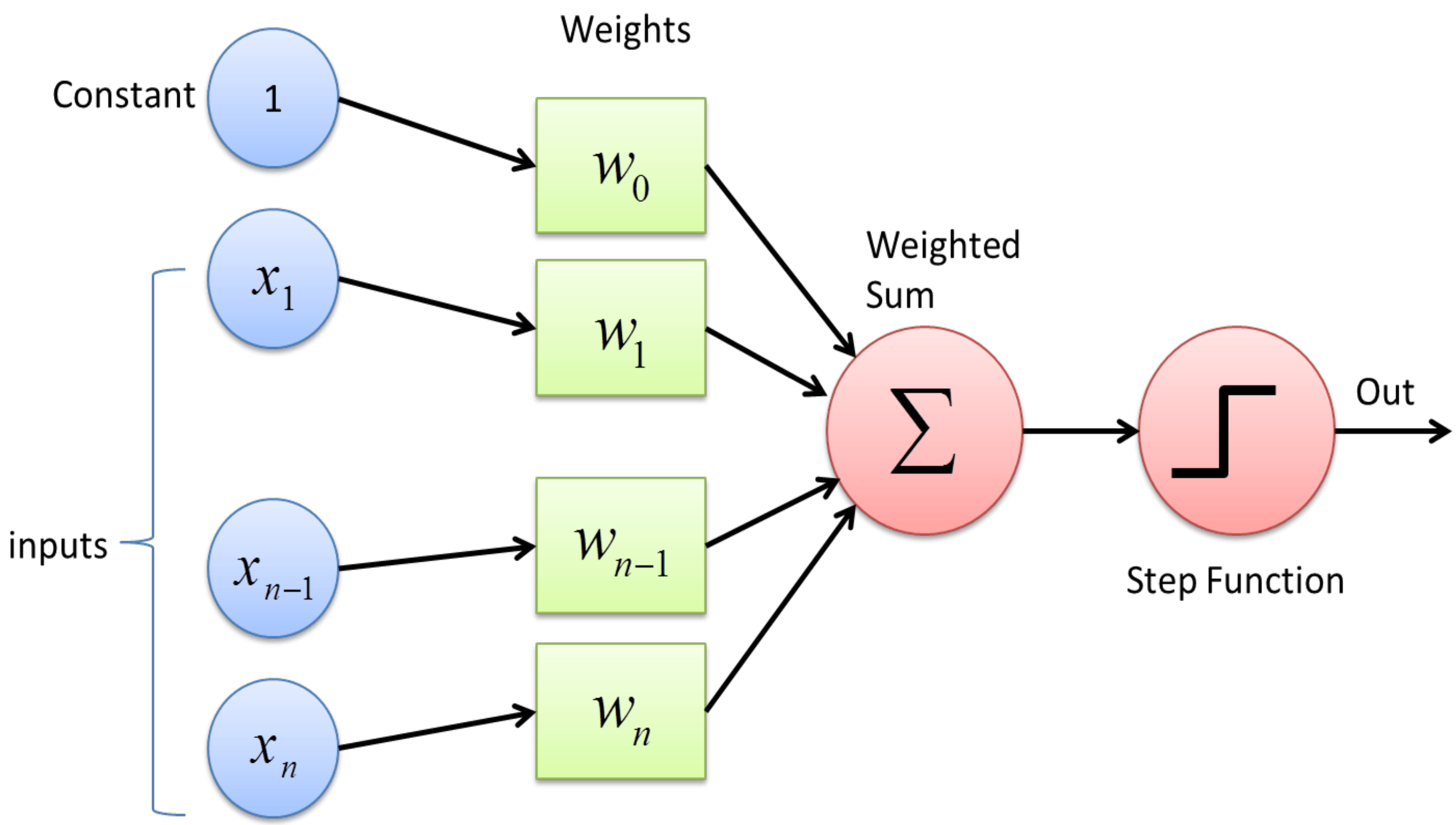
Perceptron

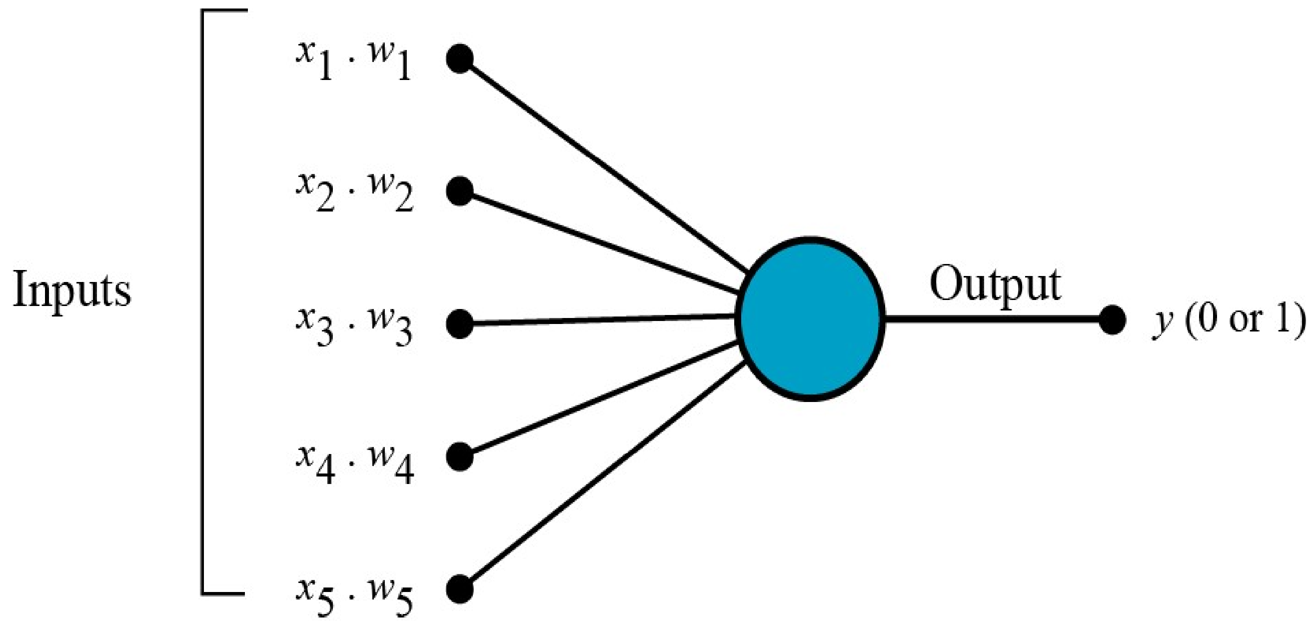
Perceptron is a linear classifier (binary). Also, it is used in supervised learning. It helps to classify the given input data.

The Perceptron consists of 4 parts.

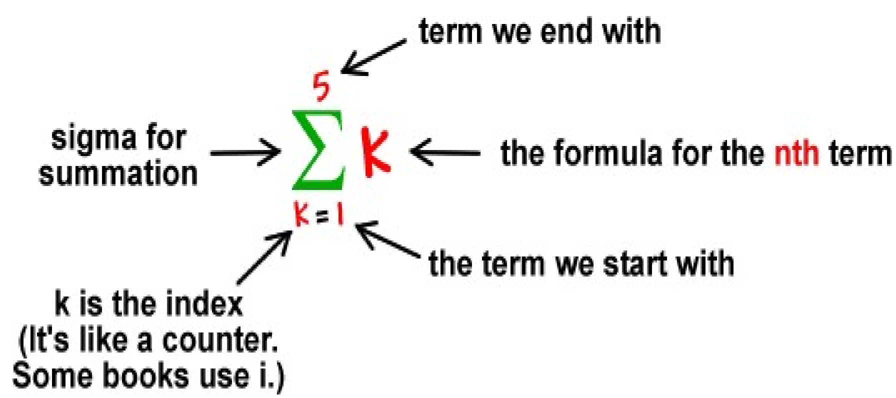
- Input values or One input layer
- Weights and Bias
- Net sum
- Activation Function



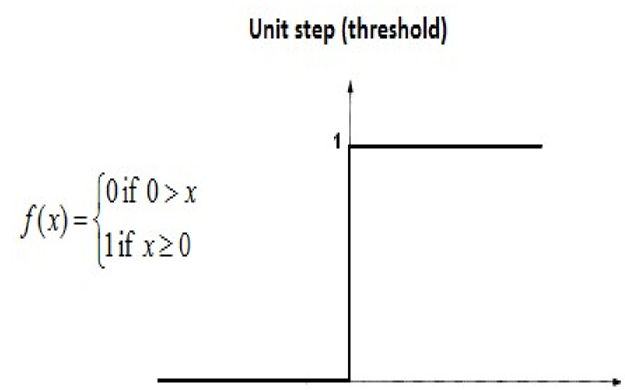
Weights
Constant
1
w_0
x_1
w_1
Weighted Sum
Out
inputs
x_{n-1}
w_{n-1}
x_n
w_n
Step Function



Inputs
$x_1 \cdot w_1$
$x_2 \cdot w_2$
$x_3 \cdot w_3$
$x_4 \cdot w_4$
$x_5 \cdot w_5$
Output
y (0 or 1)



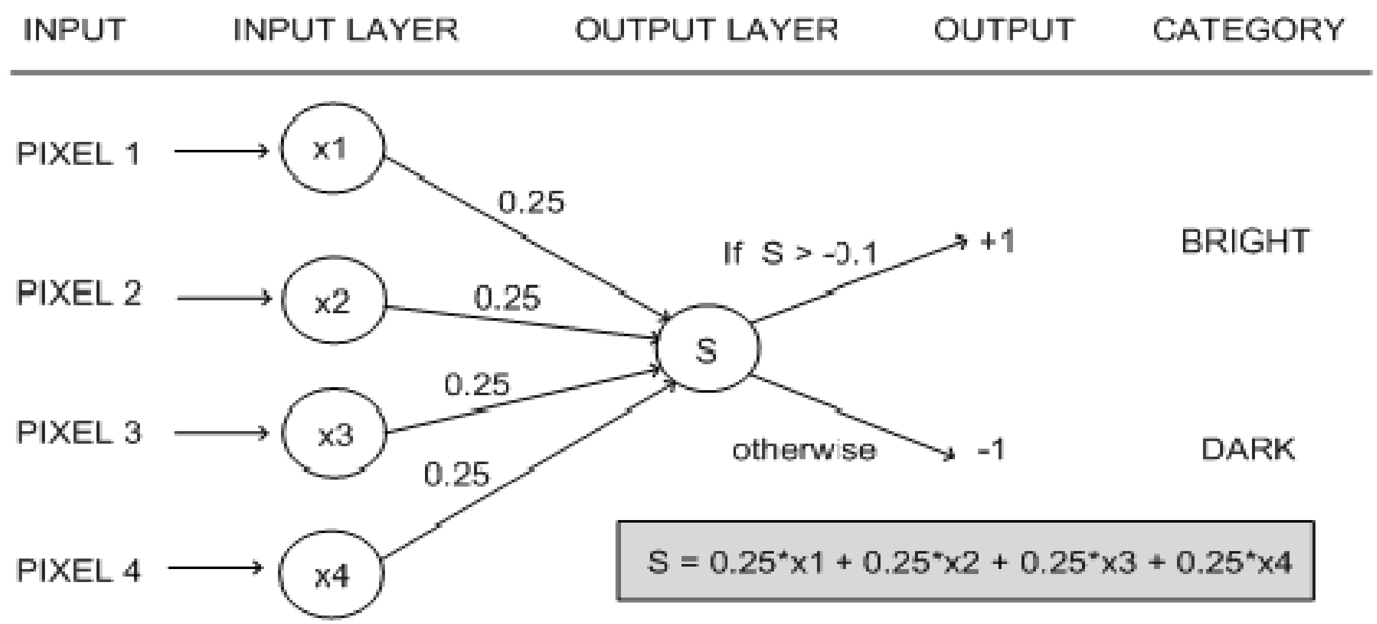
term we end with
5
sigma for summation
K
the formula for the nth term
k = 1
the term we start with
k is the index
(It's like a counter.
Some books use i.)

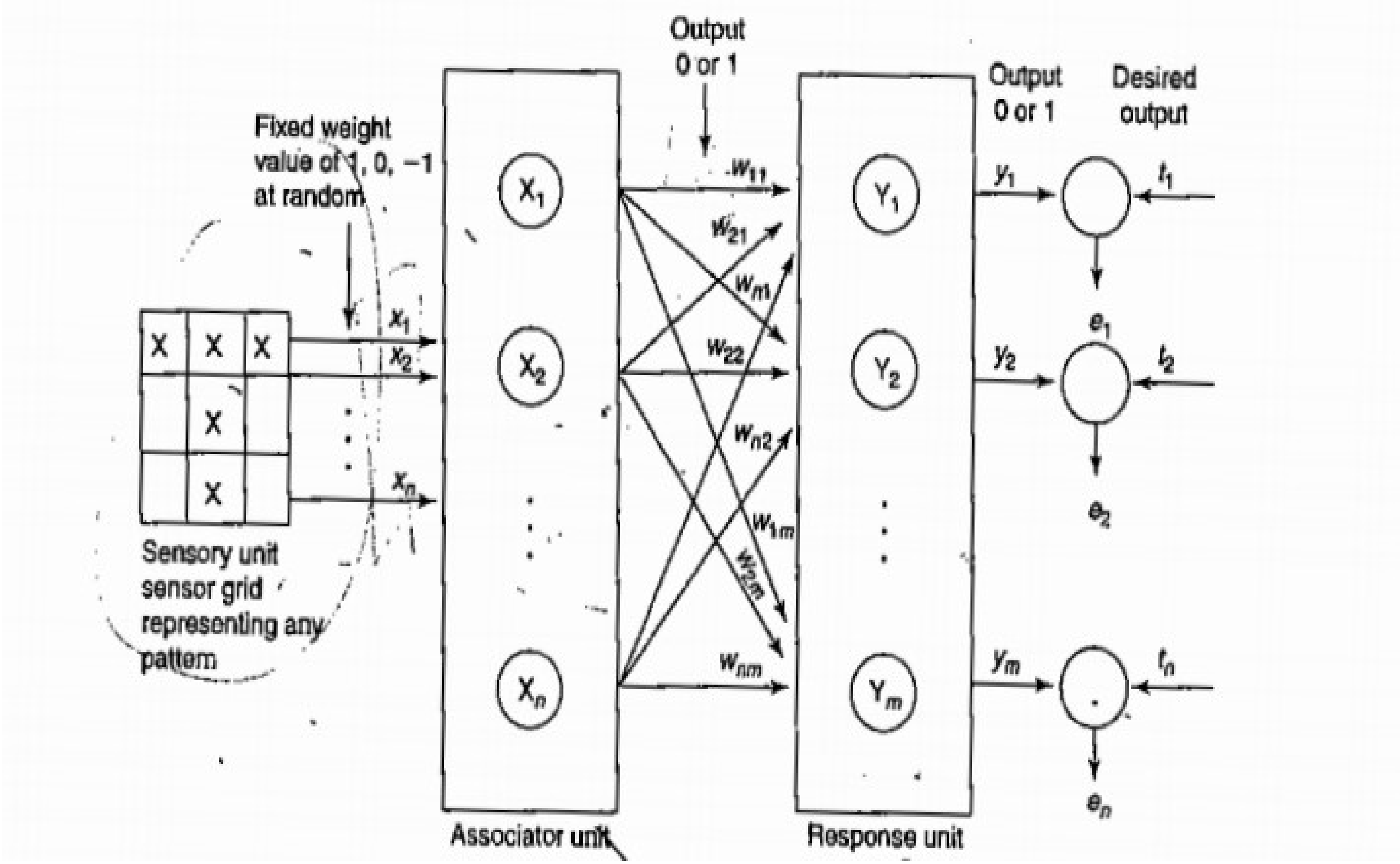


Unit step (threshold)
1
$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$

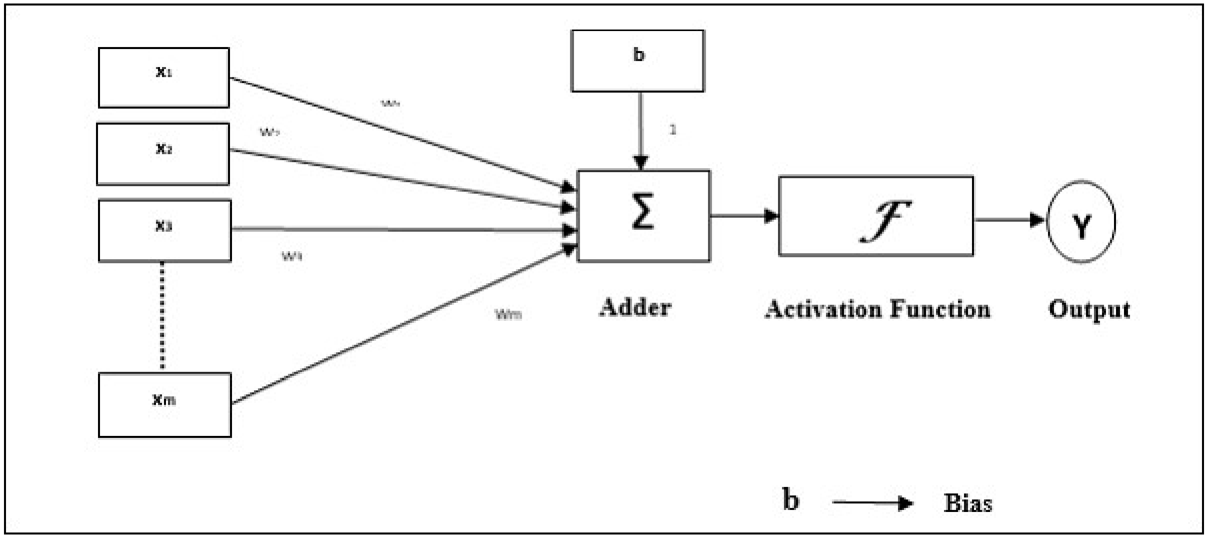
Why do we need Weights and Bias?

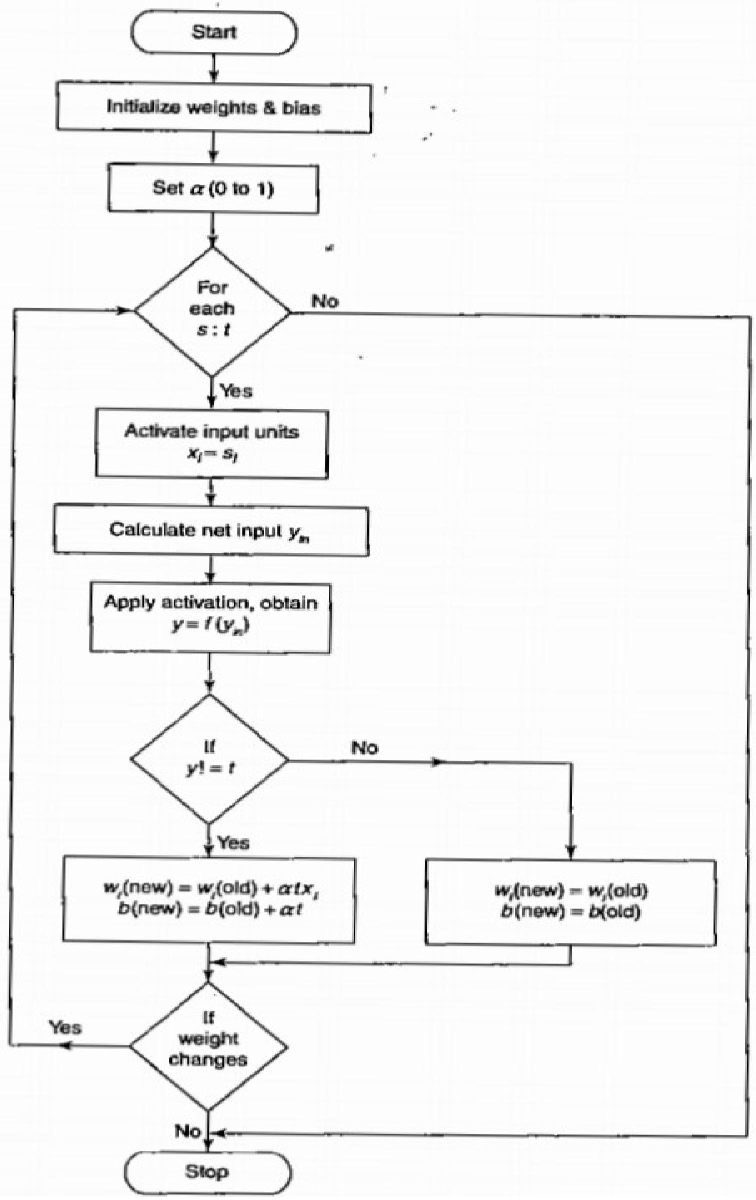
- Weights shows the strength of the particular node.
- *A bias* value allows you to shift the activation function curve up or down.




Output
0 or 1
Fixed weight
value of 1, 0, −1
at random
Output
0 or 1
Desired
output
X X X
X
X
Sensory unit
sensor grid
representing any
pattern
Associator unit
Response unit

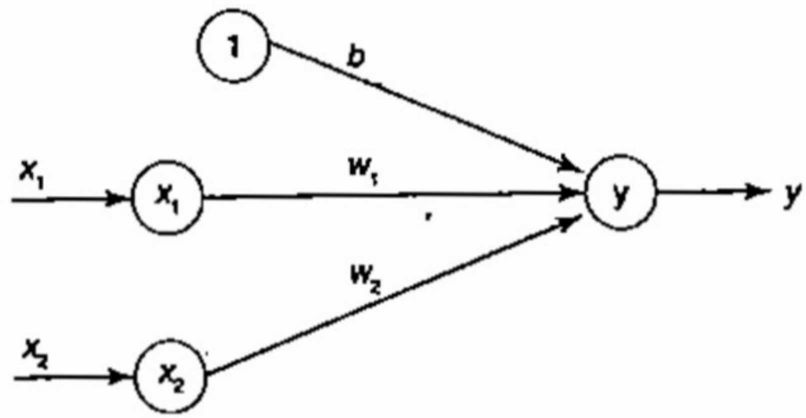
Perceptron




Start
Initialize weights & bias
Set α (0 to 1)
For each s : t
No
Yes
Activate input units
x_i = s_i
Calculate net input y_in
Apply activation, obtain
y = f (y_in)
If y! = t
No
Yes
w_i(new) = w_i(old) + αtx_i
b(new) = b(old) + αt
w_i(new) = w_i(old)
b(new) = b(old)
Yes
If weight changes
No
Stop

Implement Perceptron

X1	X2	Y
1	1	-1
1	-1	1
-1	1	-1
-1	-1	-1



Let w1=w2=0, b=0, α=1, θ=0

Applying the activation function over the net input, we obtain

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

For 1st input sample: {x1 = 1, x2 = 1, t =-1}

Yin= b+ x1 *w1 + x2 * w2 = 0 + 1*0 + 1*0 = 0

W1 (new) = W1(old) + α * t * x1 = 0 + 1 * -1 * 1 = -1

W2 (new) = W2(old) + α * t * x2 = 0 + 1 * -1 * 1 = -1

b(new) = b(old)+ α * t = 0 + 1 x -1 = -1

(W1, w2, b) = [-1, -1, -1]

For 2^{nd} input sample: {x1 = 1, x2 = -1, t =1}

Yin= b+ x1 *w1 + x2 * w2 = -1 + 1*-1 + (-1*-1) = -1

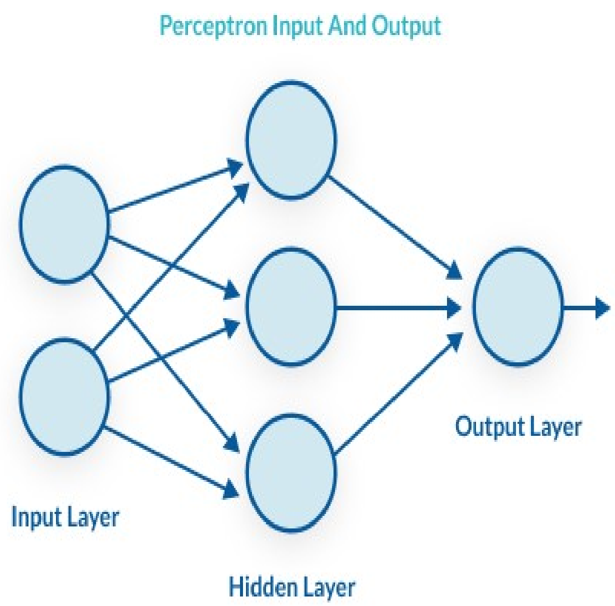
W1 (new) = W1(old) + α * t * x1 = -1 + 1 * 1 * 1 = 0

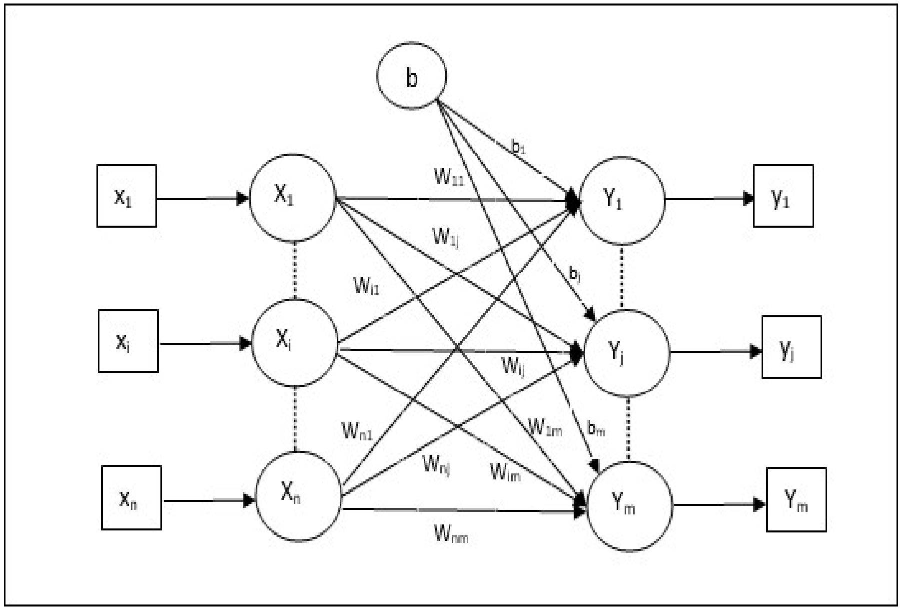
W2 (new) = W2(old) + α * t * x2 = -1 + 1 * 1 * -1 = -2

b(new) = b(old)+ α * t = -1 + 1 x 1 = 0

(W1, w2, b) = [0, -2, 0]

- Multilayer Perceptron





If y_j != t_j

Y_{in} = b + Ʃ xi * w_{ij}

W_{ij}(new)= W_{ij}(old) + α * t_j * x_i

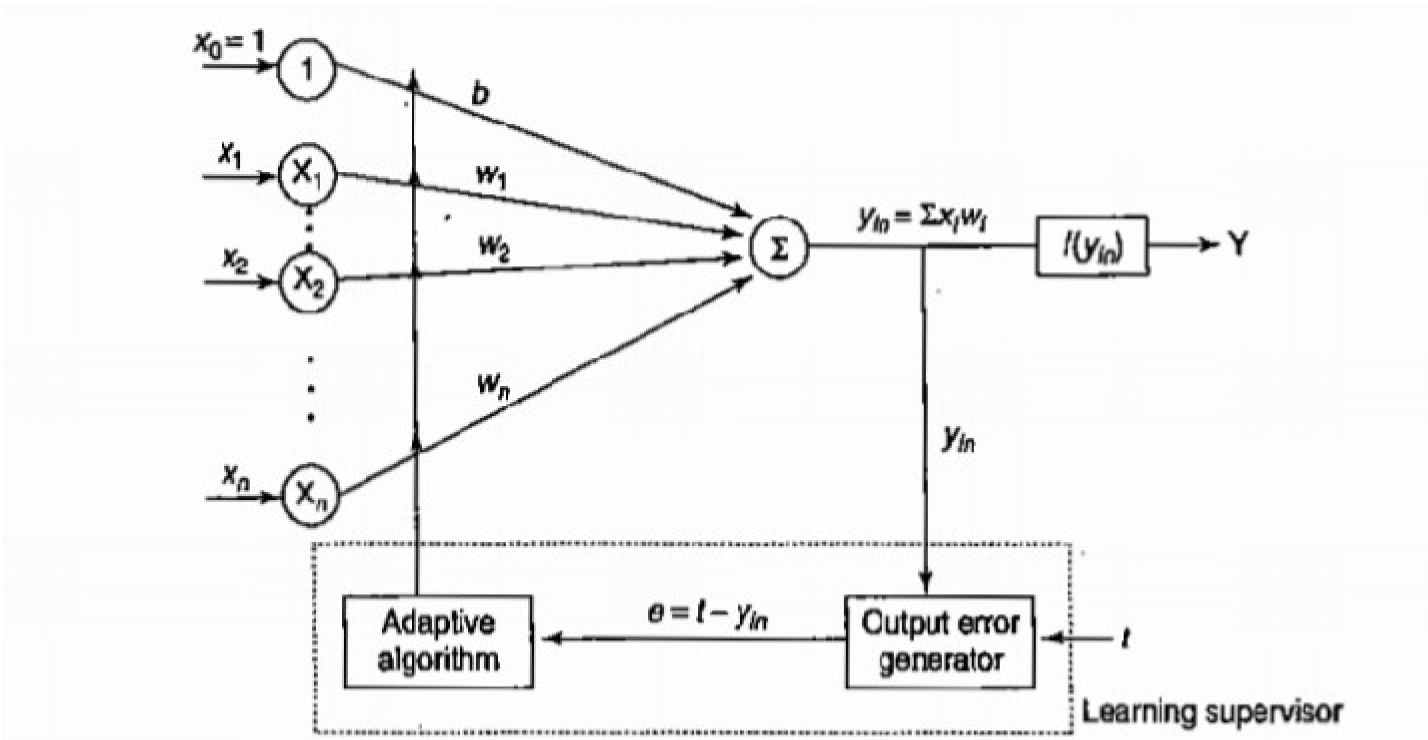
b_j(new) = b_j(old)+ α * t_j

Adaptive Linear Neuron (Adaline)

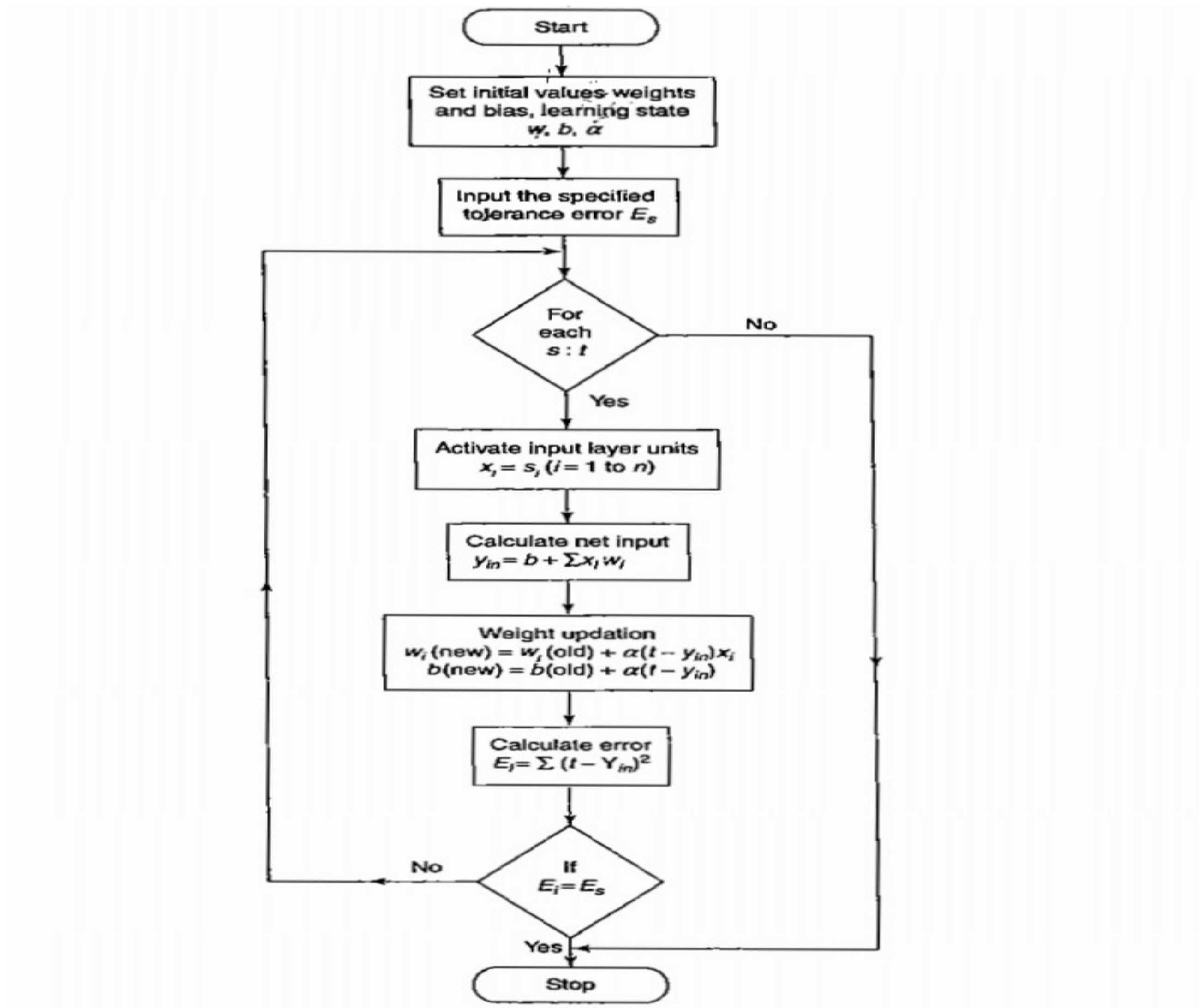
- A network with a single linear unit is called an Adaline (adaptive linear neuron).
- The Adaline network may be trained using delta rule. The delta rule may afso be called as least mean square (LMS) rule .
- The perceptron learning rule stops after a finite number of learning steps, but the gradient descent approach continues forever, converging only asymptotically to the solution.

W(new)= α(t-yin)* x1

Adaline Network

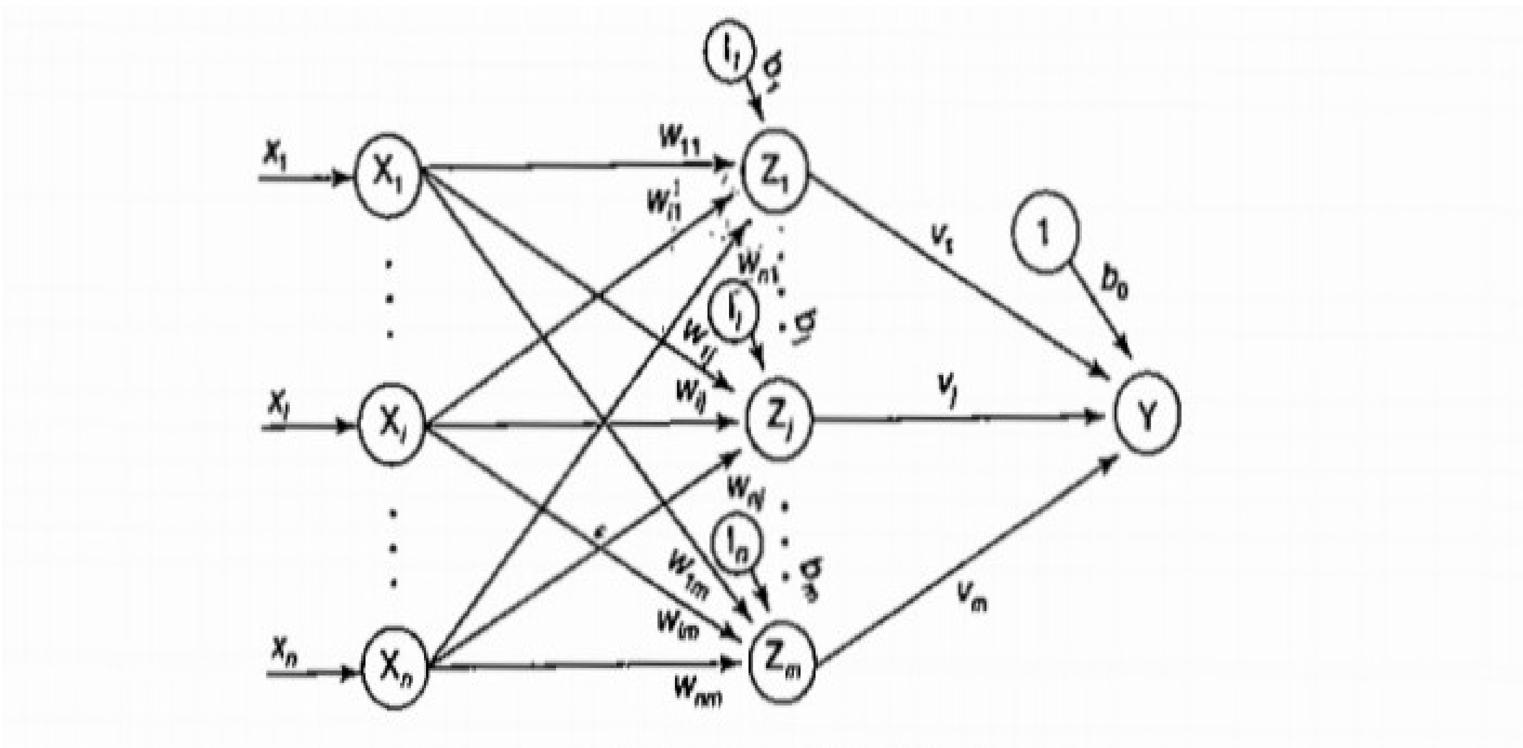


Flow Chart



Start
Set initial values weights and bias, learning state
w, b, α
Input the specified tolerance error E_s
For each $s:t$
No
Yes
Activate input layer units
$x_i = s_i$ (i = 1 to n)
Calculate net input
$y_{in} = b + \sum x_i w_i$
Weight updation
w_i (new) = w_i (old) + $\alpha(t - y_{in})x_i$
b(new) = b(old) + $\alpha(t - y_{in})$
Calculate error
$E_i = \sum (t - Y_{in})^2$
No
If $E_i = E_s$
Yes
Stop

Madaline



Steps of Madaline Layer

Step 1: When stopping condition is false, perform Steps 2–3.

Step 2: For each bipolar training pair $s:t$, perform Steps 3–7.

Step 3: Activate input layer units. For $i = 1$ to n,

$$x_i = s_i$$

Step 4: Calculate net input to each hidden Adaline unit:

$$z_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}, \quad j = 1 \text{ to } m$$

Step 5: Calculate output of each hidden unit:

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

Step 6: Find the output of the net:

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

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

Step 7: Calculate the error and update the weights.

1. If $t = y$, no weight updation is required.
2. If $t \neq y$ and $t = +1$, update weights on z_j, where net input is closest to 0 (zero):

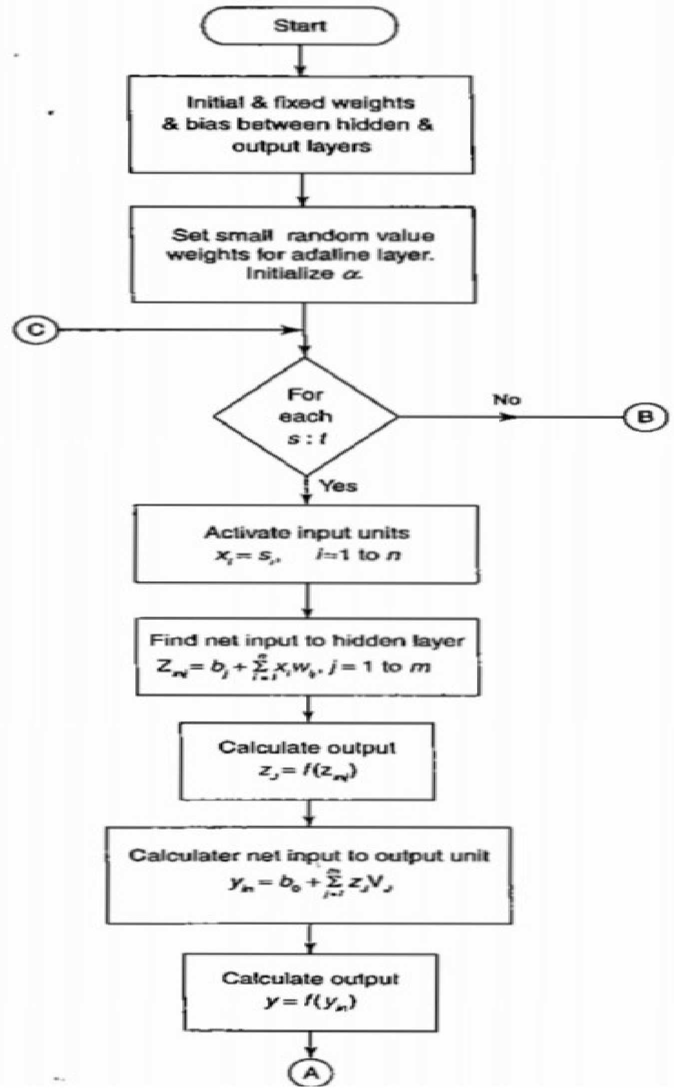
$$b_j(\text{new}) = b_j(\text{old}) + \alpha\,(1 - z_{inj})$$
$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha\,(1 - z_{inj})x_i$$

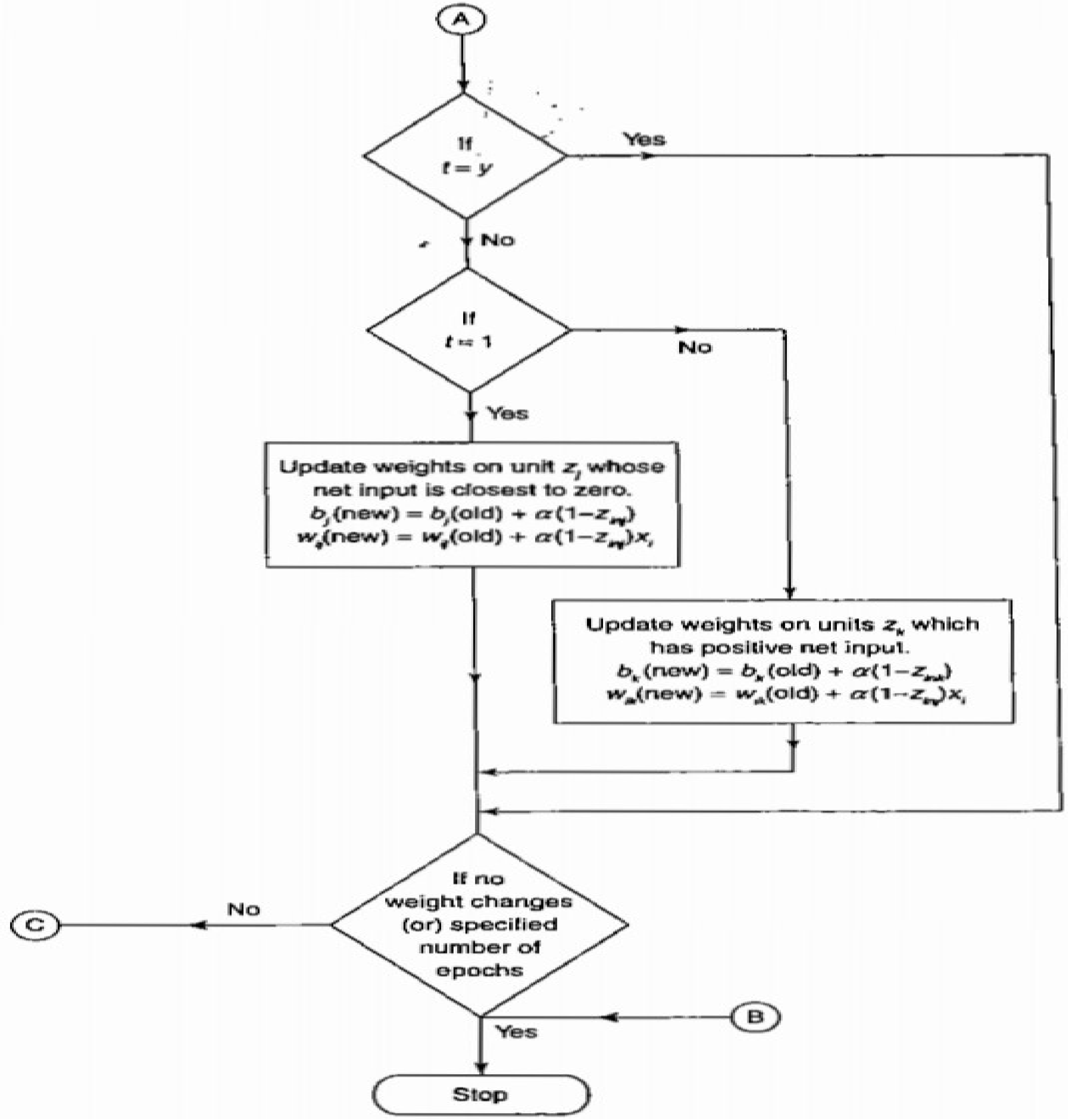
3. If $t \neq y$ and $t = -1$, update weights on units z_k whose net input is positive:

$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha\,(-1 - z_{ink})\,x_i$$
$$b_k(\text{new}) = b_k(\text{old}) + \alpha\,(-1 - z_{ink})$$

Step 8: Test for the stopping condition. (If there is no weight change or weight reaches a satisfactory or if a specified maximum number of iterations of weight updation have been performed stop, or else continue).

Flow

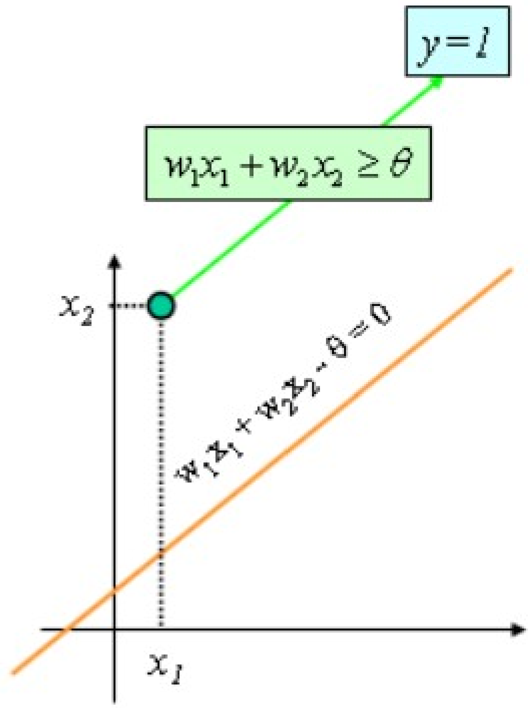

Start
Initial & fixed weights & bias between hidden & output layers
Set small random value weights for adaline layer. Initialize α.
C
For each s : t
No
B
Yes
Activate input units
$x_i = s_i, \quad i = 1$ to n
Find net input to hidden layer
$Z_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}, j = 1$ to m
Calculate output
$z_j = f(z_{inj})$
Calculater net input to output unit
$y_{in} = b_0 + \sum_{j=1}^{m} z_j V_j$
Calculate output
$y = f(y_{in})$
A



A
If
t = y
Yes
No
If
t = 1
No
Yes
Update weights on unit z_j whose
net input is closest to zero.
b_j(new) = b_j(old) + α(1−z_inj)
w_ij(new) = w_ij(old) + α(1−z_inj)x_i
Update weights on units z_k which
has positive net input.
b_k(new) = b_k(old) + α(1−z_ink)
w_ik(new) = w_ik(old) + α(1−z_ink)x_i
If no
weight changes
(or) specified
number of
epochs
No
C
Yes
B
Stop

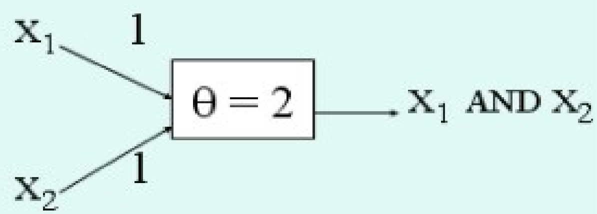
Perceptron

- The required perceptron's behavior can be obtained by adjusting appropriate weights and threshold.

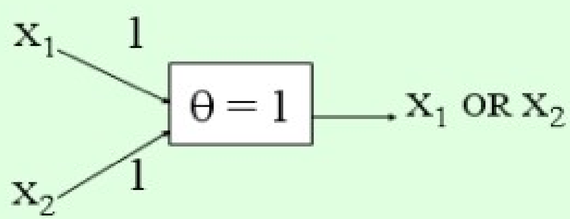


- A single perceptron with appropriately set weights and threshold can easily simulate basic logical gates:

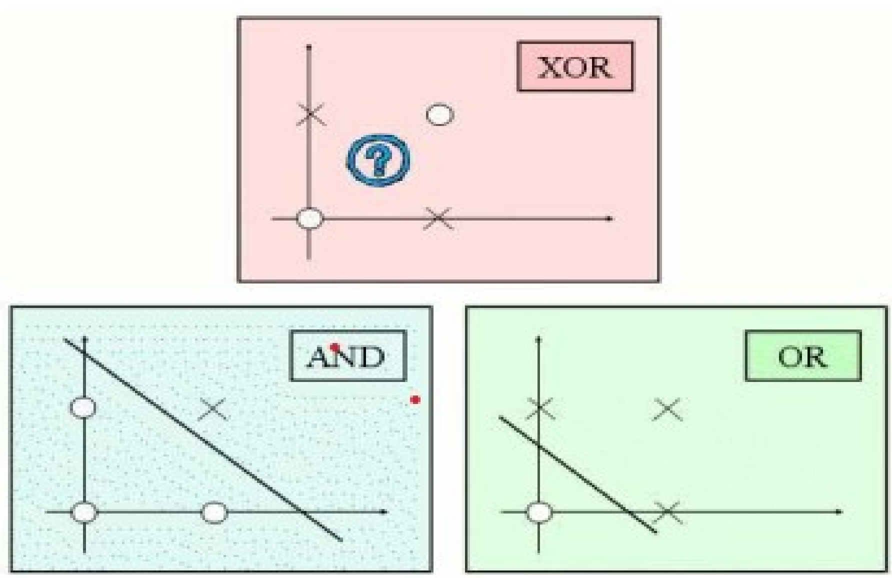
x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1



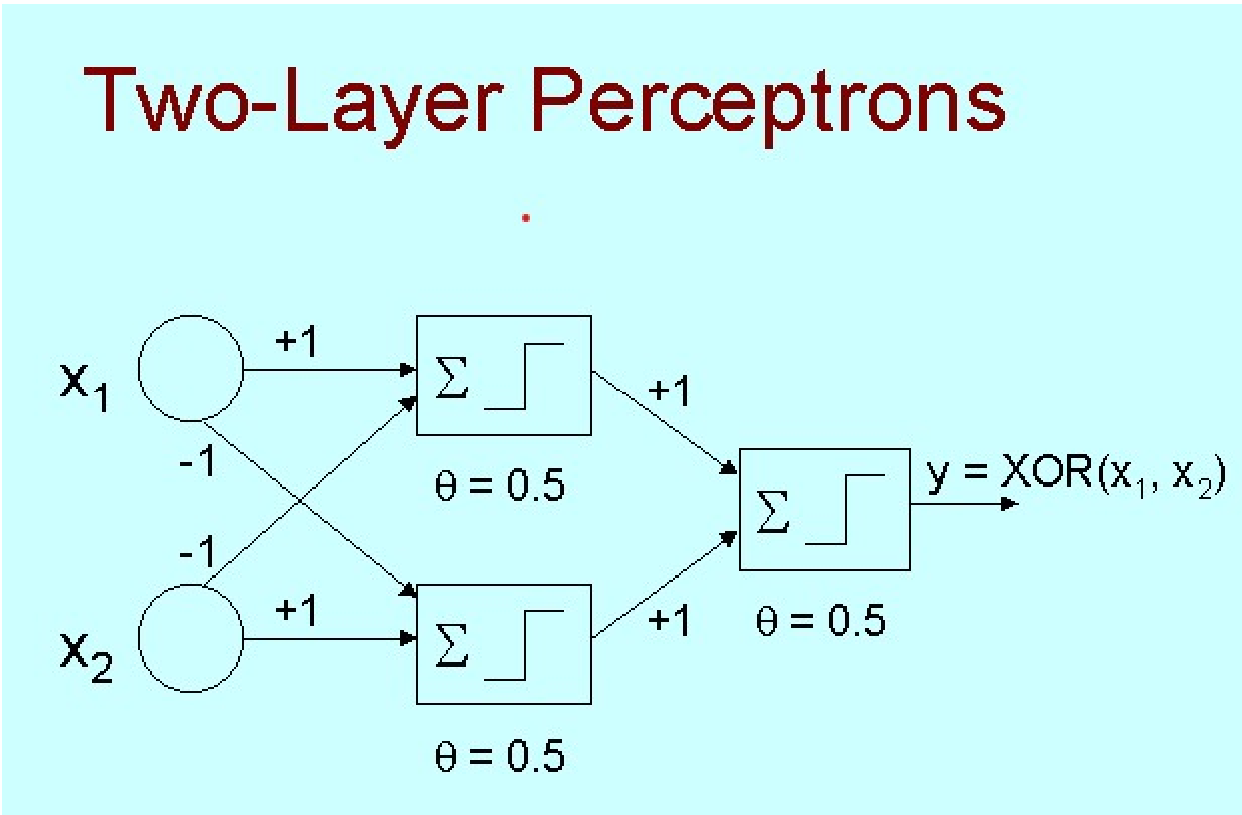
x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1



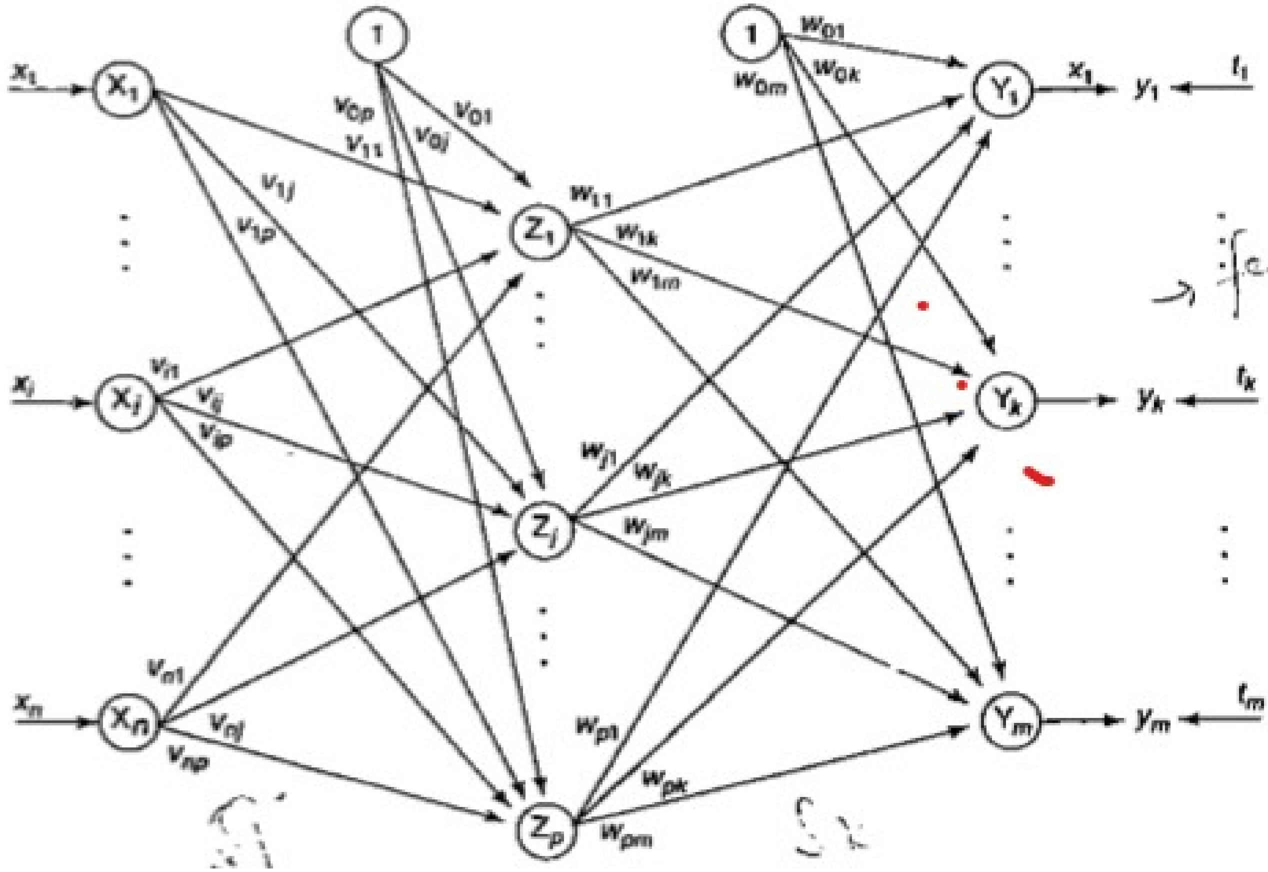
- A single perceptron can distinguish only the sets of inputs which are linearly separable in the input
- One of the simplest examples of linearly non-separable sets is logical function XOR.

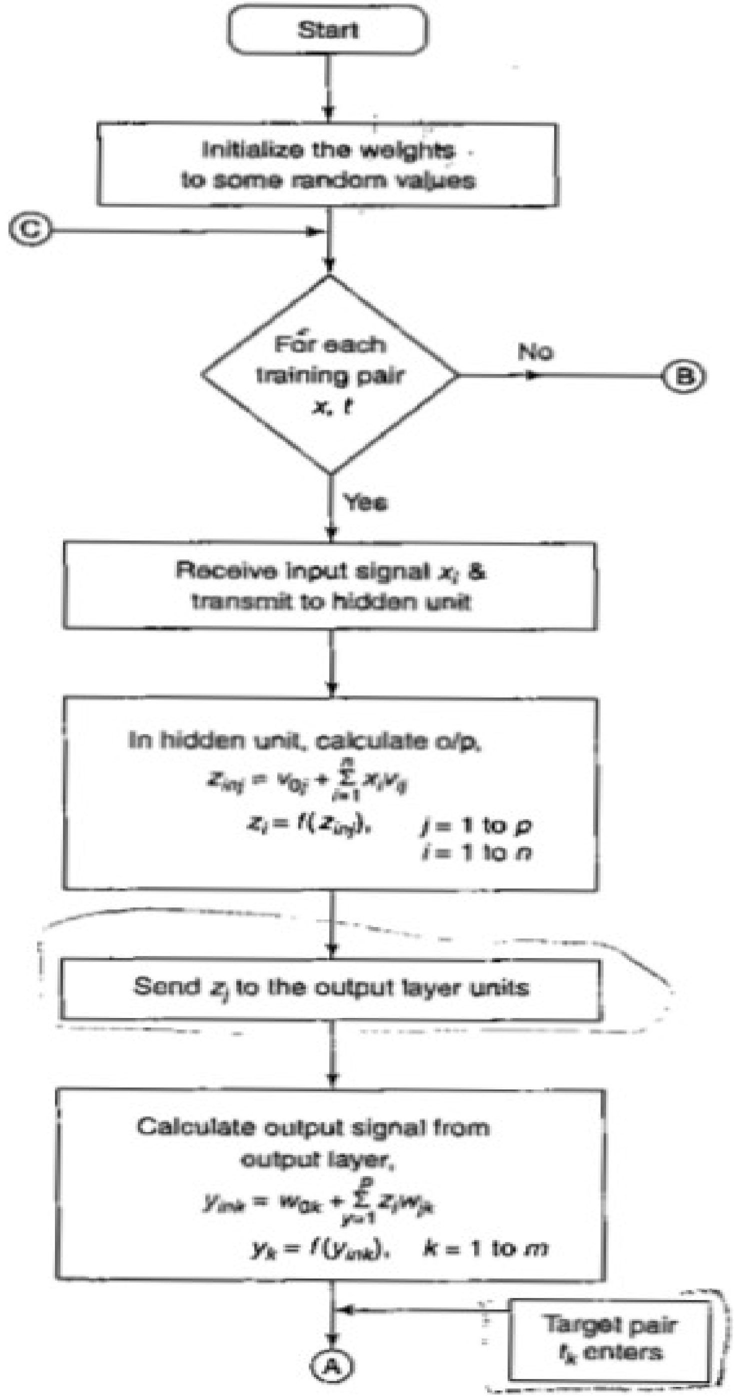


MLP

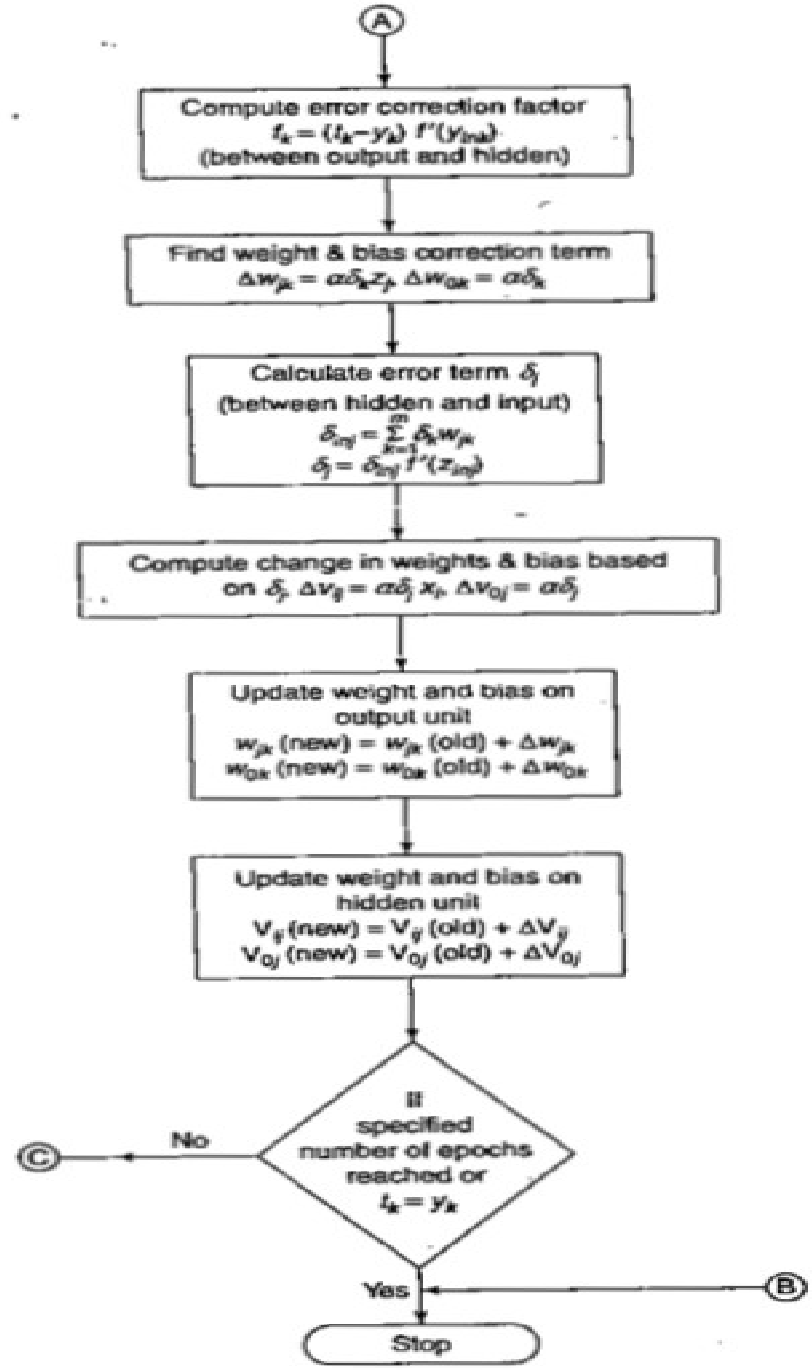


Back-Propagation NN





Start
Initialize the weights to some random values
C
For each training pair x, t
No
B
Yes
Receive input signal x_i & transmit to hidden unit
In hidden unit, calculate o/p,
$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$
$z_j = f(z_{inj}), \quad j = 1 \text{ to } p$
$i = 1 \text{ to } n$
Send z_j to the output layer units
Calculate output signal from output layer,
$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$
$y_k = f(y_{ink}), \quad k = 1 \text{ to } m$
Target pair t_k enters
A


A
Compute error correction factor
$\delta_k = (t_k - y_k)\, f'(y_{ink})$
(between output and hidden)
Find weight & bias correction term
$\Delta w_{jk} = \alpha \delta_k z_j,\ \Delta w_{0k} = \alpha \delta_k$
Calculate error term δ_j
(between hidden and input)
$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$
$\delta_j = \delta_{inj}\, f'(z_{inj})$
Compute change in weights & bias based
on δ_j, $\Delta v_{ij} = \alpha \delta_j x_i$, $\Delta v_{0j} = \alpha \delta_j$
Update weight and bias on
output unit
w_{jk} (new) = w_{jk} (old) + Δw_{jk}
w_{0k} (new) = w_{0k} (old) + Δw_{0k}
Update weight and bias on
hidden unit
V_{ij} (new) = V_{ij} (old) + ΔV_{ij}
V_{0j} (new) = V_{0j} (old) + ΔV_{0j}
If
specified
number of epochs
reached or
$t_k = y_k$
No
C
Yes
B
Stop

Step 0: Initialize weights and learning rate (take some small random values).

Step 1: Perform Steps 2–9 when stopping condition is false.

Step 2: Perform Steps 3–8 for each training pair.

Step 3: Each input unit receives input signal x_i and sends it to the hidden unit ($i = 1$ to n).

Step 4: Each hidden unit $z_j (j = 1$ to $p)$ sums its weighted input signals to calculate net input:

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

Calculate output of the hidden unit by applying its activation functions over z_{inj} (binary or bipolar sigmoidal activation function):

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

and send the output signal from the hidden unit to the input of output layer units.

Step 5: For each output unit y_k ($k = 1$ to m), calculate the net input:

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and apply the activation function to compute output signal

$$y_k = f(y_{ink})$$

Step 6: Each output unit $y_k (k = 1$ to $m)$ receives a target pattern corresponding to the input training pattern and computes the error correction term:

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

The derivative $f'(y_{ink})$ can be calculated as in Section 2.3.3. On the basis of the calculated error correction term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j; \quad \Delta w_{0k} = \alpha \delta_k$$

Also, send δ_k to the hidden layer backwards.

Step 7: Each hidden unit $(z_j, j = 1$ to $p)$ sums its delta inputs from the output units:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

The term δ_{inj} gets multiplied with the derivative of $f(z_{inj})$ to calculate the error term:

$$\boxed{\delta_j = \delta_{inj} f'(z_{inj})}$$

The derivative $f'(z_{inj})$ can be calculated as discussed in Section 2.3.3 depending on whether binary or bipolar sigmoidal function is used. On the basis of the calculated δ_j, update the change in weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i; \quad \Delta v_{0j} = \alpha \delta_j$$

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

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

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

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

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

Bipolar sigmoid function: This function is defined as

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

where λ is the steepness ~~parameter~~ and the sigmoid function range is between -1 and $+1$. The derivative of this function ~~can be~~

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

Step 8: Each output unit (y_k, $k = 1$ to m) updates the bias and weights:

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$$
$$w_{0k}(\text{new}) = w_{0k}(\text{old}) + \Delta w_{0k}$$

Each hidden unit (z_j, $j = 1$ to p) updates its bias and weights:

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$$
$$v_{0j}(\text{new}) = v_{0j}(\text{old}) + \Delta v_{0j}$$

Step 9: Check for the stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equals the target output.