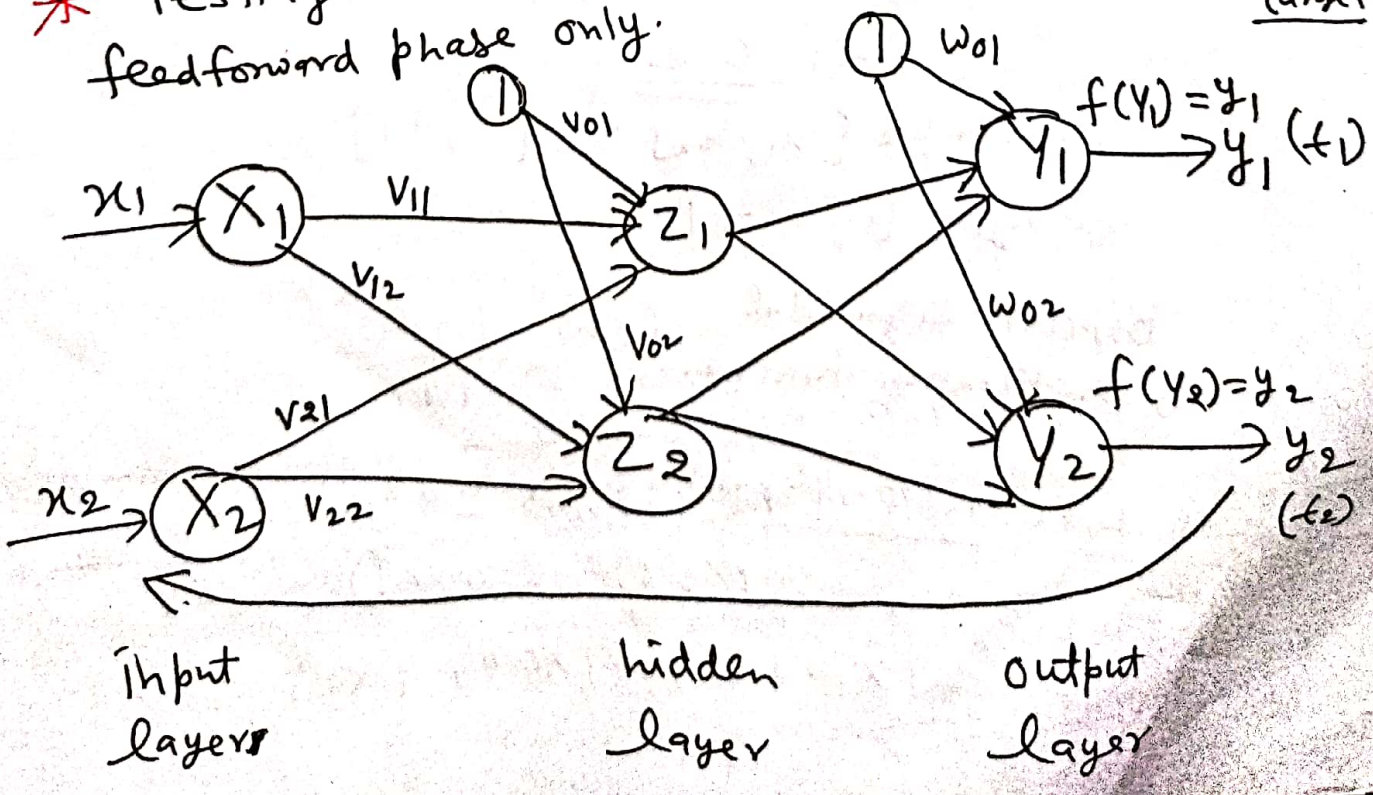
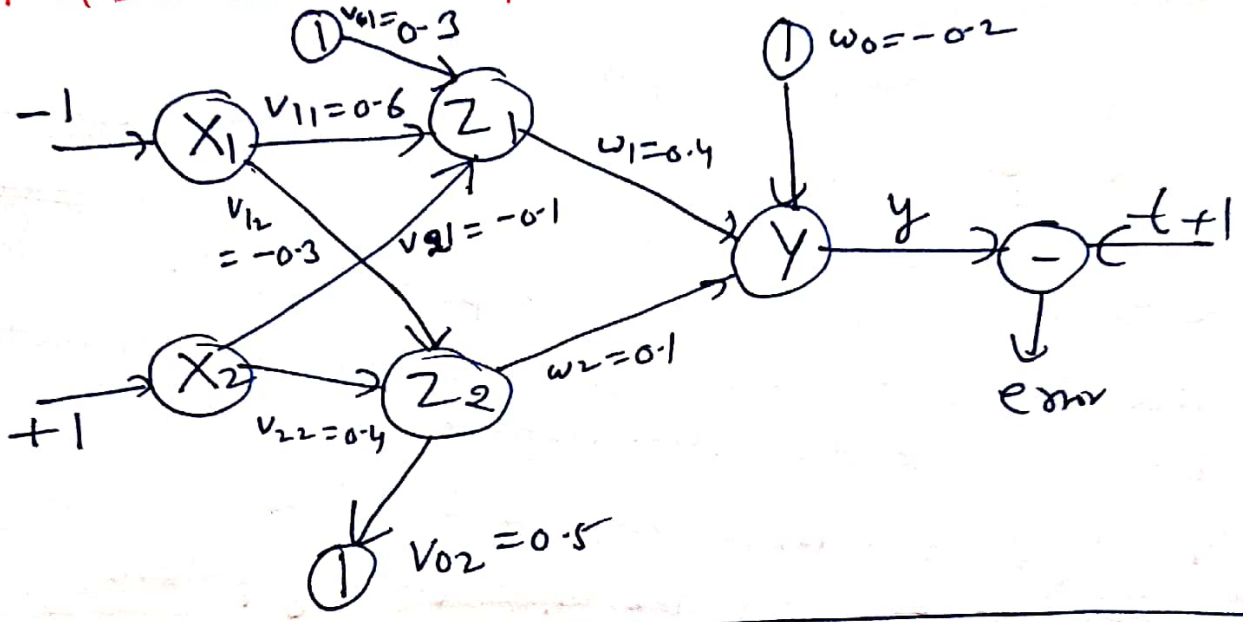
Back-Propagation Network

* The network associated with back-propagation learning algorithm are also called Back-Propagation Network (BPN).
* A BPN is a multi-layer feed forward neural network consisting of an input layer, a hidden layer and an output layer.
* The inputs are sent to the BPN and the output obtained from the network could be ~~either~~ either binary (0,1) or bipolar(-1, +1).
* The activation function could be any function, but it should be differentiable.
* Training of BPN has 3 stages:
  1. Feedforward of the input training pattern.
  2. Calculation and back propagation of error.
  3. Updation of weights.
* Testing of BPN requires the computation of feedforward phase only.



Example:

Find the new weights using back propagation network for the network shown in figure. The network is presented with the input pattern [-1, 1] and the target output is +1. Using a learning rate of $\alpha = 0.25$ and bipolar sigmoidal activation function.



Initial weights = $[v_{01}, v_{11}, v_{21}] = \{+0.3, 0.6, -0.1\}$

$= [v_{02}, v_{12}, v_{22}] = [0.5, -0.3, 0.4]$

$[w_0, w_1, w_2] = \{-0.2, 0.4, 0.1\}$

$\alpha = 0.25$

Input sample $[x_1, x_2] = [-1, +1]$

target $[t] = +1$

Bipolar sigmoidal activation function $f(x) = \dfrac{1 - e^{-x}}{1 + e^{-x}}$

Calculate net input at Z_1 and Z_2

$z_{in1} = 0.3 + (-1 \times 0.6) + (-0.1 \times 1) = -0.4$

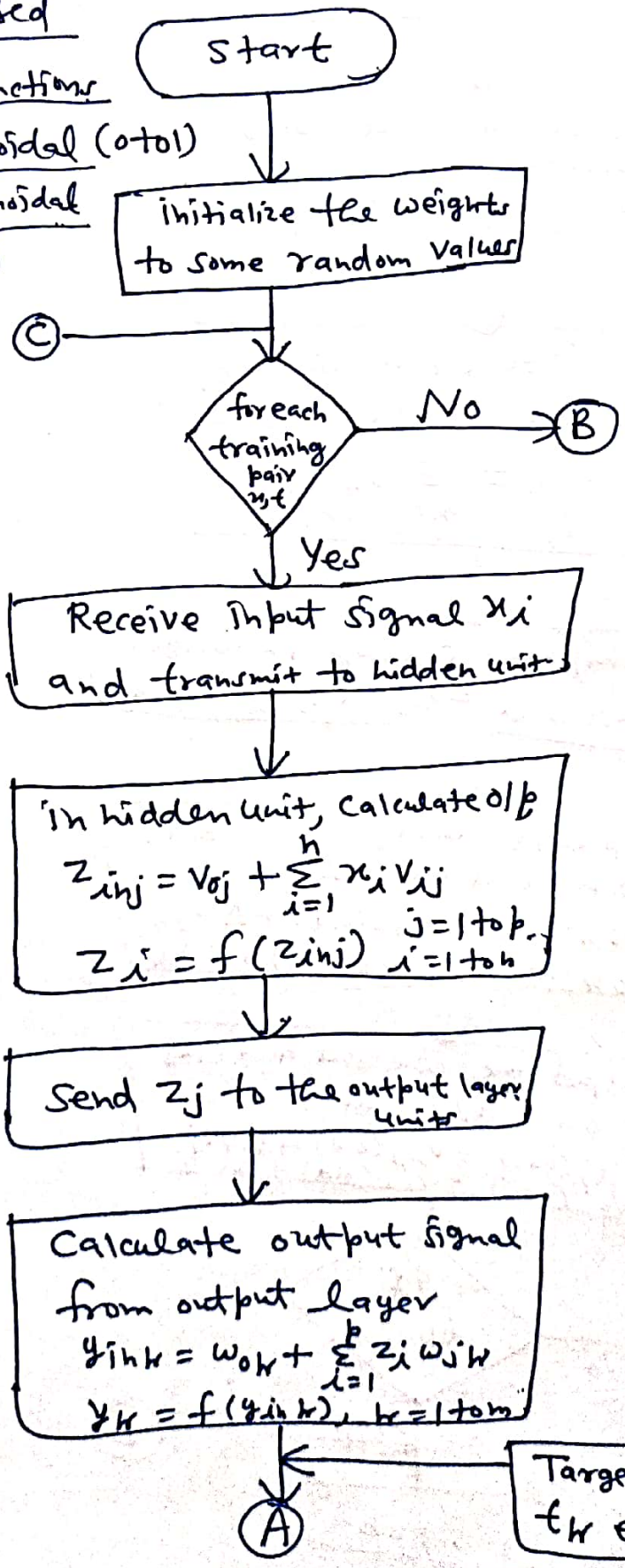
$z_{in2} = -1 \times (-0.3) + +1 \times 0.4 + 1 \times 0.5 = 1.2$

Back Propagation algorithm

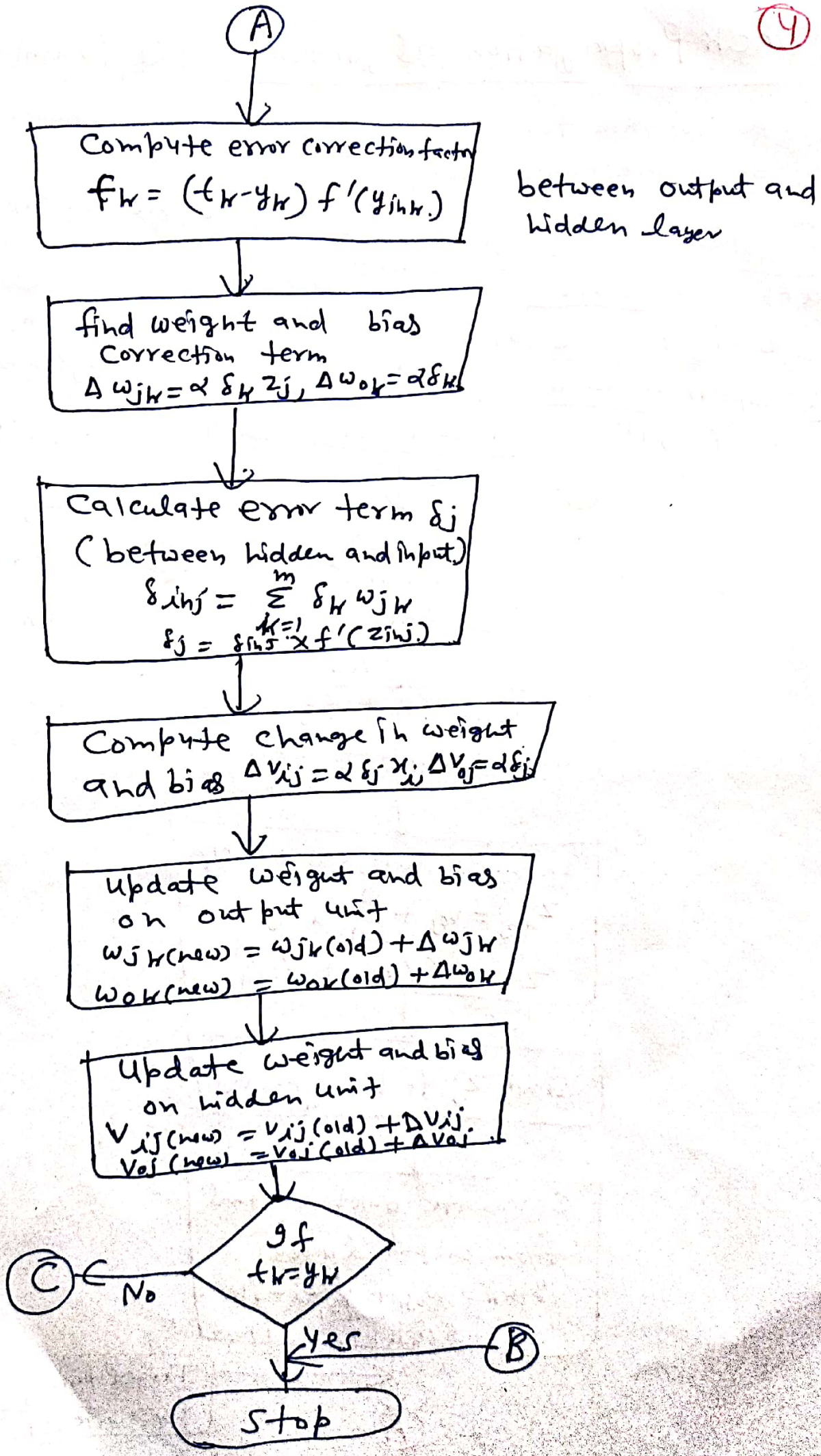
The training algorithm for a back propagation network (BPN) can be represented using flow chart.

Commonly used activation functions

1. Binary sigmoidal (0 to 1)
2. bipolar sigmoidal (−1, +1)



where x = input training vector
t = target output vector
α = learning rate
x_i = input unit i
v_{oj} = bias on jth hidden unit
w_{ok} = bias on kth output unit
z_j = hidden unit j.
y_k = output unit k
δ_k = error correction weight adjustment for w_{jk}
δ_j = error correction weight adjustment for v_{ij}



Factors Affecting the training of back propagation network (BPN)

The training of back propagation network (BPN) is based on the choice of various parameters (factors):

(1) Initial weights

* The choice of the initial weight determine how fast the network converge.
* The initial weights can not be very high.
* The weight initialization can be done by a method called Nyugen-widrow initialization. This type of initialization leads to faster convergence of network

(2) Learning Rate (α):

* The learning rate affect the convergence of the BPN
* A larger value of α may speed up the convergence but might result in overshooting, while a smaller value of α has vice-versa effect
* A larger learning rate leads to rapid learning but a lower learning rate leads to slower learning.

(3) No. of training data:

* The training data should be sufficient & proper.
* There is a rule of thumb, which states that the training data should cover the entire expected input space and while training, training-vector pairs should be selected randomly from the set.

(4) No. of hidden layers nodes:

* In case of all multilayer feed forward networks, the size of a hidden layer is very important.

* for a network of a reasonally size, the size of hidden nodes has to be only a relatively small fraction of the input layer.
* The no. of hidden layers required for an application needs to be determined separately. They are usually determined experimently.
* If a network does not converge to a solution it may need more hidden layers. However, if a network converges, there may be very few hidden nodes.

(5) Momentum factor(η)

* Momentum factor helps in faster convergence.
* It allows a larger learning rate without oscillations. It is denoted by $\eta \in [0,1]$ and the value of 0.9 is often used.
* This factor improves both training speed and accuracy.

(6) Generalization :

* A network is said to be generalized when it sensibly interpolates (interpolates) with inputs that are new to the network.
* When there are many training parameters for the given amount of training data, the network learns well, but does not generalize well. It is called overtraining.
* The solution of this problem is to monitor the test data error and terminate the training when error increases.

Perceptron Neural Network :

- The perceptron network consists of 3 units (i.e. input unit, hidden unit and output unit)
- The input units are connected to hidden units with fixed weights (-1, 0, +1) assign randomly.
- The output of the perceptron network is

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

where

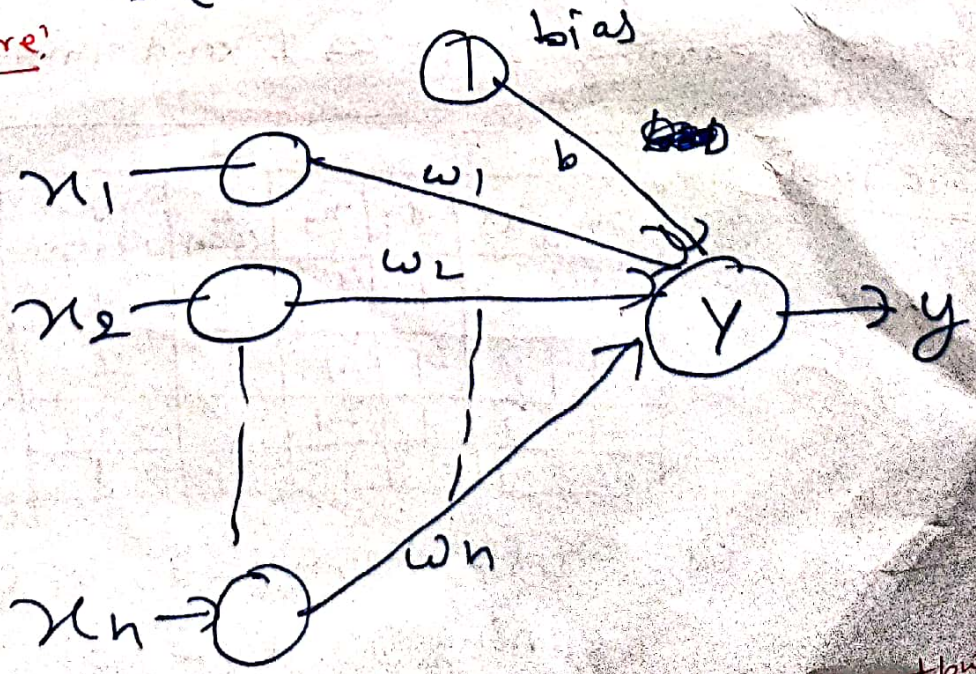
$$f(y_{in}) = \begin{cases} 1, & \text{if } y_{in} > \theta \\ 0, & \text{if } -\theta \le y_{in} \le \theta \\ -1, & \text{if } y_{in} < -\theta \end{cases}$$

- The perceptron learning rule is used in the weight updation b/w the hidden unit and output unit.
- The error is calculated as the comparison between the target value and output.
- weights and bias are adjusted as per given formula.

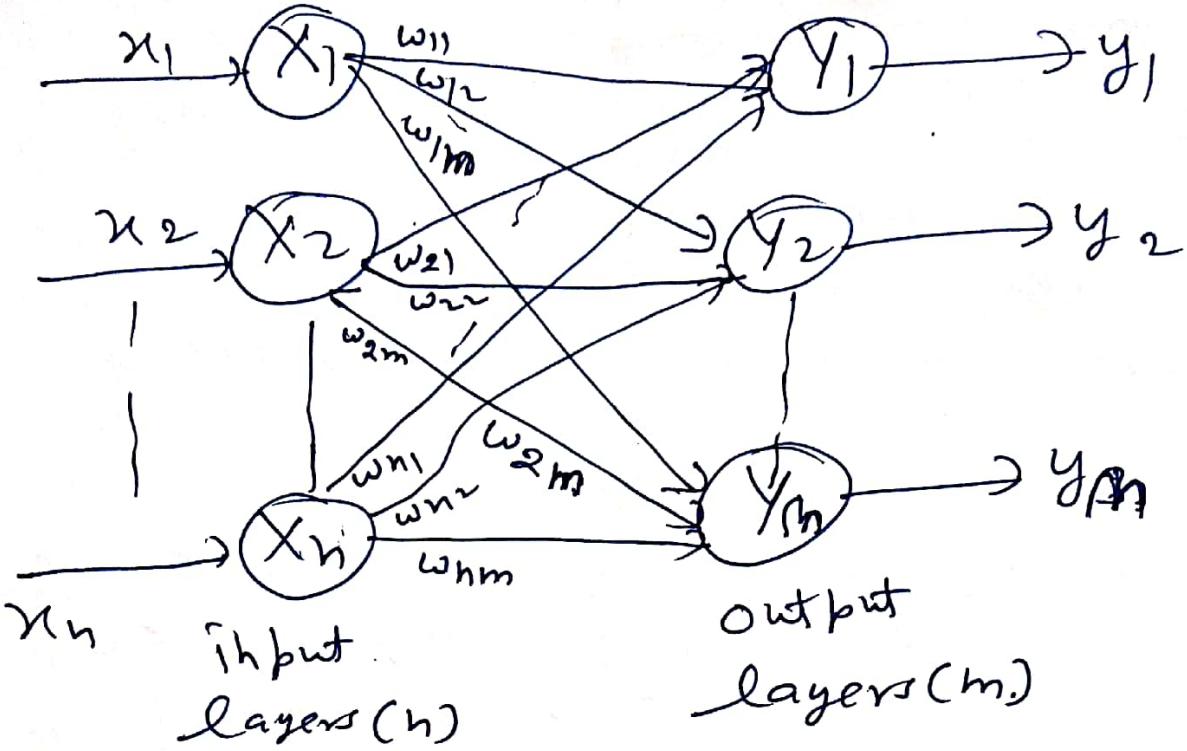
$$w_i(new) = w_i(old) + \alpha t x_i$$

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

Architecture:



Perceptron network with single output.



Perceptron network architecture with multiple output

① Find the weights required to perform the following classification using perceptron network. The vector (1, 1, 1, 1) and (-1, 1, -1, -1) are belonging to the class (so have target value 1), vector (1, 1, 1, -1) and (1, -1, -1, 1) are not belonging to the class (so have target value -1). Assume learning rate as 1 and initial weights as 0.

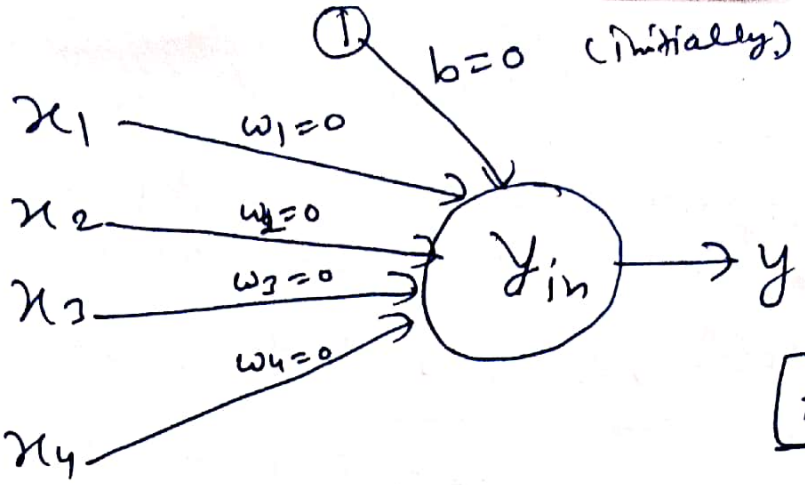
Ans Truth Table

x_1	x_2	x_3	x_4	b (bias)	t (target)
1	1	1	1	1	1
-1	1	-1	-1	1	1
1	1	1	-1	1	-1
1	-1	-1	1	1	-1

Iteration-1

Let $w_1, w_2, w_3, w_4 = 0$, $b = 0$ (given)

Learning rate (α) = 1



Assume $\theta = 0$

(Perceptron network)

Iteration-1

Input $[x_1=1, x_2=1, x_3=1, x_4=1]$

$$y_{in} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$= 0\times1 + 0\times1 + 0\times1 + 0\times1 + 0$$
$$y_{in} = 0$$

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

So $y = 0$

Since target (t) $\neq$ output (y)

$1 \neq 0$, So Calculate new weights

$$w_1(new) = w_1(old) + \Delta w_1$$
$$= 0 + \alpha t x_1$$
$$w_1(new) = 0 + 1\cdot1\cdot1 = 1$$

$$w_2(new) = w_2(old) + \Delta w_2$$
$$w_2(new) = 0 + \alpha t x_2 = 0 + 1\cdot1\cdot1 = 1$$

$$w_3(new) = w_3(old) + \Delta w_3$$
$$w_3(new) = 0 + \alpha t x_3 = 0 + 1\cdot1\cdot1 = 1$$

$$w_4(new) = w_4(old) + \Delta w_4$$
$$w_4(new) = 0 + \alpha t x_4 = 0 + 1\cdot1\cdot1 = 1$$

$$b(new) = b(old) + \Delta b$$
$$= 0 + \alpha t = 0 + 1\cdot1 = 1$$

New weights $= \{1, 1, 1, 1, 1\}$

~~for input $[x_1=1, x_2=-1, x_3=-1, x_4=1]$~~

~~$y_{in} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$~~

$= 1\times1 + 1\times-1 + 1\times-1 + 1\times1$

for input $\{x_1=-1, x_2=1, x_3=-1, x_4=-1\}, b=1$

$y_{in} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$

$= 1\times-1 + 1\times1 + 1\times-1 + 1\times-1 + 1$

$= -1 + 1 - 1 - 1 + 1$

$y_{in} = -1$

$y = -1$

$$y = \begin{cases} 1 & y_{in} > 0 \\ 0 & -0 \le y_{in} \le 0 \\ -1 & y_{in} < -0 \end{cases}$$

$\Delta w_1 = \alpha t x_1$

$= 1\cdot1\cdot(-1) = -1$

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

$\Delta w_2 = \alpha t x_2$

$= 1\cdot1\cdot(1) = 1$

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

$\Delta w_3 = \alpha t x_3 = 1\cdot1\cdot(-1) = -1$

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

~~$w_4(new) = w_4$~~ $\Delta w_4 = \alpha t x_4 = 1\cdot1\cdot(-1)$

$\Delta w_4 = -1$

$w_4(new) = w_4(old) + \Delta w_4$

$w_4(new) = 1 - 1 = 0$

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

$b(new) = 1 + 1\cdot1 = 2$

new weights $= [0, 2, 0, 0, 2]$

for inputs $\{x_1=1, x_2=1, x_3=+1, x_4=-1\}, b=1$

$t=-1$

$$y_{in} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$= 0\cdot1 + 2\cdot1 + 0\cdot1 + 0\cdot(-1) + 2$$
$$= 0 + 2 + 0 + 0 + 2$$
$$y_{in} = 4$$

$y = 1$ as $y_{in} > 0$

$\Delta w_1 = \alpha t x_1$
$= 1(-1)1 = -1$
$w_1(new) = w_1(old) + \Delta w_1$
$w_1(new) = 0 + 1 = 1$
$w_1(new) = 1$

$\Delta w_2 = \alpha t x_2$
$\Delta w_2 = 1(-1)1 = -1$
$w_2(new) = w_2(old) + \Delta w_2$
$w_2(new) = 2 + 1 = 1$

$\Delta w_3 = \alpha t x_3$
$= 1\cdot(-1)1 = -1$
$w_3(new) = w_3(old) + \Delta w_3$
$w_3 = 0 - 1 = -1$
$w_3(new) = -1$

$\Delta w_4 = \alpha t x_4 = 1\cdot(-1)\cdot(-1) = 1$
$w_4(new) = w_4(old) + \Delta w_4$
$w_4(new) = 0 + 1 = 1$

$\Delta b = \alpha t = 1\cdot(-1) = -1$
$b(new) = b(old) + \Delta b$
$b(new) = 2 - 1 = 1$

new weights $= [-1, 1, -1, 1, 1]$

for Inputs $\{x_1=1, x_2=-1, x_3=-1, x_4=1, b=1\}$, $t=-1$ (8)

$$y_{in} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$= -1\times1 + 1\times(-1) + (-1)\times(-1) + 1\times1 + 1$$
$$= -1 - 1 + 1 + 1 + 1$$

$$\boxed{y_{in} = 1}$$

$$\boxed{y = 1}$$

$$\begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & 0 \le y_{in} \le 0 \\ 1 & y_{in} < -0 \end{cases}$$

Now

$\Delta w_1 = \alpha + x_1$
$= 1.(-1)\times1$
$= -1$

$\boxed{w_1(new) = w_1(old) + \Delta w_1 = -1 + (-1) = -2}$

$\Delta w_2 = \alpha + x_2 = 1.(-1)\times(-1) = 1$

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

$\boxed{w_2(new) = 1 + 1 = 2}$

~~$\Delta w_3 = w_3(old)$~~

$\Delta w_3 = \alpha + x_3 = 1.(-1).(-1) = 1$

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

$w_3(new) = -1 + 1$

$\boxed{w_3(new) = 0}$

$\Delta w_4 = \alpha + x_4 = 1.(-1).1 = -1$

$w_4(new) = w_4(old) + \Delta w_4$

$= 1 + (-1)$

$\boxed{w_4(new) = 0}$

$\Delta b = \alpha + 1 = 1.(-1) = -1$

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

$b(new) = 1 + (-1)$

$\boxed{b(new) = 0}$

new weights $= \{w_1 = -2, w_2 = 2, w_3 = 0, w_4 = 0, b = 0\}$

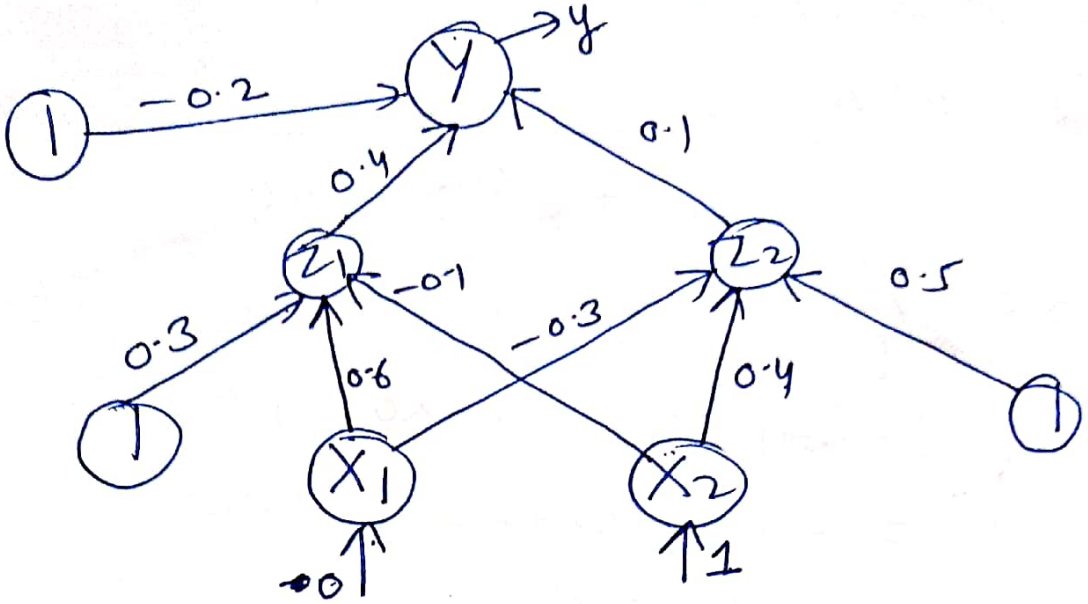
after first iteration – 1 (Loop-1)

We will repeat the process again & again until there is no weight change occur.

Perceptron - Network table

Inputs x_1 x_2 x_3 x_4 b	Target (t)	Net input (y_{in})	output (y)	weight changes (Δw) Δw_1 Δw_2 Δw_3 Δw_4 Δb	weights (new) w_1 w_2 w_3 w_4 b
iteration-1					
1 1 1 1 1	1	0	0	1 1 1 1 1	1 1 1 1 1
−1 1 −1 −1 1	1	−1	−1	−1 1 −1 −1 1	0 2 0 0 2
1 1 1 −1 1	−1	4	1	1 −1 −1 1 −1	−1 1 −1 1 1
1 −1 −1 1 1	−1	1	1	1 1 1 −1 −1	−2 2 0 0 0
iteration-2					
1 1 1 1 1	1	0	0	1 1 1 1 1	−1 3 1 1 1
−1 1 −1 −1 1	1	3	1	0 0 0 0 0	−1 3 1 1 1
1 1 1 −1 1	−1	4	1	−1 −1 −1 1 −1	−2 2 0 2 0
1 −1 −1 1 1	−1	−2	−1	0 0 0 0 0	−2 2 0 2 0
iteration-3					
1 1 1 1 1	1	2	1	0 0 0 0 0	−2 2 0 2 0
−1 1 −1 −1 1	1	2	1	0 0 0 0 0	−2 2 0 2 0
1 1 1 −1 1	−1	−2	−1	0 0 0 0 0	−2 2 0 2 0
1 −1 −1 1 1	−1	−2	−1	0 0 0 0 0	−2 2 0 2 0

(No weight change) → Stop process.



Using back propagation network, find the new weights for the net shown in figure. It is presented with the input pattern [0,1] and the target output is 1. Use a learning rate $\alpha = 0.25$ and binary sigmoidal activation function.

Binary sigmoidal activation function $f(x) = \frac{1}{1+e^{-x}}$

Input pattern $(x_1, x_2) = \{0,1\}$

target $(t) = 1$

for Z_1 layer:

Net input $= 0.3 + 0 \times 0.6 + 1 \times (-0.1)$

$z_{in1} = 0.3 + 0 - 0.1 = \underline{0.2}$

for Z_2 layer

$z_{in2} = 0 \times -0.3 + 1 \times 0.4 + 0.5$

$= \underline{0.9}$

output at Z_1

$Z_1 = f(z_{in1}) = \frac{1}{1+e^{-z_{in1}}} = \frac{1}{1+e^{-0.2}} = \underline{0.5498}$

$$z_2 = f(z_{in2}) = \frac{1}{1+\bar{e}^{z_{in2}}} = \frac{1}{1+\bar{e}^{0.9}} = 0.7109$$

So ~~$Y = Z_1 + Z_2$~~

$$Y = z_1 \times 0.4 + z_2 \times 0.1 + (-0.2)$$

$$= 0.5498 \times 0.4 + 0.7109 \times 0.1 - 0.2$$

$$Y = 0.09101$$

$$y = f(Y) = \frac{1}{1+e^{-y}} = \frac{1}{1+\bar{e}^{0.09101}} = 0.5227$$

Computation of error

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

$$f'(y_{in}) = f(y)[1 - f(y)]$$

$$= 0.5227[1 - 0.5227]$$

$t_k = 1$

$y_k = 0.5227$

$$= 0.5227 \times 0.4773$$

$$= 0.2495$$

$$\delta_1 = (1 - 0.5227)(0.2495)$$

$$= 0.1191$$

Change in weights and bias (b/w hidden layer & output layer)

$$\Delta w_{jk} = \alpha \cdot \delta_k \cdot z_j \qquad \Delta w_{ok} = \alpha \cdot \delta_k$$

$$\Delta w_1 = \alpha \delta_1 z_1 = 0.25 \times 0.1191 \times 0.5498$$

$$= 0.0164$$

$$\Delta w_2 = \alpha \delta_1 z_2 = 0.25 \times 0.1191 \times 0.7109$$

$$= 0.02117$$

$$z_1 = f(z_{in1}) = \frac{1-e^{-z_{in1}}}{1+e^{-z_{in1}}} = \frac{1-e^{0.4}}{1+e^{0.4}}$$ (3)

$$z_1 = -0.1974$$

$$z_2 = f(z_{in2}) = \frac{1-e^{-z_{in2}}}{1+e^{-z_{in2}}}$$

$$= \frac{1-e^{-1.2}}{1+e^{+1.2}} = 0.537$$

$$y_{in} = -0.2 + 0.4 \times (-0.1974) + 0.1 \times 0.537$$

$$= -0.22526$$

$$Y = f(y_{in}) = \frac{1-e^{-y_{in}}}{1+e^{+y_{in}}} = \frac{1-e^{+0.22526}}{1+e^{-0.22526}}$$

$$= -0.1122$$

Compute the error b/w output layer & hidden layer

$$\delta_{0h} = (t-y) f'(y_{in})$$

↓ derivative of $f(y_{in})$ first

$$f'(y_{in}) = \frac{1}{2}(1+f(y_{in}))(1-f(y_{in}))$$

$$= \frac{1}{2}(1-0.1122)(1+0.1122)$$

$$= 0.4937$$

$$\delta_{0h} = (1+0.1122)(0.4937) = 0.5491$$

Change in weight b/w output layer & hidden layer

$$\Delta w_0 = \alpha \delta_{0h} 1 = 0.25 \times 0.5491 = 0.1373$$

$$\Delta w_1 = \alpha \delta_{0h} z_1 = 0.25 \times 0.5491 \times (-0.1974)$$

$$\Delta w_1 = -0.0271$$

$\Delta w_2 = \alpha \delta_{oh} z_2 = 0.25 \times 0.5491 \times 0.537$ (4)

$= 0.0737$

Compute the error b/w hidden & Input layer

Error $\delta_{inh z_1} = \delta \times f'(z_{in_1})$

$= 0.5491 \times$

$\delta_{inh z_1} = \delta_{oh} \times w_1 \times f'(z_{in_1})$

$= 0.5491 \times 0.4 \times \frac{1}{2}[1-z_1][1+z_1]$

$= 0.5491 \times 0.4 \times \frac{1}{2}[1+0.1974][1-0.1974]$

$= 0.1056$

$\delta_{in} z_2 = \delta_{on} \times w_2 \times f'(z_2)$

$= 0.5491 \times 0.1 \times \frac{1}{2}(1-z_2)(1+z_2)$

$= 0.5491 \times 0.1 \times \frac{1}{2}(1-0.537)(1+0.537)$

$= 0.0195$

Change in weight b/w Hidden & Input layer

$\Delta v_{01} = \alpha \delta_{inz_1} \times 1 = 0.25 \times 0.1056 \times 1 = 0.0264$

$\Delta v_{11} = \alpha \delta_{inh z_1} \times x_1 = 0.25 \times 0.1056 \times -1 = -0.0264$

$\Delta v_{21} = \alpha \delta_{inh z_1} \times x_2 = 0.25 \times 0.1056 \times 1 = 0.0264$

$\Delta v_{02} = \alpha \delta_{inz_2} \times 1 = 0.25 \times 0.0195 \times 1 = 0.0049$

$\Delta v_{12} = \alpha \delta_{inz_2} \times x_1 = 0.25 \times 0.0195 \times -1 = -0.0049$

$\Delta v_{22} = \alpha \delta_{inz_2} \times x_2 = 0.25 \times 0.0195 \times 1 = 0.0049$

Computation of final weights of the network.

$V_{01}(new) = V_{01}(old) + \Delta V_{01} = 0.3 + 0.0264 = 0.3264$

$V_{11}(new) = V_{11}(old) + \Delta V_{11} = 0.6 - 0.0264 = 0.5736$

$V_{21}(new) = V_{21}(old) + \Delta V_{21} = -0.1 + 0.0264 = -0.0736$

$V_{02}(new) = V_{02}(old) + \Delta V_{02} = 0.5 + 0.0049 = 0.5049$

$V_{12}(new) = V_{12}(old) + \Delta V_{12} = -0.3 + (-0.049) = -0.3049$

$V_{22}(new) = V_{22}(old) + \Delta V_{22} = 0.4 + 0.0049 = 0.4049$

$W_0(new) = w_0(old) + \Delta w_0 = -0.2 + 0.1373 = -0.0627$

$w_1(new) = w_1(old) + \Delta w_1 = 0.4 + (-0.0271) = 0.3729$

$w_2(new) = w_2(old) + \Delta w_2 = 0.1 + (0.0737) = 0.1737$

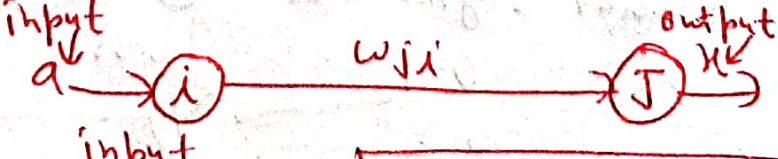
Hebb's Network (Hebb's Rule)

* This rule was proposed by Donald Hebb in 1949.
* Hebb's rule say " if neuron i is near enough to excite ~~neuron~~ neuron j and repeatedly participate in its activation, the Synaptic connection between these neurons is strengthen and neuron j becomes more sensitive to stimuli from neuron i.
* Hebb's rule can be represented using two sub rules:
  (i) If two neurons on either side of the connection are activated synchronously, then the weight of that connection is increased.
  (ii) If two neurons on either side of the connection are activated asynchronously, then the weight of that connection is decreased.
* If two neurons have no relationship, then the weight will not change.
* If inputs of both the nodes are either positive or negative, then a positive weight exist between the nodes.
* If the input of a node is either positive or negative for others, a negative weight exist b/w the nodes.

weight update Hebb's formula

~~(weight) = w_i(old) + x_i y~~.

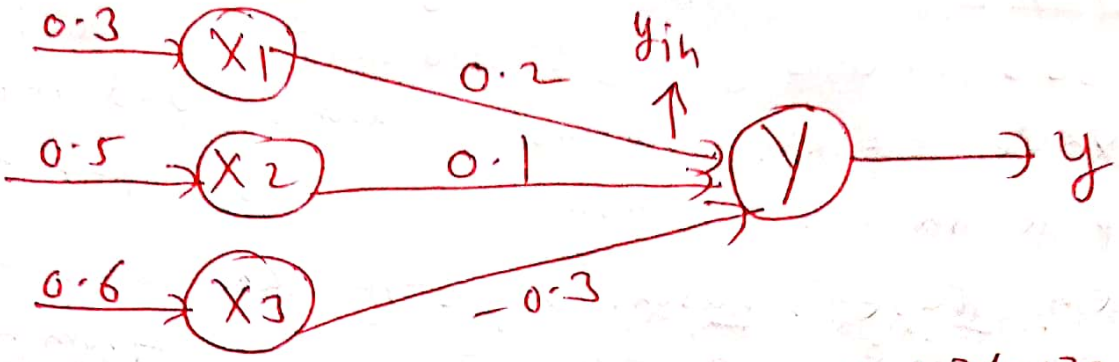
input = a
output = x
Learning = α rate



$$w_{ji(new)} = w_{ji(old)} + \Delta w_{ji}$$ → change in weight

$$\Delta w_{ji} = \alpha \cdot a \cdot x$$ ↓ Learning rate

(1) For the given network, calculate the net input to the output neuron. (4)



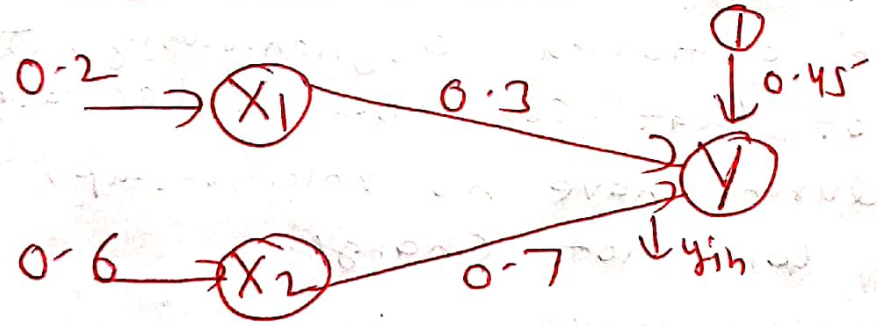
Net input $y_{in} = x_1w_1 + x_2w_2 + x_3w_3$

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

$= \underline{\underline{-0.07}}$

---

(2) Calculate the net input for the network, with bias included in the network.



Net input $y_{in} = x_1w_1 + x_2w_2 + b \times 1$

$= 0.2 \times 0.3 + 0.6 \times 0.7 + 0.45 \times 1$

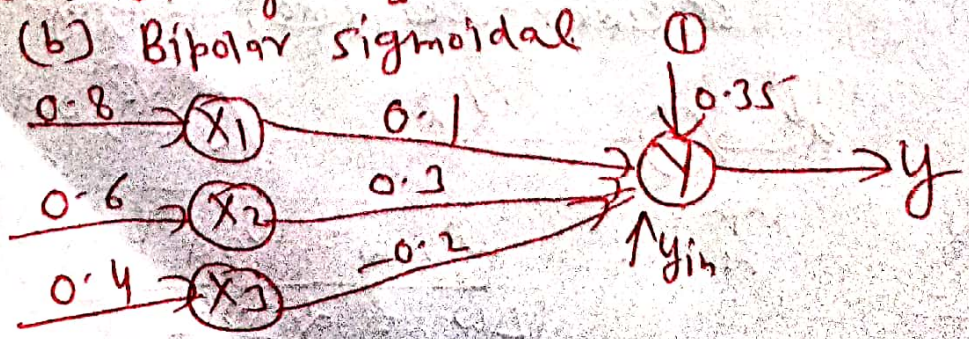
$= \underline{\underline{0.93}}$

---

(3) Obtain the output of the neuron Y for the given network using activation function as

(a) Binary sigmoidal

(b) Bipolar sigmoidal



The net input y_{in} = ~~$w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$~~ (5)

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

$y_{in} = 0.8 \times 0.1 + 0.6 \times 0.3 + 0.4 \times (-0.2) + 0.35 \times$

$y_{in} = 0.53$

(a) for Binary sigmoidal activation function

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

$$\boxed{Y = 0.625}$$

(b) for Bipolar sigmoidal activation function

$$\text{output}(Y) = f(y_{in}) = \frac{2}{1+e^{-x}} - 1$$

$$\text{output}(Y) = \frac{2}{1+e^{-0.53}} - 1 = 0.259$$

$$\boxed{Y = 0.259}$$

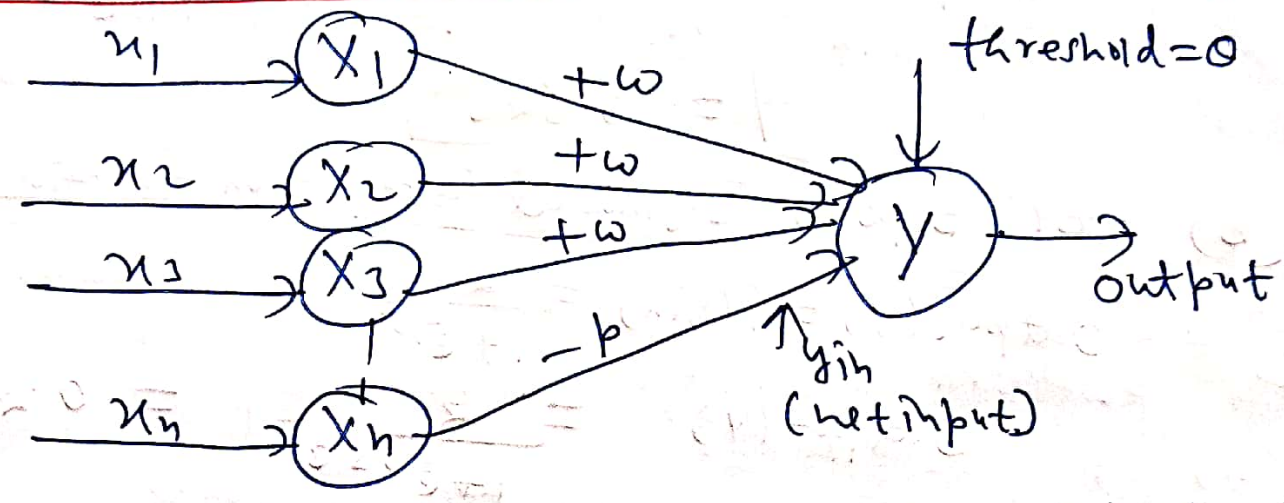
McCulloch Pitts Neuron

1. It is also called MP- Neuron model.
2. It is the first computational model of a neuron proposed by Warren McCulloch and Walter Pitts in 1943.
3. This model allows binary 0 and 1 states only.
4. These binary neuron are connected by directed weighted path.
5. The connected path can have positive weights (Excitatory) or negative weights (Inhibitory).
6. The neuron is associated with the threshold value.

(7) The neuron activates (fires), if the total inputs to the neuron is greater than or equal to ($\geq$) to the threshold value. (6)

(8) The M-P neuron model has no particular training algorithm.

Architecture of MP-Neuron Model



Net input (y_{in}) = $x_1 \times w + x_2 \times w + x_3 \times w + \cdots - x_n \times (-p)$

output (Y) = $f(y_{in}) = 1$, if $y_{in} \geq \theta$
$= 0$, if $y_{in} < \theta$

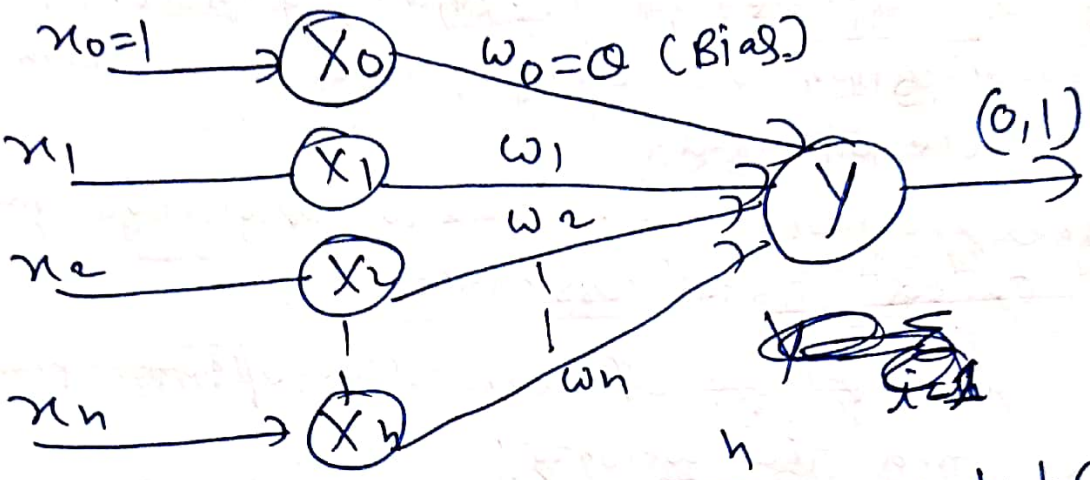
~~Rosenblatt~~ Rosenblatt's Perceptron Model

(1) Rosenblatt's perceptron was designed by Rosenblatt in 1958 to overcome most of the issues of the ~~neu~~ McCulloch-Pitts neuron.
   (a) It can process non-boolean inputs.
   (b) It can assign different weights to each input automatically.
   (c) The threshold θ is computed automatically.

(2) It is a single layer neural network.

(3) Rosenblatt's perceptron can be seen as a set of inputs that are weighted and to which we apply an activation function.

(4) We can represent the perceptron as



$$\text{output } (Y) = \sum_{i=1}^{n} w_i x_i + 1.0$$

$$Y = w_1 x_1 + w_2 x_2 + \cdots w_n x_n + 0$$

(5) The inputs can be seen as neurons and will be called the input layer.

(6) These neurons (input layer) and the activation function form a perceptron.

(7) Activation function is used to calculate the output.

$$y = \begin{cases} 1 & , \quad Y \geq 0 \\ 0 & , \quad Y < 0 \end{cases}$$

(8) This model is considered as the first generation of neural network. It has main limitation of not being able to solve non-linear problems.

Soft Computing

ADALINE (Part-3)

1. ADALINE is a single layer neural network which uses linear activation function (Bipolar activation function) for its input and target outputs.
2. Learning rate is between 0.1 to 1 and bias to.
3. ADALINE use delta rule to update the weights between the connections to minimize the difference between output value and target value.
4. Delta rule for change in weight

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

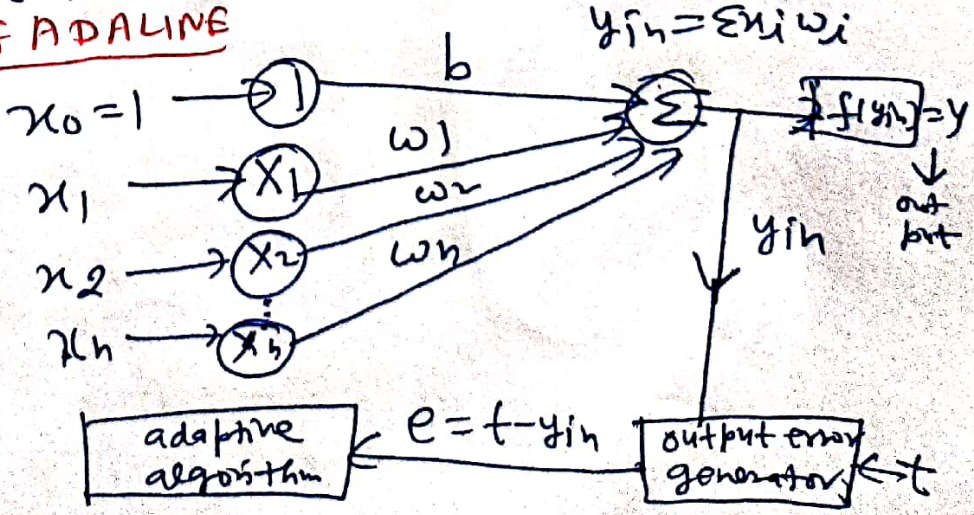
where Δw_i = change in weight
α = learning rate
x_i = input vector (-1, +1)
y_{in} = net input to output unit

$$Y = f(y_{in}) = \Sigma (x_i w_i)$$

t = target output

$$\text{Error } (E_i) = \Sigma (t - y_{in})^2$$

Architecture of ADALINE



ADALINE Training algorithm

(i) weights and bias are set to some random values but not zero. Set the learning rate parameter α.

(ii) Perform steps (iii) to (VII) when stopping condition is false.

(iii) Perform steps (IV) to (VI) for each bipolar training pair s:t

(iv) Set activations for input unit $i = 1$ to n

$$x_i = s_i$$

(v) Calculate the net input to the output unit

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

(vi) Update the weights and bias for $i = 1$ to n

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

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

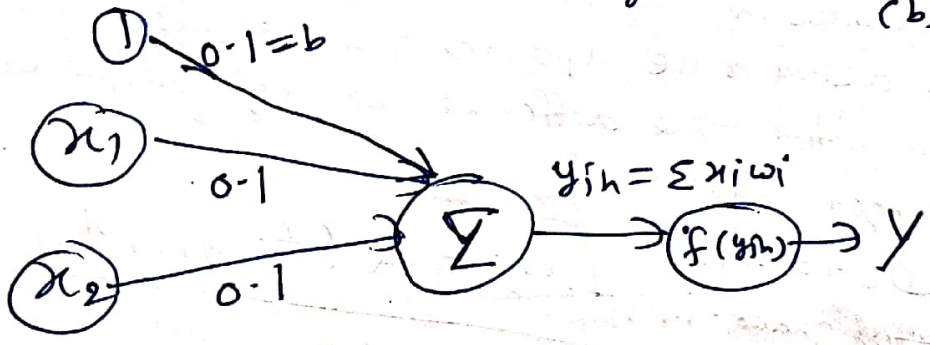
(vii) If the highest weight change that occured during training is smaller than a specific tolerance, then stop the training process else continue.

Q= Implement OR function with bipolar inputs and targets using ~~adaline~~ ADALINE network

Ans The truth table for OR function with bipolar inputs and targets is shown in table 1.

x_1	x_2	~~output~~ Bias (b)	target (t)
-1	-1	1	-1
-1	1	1	1
1	-1	1	1
1	1	1	1

Assume random initial weights and bias (b) = 0.1



$$y_{in} = \Sigma x_i w_i + b$$

for $x_1 = 1, x_2 = 1$ $= 1 \times 0.1 + 1 \times 0.1 + 0.1 = 0.3$

$$\Delta w_i = \alpha (t - y_{in}) x_i \quad , i = 1$$

$$\Delta w_1 = 0.1 (1 - 0.3) \times 1$$
$$= 0.1 \times 0.7 \times 1 = 0.07$$

$$w_1(new) = w_1(old) + \Delta w_1$$
$$w_1(new) = 0.1 + 0.07 = 0.17$$

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

$$\Delta w_2 = 0.1 (1 - 0.3) \times 1 = 0.1 \times 0.7$$
$$\Delta w_2 = 0.07$$

$$w_2(new) = w_2(old) + \Delta w_2$$
$$= 0.1 + 0.07$$

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

$b(new) = b(old) + \Delta b$, $\Delta b = 0.1 \times (1-0.3) \times 1$

$= 0.1 + 0.07$ $\Delta b = 0.07$

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

Now calculate the error

$E = (t - y_{in})^2$

$E = (1-0.3)^2 = 0.49$

for first input sample ($x_1 = 1, x_2 = 1$), the final weights are $w = [0.17, 0.17, 0.17]$

$\boxed{error(E) = 0.49}$

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

$y_{in} = \sum w_i x_i$

$= w_1 x_1 + w_2 x_2 + b$

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

$\boxed{y_{in} = 0.17 - 0.17 + 0.17 = 0.17}$

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

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

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

$= 0.17 + 0.1 \times 0.83$

$= 0.17 + 0.083$

$\boxed{w_1(new) = 0.253}$

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

$w_2(new) = 0.17 + \alpha (t - y_{in}) x_2$

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

$= 0.17 + 0.1 \times 0.83 \times -1$

$= 0.17 - 0.083$

$\boxed{w_2(new) = 0.087}$

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

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

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

$b(new) = 0.17 + 0.083 = 0.253$

So for $\{x_1 = 1, x_2 = -1\}$, new weights are

$w = \{0.253, 0.087, 0.253\}$

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

for $x_1 = -1, x_2 = 1,$

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

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

$y_{in} = -0.253 + 0.087 + 0.253$

$y_{in} = 0.087$

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

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

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

$= 0.253 + 0.1 \times (0.913) \times -1$

$w_1(new) = 0.253 - 0.0913$

$w_1(new) = 0.1617$

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

$= 0.087 + \alpha(t - y_{in})x_2$

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

$= 0.087 + 0.0913$

$w_2(new) = 0.1783$

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

$= 0.253 + \alpha (t - y_{in})$

$= 0.253 + 0.1 (1 - 0.087)$

$= 0.253 + 0.0913$

$b(new) = 0.3443$

New weights are $\{0.1617, 0.1783, 0.3443\}$

error $= (t - y_{in})^2 = (1 - 0.087)^2$

$= 0.83$

for $x_1 = -1$, $x_2 = -1$, $t = -1$

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

~~$= 0.1617 \times (-1)$~~

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

$= -0.1617 - 0.1783 + 0.3443$

~~$y_{in} = 0.3227$~~ $y_{in} = 0.0043$

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

$= 0.1617 + \alpha (t - y_{in}) x_1$

$= 0.1617 + 0.1 (-1 - 0.0043) \times (-1)$

$= 0.1617 + 0.1 (+1.0043)$

$= 0.1617 + 0.10043$

$w_1(new) = 0.2621$

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

$= 0.1783 + \alpha (t - y_{in}) x_2$

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

$= 0.1783 + 0.10043$

$w_2(new) = 0.2787$

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

$= 0.3443 + \alpha(t - y_{in})$

$= 0.3443 + 0.1(-1 - 0.0043)$

$= 0.3443 + 0.1(-1.0043)$

$b(new) = 0.3443 - 0.10043$

$b(new) = 0.2439$

$error = (t - y_{in})^2$

$= (-1 - 0.0043)^2 = (-1.0043)^2$

$error = 1.01$

Total mean square error $= 0.49 + 0.69 + 0.83 + 1.01$

$= 3.02$

after loop 1

after loop 2: $= 1.94$

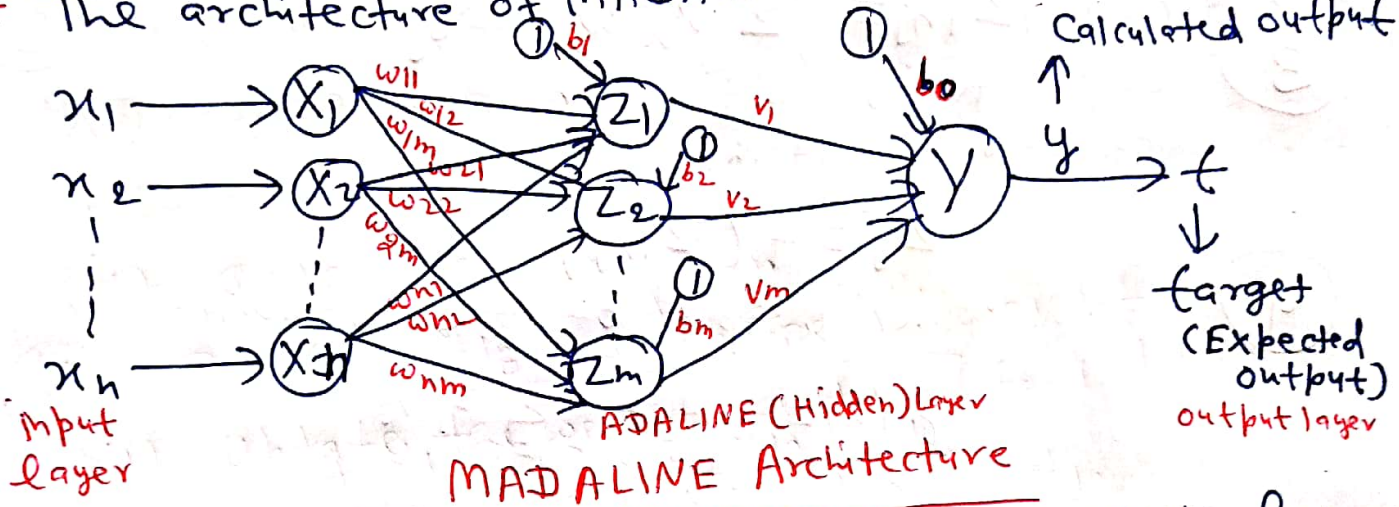
after loop 3: $= 1.55$

after loop 4 $= 1.387$

Error reduces after each loop

MADALINE :

* This model consists of many ADALINE in parallel with a single output unit whose value is based on certain selection rule.
* The architecture of MADALINE is as given:



* The weights between input layer and ADALINE layer are adjusted during the training process.
* The weights between ADALINE layer and MADALINE (output) layer are fixed, positive and posses equal value.
* The ADALINE and MADALINE layer neuron have a bias of input (1) connected to them.
* we will initialize the weight between input layer and ADALINE layer. ($w_{11}, w_{12}, w_{21}, w_{22}, \dots w_{n1}, w_{n2}, \dots w_{nm}$) to some random value.
* initiate weight between ADALine and MADALINE layer to some random value ($v_1, v_2 \dots v_m$). Also set learning rate to some fix value.
* Activation function for MADALINE is bipolar step function.

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

Training algorithm for MADALINE

(1) Initialize the weight between input layer and hidden layer, hidden layer and output layer to some random value. Also set learning rate to random value.

(2) Calculate net input to each hidden ~~layer~~ unit.

$$z_{inj} = \sum x_i w_{ij} + b_j$$

(3) Calculate output of each hidden unit using activation function.

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

(4) find the output of the network.

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

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

(5) (i) If target output = Calculated output then no weight updation is required.

(ii) If target output (t) ≠ Calculated output(y), then update the weight and Calculate new weights.

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

(6) Repeat the process till there is no weight change or min. weight change.

Auto Associative memory Network

(1) Auto Associative neural network have same value of training input and target output.

(2) The determination of the weights of the associative network is called storing of vectors.

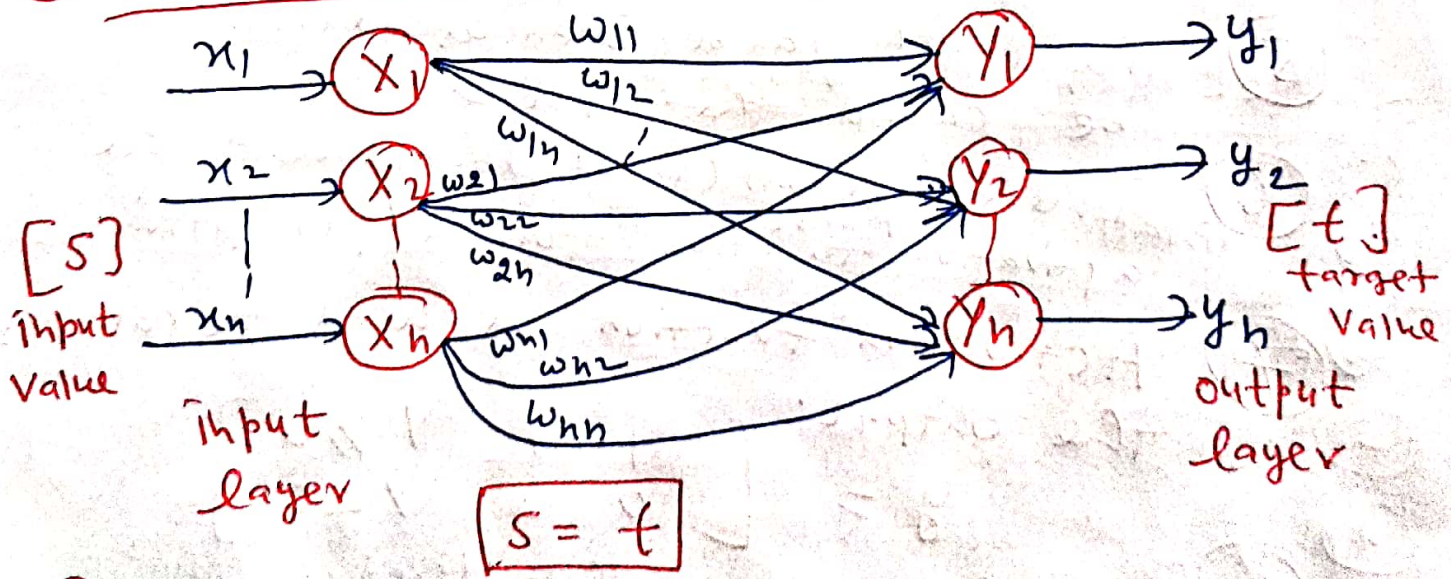
(3) It is capable of retriving a piece of data upon presentation of only partial information from that piece of data.

(4) Neural network using auto associative memory are called auto-associative network.

(5) It makes use of hebb's rule/outer product rule to find the weights.

(6) The input and output layers are connected through weighted connection.

(7) Architecture of Auto associative network



$S = t$

(8) Training algorithm for auto associative network

(a) initialize all weights to zero.

(b) for each of the vector that has to be stored, perform steps c to e.

(c) Activate each of input unit $x_i = S_i$ ($i = 1$ to n)

(d) Activate each of the output unit

$$y_j = s_j \quad (j = 1 \text{ to } n)$$

(e) Adjust the weights

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j$$

or using outer product rule

$$w_{\text{new}} = [s]^T [t]$$

where T = Transpose of matrix

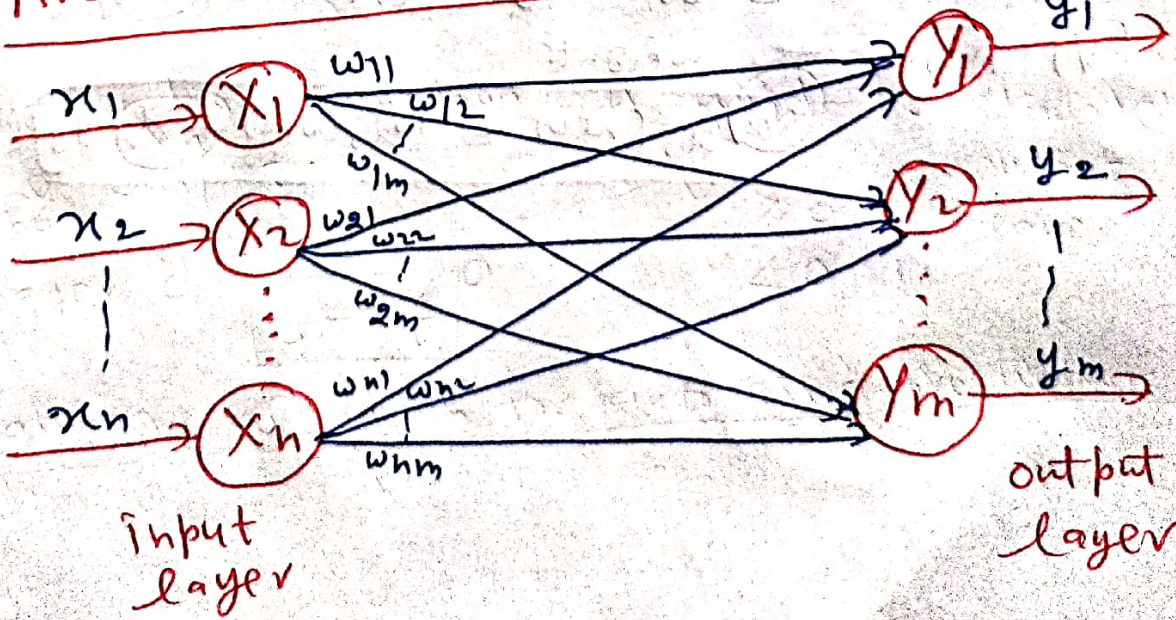
Hetero associative memory network:

(1) Hetero associative memory is capable of retrieving a piece of data of one category upon presentation of data from another category.

(2) Training input vector and target output vector are different.

(3) uses Hebb's rule/delta rule/outer product rule to find the weight matrix.

(4) Architecture of hetero-associative memory network



(5) The net finds an appropriate output vector which corresponds to an input vector x, that may be either one of the stored pattern or a new pattern.

(6) There exist weighted interconnections between the input and output layers.

(7) The input and output layer units are not correlated with each other.

(8) Hebb's ~~auto-asso~~ training algorithm for hetero-associative neural network.

(i) Set all the initial weights to zero.

$$w_{ij} = 0 \quad (i = 1 \text{ to } n,\ j = 1 \text{ to } m.)$$

(ii) for each training target input output pair, perform steps (iii) to (iv)

(iii) Activate the input layer units to current training input

$$x_i = s_i \quad (\text{for } i = 1 \text{ to } n.)$$

(iv) Activate the output layer units to current ~~target~~ output.

$$y_j = t_j \quad (\text{for } j = 1 \text{ to } m.)$$

(v) Adjust the weights

$$\boxed{w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j} \quad (\text{for } i = 1 \text{ to } n,\ j = 1 \text{ to } m.)$$

Bi-directional Associative Memory (BAM)

(1) It is a hetero associative recurrent neural network consisting of two layers
(i) X-layer (ii) Y-layer.
which are connected by means of directional weighted connection paths.

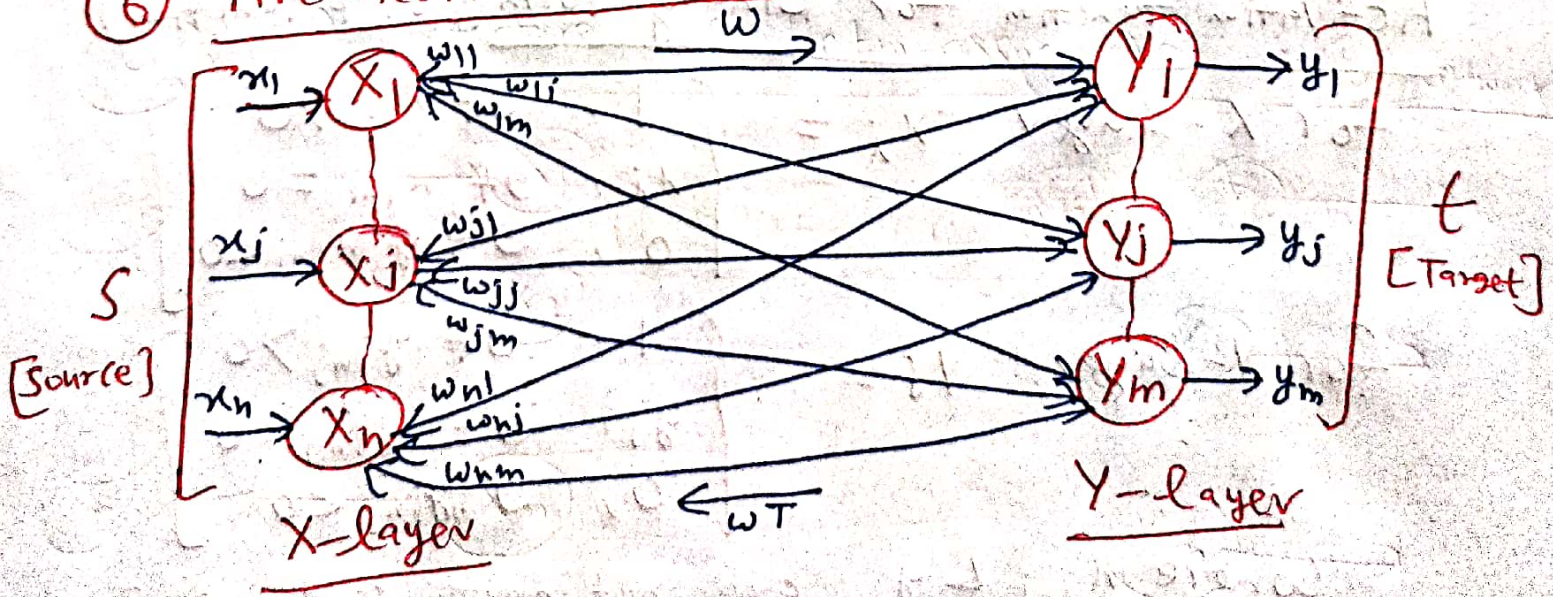
(2) This network iterates by sending a signal back and forth between the two layers until all neurons reach equilibrium.

(3) The network can respond to input on either layer.

(4) The weight matrix is calculated in both directions.

(5) If the weight matrix ~~the~~ for signal sent from the X-layer to Y layer is W then weight matrix for signal sent from Y-layer to X-layer is W^T.

(6) Architecture of BAM



Types of BAM

(i) Discrete BAM (ii) Continuous BAM

(i) Discrete BAM

Discrete BAM has the same structure as of BAM. It is available in two forms.

(1) Binary form (0,1)
(2) Bipolar form (−1, 1)

for Binary input vector, the weight matrix w_{ij} is

$$w_{ij} = \sum_{p=1}^{P} [2s_i(p)-1][2t_j(p)-1]$$

where $s(p)$ = input vector
$t(p)$ = target vector
and $p = 1, 2, 3, \cdots P$

for Bipolar input vector

$$\text{weight matrix } w_{ij} = \sum_{p=1}^{P} s_i(p)\, t_j(p)$$

Activation function for Binary form (step function)

for X-layer $x_i = \begin{cases} 1, & \text{if } x_{ini} > 0 \\ x_i, & \text{if } x_{ini} = 0 \\ 0, & \text{if } x_{ini} < 0 \end{cases}$

for Y-layer $y_j = \begin{cases} 1, & \text{if } y_{inj} > 0 \\ y_j, & \text{if } y_{inj} = 0 \\ 0, & \text{if } y_{inj} < 0 \end{cases}$

where x_{ini} & y_{inj} are net input at X-layer and net input at Y-layer

for Bipolar inputs (Activation functions)

for x-layer

$$x_i = \begin{cases} 1, & \text{if } x_{ini} > \theta_i \\ x_i, & \text{if } x_{ini} = \theta_i \\ -1, & \text{if } x_{ini} < \theta_i \end{cases}$$

for y-layer

$$y_j = \begin{cases} 1, & \text{if } y_{inj} > \theta_j \\ y_j, & \text{if } y_{inj} = \theta_j \\ -1, & \text{if } y_{inj} < \theta_j \end{cases}$$

(ii) Continuous BAM

1. A continuous BAM has the capability to transfer the input smoothly and continuously into the respective output.
2. The continuous BAM uses logistic sigmoid activation function for all units.
3. The weights are determined using hebb's rule

$$w_{ij} = \sum_{p=1}^{P} [2s_i(p)-1][2t_j(p)-1]$$

4. Activation function for Binary logistic function

$$f(y_{inj}) = \frac{1}{1+e^{-y_{inj}}}$$

Activation function for Bipolar logistic function

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

where net input $y_{inj} = \sum x_i w_{ij} + b_j$ using hebb's rule

Hopfield Network :

1* John J. Hopfield proposed a network in 1982 which consists of a set of neurons with output of each neuron is connected as feedback to all other neurons.

2* Hopfield network used in many applications of associative memory and many optimization problem (Travelling Salesman problem).

3 It is basically of two types
(i) Discrete hopfield network.
(ii) Continuous hopfield network.

4 The hopfield network is an auto associative fully interconnected single layer feedback network.

5 It is a symmetrically weighted network

(i) Discrete hopfield network:

(a) Hopfield network when operates in discrete line fashion, then it is called 'Discrete hopfield network'.

(b) This network has two values inputs
(i) Binary (0,1) (ii) Bipolar (1,-1)

(c) The network has symmetrical weights with no self-connections

$$w_{ij} = w_{ji} \qquad w_{ij} = 0$$

(d) The architecture of discrete hopfield network model consists of two

One inverting and other non-inverting. (15)

(e) The output from each processing element are feedback to the input of the other processing elements but not to itself

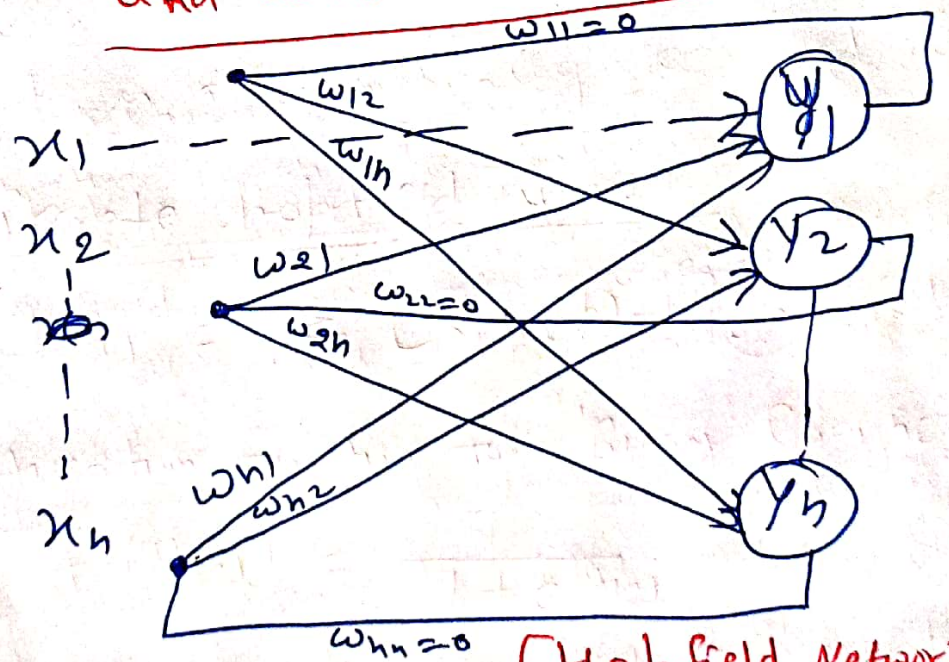
(f) for storing a set of binary patterns $s(p)$ where $p = 1$ to P, The weight matrix is

$$w_{ij} = \sum_{p=1}^{P} [2s_i(p)-1][2s_j(p)-1], \quad (i \neq j)$$

(g) for storing a set of bipolar pattern $s(p)$, the weight matrix is

$$w_{ij} = \sum_{p=1}^{P} s_i(p) \cdot s_j(p), \quad (i \neq j)$$

and weight for self connection $w_{ij} = 0$



[Hopfield Network Architecture]

Binary pattern is like $s(p) = [0\ 1\ 1\ 0\ 1]$ ($s_1\ s_2\ s_3\ s_4\ s_5$)

Bipolar pattern is like $s(p) = [1\ -1\ 1\ 1\ -1]$

(ii) Continuous Hopfield network

(a) A discrete hopfield network can be modified to a continuous model.

(b) The nodes of this network have a continuous output rather than a two state discrete output.

(c) The output value is between 0 to 1.

(d) The continuous hopfield network can be realized as an electronic circuit, which uses non-linear amplifier and resistors

Self-organizing Maps (SOMs) (17)

1. This technique was developed by T. Kohonen in 1982.
2. SOMs are named as "self organizing" because no supervision is required.
3. Self organizing maps learn on their own through unsupervised competitive learning.
4. Self-organizing Maps are ~~are~~ neural network that uses unsupervised learning approach and trained its network through a competetive learning algorithm to map multidimensional data into lower-dimensional which allow easy interpretation of complex problems
5. SOMs have two layers.
   (i) input layer (ii) output layer.
   There are n units in the input layer and m unit in the output layer.
6. **SOMs Training algorithm:**
   (i) **Initialization:** Choose random number for the initial weights w_{ij}
   (ii) **Sampling:** Take a sampling training input vector $x = [x_1, x_2, \dots x_n]$ for the input layer.
   (iii) **Matching:** find the winning neuron from the output layer that has weight vector closest to the input vector. It can be calculated by taking the square of Euclidean distance for each output unit and find the output unit that has minimum Euclidean distance from the input vector.

   for each $j = 1$ to m

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

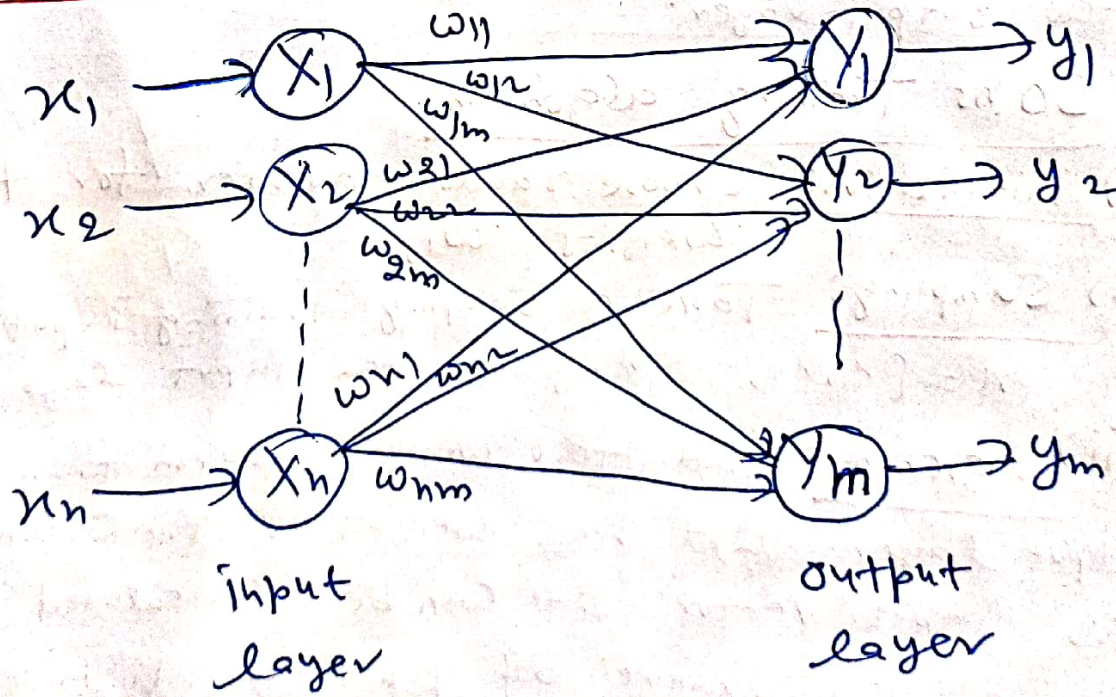
(iv) **New weight Calculation** find the new weights between input vector sample and winning output unit.

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

where α = learning rate

(v) **Continuation:** Repeat step-2 (ii) to (iv) until weight updation is negligible (i.e. how weights are similar to old weights) or feature map stop changing.

Architecture of Self-organizing Map



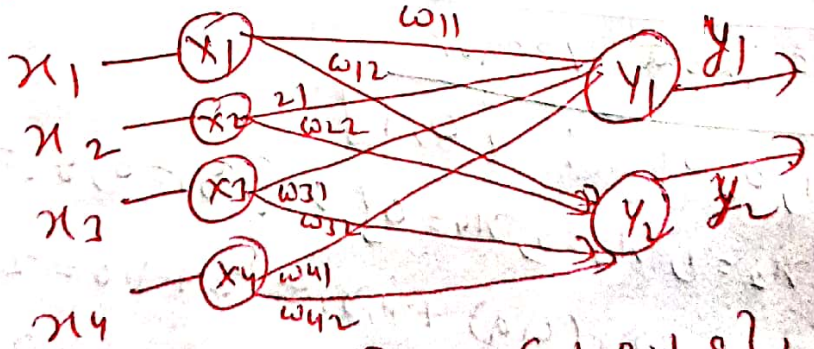
Numerical on hetro associative memory using hebb's rule ①

for the given input vector $S=(S_1, S_2, S_3, S_4)$ and output vector $T=(T_1, T_2)$ find the weight matrix using Hebb's rule in hetro associative training algorithm.

	$S=[S_1, S_2, S_3, S_4]$	$T=[T_1, T_2]$
I	(1, 0, 1, 0)	(1, 0)
II	(1, 0, 0, 1)	(1, 0)
III	(1, 1, 0, 0)	(0, 1)
IV	(0, 0, 1, 1)	(0, 1)

Ans for first input vector (1, 0, 1, 0) (1, 0)

initialize the weight to 0



$[x_1, x_2, x_3, x_4] = [1, 0, 1, 0]$,

$(y_1, y_2) = [1, 0]$

$w_{ij}(new) = w_{ij}(old) + x_i y_j$

$w_{11}(new) = w_{11}(old) + x_1 y_1 = 0 + 1 \cdot 1 = 1$

$w_{21}(new) = w_{21}(old) + x_2 y_1 = 0 + 0 \cdot 1 = 0$

$w_{31}(new) = w_{31}(old) + x_3 y_1 = 0 + 1 \cdot 1 = 1$

$w_{41}(new) = w_{41}(old) + x_4 y_1 = 0 + 0 \times 1 = 0$

$w_{12}(new) = w_{12}(old) + x_1 y_2 = 0 + 1 \times 0 = 0$

$w_{22}(new) = w_{22}(old) + x_2 y_2 = 0 + 0 \times 0 = 0$

$w_{32}(new) = w_{32}(old) + x_3 y_2$

$w_{32}(new) = 0 + 1 \times 0 = 0$

$w_{42}(new) = w_{42}(old) + x_4 y_2$

$= 0 + 0 \times 0 = 0$

for 2nd input vector

[1, 0, 0] : [1, 0]

$\{x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1\}$, $\{y_1 = 1, y_2 = 0\}$

$w_{11}(new) = w_{11}(old) + x_1 y_1 = 1 + 1 \times 1 = 2$

$w_{21}(new) = w_{21}(old) + x_2 y_1 = 0 + 0 \times 1 = 0$

$w_{31}(new) = w_{31}(old) + x_3 y_1 = 1 + 0 \cdot 1 = 1$

$w_{41}(new) = w_{41}(old) + x_4 y_1 = 0 + 1 \cdot 1 = 1$

$w_{12}(new) = w_{12}(old) + x_1 y_2 = 0 + 1 \cdot 0 = 0$

$w_{22}(new) = w_{22}(old) + x_2 y_2 = 0 + 0 \cdot 0 = 0$

$w_{32}(new) = w_{32}(old) + x_3 y_2 = 0 + 0 \cdot 0 = 0$

$w_{42}(new) = w_{42}(old) + x_4 y_2 = 0 + 1 \cdot 0 = 0$

for 3rd input vector

[1, 1, 0, 0] : [0, 1]

$\{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0\}$: $\{y_1 = 0, y_2 = 1\}$

$w_{11}(new) = w_{11}(old) + x_1 y_1 = 2 + 1 \cdot 0 = 2$

$w_{21}(new) = w_{21}(old) + x_2 y_1 = 0 + 1 \cdot 0 = 0$

$w_{31}(new) = w_{31}(old) + x_3 y_1 = 1 + 0 \cdot 0 = 1$

$w_{41}(new) = w_{41}(old) + x_4 y_1 = 1 + 0 \cdot 0 = 1$

$w_{12}(new) = w_{12}(old) + x_1 y_2 = 0 + 1 \cdot 1 = 1$

$w_{22}(new) = w_{22}(old) + x_2 y_2 = 0 + 1 \cdot 1 = 1$

$w_{32}(new) = w_{32}(old) + x_3 y_2 = 0 + 0 \cdot 1 = 0$

$w_{42}(new) = w_{42}(old) + x_4 y_2 = 0 + 0 \cdot 1 = 0$

for 4th input vector

$[0, 0, 1, 1] : \{0, 1\}$

$\{x_1=0, x_2=0, x_3=1, x_4=1\}, \{y_1=0, y_2=1\}$

$\omega_{11}(new) = \omega_{11}(old) + x_1 y_1 = 2 + 0 \cdot 0 = 2$

$\omega_{21}(new) = \omega_{21}(old) + x_2 y_1 = 0 + 0 \cdot 0 = 0$

$\omega_{31}(new) = \omega_{31}(old) + x_3 y_1 = 1 + 1 \cdot 0 = 1$

$\omega_{41}(new) = \omega_{41}(old) + x_4 y_1 = 1 + 1 \cdot 0 = 1$

$\omega_{12}(new) = \omega_{12}(old) + x_1 y_2 = 1 + 0 \cdot 1 = 1$

$\omega_{22}(new) = \omega_{22}(old) + x_2 y_2 = 1 + 0 \cdot 1 = 1$

$\omega_{32}(new) = \omega_{32}(old) + x_3 y_2 = 0 + 1 \cdot 1 = 1$

$\omega_{42}(new) = \omega_{42}(old) + x_4 y_2 = 0 + 1 \cdot 1 = 1$

So final weight matrix is

$$\omega = \begin{bmatrix} \omega_{11} & \omega_{12} \\ \omega_{21} & \omega_{22} \\ \omega_{31} & \omega_{32} \\ \omega_{41} & \omega_{42} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$