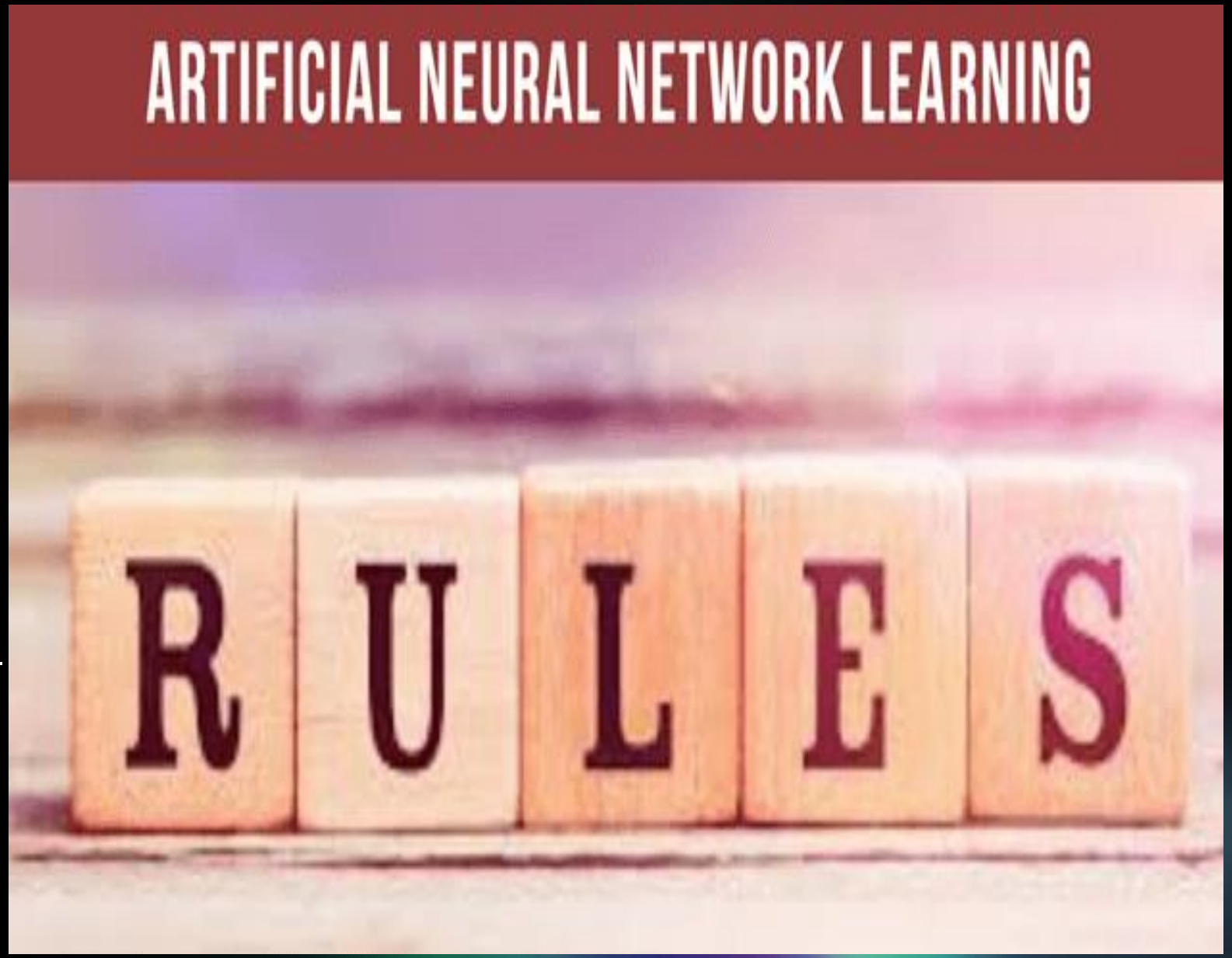
HEBB RULE

PERCEPTRON LEARNING RULE

DELTA RULE



Learning: Updation of weight matrix or updation of weight and biases in ANN is known as learning. It is also known as training.

The purpose of learning rule is to train the network to perform some task or to reduce the error.

Many types of neural network learning rules

1. Supervised
2. Unsupervised
3. Reinforcement

HEB LEARNING RULE

- By Donald Hebb in 1949
- Learning is performed by change in synaptic gaps
- He said " When axon of cell A is near enough to excite cell B, and repeatedly or permanently takes place in firing it, some changes takes place in one or both cells such as A's efficiency as one of the cell firing B, is increased"
- More suited for Bipolar data then binary data
- The Hebb rule can be used for pattern association, pattern categorization, pattern classification and over a range of other areas.
- Unsupervised Learning Rule

HEBB RULE

- Weight vector is found to increase proportionally to the product of input and learning signal, learning signal is equal to neuron's output
- If two interconnected neurons are "on" simultaneously then the weight associated with with these neurons can be increased by the modification made in synaptic gap(strength).This weight update in hebb rule is given by

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

Why more suited for bipolar data

If binary data used above weight updating can't distinguish two conditions

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

.

Algorithm

- Step 0: Initialize all weights
  - For simplicity, set weights and bias to zero
- Step 1: For each input training vector do steps 2-4
- Step 2: Set activations of input units

$$x_i = s_i$$

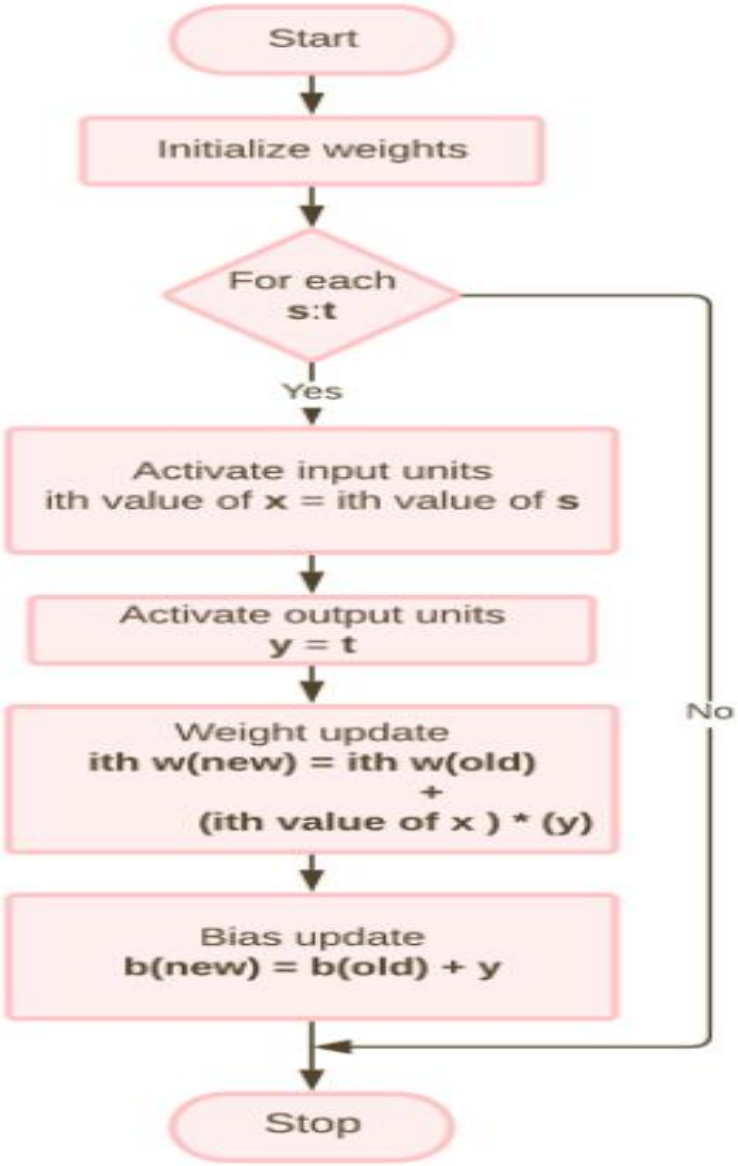
- Step 3: Set the activation for the output unit

$$y = t$$

- Step 4: Adjust weights and bias

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

Flowchart



Hebb Rule example

- Design a Hebb net to implement logical AND function (use bipolar inputs and targets).

Solution: The training data for the AND function is given in Table 9.

Table 9

Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

The network is trained using the Hebb network training algorithm discussed in Section 2.7.3. 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

$$w_i(\text{new}) = w_i(\text{old}) + x_i y$$
$$w_1(\text{new}) = w_1(\text{old}) + x_1 y = 0 + 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2 y = 0 + 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + y = 0 + 1 = 1$$

The weights calculated above are the final weights that are obtained after presenting the first input. These weights are used as the initial weights when the second input pattern is presented. The weight change here is $\Delta w_i = x_i y$. Hence weight changes relating to the first input are

$$\Delta w_1 = x_1 y = 1 \times 1 = 1$$
$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$
$$\Delta b = y = 1$$

- **Second input** $[x_1\ x_2\ b] = [1\ -1\ 1]$ **and** $y = -1$: The initial or old weights here are the final (new) weights obtained by presenting the first input pattern, i.e.,

$$[w_1\ w_2\ b] = [1\ 1\ 1]$$

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

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 1 - 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 1 + 1 = 2$$
$$b(\text{new}) = b(\text{old}) + \Delta b = 1 - 1 = 0$$

Similarly, by presenting the third and fourth input patterns, the new weights can be calculated. Table 10 shows the values of weights for all inputs.

Table 10

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	0	2	0
−1	1	1	−1	1	−1	−1	1	1	−1
−1	−1	1	−1	1	1	−1	2	2	−2

The separating line equation is given by

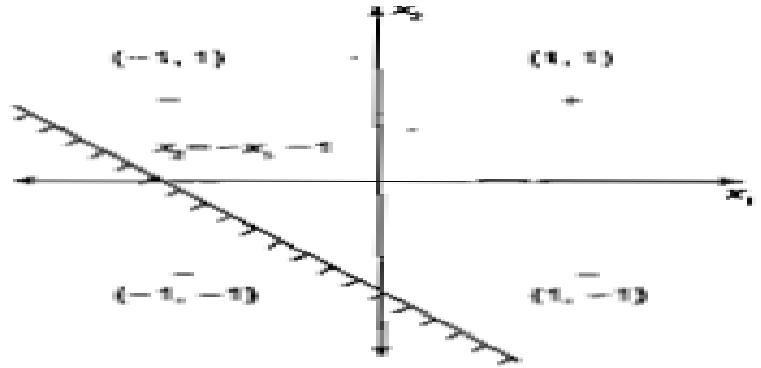
$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}$$

For all inputs, use the final weights obtained for each input to obtain the separating line. For the first input [1 1 1], the separating line is given by

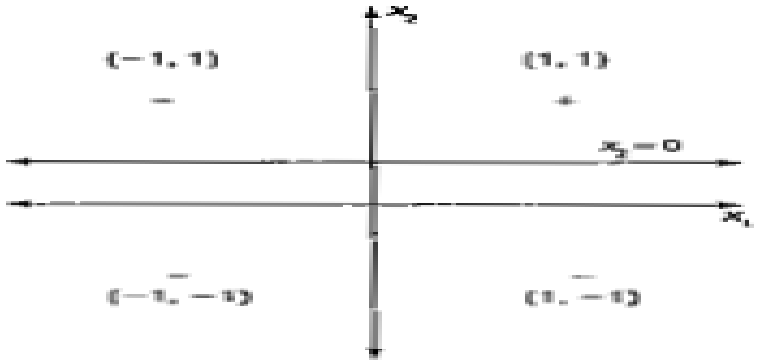
$$x_2 = \frac{-1}{1} x_1 - \frac{1}{1} \Rightarrow x_2 = -x_1 - 1$$

Similarly, for the second input [1 −1 1], the separating line is

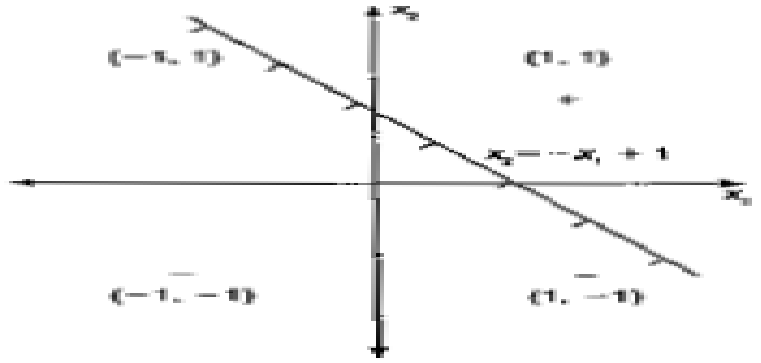
$$x_2 = \frac{-0}{2} x_1 - \frac{0}{2} \Rightarrow x_2 = 0$$



(A) First input



(B) Second input



(C) Third and fourth inputs

Figure 11 Decision boundary for AND function using Hebb rule for each training pair.

For the third input [−1 1 1], it is

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

Finally, for the fourth input [−1 −1 1], the separating line is

$$x_2 = \frac{-2}{2}x_1 + \frac{2}{2} \Rightarrow x_2 = -x_1 + 1$$

The graphs for each of these separating lines obtained are shown in Figure 11. In this figure "+" mark is used for output "1" and "−" mark is used for output "−1." From Figure 11, it can be noticed that for the first input, the decision boundary differentiates only the first and fourth inputs, and not all negative responses are separated from positive responses. When the second input pattern is presented, the decision boundary separates (1, 1) from (1, −1) and (−1, −1) and not (−1, 1). But the boundary line is same for the both third and fourth training pairs. And, the decision boundary line obtained from these input training pairs separates the positive response region from the negative response region. Hence, the weights obtained from this are the final weights and are given as

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

The network can be represented as shown in Figure 12.

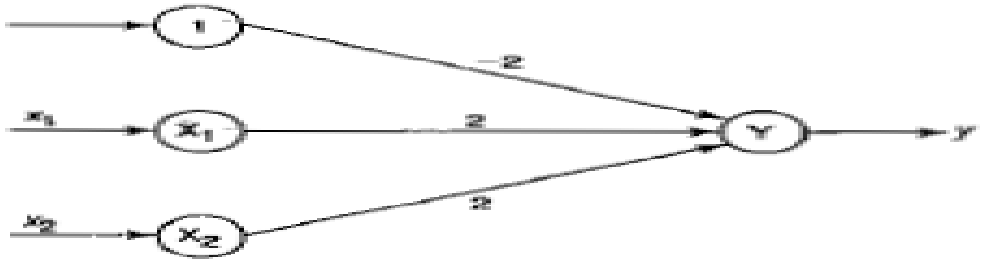


Figure 12 Hebb net for AND function.

Implement OR function using Hebb rule(consider bipolar input and targets

Solution: The training pair for the OR function is given in Table 11.

Table 11

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.,

$$w_1 = w_2 = b = 0$$

The network is trained and the final weights are outlined using the Hebb training algorithm discussed in Section 2.7.3. The weights are considered as final weights if the boundary line obtained from these weights separates the positive response region and negative response region.

By presenting all the input patterns, the weights are calculated. Table 12 shows the weights calculated for all the inputs.

Table 12

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 separating line equation is

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2} = \frac{-2}{2} x_1 - \frac{2}{2} = -x_1 - 1$$

The decision region for this net is shown in Figure 13. It is observed in Figure 13 that straight line $x_2 = -x_1 - 1$ separates the pattern space into two regions. The input patterns [(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

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

The network can be represented as shown in Figure 14.

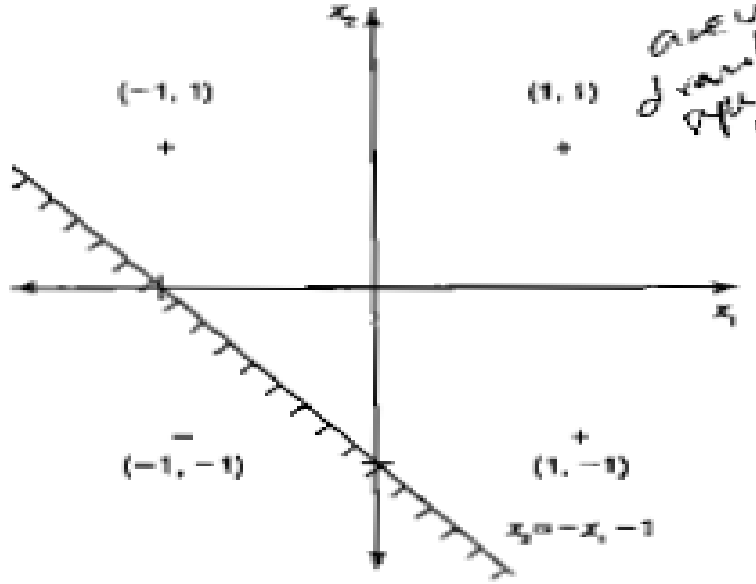


Figure 13 Decision boundary for OR function.

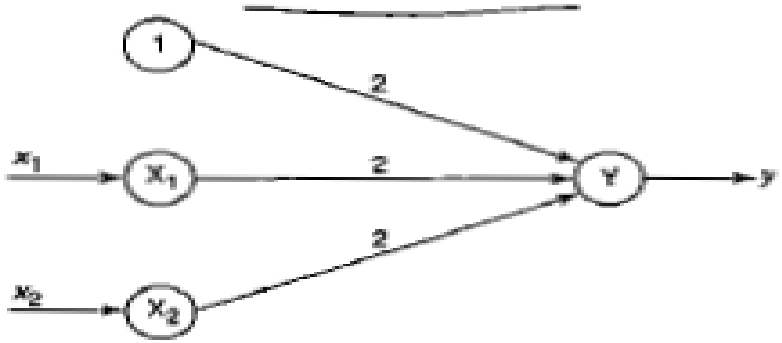


Figure 14 Hebb net for OR function.

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

Solution: The training patterns for an XOR function are shown in Table 13.

Table 13

Inputs			Target
x_1	x_2	b	y
1	1	1	−1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

Here, a single-layer network with two input neurons, one bias and one output neuron is considered. In this case also, the initial weights are assumed to be zero:

$$w_1 = w_2 = b = 0$$

By using the Hebb training algorithm, the network is trained and the final weights are calculated as shown in the following Table 14.

Table 14

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	0	−2	0
−1	1	1	1	−1	1	1	−1	−1	1
−1	−1	1	−1	1	1	−1	0	0	0

The final weights obtained after presenting all the input patterns do not give correct output for all patterns. Figure 15 shows that the input patterns are linearly non-separable. The graph shown in Figure 15 indicates that the four input pairs that are present cannot be divided by a single line to separate them into two regions. Thus XOR function is a case of a pattern classification problem, which is not linearly separable.

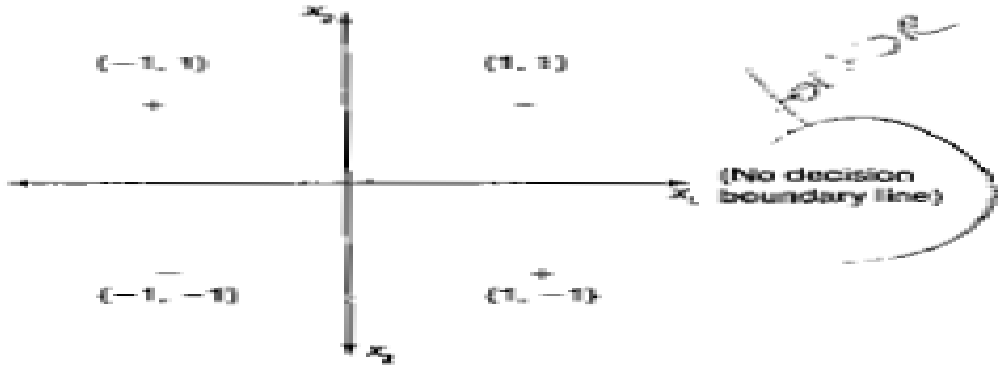


Figure 15 Decision boundary for XOR function.

11. Using the Hebb rule, find the weights required to perform the following classifications of the given input patterns shown in Figure 16. The pattern is shown as 3×3 matrix form in the squares. The "+" symbols represent the value "1" and empty squares indicate "−1." Consider "I" belongs to the members of class (so has target value 1) and "O" does not belong to the members of class (so has target value −1).

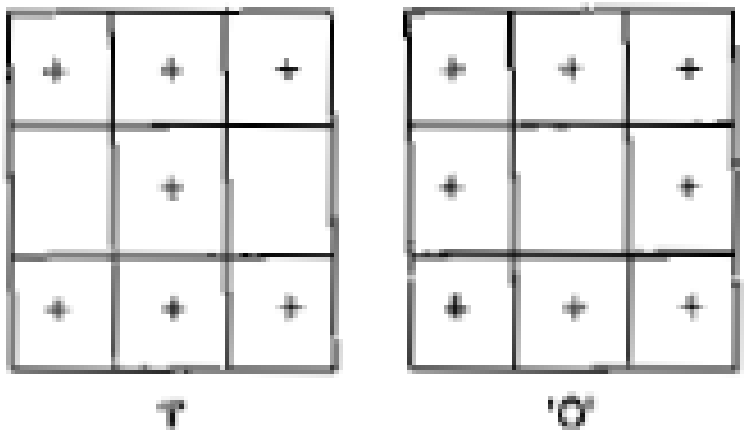


Figure 16 Data for input patterns.

Solution: The training input patterns for the given net (Figure 16) are indicated in Table 15.

Table 15

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

Here a single-layer network with nine input neurons, one bias and one output neuron is formed. Set the initial weights and bias to zero, i.e.,

$$w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = w_7 = w_8 = w_9 = b = 0$$

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

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

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

$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$
$$\Delta w_3 = x_3 y = 1 \times 1 = 1$$
$$\Delta w_4 = x_4 y = -1 \times 1 = -1$$
$$\Delta w_5 = x_5 y = 1 \times 1 = 1$$
$$\Delta w_6 = x_6 y = -1 \times 1 = -1$$
$$\Delta w_7 = x_7 y = 1 \times 1 = 1$$
$$\Delta w_8 = x_8 y = 1 \times 1 = 1$$
$$\Delta w_9 = x_9 y = 1 \times 1 = 1$$
$$\Delta b = y = 1$$

We now calculate the new weights using the formula

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

Setting the old weights as the initial weights here, we obtain

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0 + 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0 + 1 = 1$$
$$w_3(\text{new}) = w_3(\text{old}) + \Delta w_3 = 0 + 1 = 1$$

Similarly, calculating for other weights we get

$$w_4(\text{new}) = -1, \quad w_5(\text{new}) = 1, \quad w_6(\text{new}) = -1,$$
$$w_7(\text{new}) = 1, \quad w_8(\text{new}) = 1, \quad w_9(\text{new}) = 1,$$
$$b(\text{new}) = 1$$

The weights after presenting first input pattern are

$$W_{(\text{new})} = [1\ 1\ 1\ -1\ 1\ -1\ 1\ 1\ 1\ 1]$$

Case 2: Now we present the second input pattern (O). The initial weights used here are the final weights obtained after presenting the first input pattern. Here, the weights are calculated as shown below ($y = -1$ with the initial weights being [1 1 1 −1 1 −1 1 1 1 1]).

$$w_i(\text{new}) = w_i(\text{old}) + \Delta x_i \qquad [\Delta w_i = x_i y]$$
$$w_1(\text{new}) = w_1(\text{old}) + x_1 y = 1 + 1 \times -1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2 y = 1 + 1 \times -1 = 0$$
$$w_3(\text{new}) = w_3(\text{old}) + x_3 y = 1 + 1 \times -1 = 0$$
$$w_4(\text{new}) = w_4(\text{old}) + x_4 y = -1 + 1 \times -1 = -2$$
$$w_5(\text{new}) = w_5(\text{old}) + x_5 y = 1 + -1 \times -1 = 2$$
$$w_6(\text{new}) = w_6(\text{old}) + x_6 y = -1 + 1 \times -1 = -2$$
$$w_7(\text{new}) = w_7(\text{old}) + x_7 y = 1 + 1 \times -1 = 0$$
$$w_8(\text{new}) = w_8(\text{old}) + x_8 y = 1 + 1 \times -1 = 0$$
$$w_9(\text{new}) = w_9(\text{old}) + x_9 y = 1 + 1 \times -1 = 0$$
$$b(\text{new}) = b(\text{old}) + y = 1 + 1 \times -1 = 0$$

The final weights after presenting the second input pattern are given as

$$W_{(\text{new})} = [0\ 0\ 0\ -2\ 2\ -2\ 0\ 0\ 0\ 0]$$

The weights obtained are indicated in the Hebb net shown in Figure 17.

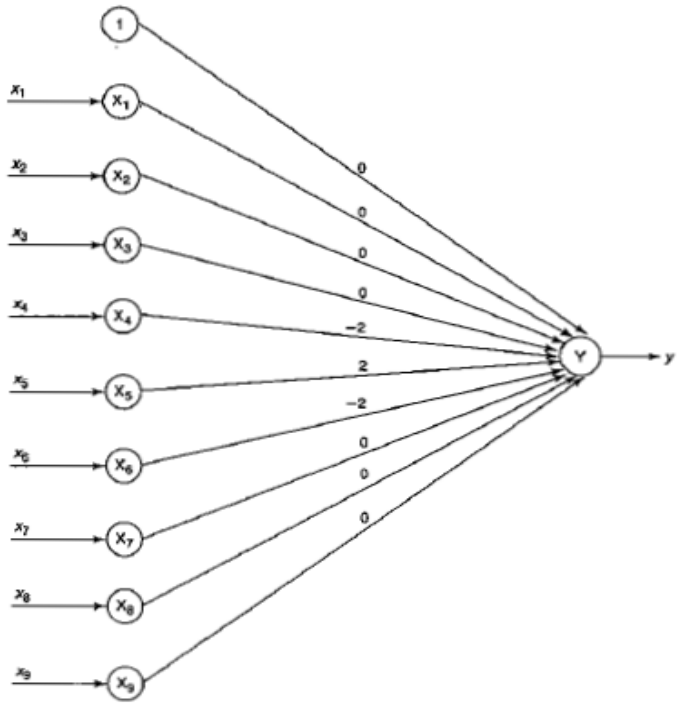


Figure 17 Hebb net for the data matrix shown in Figure 16.

12. Find the weights required to perform the following classifications of given input patterns using the Hebb rule. The inputs are "1" where "+" symbol is present and "−1" where "." is present. "L" pattern belongs to the class (target value +1) and "U" pattern does not belong to the class (target value −1).

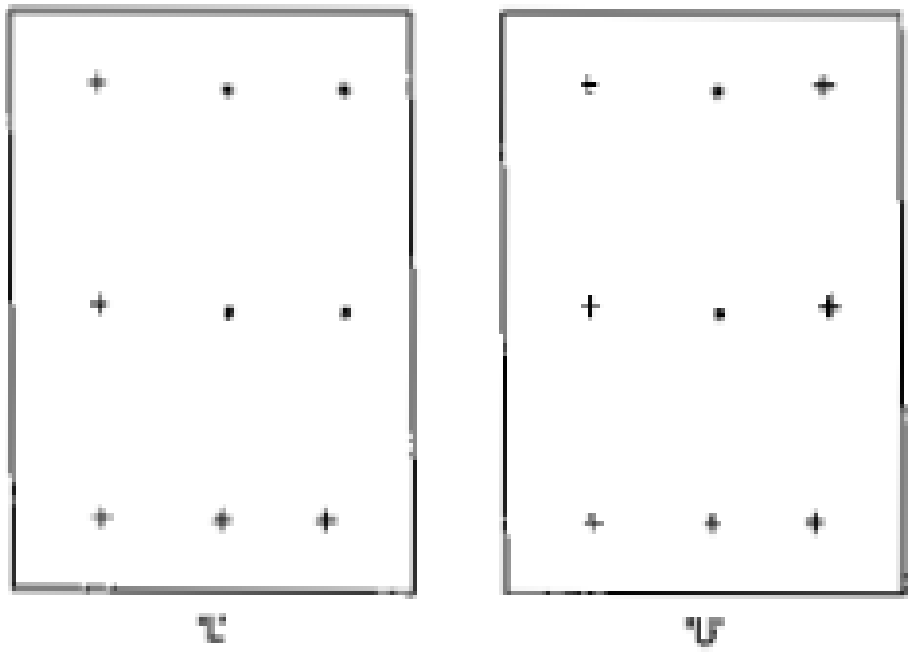


Figure 18 Input data for given patterns.

Solution: The training input patterns for Figure 18 are given in Table 16.

Table 16

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

A single-layer network with nine input neurons, one bias and one output neuron is formed. Set the initial weights and bias to zero, i.e.,

$$w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = w_7 = w_8 = w_9 = b = 0$$

The weights are calculated using

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

The calculated weights are given in Table 17.

Table 17

Inputs										Target	Weights									
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	b
											(0	0	0	0	0	0	0	0	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	1	1	−1	1	1	1	1	1	−1	0	0	−2	0	0	−2	0	0	0	0

The final weights after presenting the two input patterns are

$$W_{(new)} = [0\ 0\ -2\ 0\ 0\ -2\ 0\ 0\ 0\ 0]$$

The obtained weights are indicated in the Hebb net shown in Figure 19.

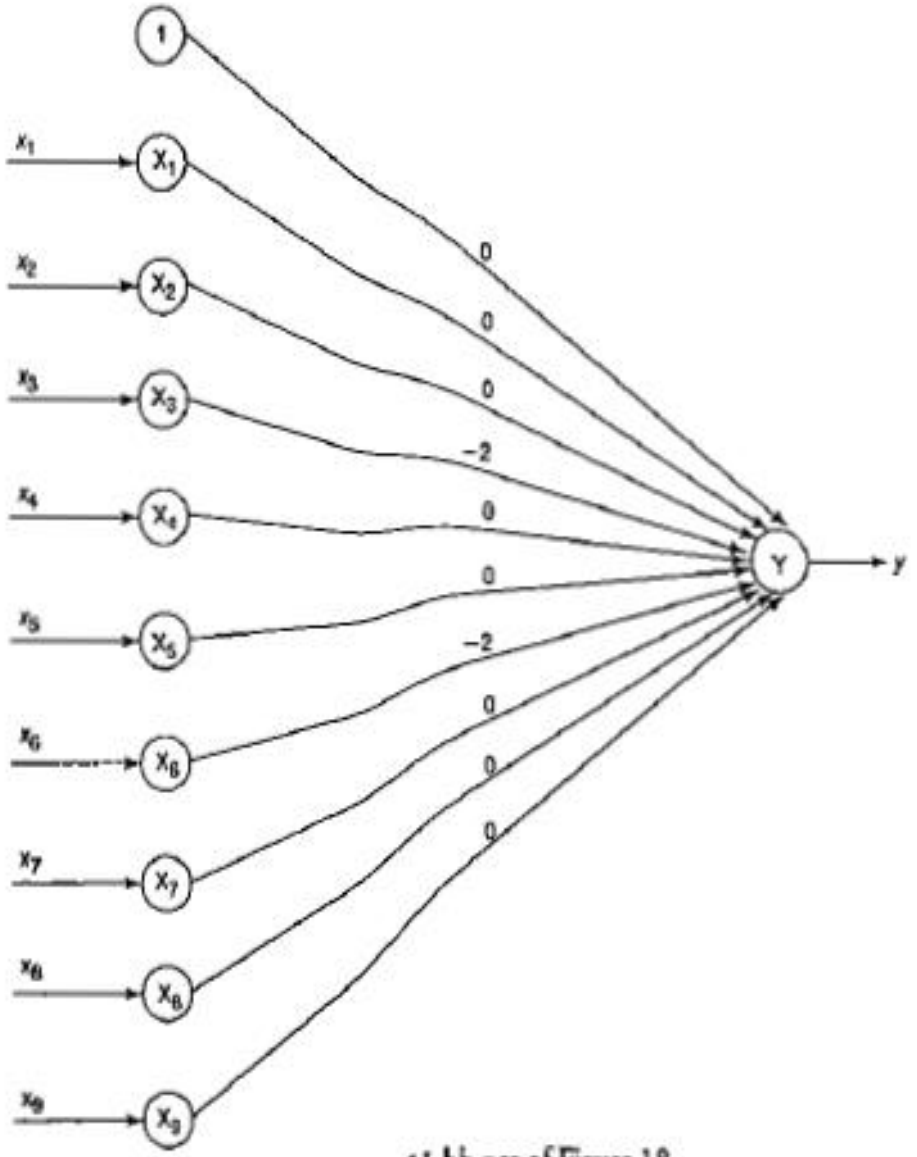


Figure 19 Hebb net of Figure 18.

Perceptron Learning Rule

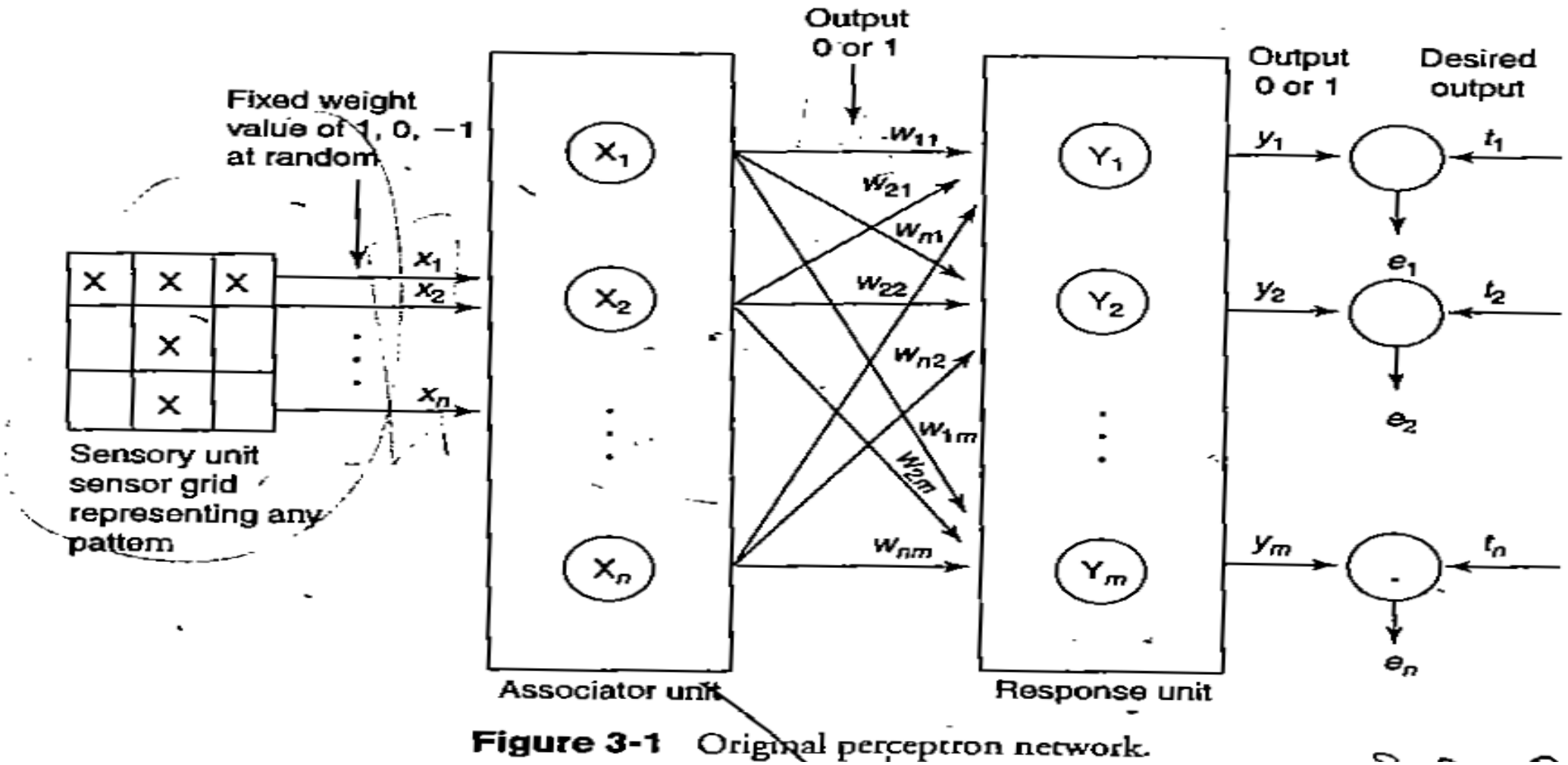


Figure 3-1 Original perceptron network.

PERCEPTRON LEARNING RULE

Learning signal is difference between the desired and actual response of neuron

Output is calculated as

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

Weight updation is done as

If $y \neq t$, then

$$w(\text{new}) = w(\text{old}) + \alpha t x \quad (\alpha - \text{learning rate})$$

else, we have

$$w(\text{new}) = w(\text{old})$$

Algorithm

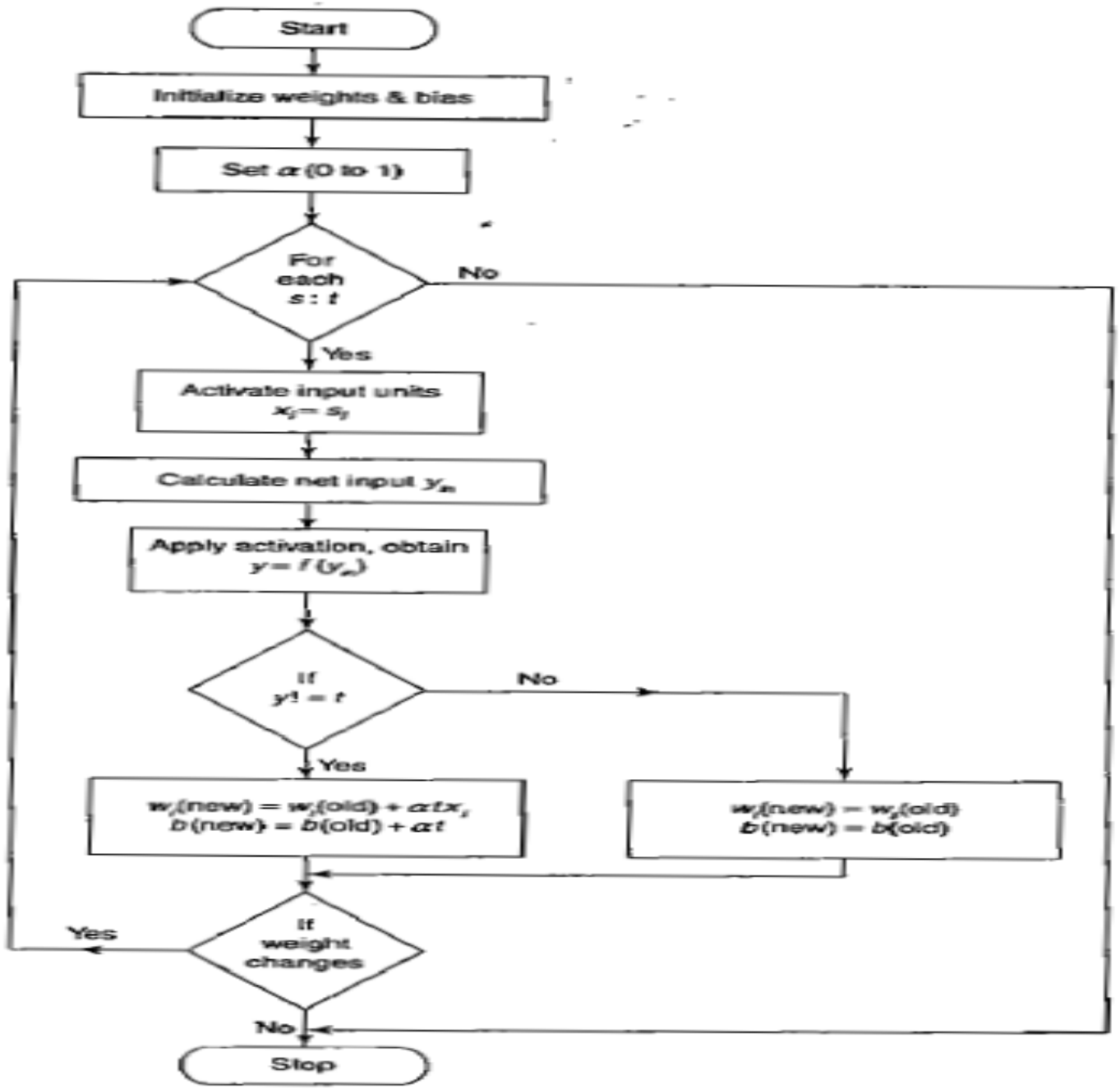


Figure 3-3 Flowchart for perceptron network with single output.

Perceptron Training algo for single output class

Step 0: Initialize the weights and the bias (for easy calculation they can be set to zero). Also initialize the learning rate $\alpha (0 < \alpha \leq 1)$. For simplicity α is set to 1.

Step 1: Perform Steps 2–6 until the final stopping condition is false.

Step 2: Perform Steps 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. To do so, first obtain the net input:

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

where "n" is the number of input neurons in the input layer. Then apply activations over the net input calculated to obtain the output:

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

Step 5: *Weight and bias adjustment:* Compare the value of the actual (calculated) output and desired (target) output.

If $y \neq t$, then

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

else, we have

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

Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again from Step 2.

The algorithm discussed above is not sensitive to the initial values of the weights or the value of the learning rate.

Perceptron Training algo for multiple output class

- The only change is in bias values

Calculate output response of each output unit $j = 1$ to m: First, the net input is calculated as

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

Then activations are applied over the net input to calculate the output response:

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

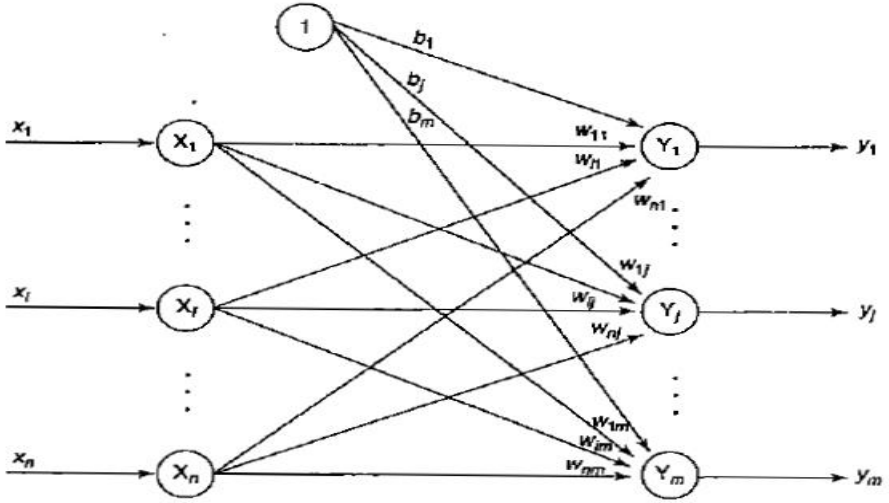


Figure 3-4 Network architecture for perceptron network for several output classes.

Implement AND function using perceptron networks for bipolar input and targets

Solution: Table 1 shows the truth table for AND function with bipolar inputs and targets:

Table 1

x_1	x_2	t
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1

The perceptron network, which uses perceptron learning rule, is used to train the AND function. The network architecture is as shown in Figure 1. The input patterns are presented to the network one by one. When all the four input patterns are presented, then one epoch is said to be completed. The initial weights and threshold are set to zero, i.e., $w_1 = w_2 = b = 0$ and $\theta = 0$. The learning rate α is set equal to 1.

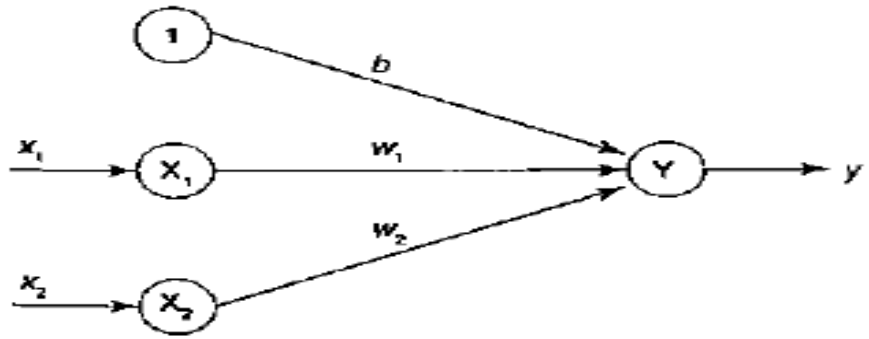


Figure 1 Perceptron network for AND function.

For the first input pattern, $x_1 = 1, x_2 = 1$ and $t = 1$, with weights and bias, $w_1 = 0$, $w_2 = 0$ and $b = 0$:

- Calculate the net input

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= 0 + 1 \times 0 + 1 \times 0 = 0$$

- The output y is computed by applying activations over the net input calculated:

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

Here we have taken $\theta = 0$. Hence, when, $y_{in} = 0$, $y = 0$.

- Check whether $t = y$. Here, $t = 1$ and $y = 0$, so $t \neq y$, hence weight updation takes place:

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times 1 = 1$$

Here, the change in weights are

$$\Delta w_1 = \alpha t x_1;$$
$$\Delta w_2 = \alpha t x_2;$$
$$\Delta b = \alpha t$$

The weights $w_1 = 1$, $w_2 = 1$, $b = 1$ are the final weights after first input pattern is presented. The same process is repeated for all the input patterns. The process can be stopped when all the targets become equal to the calculated output or when a separating line is obtained using the final weights for separating the positive responses from negative responses. Table 2 shows the training of perceptron network until its

Table 2

Input x_1	Input x_2	Input 1	Target (t)	Net input (y_{in})	Calculated output (y)	Weight changes Δw_1	Weight changes Δw_2	Weight changes Δb	Weights w_1	Weights w_2	Weights b
									(0	0	0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	−1	1	−1	1	1	−1	1	−1	0	2	0
−1	1	1	−1	2	1	+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	1	0	0	0	1	1	−1
1	−1	1	−1	−1	−1	0	0	0	1	1	−1
−1	1	1	−1	−1	−1	0	0	0	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1

target and calculated output converge for all the patterns.

The final weights and bias after second epoch are

$$w_1 = 1, w_2 = 1, b = -1$$

Since the threshold for the problem is zero, the equation of the separating line is

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

Here

$$w_1x_1 + w_2x_2 + b > \theta$$
$$w_1x_1 + w_2x_2 + b > 0$$

Thus, using the final weights we obtain

$$x_2 = -\frac{1}{1}x_1 - \frac{(-1)}{1}$$
$$x_2 = -x_1 + 1$$

It can be easily found that the above straight line separates the positive response and negative response region, as shown in Figure 2.

The same methodology can be applied for implementing other logic functions such as OR, AND-NOT, NAND, etc. If there exists a threshold value $\theta \neq 0$, then two separating lines have to be obtained, i.e., one to separate positive response from zero and the other for separating zero from the negative response.

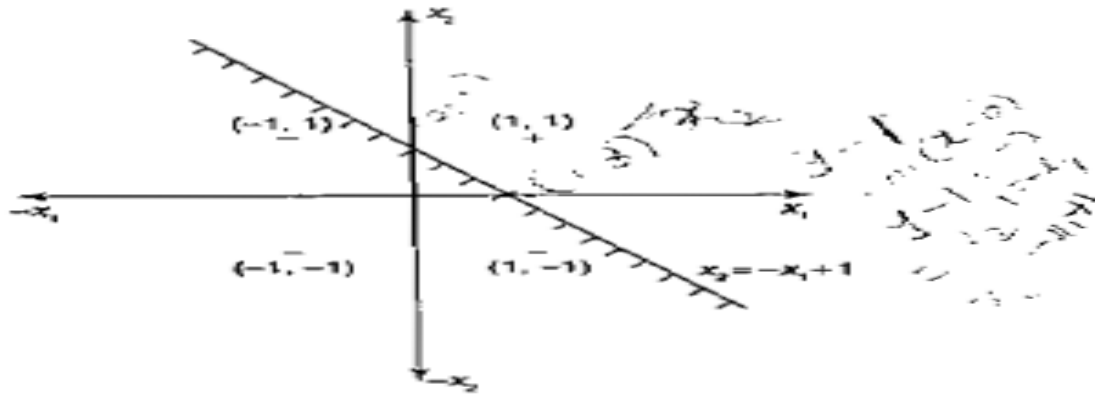


Figure 2 Decision boundary for AND function in perceptron training ($\theta = 0$).

2. Implement OR function with binary inputs and bipolar targets using perceptron training algorithm upto 3 epochs.

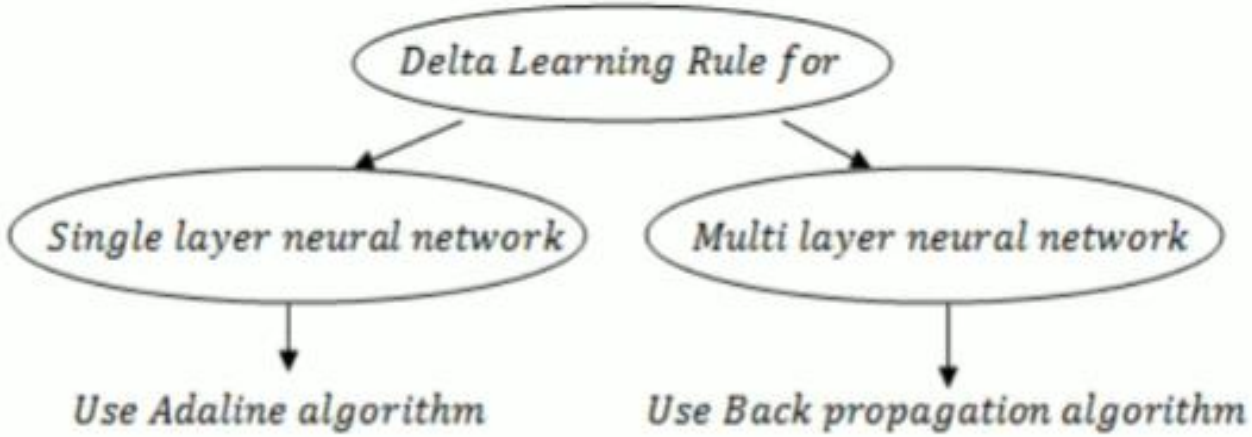
Solution: The truth table for OR function with binary inputs and bipolar targets is shown in Table 3.

Table 3

x_1	x_2	t
1	1	1
1	0	1
0	1	1
0	0	−1

DELTA RULE

- Developed by Widrow and Hoff.
- Modification in weight of node is equal to the multiplication of error and the input,
- Error is difference between target and actual output value
- Also called LMS(Least Mean Square) Learning rule, Widrow and Hoff rule and continuous perceptron training rule.
- Delta rule uses gradient descent for updating the weights to minimize the error from perceptron.
- The major aim is to minimize the error(mean squared error) over all training patterns. This is done by reducing the error for each pattern, one at a time.
- It is independent of activation function
- Supervised learning rule



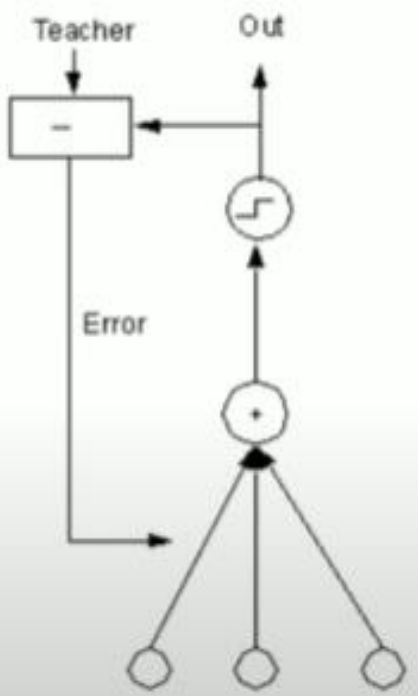
Single layer neural network trained with Delta learning is known as Adaptive Linear, (ADALINE) neural network.

See the differance to send back error to network, after and before activation function.

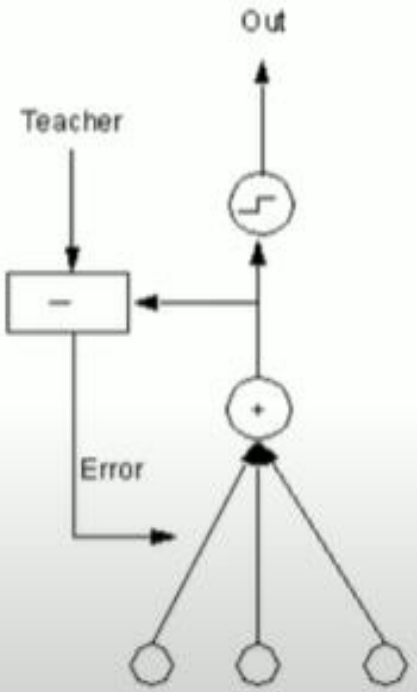
$$Wi(new) = Wi(old) + \alpha(Target\ output - Yin)xi$$

$$Net\ input, Yin = \Sigma wixi$$

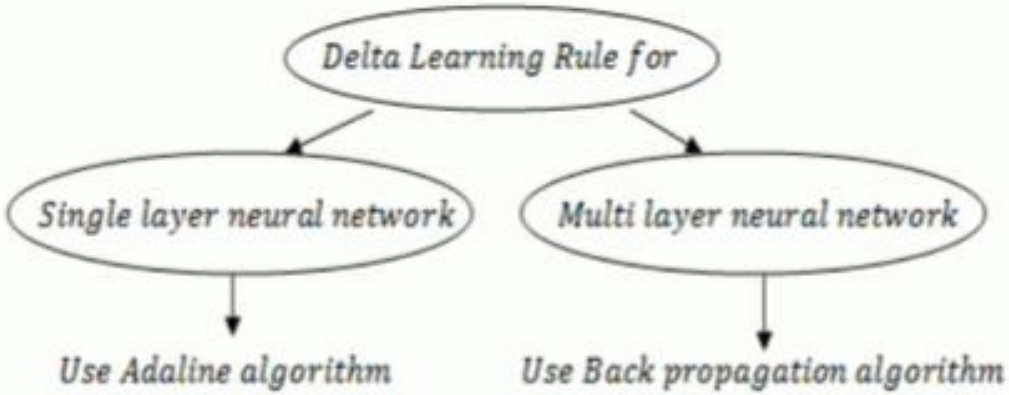
$$b(new) = b(old) + \alpha(Target\ output - Yin)$$



Perceptron Learning



Delta Rule



Multi layer neural network trained with Delta learning is known as Back propagation algorithm or Back propagtion neural network.

$$Wi(new) = Wi(old) + \Delta wij$$

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment — learning rate — weight gradient

$\frac{\partial E}{\partial wi}$ = *Partial derivative of total error with respect to wi*

OR

$$Wi(new) = Wi(old) + \alpha(t - y)f'(net)xi$$

Here, α = *Learning rate,* t = *target output,* y = *actual output,* $f'(net)$ = *Derivative of activation function,* xi = *input.*

Ex- Implement logic OR operation using Delta Learning Rule. Use bipolar inputs and target. Consider learning rate, α=0.1, Perform 2 Epoch for Network training

Truth table of OR gate		
Input		Output
X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

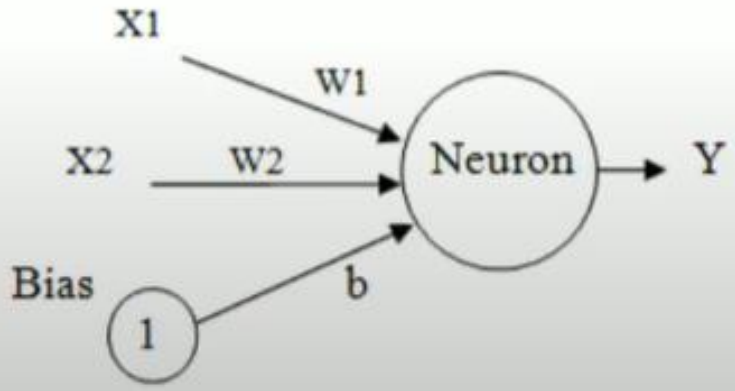
Solution-

Binary input and output represented as = 0 & 1

Bipolar input and output represented as = -1 & 1

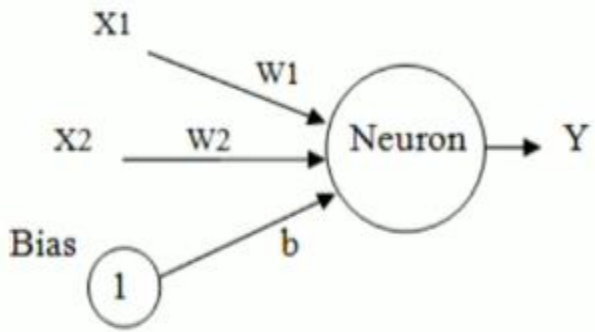
So replace '0' by '-1' in truth table of OR gate.

Bipolar Input			Target Output
X1	X2	Bias, b	t
1	1	1	1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1



Single layer Perceptron network for logic OR function

Learning Of Neural Network



Input			Target O/p	Net Input	t-Yin	Change in weights and bias			New weights and bias			Error $(t-Yin)^2$
X1	X2	1	t	Yin		ΔW1	ΔW2	Δb	W1	W2	b	
EPOCH=1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49

Step 1

Initialize weights and bias,

Let us take, $w1 = w2 = b = 0.1$

Learning rate given, $\alpha = 0.1$

Step 2

From truth table, take first input pattern, $x1 = 1, \quad x2 = 1$ *and* $t = 1$

$$Net\ Input = Yin = w1x1 + w2x2 + b$$

$$Yin = 0.1 * 1 + 0.1 * 1 + 0.1 = 0.3$$

$$t - Yin = 1 - 0.3 = 0.7$$

$$Error, E = (t - Yin)^2 = 0.49$$

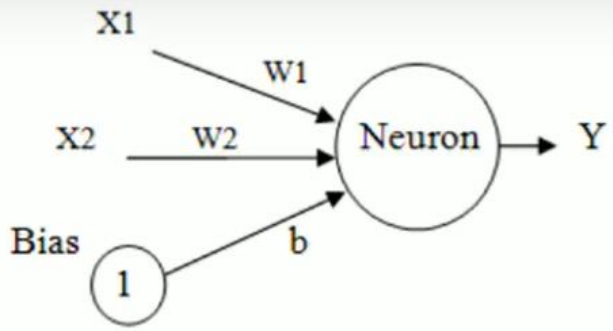
Change weight and bias as per Delta learning rule

$$\boldsymbol{Wi(new) = Wi(old) + \alpha(Target\ output - Net\ input)xi}$$

$$wi(new) = wi(old) + \alpha(t - Yin)xi$$

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

$$\Delta wi = \alpha(t - Yin)xi$$



Input			Target O/p	Net Input	t-Yin	Change in weights and bias			New weights and bias			Error $(t-Yin)^2$
X1	X2	1	t	Yin		ΔW1	ΔW2	Δb	W1	W2	b	
EPOCH=1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49

$$bi(new) = bi(old) + \alpha(Target\ output - Net\ input)$$

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

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

$$\Delta b = \alpha(t - Yin)$$

$$\Delta w1 = \alpha(t - Yin)x1 = 0.1(1 - 0.3) * 1 = 0.07$$

$$\Delta w2 = \alpha(t - Yin)x2 = 0.1(1 - 0.3) * 1 = 0.07$$

$$\Delta b = \alpha(t - Yin) = 0.1(1 - 0.3) = 0.07$$

$$w1(new) = w1(old) + \Delta w1$$

$$w1(new) = w1(old) + \alpha(t - Yin)x1 = 0.1 + 0.07 = 0.17$$

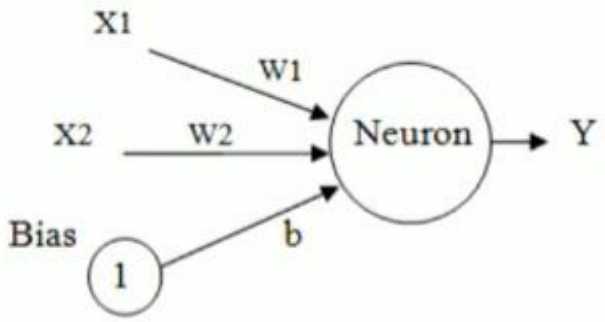
$$w2(new) = w2(old) + \Delta w2$$

$$w2(new) = w2(old) + \alpha(t - Yin)x2 = 0.1 + 0.07 = 0.17$$

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

$$b(new) = b(old) + \alpha(t - Yin) = 0.1 + 0.07 = 0.17$$

Learning Of Neural Network



Input			Target O/p	Net Input	t-Yin	Change in weights and bias			New weights and bias			Error $(t-Yin)^2$
X1	X2	1	t	Yin		ΔW1	ΔW2	Δb	W1	W2	b	
EPOCH=1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	1	0.17	0.83	0.083	-0.083	0.083	*0.253*	*0.087*	*0.253*	0.69

Step 3

From truth table, take Second input pattern,

$x1 = 1, \quad x2 = -1$ *and* $t = 1$

Old weights used here are final weights obtained after the first input pattern

$w(old) = [0.17,\ 0.17]$ And $b(old) = 0.17$

Net input $= Yin = w1x1 + w2x2 + b$

$$Yin = 0.17 * 1 + 0.17 * (-1) + 0.17 = 0.17$$

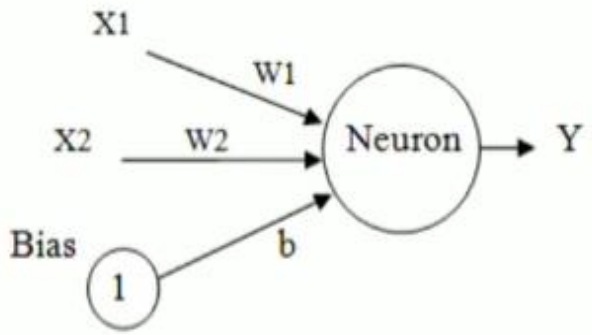
$$t - Yin = 1 - 0.17 = 0.83$$

$$Error, E = (t - Yin)^2 = 0.69$$

$$\Delta w1 = \alpha(t - Yin)x1 = 0.1(1 - 0.17) * 1 = 0.083$$

$$\Delta w2 = \alpha(t - Yin)x2 = 0.1(1 - 0.17) * (-1) = -0.083$$

$$\Delta b = \alpha(t - Yin) = 0.1(1 - 0.17) = 0.083$$



Input			Target O/p	Net Input	t-Yin	Change in weights and bias			New weights and bias			Error $(t-Yin)^2$
X1	X2	1	t	Yin		ΔW1	ΔW2	Δb	W1	W2	b	
EPOCH=1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	1	0.17	0.83	0.083	-0.083	0.083	0.253	0.087	0.253	0.69
-1	1	1	1	0.087	0.913	-.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83

Step 4

From truth table, take third pattern,

$x1 = -1, \quad x2 = 1$ *and* $t = 1$

Old weights used here are final weights obtained after the Second input pattern

$w(old) = [0.253, 0.087]$ And $b(old) = 0.253$

Net input $= Yin = w1x1 + w2x2 + b$

$$Yin = 0.253 * (-1) + 0.087 * 1 + 0.253 = 0.087$$

$$t - Yin = 1 - 0.087 = 0.913$$

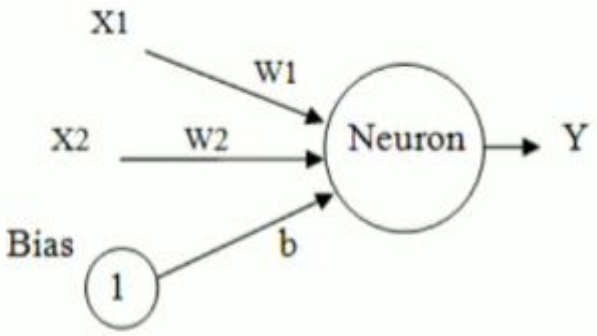
$$Error, E = (t - Yin)^2 = 0.83$$

Change weight and bias as per Delta learning rule

$$\Delta w1 = \alpha(t - Yin)x1 = 0.1(1 - 0.087) * (-1) = -0.0913$$

$$\Delta w2 = \alpha(t - Yin)x2 = 0.1(1 - 0.087) * (1) = 0.0913$$

$$\Delta b = \alpha(t - Yin) = 0.1(1 - 0.087) = 0.0913$$



Input			Target O/p	Net Input	t-Yin	Change in weights and bias			New weights and bias			Error $(t-Yin)^2$
X1	X2	1	t	Yin		ΔW1	ΔW2	Δb	W1	W2	b	
EPOCH=1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	1	0.17	0.83	0.083	-0.083	0.083	*0.253*	*0.087*	*0.253*	0.69
-1	1	1	1	0.087	0.913	-.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
-1	-1	1	-1	0.0043	-1.0043	0.1004	0.1004	-.1004	*0.2621*	*0.2787*	*0.2439*	*1.01*

Step 5

From truth table, take fourth input pattern,

$x1 = -1, \quad x2 = -1$ *and* $t = -1$

Old weights used here are final weights obtained after the Third input pattern

$w(old) = [0.1617, 0.1783]$ And $b(old) = 0.3443$

$$Net\ input = Yin = w1x1 + w2x2 + b$$

$$Yin = 0.1617 * (-1) + 0.1783 * (-1) + 0.3443 = 0.0043$$

$$t - Yin = -1 - 0.0043 = -1.0043$$

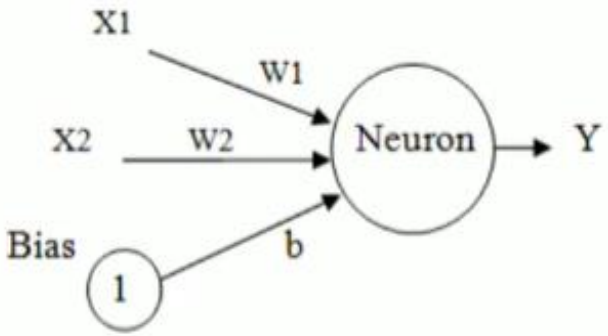
$$Error, E = (t - Yin)^2 = 1.0086$$

Change weight and bias as per Delta learning rule

$$\Delta w1 = \alpha(t - Yin)x1 = 0.1(-1 - 0.0043) * (-1) = 0.1004$$

$$\Delta w2 = \alpha(t - Yin)x2 = 0.1(-1 - 0.0043) * (-1) = 0.1004$$

$$\Delta b = \alpha(t - Yin) = 0.1(-1 - 0.0043) = -0.1004$$



Input			Target O/p	Net Input	t-Yin	Change in weights and bias			New weights and bias			Error $(t-Yin)^2$
X1	X2	1	t	Yin		ΔW1	ΔW2	Δb	W1	W2	b	
EPOCH=1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	1	0.17	0.83	0.083	-0.083	0.083	*0.253*	*0.087*	*0.253*	0.69
-1	1	1	1	0.087	0.913	-.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
-1	-1	1	-1	0.0043	-1.0043	0.1004	0.1004	-.1004	*0.2621*	*0.2787*	*0.2439*	*1.01*

$$w1(new) = w1(old) + \Delta w1$$

$$w1(new) = w1(old) + \alpha(t - Yin)x1 = 0.1617 + 0.1004 = 0.2621$$

$$w2(new) = w2(old) + \Delta w2$$

$$w2(new) = w2(old) + \alpha(t - Yin)x2 = 0.1783 + 0.1004 = 0.2787$$

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

$$b(new) = b(old) + \alpha(t - Yin) = 0.3443 - 0.1004 = 0.2439$$

$$So\ for\ first\ Epoch, Mean\ square\ error = \frac{\sum(t - Yin)^2}{4}$$

$$Mean\ square\ error = \frac{(0.49) + (0.69) + (0.83) + (1.0086)}{4} = 0.75465$$

Learning Of Neural Network

Input			Target O/p	Net Input	t-Yin	Change in weights and bias			New weights and bias			Error $(t-Yin)^2$
X1	X2	1	t	Yin		ΔW1	ΔW2	Δb	W1	W2	b	
EPOCH=1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	1	0.17	0.83	0.083	-0.083	0.083	*0.253*	*0.087*	*0.253*	0.69
-1	1	1	1	0.087	0.913	-.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
-1	-1	1	-1	0.0043	-1.0043	0.1004	0.1004	-.1004	*0.2621*	*0.2787*	*0.2439*	*1.01*
EPOCH=2												
1	1	1	1	0.7847	0.2153	0.0215	0.0215	0.0215	0.2837	0.3003	0.2654	0.046
1	-1	1	1	0.2488	0.7512	0.7512	-0.0751	0.0751	0.3588	0.2251	0.3405	0.564
-1	1	1	1	0.2069	0.7931	-0.793	0.0793	0.0793	*0.2795*	*0.3044*	*0.4198*	*0.629*
-1	-1	1	-1	-0.1641	-0.8359	0.0836	0.0836	-0.083	0.3631	0.388	0.366	0.699

Step 6 *Repeat the same procedure for Epoch II.*

$$\textit{Mean square error for epoch II} = \frac{(0.046) + (0.564) + (0.629) + (0.699)}{4}$$

$$= 0.4845$$

$$Mean\ square\ error\ for\ epoch\ I = \frac{(0.49) + (0.69) + (0.83) + (1.0086)}{4} = 0.75465$$

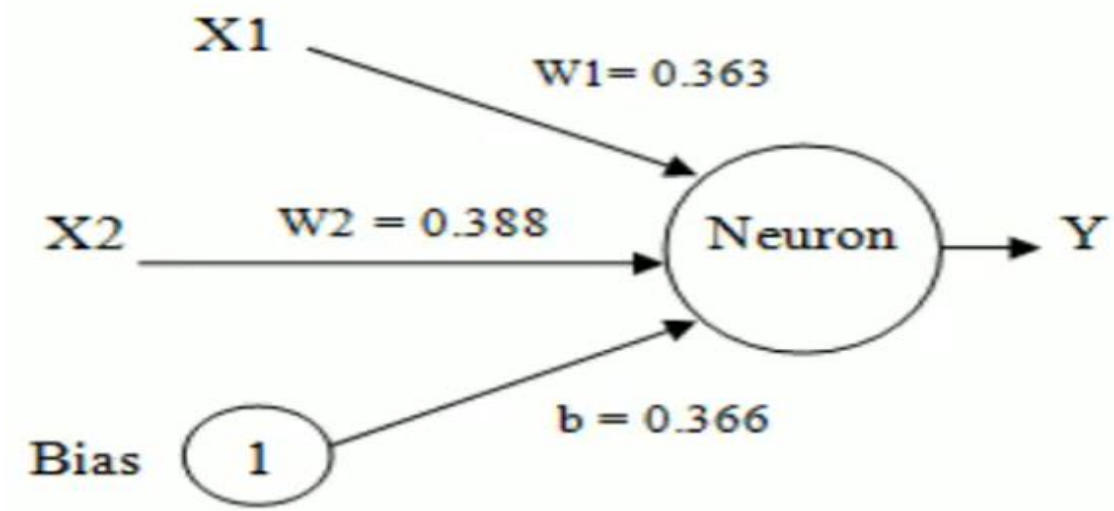
$$Mean\ square\ error\ for\ epoch\ II = \frac{(0.046) + (0.564) + (0.629) + (0.699)}{4} = 0.4845$$

In this way, repeat the process to get minimum Mean square error

Last updated weights and bias are final weights and bias of neural network.

The process can be stopped when ***all target outputs*** *become equal to the actual outputs*

OR when a separating line is obtained using the final weights for separating the positive response from negative response.



Classification Of Supervised Learning Algorithms

1. Gradient Descent
2. Stochastic

#1) Gradient Descent Learning

In this type of learning, the error reduction takes place with the help of weights and the activation function of the network. The activation function should be differentiable.

The adjustment of weights depends on the error gradient E in this learning. The backpropagation rule is an example of this type of learning. Thus the weight adjustment is defined as

$\Delta W = \dot{\eta} (\partial E / \partial W)$, $\dot{\eta}$ is learning rate parameter and $(\partial E / \partial W)$ is error gradient with respect to W.

Classification Of Unsupervised Learning Algorithms

1. Hebbian
2. Competitive

#1) Hebbian Learning

This learning was proposed by Hebb in 1949. It is based on correlative adjustment of weights. The input and output patterns pairs are associated with a weight matrix, W.

$$W = \sum X * Y^T$$

Where X represent input and Y output.

The transpose of the output is taken for weight adjustment.

#2) Competitive Learning

It is a winner takes all strategy. In this type of learning, when an input pattern is sent to the network, all the neurons in the layer compete and only the winning neurons have weight adjustments.