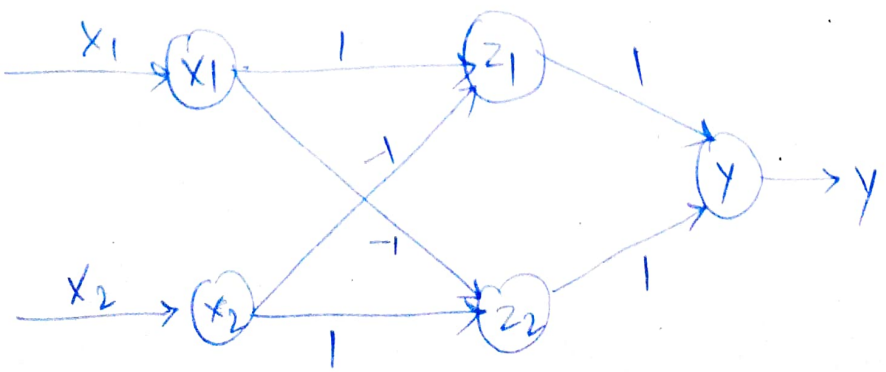
final $v_1 = v_2 = 1$

$w_{11} = 1$ $w_{21} = -1$

$w_{12} = -1$ $w_{22} = 1$



Hence final:



• 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.

→ A decision line is drawn to separate positive and negative responses. The decision line may also be called as the decision-making or decision-support line or linear-seperable line.

→ It is used to classify the patterns based upon their output responses.

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

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

For example, if a bipolar step activation function is used over the calculated y_{in} then the value of the function is 1 for positive net input and -1 for a negative net input. So, the boundary exists in between the regions where $y_{in} > 0$ and $y_{in} < 0$. This region may be called as decision boundary and can be calculated when y_{in} is equal to 0

i.e. $$b + \sum_{i=1}^{n} x_i w_i = 0$$

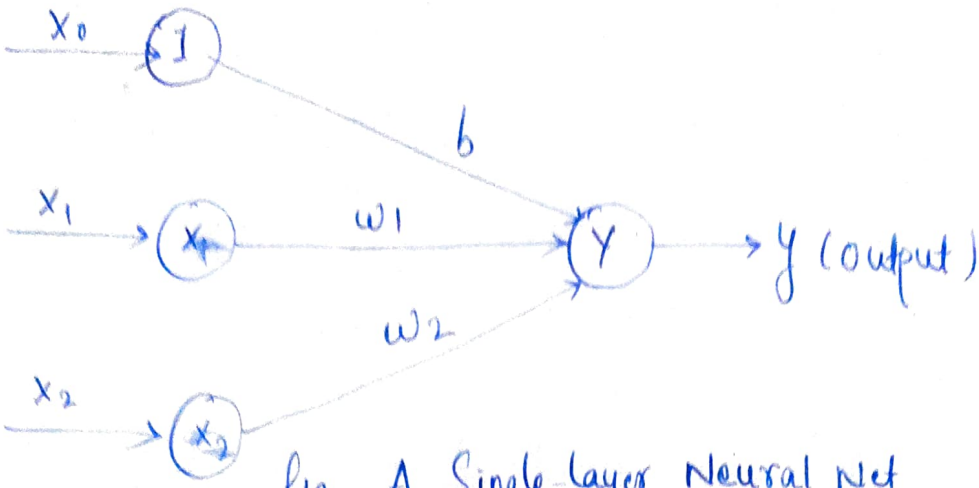


fig - A Single layer Neural Net

Requirement for the response of the net is.

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

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

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

← equation of line.

(here we can take example for AND function.

→ During training process, value of w_1, w_2 and b are determined so the net will produce a positive response for the training data.

→ If on the other hand, threshold value is being used, then the condition for obtaining the positive response from output unit is

Net input received $> \theta$ (threshold)

$$y_{in} > \theta$$

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

The Separating line equation will then be

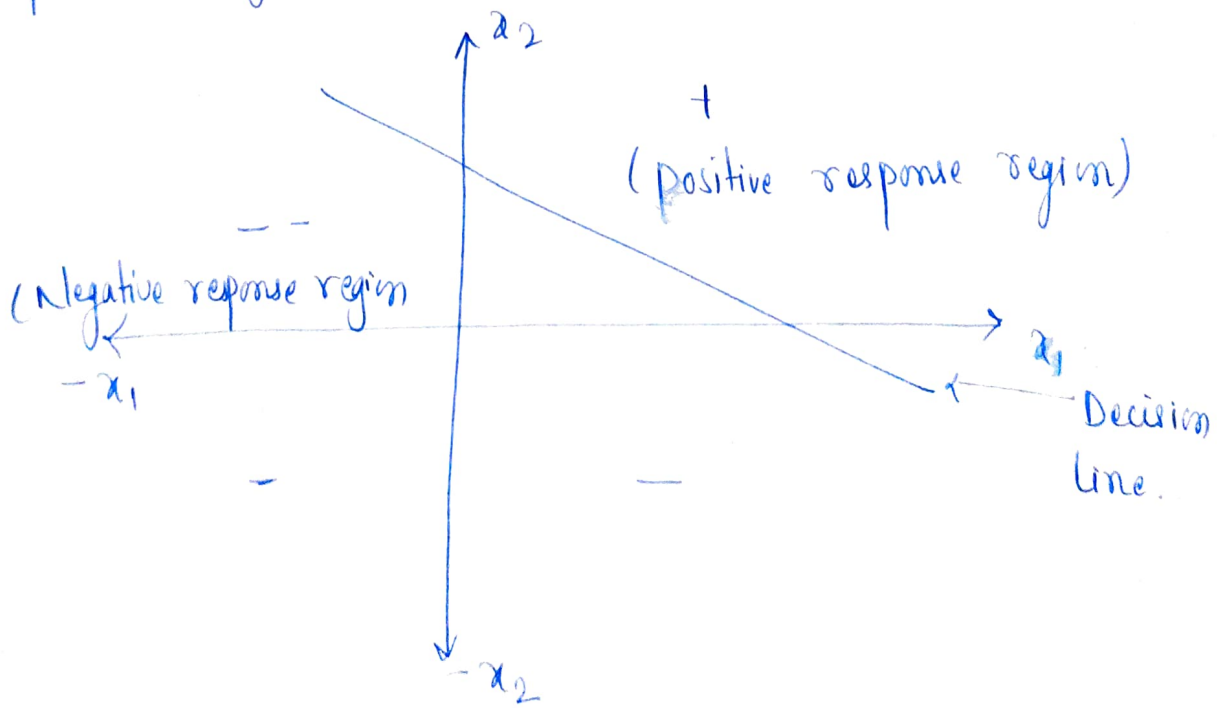
$$x_1 w_1 + x_2 w_2 = \theta$$

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

For Example: AND function we have

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

→ Means network have positive response in the first quadrant and negative response in all other quadrants with either binary or bipolar data, then the decision line is drawn Separating the positive response region from the negative response region.



Hebb Network:

→ Hebb Learning rule is a simple learning rule which was proposed by Donald Hebb in 1949.

→ According to Hebb Rule, the weight vector is found to increase proportionately to the product of the input and the learning signal.

→ Here, the learning signal is equal to the neuron's output. In Hebb learning, if two interconnected neurons are 'on' simultaneously then the weights associated with these neurons can be increased by the modification made in their synaptic gap(strength).

→ The weight update in Hebb rule is given by →

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

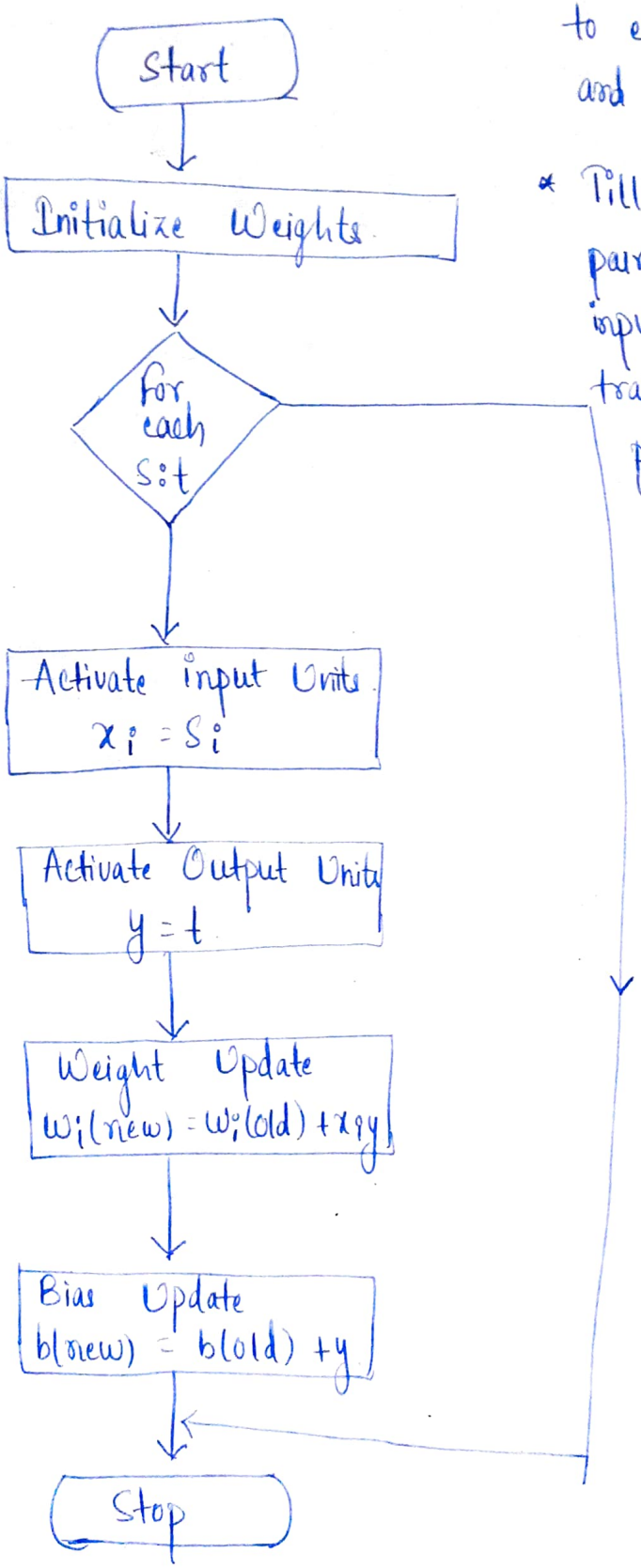
→ The Hebb Rule is more suited for bipolar data than binary data. If binary data is used, the above weight updation formula cannot distinguished two conditions namely:

1. A training pair in which an input unit is "on" and target value is "off."
2. A training pair in which both the input unit and target value are "off".

- Flow chart of Training Algorithm:

→ The training algorithm is used for the calculations and adjustment of weights.

* where, s:t → refers to each training input and target output pair.
* Till there exists a pair of training input & target output, training process takes place else it stops.



- Flow Chart of Hebb training Algorithm.

Training Algorithm:

Step 0 : First Initialize the Weights. Basically in this n/w they may be set to Zero.
i.e. $w_i = 0$ for $i = 1$ to n, where n = total no. of input neurons.

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

Step 2 : Input Units activations are set.
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:

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

$$b(new) = b(old) + y$$

In Vector form, it is written as

$$w(new) = w(old) + xy$$

Here, change in Weight can be $\boxed{\Delta w = xy}$

As a result,

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

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

Q. Design a Hebb net to implement Logical AND function (Use bipolar inputs and targets).

Solution: The training data for AND function is

Inputs			Target
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 and bias are set to zero. i.e.

$$w_1 = w_2 = b = 0$$

- First input $[x_1 x_2 b] = [1\ 1\ 1]$ and target = 1 [i.e. $y = 1$]

- Setting the initial weights as old weights and applying the Hebb rule. We get

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

$\Delta w_i = x_i y$ (x_i ← input, y ← target in this case.)

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

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

$\Delta b = y = 1$

So, new weight will be

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

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

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

Second Input: $[x_1 \; x_2 \; b] = [1 \quad -1 \quad 1]$ and $y = -1$.

↳ The weight change here is

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

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

$\Delta b = y = -1$

- The new weights here are.

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

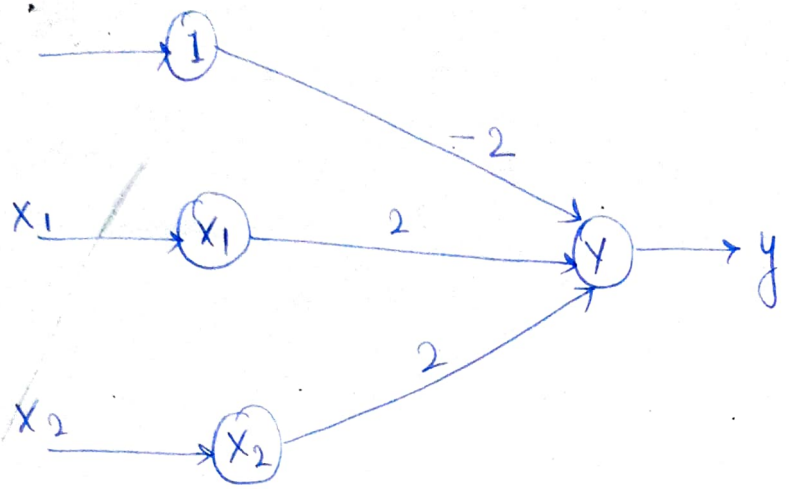
$w_2(new) = w_1(old) + \Delta w_2 = 1 + 1 = 2$

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

Similarly, by presenting the third and fourth input patterns, the new weights can be calculated.

Inputs				Weight changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1	w_2	b
							(0	0	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

Hebb Net for AND function

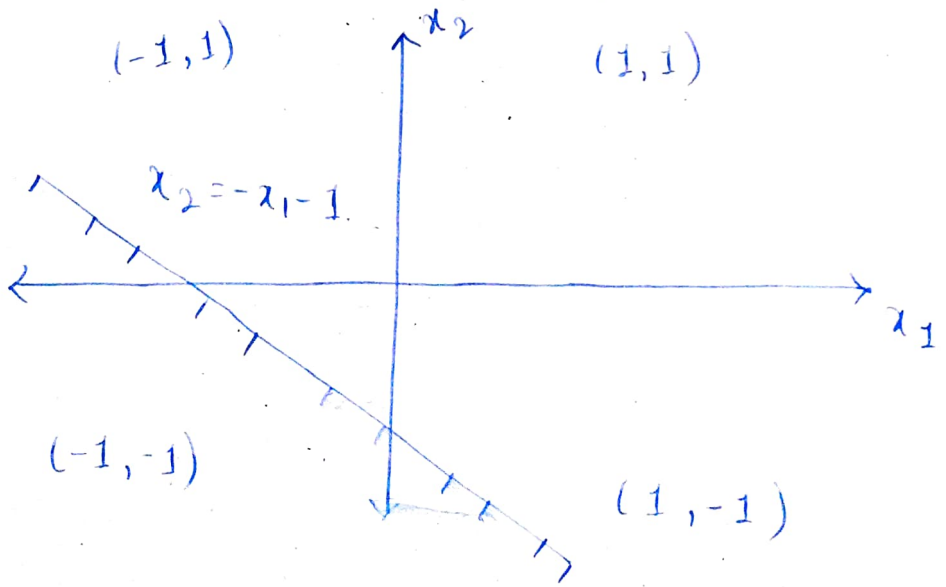


Now, the Separating line equation is given by

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

for first input [1 1 1], the Separating line is given by

$$x_2 = \frac{-1}{1} x_1 - \frac{1}{1} \Rightarrow \boxed{x_2 = -x_1 - 1}$$



(A) First Input

Q: Design a Hebb net to implement OR function. (4)

Sol^{n}: The training pair for OR function is.

Inputs			Target
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 and Bias are Set to Zero.

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

→ By presenting all the input patterns, the weights are Calculated. Here, table shows the weights Calculated for all the inputs.

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

Using the final weights, the boundary line equation can be obtained. The Seperating line equation is -

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

$$= \frac{-2}{2} x_1 - \frac{2}{2} = -x_1 - 1$$

So, the decision region for this net will be.

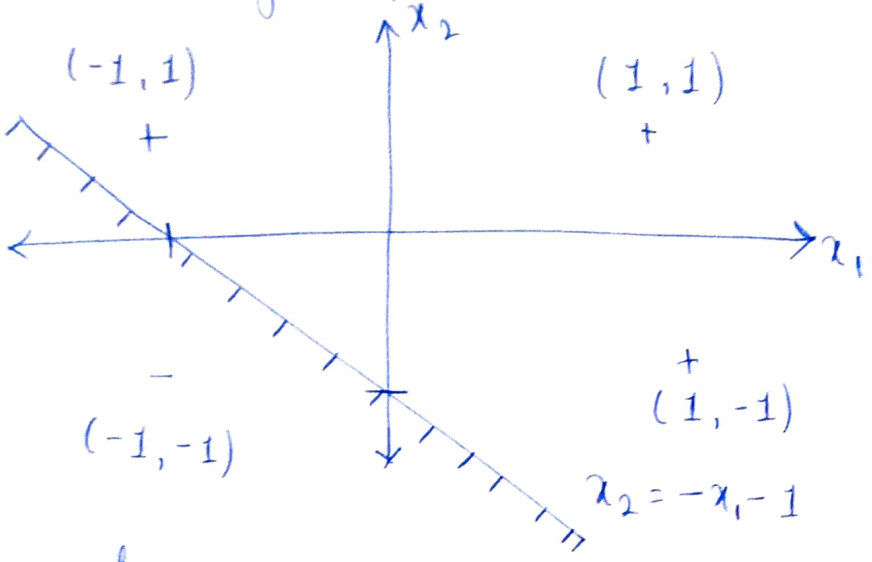


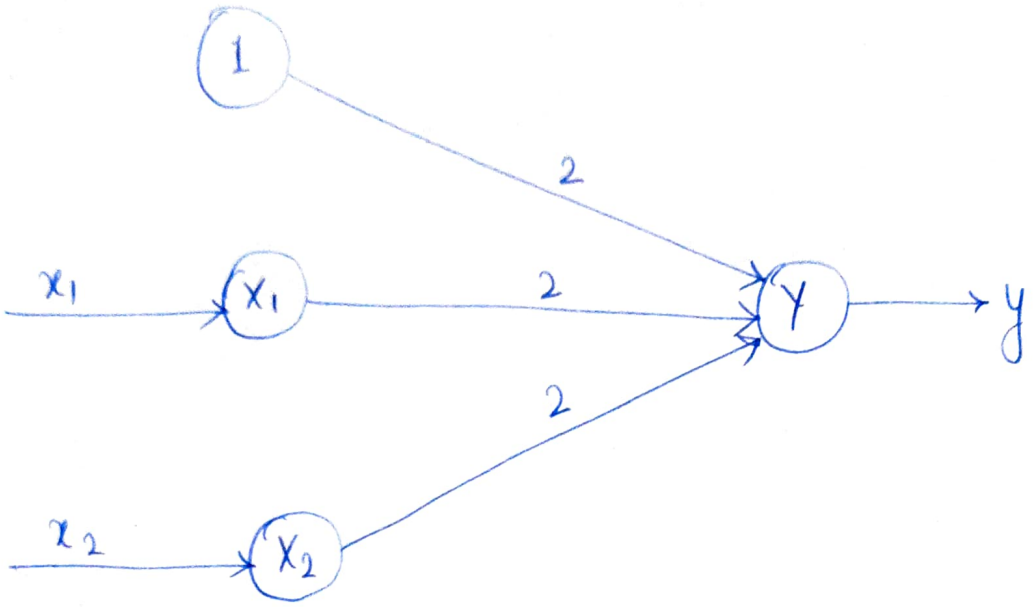
fig - Decision boundary for OR function.

Here, Straight line $x_2 = -x_1 - 1$ Seperates the pattern Space into two regions. The input pattern [(1, 1), (1, -1), (-1, 1)] for which the Output response is "1" lie on One Side of the boundary, and the input pattern (-1, -1) for which the Output response is "-1" lies on the other Side of the boundary.

Thus, the final weights are

$w_1 = 2$; $w_2 = 2$; $b = 2$.

So, The network can be represented as shown in below figure:



Questions to Solve:

Q. Use the Hebb rule method to implement XOR function (take bipolar inputs and targets).

Sol^n:- hint:

Input			Target
x_1	x_2	b	y
1	1	1	-1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

Inputs				Weight changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1	w_2	b
							(0	0	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	0	0	0

* XOR function is a case of a pattern classification problem, which is not linearly seperable.

• Perceptron Networks

→ Perceptron networks come under single-layer feed-forward networks and are also called simple perceptrons.

→ A simple perceptron network was discovered by Block in 1962.

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

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

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

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

→ The output of the perceptron network is given by

$$\boxed{y = f(y_{in})}$$

i.e.

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

(12)

→ The perceptron learning rule is used in the weight updation between the associator unit and the response unit.

For each training input, the net will calculate the response and it will determine whether or not an error has occurred.

→ The Error calculation is based on the comparison of the values of targets with those of the calculated outputs.

→ The weight on the connections from the units that send the non-zero signal will get adjusted suitably.

→ The weights will be adjusted on the basis of the learning rule if an error has occurred for a particular training pattern. i.e.

$$w_i(new) = w_i(old) + \alpha t x_i$$

$$b(new) = b(old) + \alpha t$$

Here, if no error occurs, there is no weight updation and hence training process may be stopped.

t = target value is $+1$ or -1

α = learning rate.

Fig - Original perceptron network

Sensory Unit

Sensory grid representing the pattern

Fixed Weight Value
+ 1, 0, -1
at random.

Association Unit

Response Unit

Output
0 or 1

Desired output
+1 -1

Perceptron Learning Rule:

→ Consider a finite "n" number of input training vectors, with their associated target (desired) values $x(n)$ and $t(n)$, where "n" ranges from 1 to N.

The target (t) is either +1 or -1.

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

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

The Weight Updation in Case of perceptron Learning is.

If $y \neq t$ then

$w(new) = w(old) + \alpha t x$ → α = Learning rate)

else, we have

$w(new) = w(old)$.

→ The Weights Can be initialized at any Values in this method. The perceptron rule Convergence theorem States that

"If there is a Weight Vector W, Such that

$f(x(n)w) = t(n)$, for all n,

then for any starting vector w_1, the perceptron learning rule will converge to a weight vector that gives the correct response for all training patterns, and this learning takes place within a finite number of steps provided that the solution exists".

Architecture :

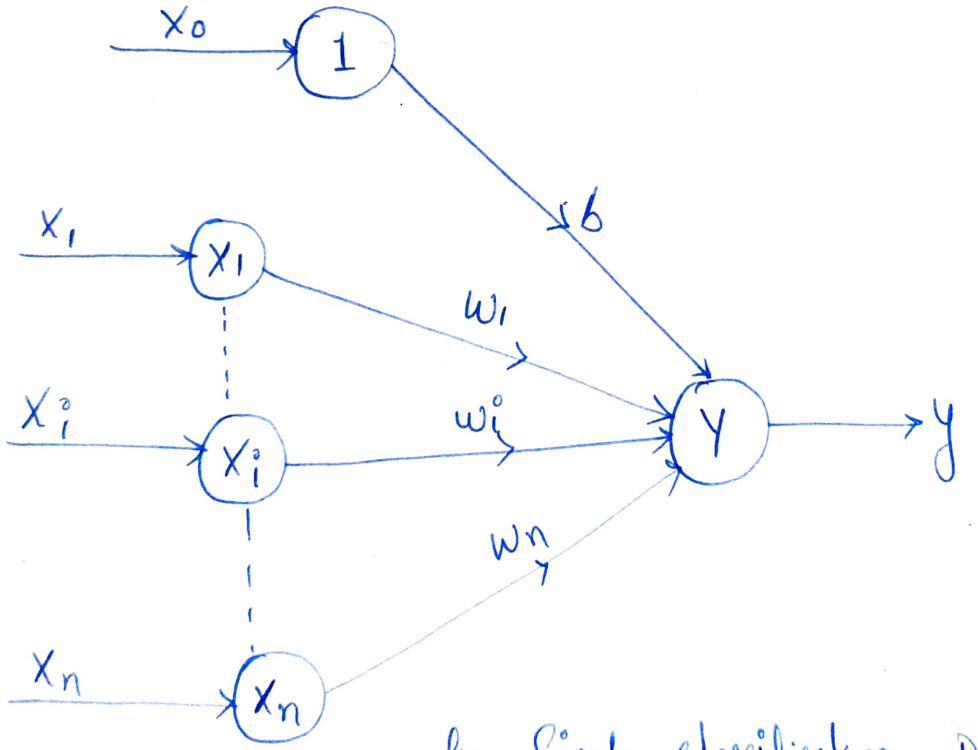


fig - Single classification perceptron network.

n = input neurons.
yy = Output Neuron.
b = bias.

{ Main Goal = to classify the input pattern as a member or not a member to a particular class. }

Flow Chart for Training Process :

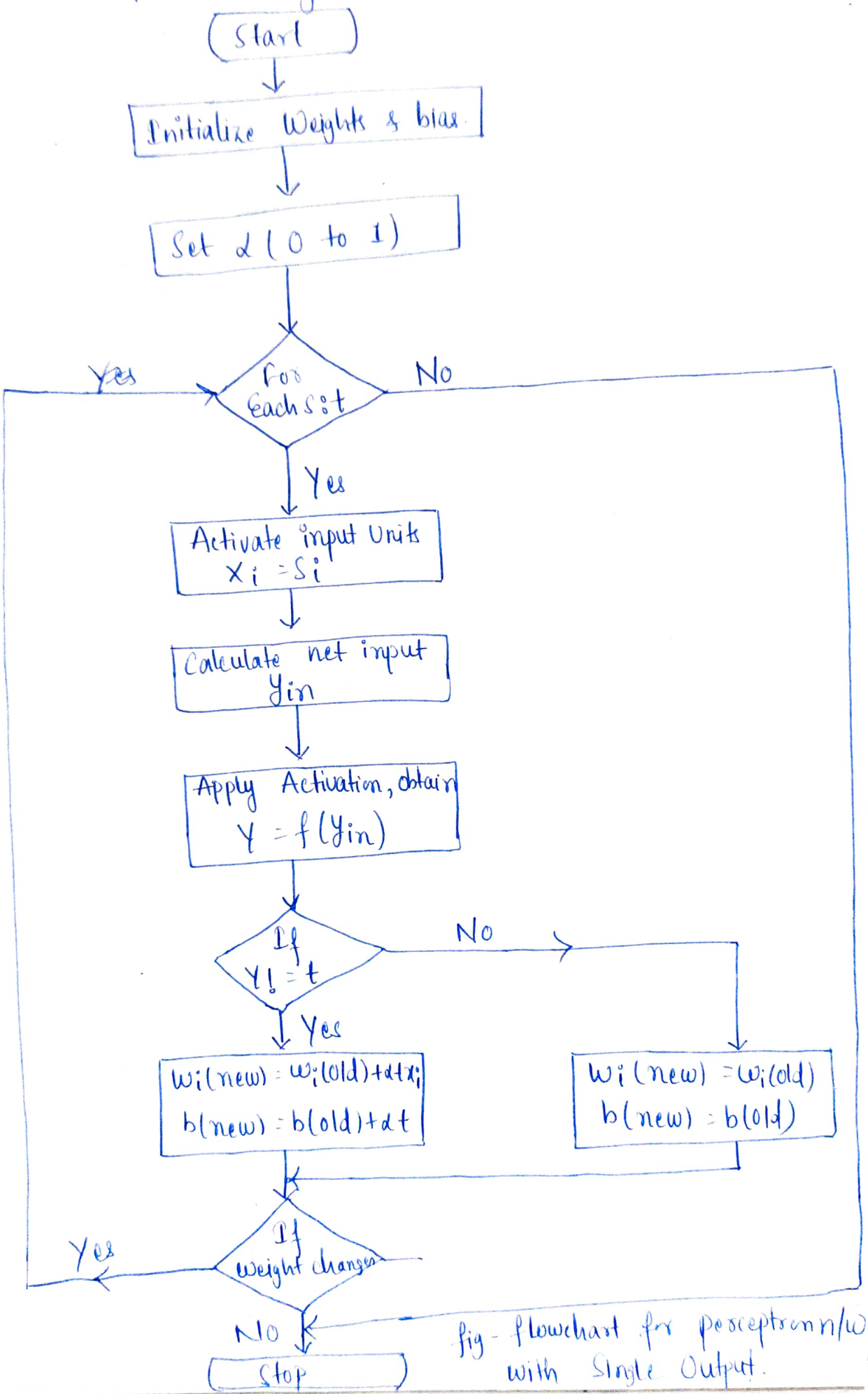


fig - flowchart for perceptronn n/w with single Output.

Perceptron Training Algorithm for Single Output classes:

Step 0 : Initialize the weights and the bias. Also initialize the learning rate α ($0 < \alpha \leq 1$). For simplicity α is set to 1.

Step 1 : Perform step 2-6 Until the final stopping condition is false.

Step 2 : Perform step 3-5 for each training pair indicated by $s:t$

Step 3 : The input layer containing input units is applied with identity activation functions:

$$x_i = s_i$$

Step 4 : Calculate the Output of the network.

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

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

Step 5 : Weight and bias adjustment :

If $y \neq t$ then

$$w_i(new) = w_i(old) + \alpha t x_i$$

$$b(new) = b(old) + \alpha t$$

else,

$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

Step 6 : Train the network Until there is no weight change.

This is the stopping Condition for the network.

If Condition is not met, then start again from step 2.

Perceptron Training Algorithm for Multiple Output classes:

Step 0: Initialize the weights, biases and learning rate suitably.

Step 1: check for stopping condition; if false, perform steps 2 to 6.

Step 2: Perform step 3-5 for each bipolar or binary training vector pair $s:t$.

Step 3: Set activation of each input unit $i = 1$ to n:

$$x_i = s_i$$

Step 4: Calculate output response of each output unit $j = 1$ to m.

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

Then,

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

Step 5 : Make adjustment in weights and bias for $j = 1$ to m and $i = 1$ to n.

If $t_j \neq y_j$ then

$$w_{ij}(new) = w_{ij}(old) + \alpha t_j x_i$$
$$b_j(new) = b_j(old) + \alpha t_j$$

else,

$$w_{ij}(new) = w_{ij}(old)$$
$$b_j(new) = b_j(old)$$

Step 6 : Test for the stopping condition. If no change in weights then stop else start Again from Step 2.

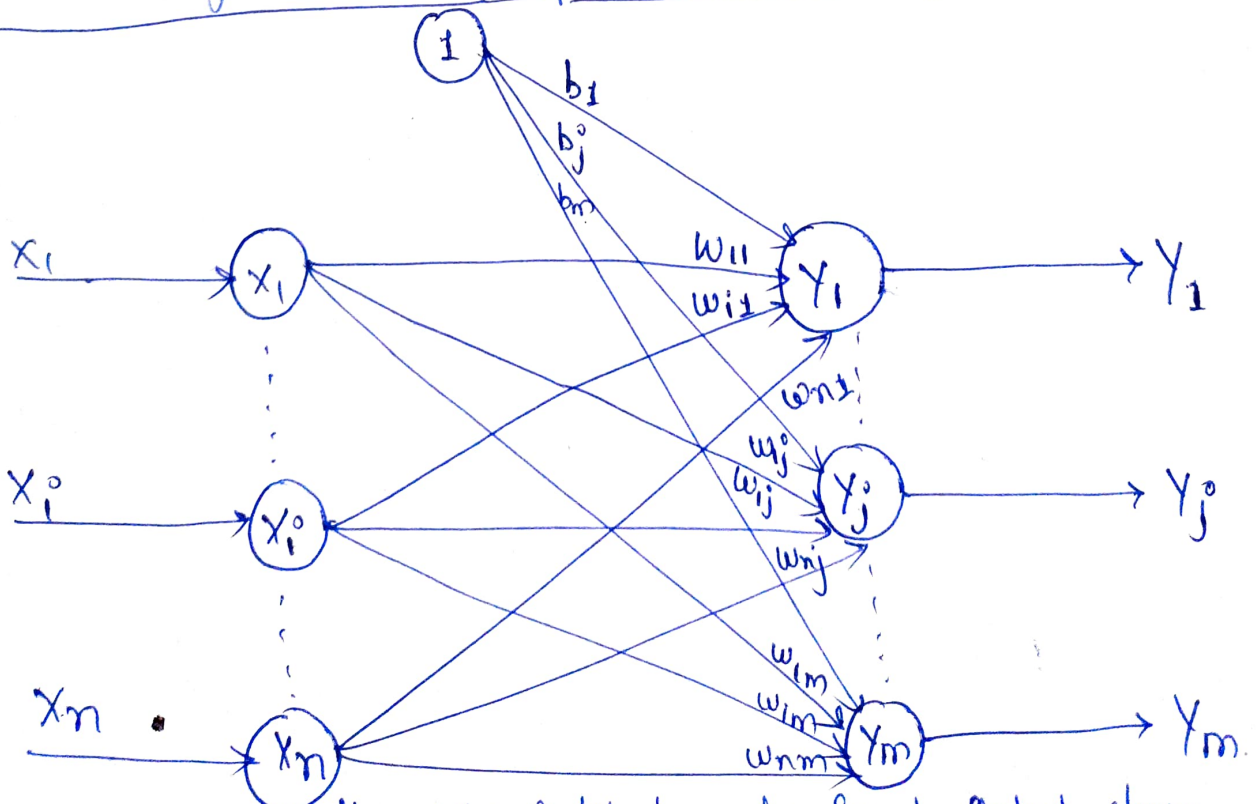


fig - N/w Architecture for Several Output classes.

Perceptron Network Testing Algorithm.

→ Once the training process is complete, it is best to test the network performance.

→ The testing Algorithm is as follows:

Step 0: The initial weights to be used here are taken from the training Algorithm. means the final weights during training.

Step 1: For each input vector X to be classified perform step 2-3.

Step 2: Set activations of the input Unit.

Step 3: Obtain the response of Output Unit.

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

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

→ Thus, the testing Algorithm tests the performance of network.

Sol^{n} :- $y_{in} = b + x_1 w_1 + x_2 w_2$

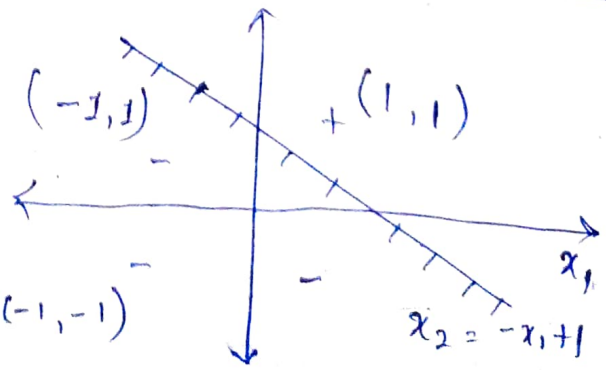
$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & y = 0 \\ -1 & y_{in} < 0 \end{cases}$$

$\Delta w_1 = \alpha t x_1$;

$\Delta w_2 = \alpha t x_2$;

$\Delta b = \alpha t$

Input x_1	Input x_2	Target (t)	Net input (y_{in})	Calculated Output (y)	Weight changes Δw_1	Δw_2	Δb	Weights w_1	w_2	b
								(0	0	0)
								for $\alpha = 1$.		
EPOCH-1										
1	1	1	0	0	1	1	1	1	1	1
1	-1	-1	1	1	-1	1	-1	0	2	0
-1	1	-1	2	1	+1	-1	-1	1	1	-1
-1	-1	-1	-3	-1	0	0	0	1	1	-1
EPOCH-2										
1	1	1	1	1	0	0	0	1	1	-1
1	-1	-1	-1	-1	0	0	0	1	1	-1
-1	1	-1	-1	-1	0	0	0	1	1	-1
-1	-1	-1	3	-1	0	0	0	1	1	-1



Perceptron learning Rule:

→ In case of perceptron learning rule, the learning signal is the difference between the calculated output and actual (target) output of a neuron.

→ The Output "y" is calculated.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i \qquad y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & -\theta \le y_{in} \le \theta \\ -1 & y_{in} < -\theta \end{cases}$$

If $y \neq t$ then.

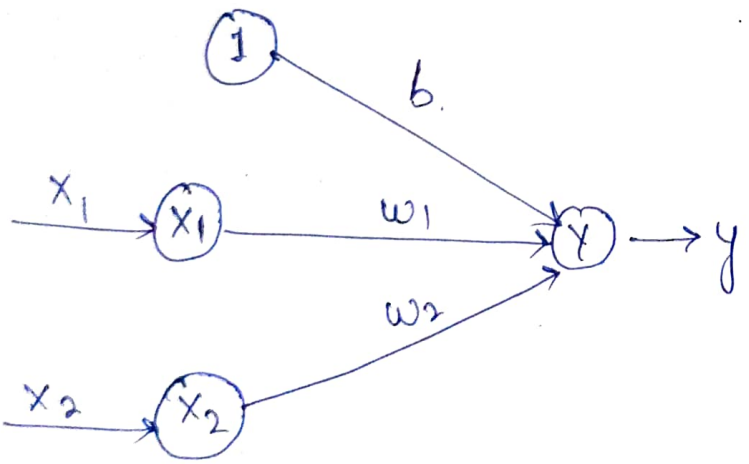
$$w(new) = w(old) + \alpha t x.$$

$\alpha \rightarrow$ Learning rate.

else $w(new) = w(old)$

→ weights are updated using this data.

AND function Using Perceptron Rule.



x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

• Adaptive Linear Neuron (Adaline) [Supervised]

→ It is a Single-layer neural network which uses Linear activation function (bipolar activation function) for its input and target output.

→ Learning rate lies in between (0.1 to 1 and Bias ≠ 0 means weight and Bias can not be zero).

→ It uses delta rule means need to change the value of weight so that we can minimize the difference between output value and target value.

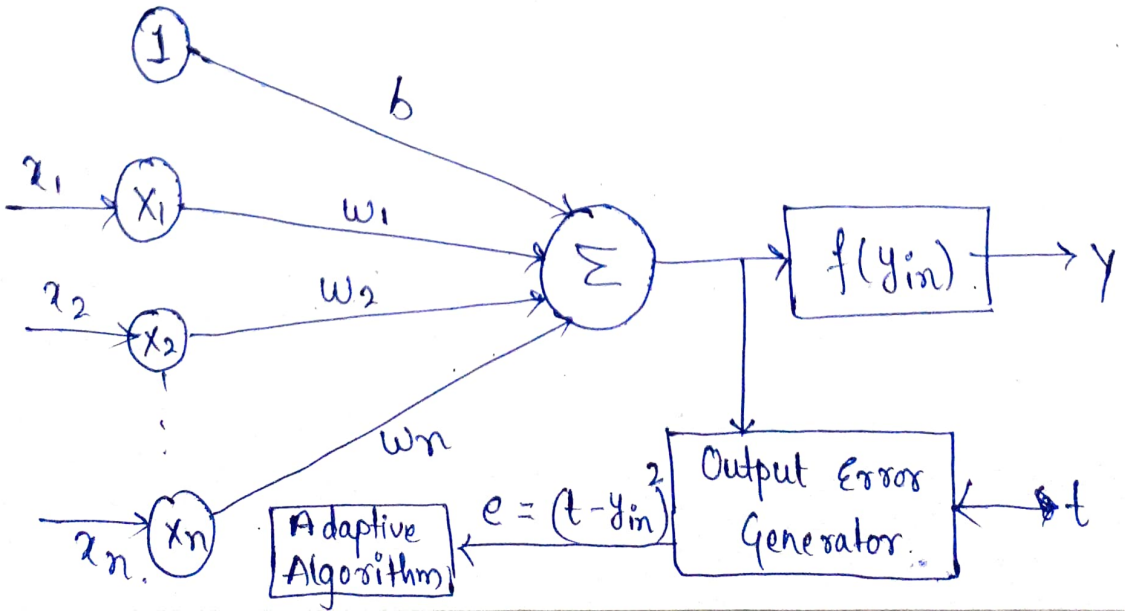
$$\Delta w_i = \alpha (t - y_{in}) x_i$$

- α → learning rate – $(0.1 \text{ to } 1,\ \alpha \neq 0)$.
- x_i → input vector (1 or −1 only)

here,

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

$$Y = f(y_{in}) = \sum x_i w_i + b$$



Q. Implement OR function Using ADALINE.

Inputs			Target t
x_1	x_2	b	t
1	1	1	1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

Sol^n:- Let Assume

initial Weights and bias (b) = 0.1

for input $[x_1 = 1, x_2 = 1]$

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

$= 1 \times 0.1 + 1 \times 0.1 + 0.1$

$y_{in} = 0.3$

$\Delta w_1 = \alpha (t - y_{in}) x_i$

$= 0.1 (1 - 0.3) \times 1$

$\boxed{\Delta w_1 = 0.07}$

~~for~~ Second input ~~$[x_1 = 1, x_2 = 1]$~~

$\Delta w_2 = \alpha (t - y_{in}) x_i \quad (x_2 = 1)$

$= 0.1 (1 - 0.3) \times 1$

$\boxed{\Delta w_2 = 0.07}$

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

$w_1 (new) = 0.1 + 0.07$

$\boxed{w_1 (new) = 0.17}$

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

$= 0.1 + 0.07$

$\boxed{w_2 (new) = 0.17}$

Bias $b (new) = b (old) + \Delta b$

$= 0.1 + 0.07$

$\boxed{b (new) = 0.17}$

$[\Delta b = \alpha (t - y_{in}) \times 1]$

$= 0.1 (1 - 0.3) = 0.07$

Calculation of Error :

$E = (t - y_{in}) = (1 - 0.3)^2 = (0.7)^2 = 0.49$

So for first input

$[x_1 = 1, x_2 = 1]$, Weights $= [w_1 \; w_2 \; b] = [0.17, 0.17, 0.17]$

Error $E = 0.49$

Now for Second input $(x_1 = 1, x_2 = -1)$, $t = 1$

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

$= 0.17 \times 1 + 0.17 \times (-1) + 0.17$

$y_{in} = 0.17$

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

$= 0.17 + \alpha (t - y_{in}) x_1$

$= 0.17 + 0.1 (1 - 0.17) \times 1$

$w_1(new) = 0.253$

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

$= 0.17 + 0.1(1 - 0.17) \times (-1)$

$w_2(new) = 0.087$

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

$= 0.17 + 0.1(1 - 0.17)$

$b(new) = 0.253$

Error $= (t - y_{in}) = (1 - 0.17)^2 = (0.83)^2 = 0.69 = E$

So,

$[x_1 = 1, x_2 = -1]$ = Weights $= [0.253, 0.087, 0.253]$

Error $= 0.69$

For Third input : $[x_1 = -1 \ , \ x_2 = 1]$

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

$$= 0.253 \times (-1) + 0.087 \times 1 + 0.253$$

$$\boxed{y_{in} = 0.087}$$

$$w_1(new) = w_1(old) + \alpha (t - y_{in}) \times x_1$$

$$= 0.253 + 0.1 (1 - 0.087) \times (-1)$$

$$\boxed{w_1(new) = 0.1617}$$

$$w_2(new) = 0.087 + 0.1 (1 - 0.087) \times 1$$

$$\boxed{w_2(new) = 0.1783}$$

$$b(new) = b(old) + \alpha (t - y_{in}) \times 1$$

$$\boxed{b(new) = 0.3443}$$

$$Error = (1 - 0.087)^2$$

$$\boxed{Error = 0.83}$$

So, New Weight,

$$[x_1 = -1 \ , x_2 = 1] \quad t = +1 \quad = (0.1617 \ , 0.1783 \ , 0.3443)$$

$$\boxed{Error = 0.83}$$

For fourth input : $[x_1 = -1 \ , \ x_2 = -1 \ , \ t = -1]$

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

$$= 0.1617 \times -1 + 0.1783 \times (-1) + 0.3443$$

$$\boxed{y_{in} = 0.0043}$$

$w_i(new) = w_i(old) + \alpha(t - y_{in}) \times x_1$

$= 0.1617 + 0.1(-1 - 0.0043)x_1$

$w_1(new) = 0.1617 \quad 0.2621$

$w_2(new) = w_2(old) + \alpha(t - y_{in})x_2$

$= 0.1783 + 0.1(-1 - 0.0043) \times (-1)$

$w_2(new) = 0.2787$

$b(new) = 0.3443 + 0.1(-1 - 0.0043) \times 1$

$b(new) = 0.2439$

$Error = (t - y_{in})^2 = (0.1 - 0.043)^2 = 1.01$

$Error = 1.01$

Total Error = 0.49 + 0.69 + 0.83 + 1.01

after 1st iteration = 3.02

Total Error 2nd Iteration = 1.94

Total Error 3rd Iteration = 1.55

Total Error 4th Iteration = 1.38 → Hence, error reduces after each loop.

Associative Memory :

→ It is also known as Content Addressable Memory (CAM) or associative storage or associative Array.

→ It is a special type of memory that is optimized for performing searches through data, as opposed to providing a simple direct access to the data based on the address.

→ Associative Memory makes a parallel search within a stored data file. The concept behind this search is to output any one or all stored items which match the given search argument and to retrieve the stored data either completely or partially.

* Two types of associative Memory :-

(a). Auto-associative Memory.

(b). Hetro-associative Memory.

→ Both these nets are single-layer nets in which the weights are determined in a manner that the net stores a set of pattern associations.

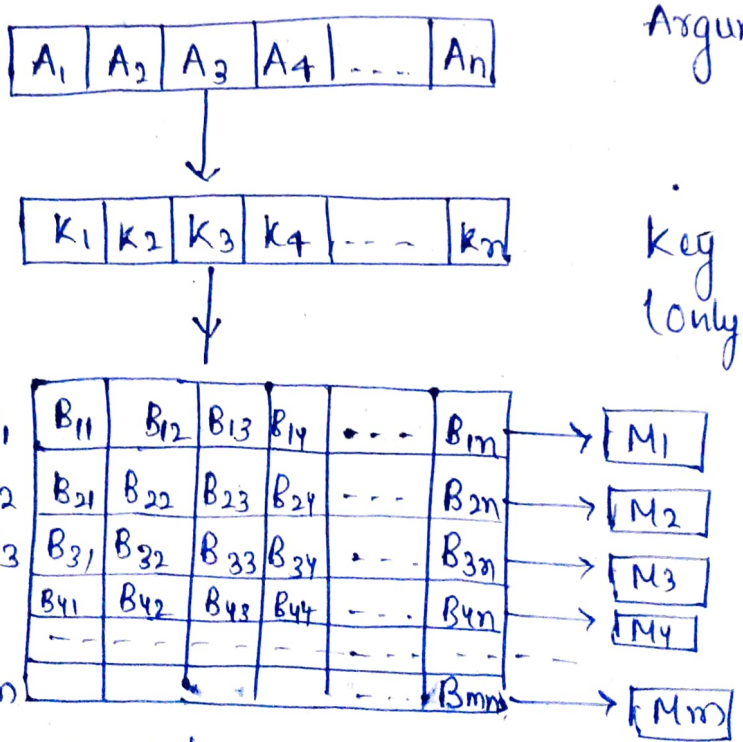
→ Each of these association is an input-output vector pair. say s:t.

→ If each of the Output Vector is same as the input Vectors with which it is associated, then the net is a said to be autoassociative memory net.

→ If the Output Vectors are different from the input Vectors then the net is said to be hetroassociative memory net.

→ This type of Memory is associative means accessed Simultaneously and in parallel on the basis of data content rather than by their specific address or location.

Block Diagram of Associative Memory:



Associative Memory.

Example :

A = 101 111100

K = 111 00000

W_1 = 100 111100 (No Match)

W_2 = 101 000001 (Match).

Advantage :- Used for high speed

Disadvantage - Costly.