Soft Computing
Module 2

Dr.Priya S

MODULE 2

- Perceptron Networks
- ADALINE
- Back Propagation Networks

Topics in Perceptron

- Introduction
- Theory
- Perceptron Learning Rule
- Learning rule convergence theorem
- Architecture
- Flowchart for training process
- Training algorithm -for single and multiple output class
- -Testing algorithm

Introduction-Perceptron Network

- Perceptron ia a neuron in ANN
- Perceptron network is the simplest of NN used for classification of patterns
- More powerful than Hebb Network
- -bipolar data
- -iterative weight adjustment
- Simple Perceptron was developed by Block in 1962
- Various types of perceptron was developed by Rosenblatt and Minsky

Introduction-contd.

- Perceptron is limited to perform only binary classification of patterns
- It can learn only lineraly seperable problems
- Perceptron use binary activation fn./step fn./thresholding fn./heaviside fn.
- Iterative learning converges to correct weights
- 2 types of perceptron-single layer and multilayer
- Learning rate parameter α is set (0 and 1)

Theory or Characteristics

-Perceptron network consists of 3 units: *sensory unit (input unit), associator unit (hidden unit)*, and *response unit (output unit)*.

-Sensory units are connected to associator units with fixed weights having values *1,0 or -1* .

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

-The response unit has an activation of *1,0 or -1* .

- The output of the perceptron network is given by;

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

where $f(y_{in})$ is activation function and is defined as;

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

- The perceptron learning rule is used in the weight updation between *associator unit and response unit*.
- The error calculation is based on the comparison of the values of targets with those of the calculated outputs.

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

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

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

where,

t= target value(+1or 1)

α = learning rate

If no error occurs, there is no weight updation and training process may be stopped

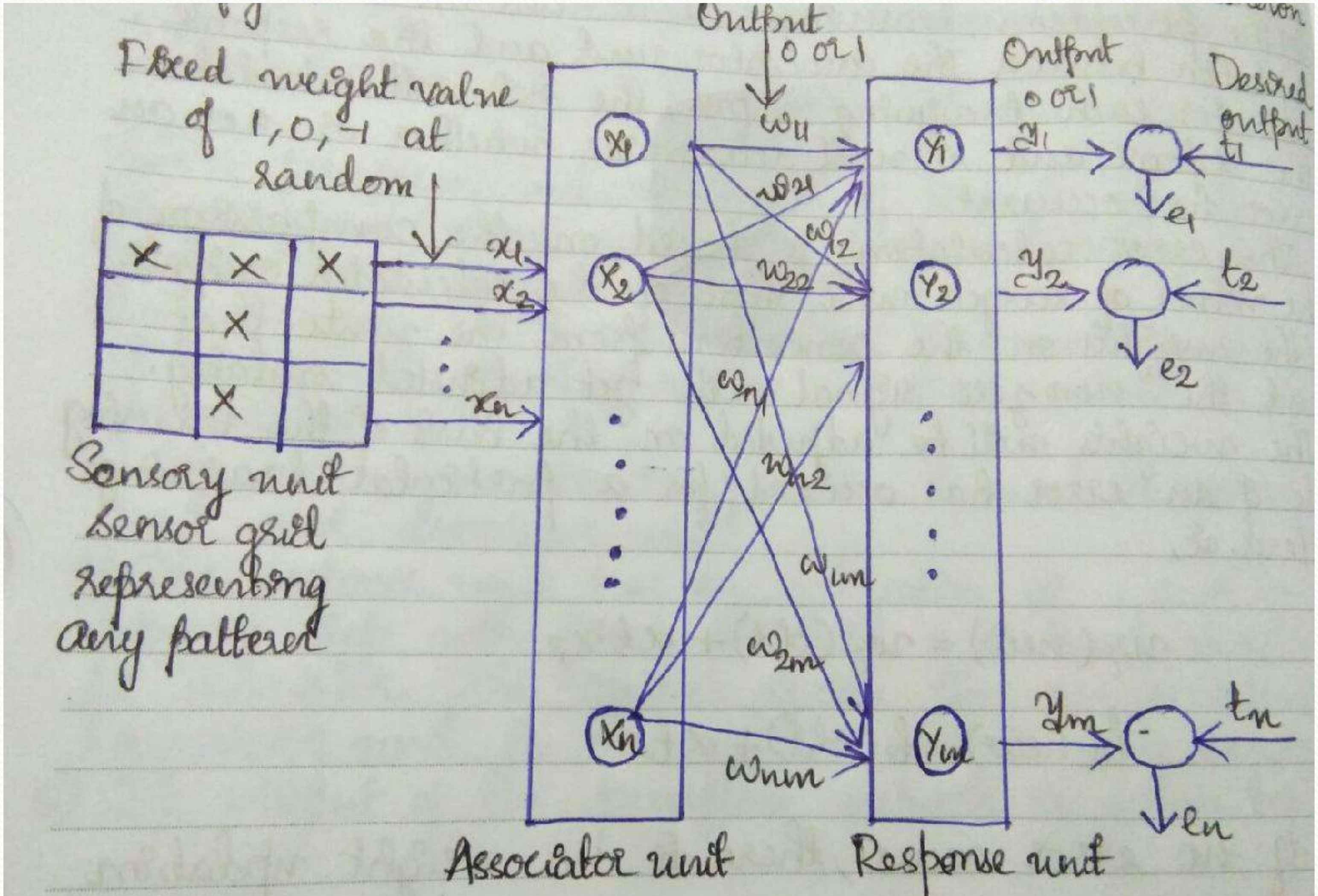


Figure : A perceptron network with its three units

Learning Rule

- A finite n number of input training vectors with their associated target values; $x(n)$ *and* $t(n)$.
- The output y is obtained on the basis of the net input calculated and activation function being applied over the net input.

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

- The weight updation is as follows:
  If $y \neq t$ then ,
  $$w_i(new) = w_i(old) + \alpha t x_i$$
  else, we have
  $$w(new) = w(old)$$

Perceptron Learning Rule Convergence Theorem

- "If there is a weight vector W, such that f(x (n) W)= t(n), then for any starting vector w_1, the perceptron learning rule will converge to a weight vector that gives the correct response for all training patterns, and this learning takes place within a finite number of steps provided that the solution exists"

Architecture

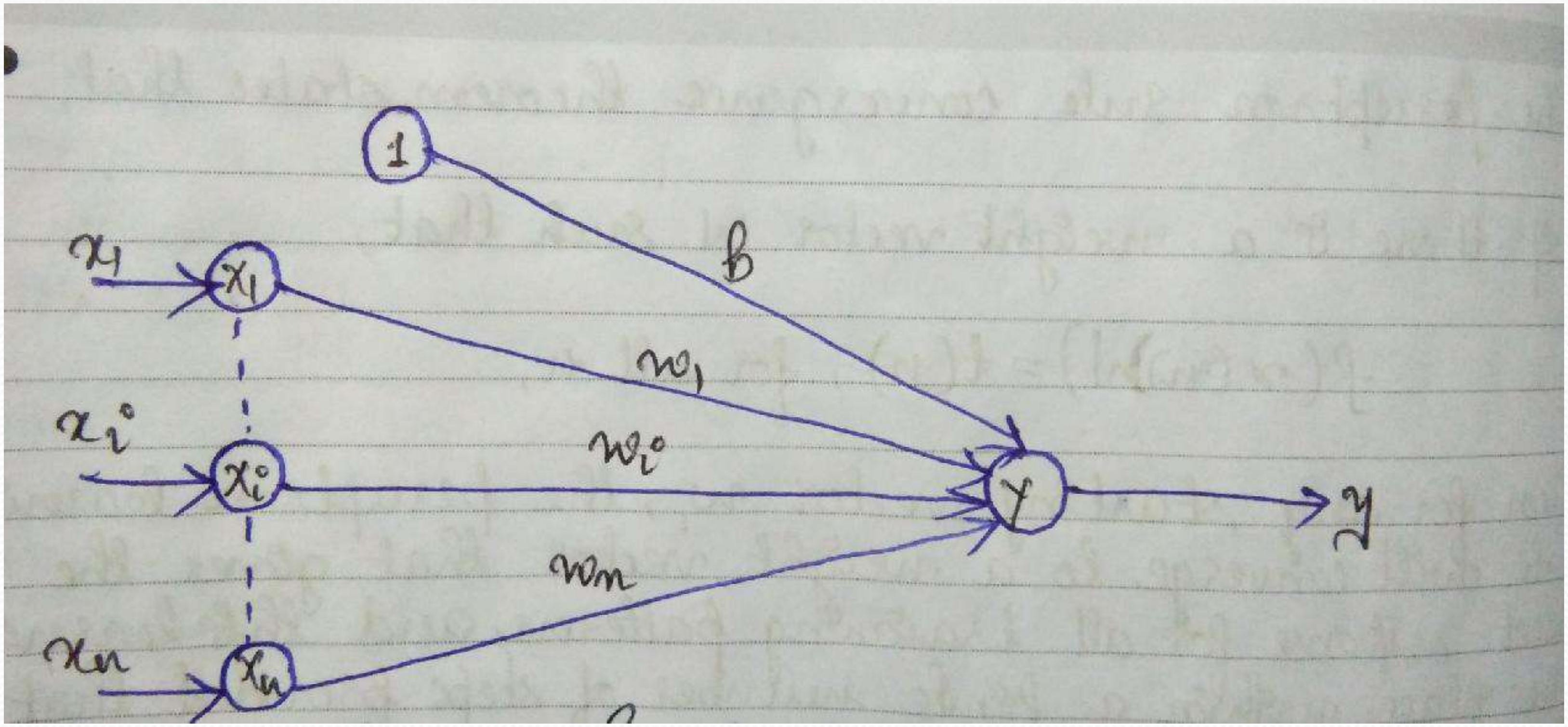
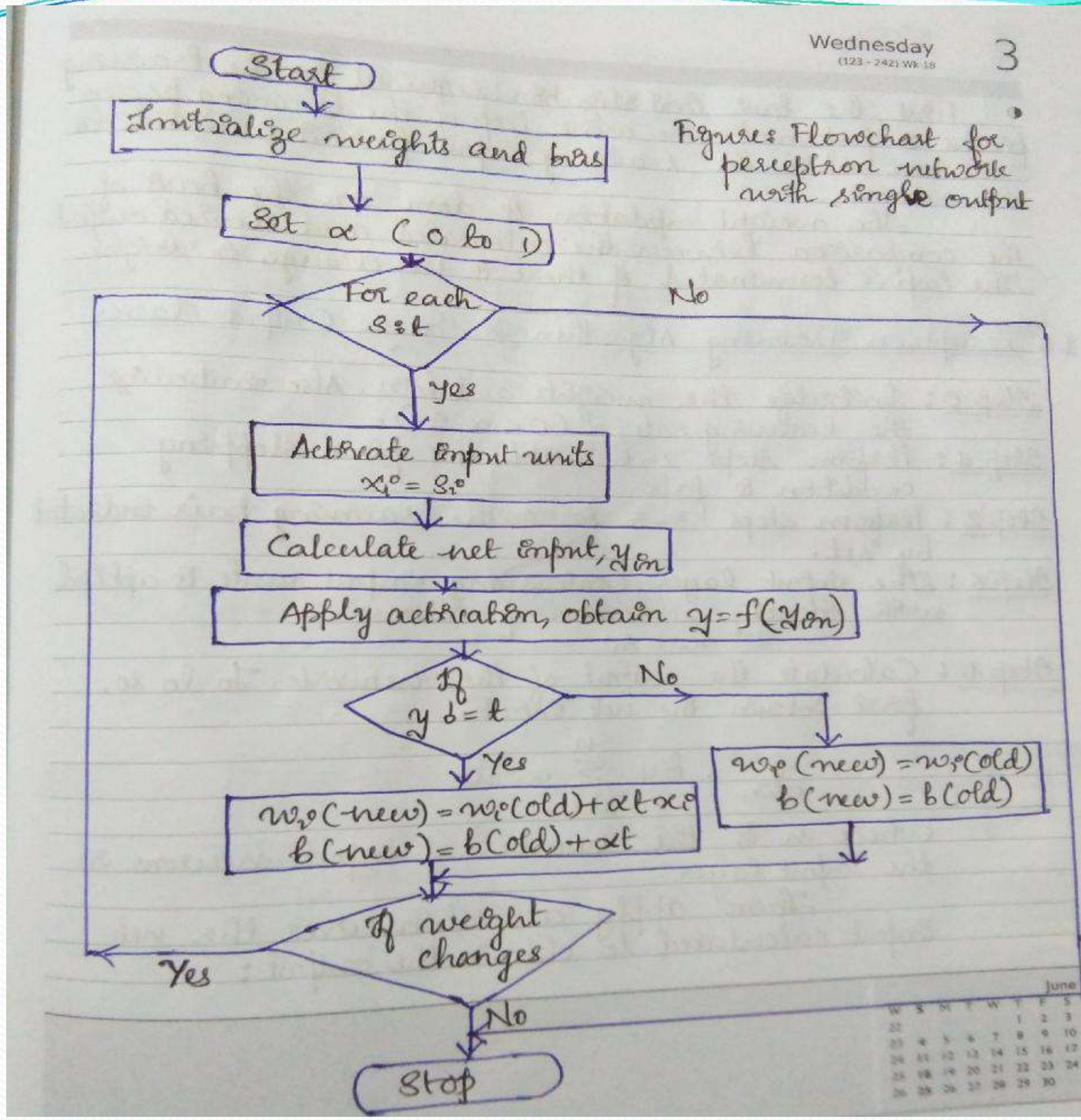


Figure : Single classification perceptron network

- Perceptron has sensory, associator and response unit
- In this architecture only the associator and response unit is shown and sensory unit is hidden because only the weights between the associator and the response unit are adjusted
- Input layer consists of input neurons from $X_1 \ldots X_i \ldots X_n$
- There always exist a common bias of 1
- This is a single layer network

Flowchart



Wednesday 3
Start
Initialize weights and bias
Figure: Flowchart for perceptron network with single output
Set α (0 to 1)
For each s:t
No
yes
Activate input units
xi = si
Calculate net input, yin
Apply activation, obtain y = f(yin)
If y ≠ t
No
Yes
wi (new) = wi (old)
b(new) = b(old)
wi (new) = wi (old) + α t xi
b (new) = b (old) + α t
If weight changes
Yes
No
Stop

Perceptron Training Algorithm for Single Output Class

- *Step 0*: Initialize the weights and bias. Also, initialize the learning rate, $\alpha (0 < \alpha \leq 1)$.
- *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 unit is applied with identity activation functions:

$$x_i = s_i$$

- *Step 4*: Calculate the output of the network.

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

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

- *Step 5*: Weight and bias adjustment:
  If $y \neq t$ then ,

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

  else, we have

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

- *Step 6*: Train the network until there is no weight change.
  Otherwise, start again from Step 2.

Perceptron Training Algorithm for Multiple Output Class

- *Step 0*: Initialize the weights and bias. Also, initialize the learning rate, $\alpha (0 < \alpha \leq 1)$.
- *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 unit is applied with identity activation functions:

$$x_i = s_i$$

- *Step 4*: Calculate the output of the network.

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

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

- *Step 5*: Make adjustment in weight and bias for $j = 1$ *to* m and $i = 1$ *to* n
  If $t_i \neq y_j$ then ,

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

  else, we have

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

- *Step 6*: Train the network until there is no weight change. Otherwise, start again from Step 2.

Perceptron Network Testing Algorithm

- *Step 0*: Initial weights is equal to the final weights obtained during training.
- *Step 1*: For each input vector X to be classified, perform Steps 2-3.
- *Step 2*: Set activations of the input unit.
- *Step 3*: Obtain the response of output unit.

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

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

Problems

1.Develop a perceptron for the AND function with bipolar inputs and targets

Input			Target
X_1	X_2	b	t
1	1	1	1
-1	1	1	-1
1	-1	1	-1
-1	-1	1	-1

Step 1: Initial weights $w_1 = w_2 = 0$ and $b = 0$, $\alpha = 1$, $\theta = 0$.

Step 2: Begin computation.

Step 3: For input pair (1, 1): 1, do Steps 4–6

Step 4: Set activations of input units

$$x_i = (1, 1).$$

Step 5: Calculate the net input.

$$y_{-in} = b + \sum x_i w_i = 0 + 1 \times 0 + 1 \times 0 = 0$$

Applying the activation,

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

Therefore $y = 0$.

Step 6: $t = 1$ and $y = 0$

Since $t \neq y$, the new weights are,

$$w_{i(new)} = w_{i(old)} + \alpha t x_i$$

$$w_{1(new)} = w_{1(old)} + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$$

$$w_{2(n)} = w_{2(o)} + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$

$$b_{(new)} = b_{(old)} + \alpha t$$

$$b_{(n)} = b_{(0)} + \alpha t = 0 + 1 \times 1 = 1$$

The new weights and bias are [1 1 1].

The algorithmic steps are repeated for all the input vectors with their initial weights as the previously calculated weights.

By presenting all the input vectors, the updated weights are shown in table below:

Input			Net	Output	Target	Weight Changes			Weights		
x_1	x_2	B	y_{in}	y	t	Δw_1	Δw_2	Δb	w_1	w_2	B
								(0)	0	0	0
1	1	1	0	0	1	1	1	1	1	1	1
−1	1	1	1	1	−1	1	−1	−1	2	0	0
1	−1	1	2	1	−1	−1	1	−1	1	1	−1
−1	−1	1	−3	−1	−1	0	0	0	1	1	−1

This completes one epoch of the training.

The final weights after the first epoch is completed are, $w_1 = 1$, $w_2 = 1$, $b = -1$

Linear Separability

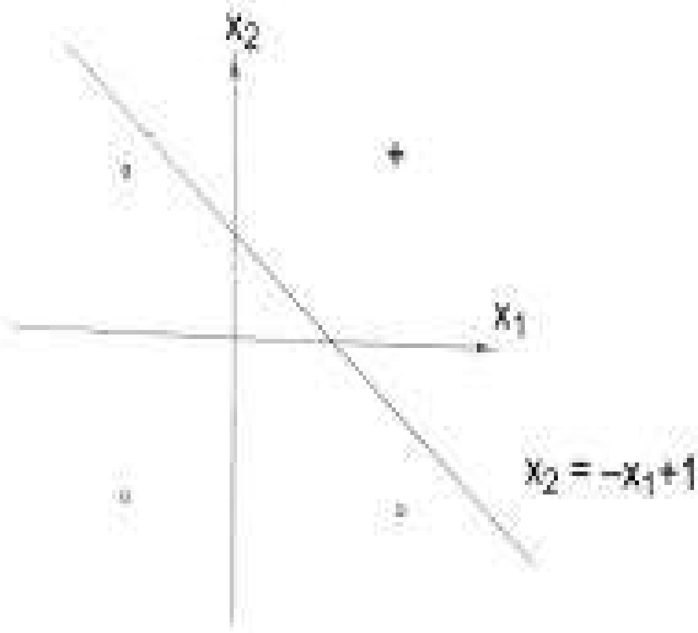
We know that $b + x_1 w_1 + x_2 w_2 = 0$

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

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

$x_2 = -x_1 + 1$ is the separating line equation.

The decision boundary for AND function trained by perceptron network is given as,



In a similar way, the perceptron network can be developed for logic functions OR, NOT, AND NOT etc.

Problem 2

Implement AND function using perceptron with 2 epochs

- The final weights and bias after epoch 1 is used as the initial weight and bias for the second epoch
- $w1 = 1$, $w2 = 1$, $b = -1$

Input			Target	Net input	Calculated output	Weight changes			Weights		
									w_1	w_2	b
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	(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

Problem 3 Implement OR function with binary inputs and bipolar targets using perceptron training algorithm up to 3 epochs

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

Table

x_1	x_2	t
1	1	1
1	0	1
0	1	1
0	0	−1

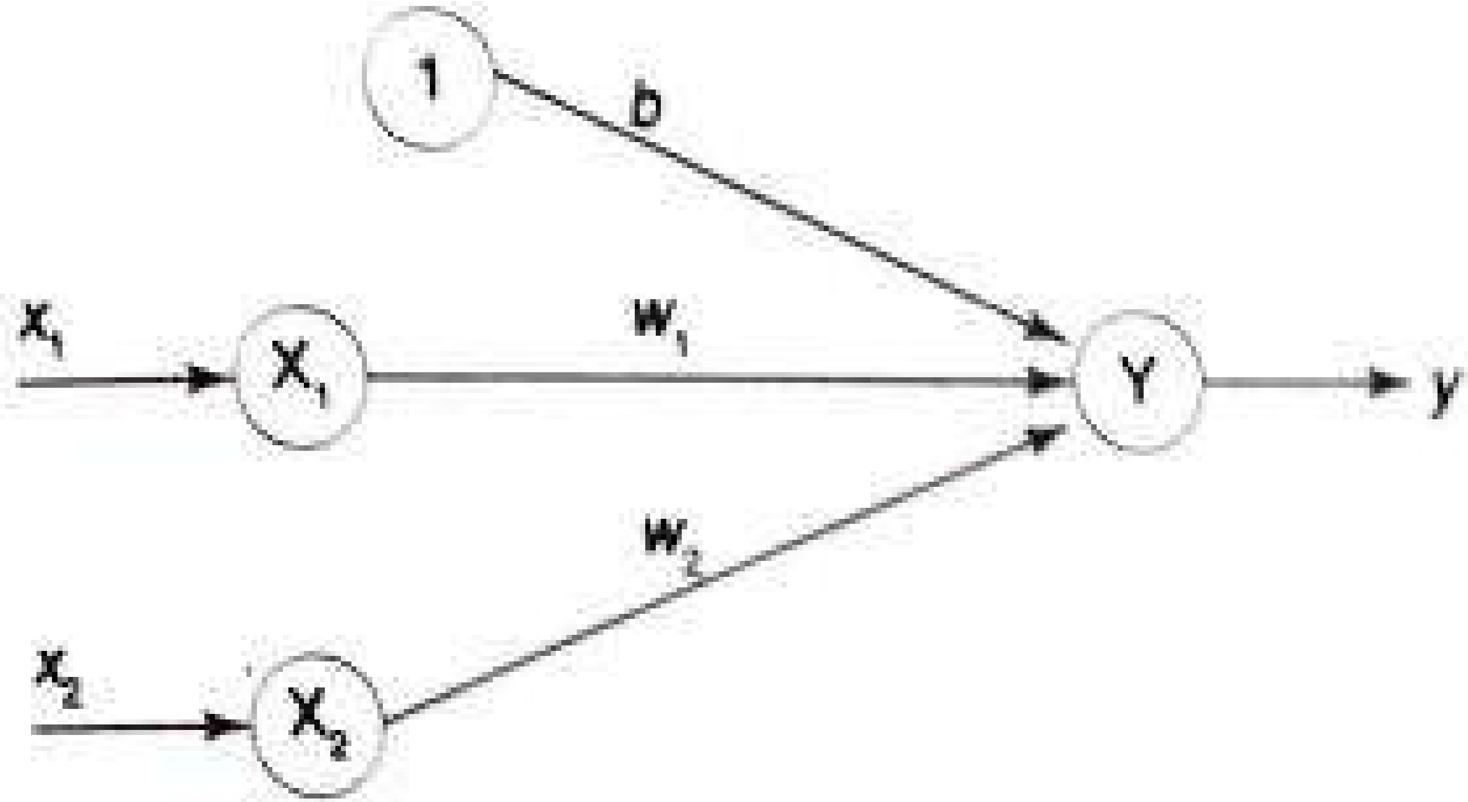


Figure Perceptron network for OR function.

The initial values of the weights and bias are taken as zero, i.e.,

$$w_1 = w_2 = b = 0$$

Also the learning rate is 1 and threshold is 0.2. So, the activation function becomes

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \le y_{in} \le 0.2 \end{cases}$$

The final weights at the end of third epoch are

$$w_1 = 2, w_2 = 1, b = -1$$

Further epochs have to be done for the convergence of the network.

Adaline-Adaptive Linear Neuron

Topics in Adaline

- Introduction
- Theory
- Delta Learning Rule
- Architecture
- Flowchart for training process
- Training algorithm
- Testing algorithm

Introduction

- Adaptive Linear Neuron/ Element
- Single layer ANN developed by Widrow& Hoff at Stanford University in 1960
- Based on Mc-Culloch Pitts neuron
- Net input is not passed through activation function for weight updation ie. $\Delta w=\alpha\ (t-y_{in})\ x_i$
- Used as a classifier for binary classification
- Can learn iteratively and has linear decision boundary.

Adaptive Linear Neuron(Adaline) Theory

- The units with linear activation function are called *linear units*.
- A network with single linear unit is called an *Adaline*.
- It uses bipolar activation for its input signals and its target output.
- The weights between the input and the output units are adjustable.
- Adaline is a net which has only one output unit.
- It is trained using *delta rule*.

Perceptron	Adaline
Uses perceptron learning rule	Uses Delta learning rule
Learning rule originates from Hebbian assumption	Delta rule derived from gradient –descent method
Learning rule stops after a finite number of steps	Gradient –descent method continues
If there is error ,weight and bias are adjusted using $w_i(new)= w_i(old)+\alpha\ t\ x_i$ $b_i(new)= b_i(old)+\alpha\ t$	If there is error ,weight and bias are adjusted using $w_i(new)= w_i(old)+\alpha\ (t-y_{in})x_i$ $b_i(new)= b_i(old)+\alpha\ (t-y_{in})$
Does not allow real values in output	Allow real values in output
Thresholding activation function	Linear activation function

Delta rule for single output unit

- Also known as *Least Mean Square(LMS) rule* or *Widrow Hoff rule*.
-

.
- *Widrow Hoff rule* vs *Perceptron learning rule*:
  1. Perceptron learning rule originates from the Hebbian assumption while the delta rule is derived from the gradient descent method.
  2. Perceptron learning rule stops after a finite number of learning steps. But the gradient descent approach continues forever.
- Updates the weights so as to minimize the difference between the net input and the target value.

- The delta rule for adjusting the weight of i^{th} pattern *(i = 1 to n)* is,

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

where,

Δw_i- weight change

$\alpha-$ learning rate

x_i- activation of input unit

$y_{in}-$ net input to the output unit. *ie*,

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

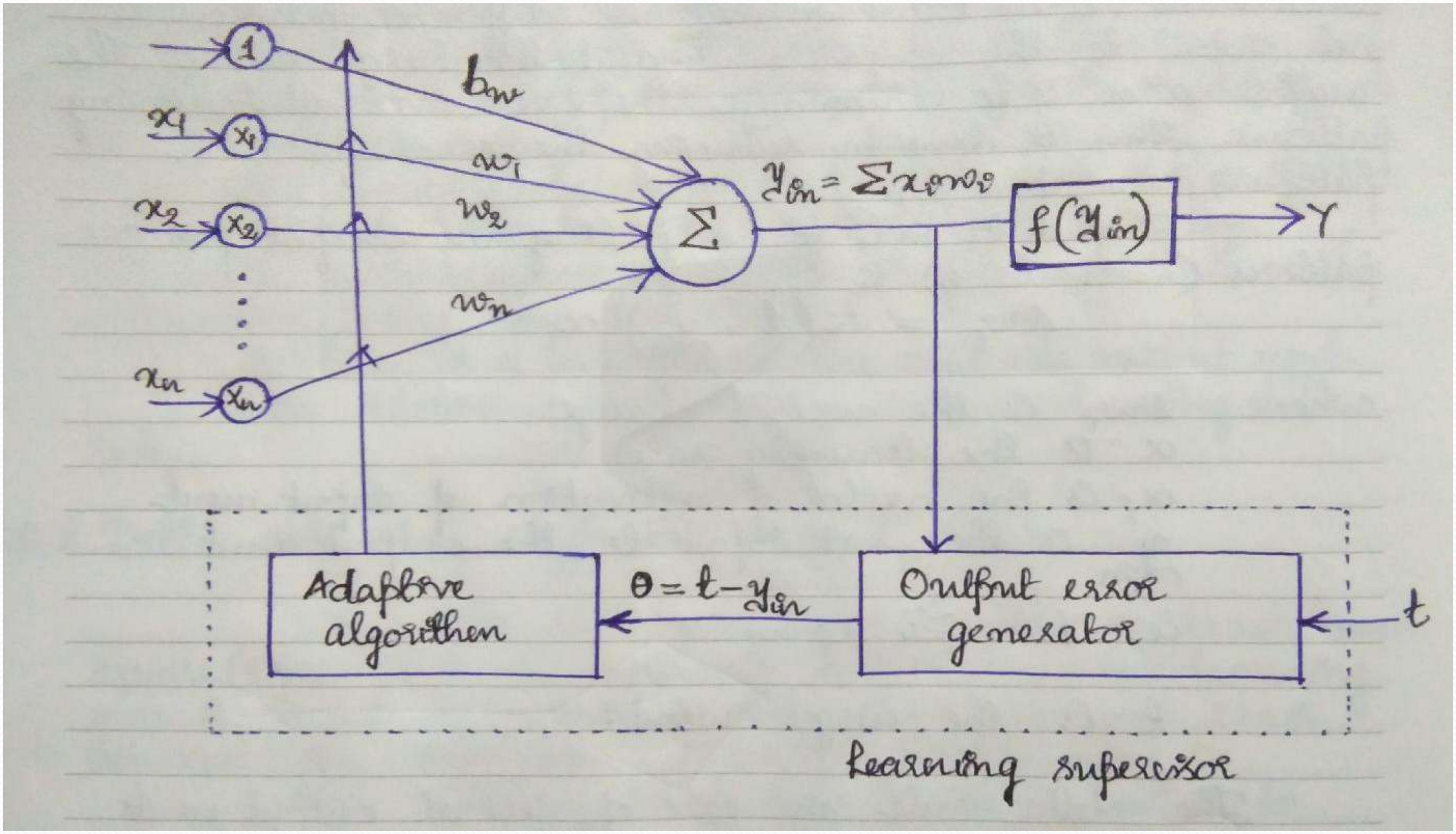
t – target output

- Delta Rule in the case of Several Output Units

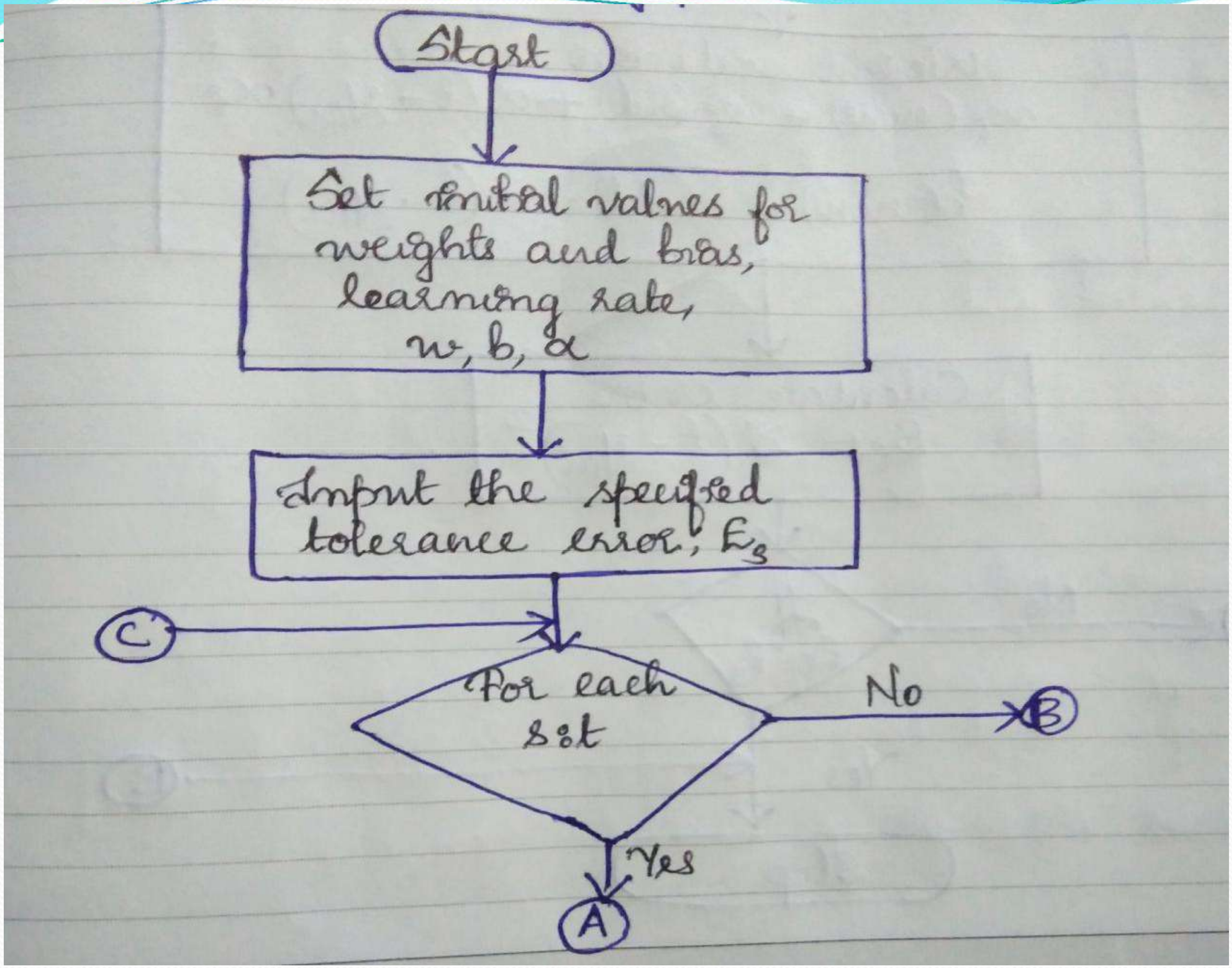
- The delta rule for adjusting the weight from i^{th} input unit to the j^{th} output unit is :

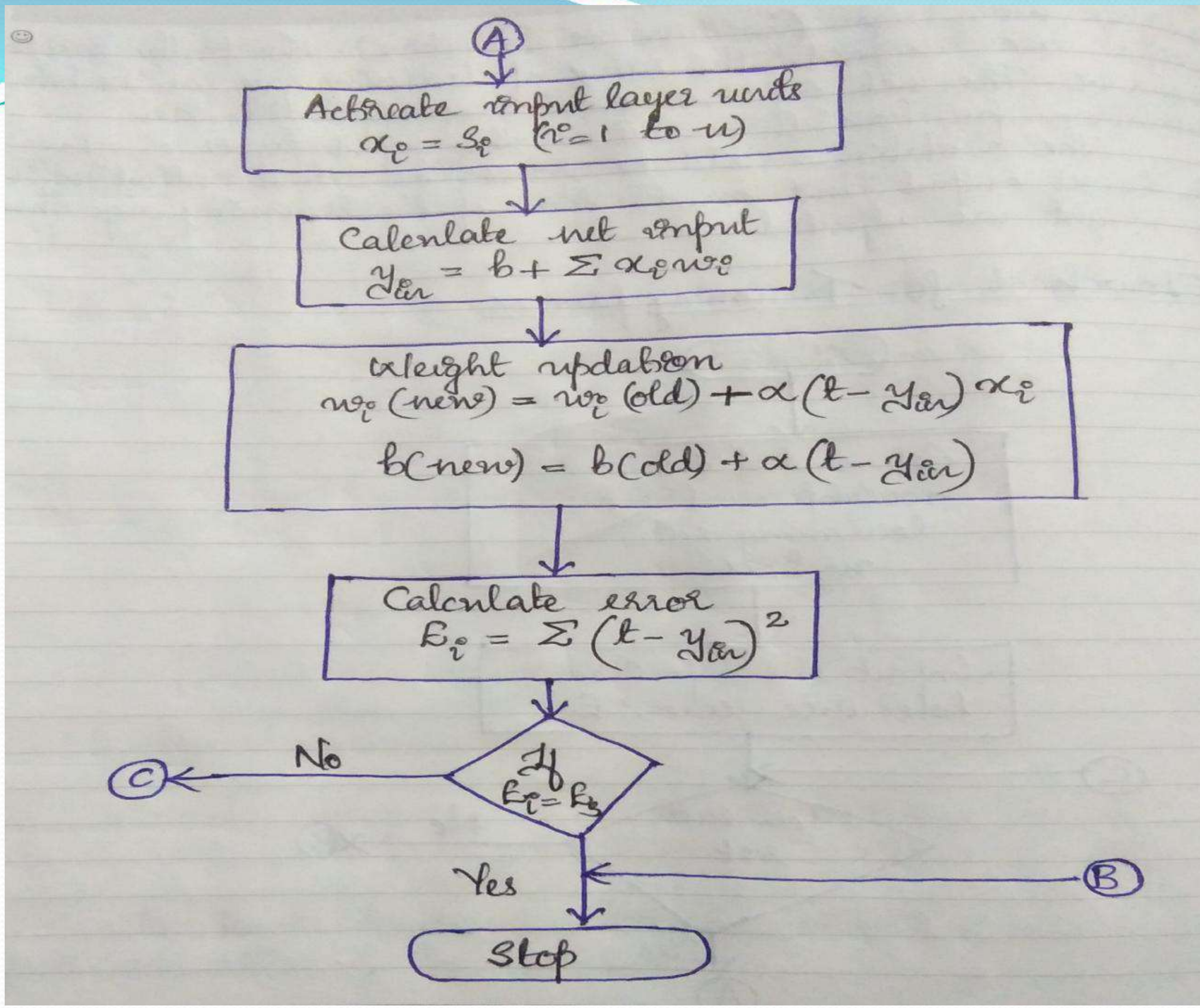
$$\Delta w_{ij} = \alpha(t_j - y_{inj})x_i$$

Architecture



Flowchart of training process




A
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
$E_i = \sum (t - y_{in})^2$
If $E_i = E_s$
No
C
Yes
B
Stop

Training algorithm

- *Step 0* : Weights and bias are set to some random values but not zero. Set the learning rate parameter, .
- *Step 1* : Perform Steps 2-6 when stopping condition is false.
- *Step 2* : Perform Steps 3-5 for each bipolar training pair; $s : t$.
- *Step 3* : Set activations for input units $i = 1 \ to \ n$:

$$x_i = s_i$$

- *Step 4* : Calculate the net input to the output unit:

$$y_{in}$$

Step 5 : Update the weights and bias for $i = 1$ *to* n :

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

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

Where α lies between 0.1 and 1.0

Step 6 : If the highest weight change that occurred during training is smaller than a specified tolerance then stop the training process; else continue.

Testing algorithm

- *Step 0* : Initial weights is equal to the final weights obtained during training.
- *Step 1* : Perform Steps 2- 4 for each bipolar input vector; x.
- *Step 2* : Set activations of the input units to x.
- *Step 3* : Calculate the net input to the output unit: y_{in}
- *Step 4* : Apply the activation function over the net input calculated:

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

Problems

Implement OR function with bipolar inputs and targets using Adaline network.

Solution: The truth table for OR function with bipolar inputs and targets is shown in Table 10.

Table 10

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

Initially all the weights and links are assumed to be small random values, say 0.1, and the learning rate is also set to 0.1. Also here the least mean square error may be set. The weights are calculated until the least mean square error is obtained.

The initial weights are taken to be $w_1 = w_2 = b = 0.1$ and the learning rate $\alpha = 0.1$. For the first input sample, $x_1 = 1, x_2 = 1, t = 1$, we calculate the net input as

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i = b + \sum_{i=1}^{2} x_i w_i$$

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

$$= 0.1 + 1 \times 0.1 + 1 \times 0.1 = 0.3$$

Now compute $(t - y_{in}) = (1 - 0.3) = 0.7$. Updating the weights we obtain,

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

where $\alpha(t - y_{in})x_i$ is called as weight change Δw_i. The new weights are obtained as

$$\begin{aligned} w_1(\text{new}) &= w_1(\text{old}) + \Delta w_1 = 0.1 + 0.1 \times 0.7 \times 1 \\ &= 0.1 + 0.07 = 0.17 \\ w_2(\text{new}) &= w_2(\text{old}) + \Delta w_2 = 0.1 \\ &\quad + 0.1 \times 0.7 \times 1 = 0.17 \\ b(\text{new}) &= b(\text{old}) + \Delta b = 0.1 + 0.1 \times 0.7 = 0.17 \end{aligned}$$

where

$$\Delta w_1 = \alpha(t - y_{in})x_1$$
$$\Delta w_2 = \alpha(t - y_{in})x_2$$
$$\Delta b = \alpha(t - y_{in})$$

Now we calculate the error:

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

The final weights after presenting first input sample are

$$w = [0.17 \quad 0.17 \quad 0.17]$$

and error $E = 0.49$.

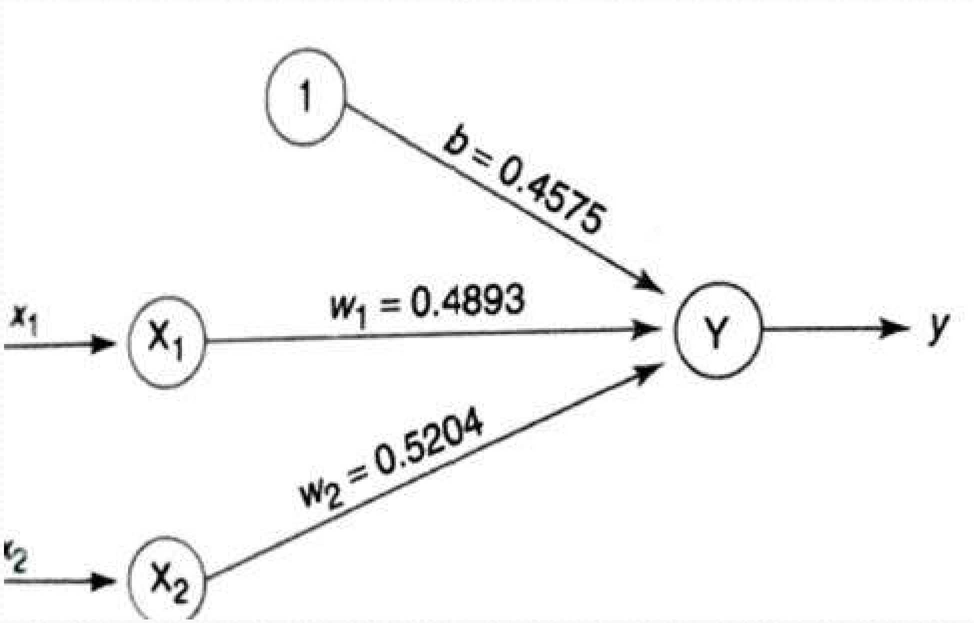
These calculations are performed for all the input samples and the error is calculated. One epoch is completed when all the input patterns are presented. Summing up all the errors obtained for each input sample during one epoch will give the total mean square error of that epoch. The network training is continued until this error is minimized to a very small value.

Inputs			Target	Net input		Weight changes			Weights			Error
x_1	x_2	1	t	y_{in}	$(t - y_{in})$	Δw_1	Δw_2	Δb	w_1 (0.1	w_2 0.1	b 0.1)	$(t - y_{in})^2$
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	−0.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
−1	−1	1	−1	0.0043	−1.0043	0.1004	0.1004	−0.1004	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	−0.7931	0.0793	0.0793	0.2795	0.3044	0.4198	0.629
−1	−1	1	−1	−0.1641	−0.8359	0.0836	0.0836	−0.0836	0.3631	0.388	0.336	0.699
EPOCH-3												
1	1	1	1	1.0873	−0.0873	−0.087	−0.087	−0.087	0.3543	0.3793	0.3275	0.0076
1	−1	1	1	0.3025	+0.6975	0.0697	−0.0697	0.0697	0.4241	0.3096	0.3973	0.487
−1	1	1	1	0.2827	0.7173	−0.0717	0.0717	0.0717	0.3523	0.3813	0.469	0.515
−1	−1	1	−1	−0.2647	−0.7353	0.0735	0.0735	−0.0735	0.4259	0.4548	0.3954	0.541
EPOCH-4												
1	1	1	1	1.2761	−0.2761	−0.0276	−0.0276	−0.0276	0.3983	0.4272	0.3678	0.076
1	−1	1	1	0.3389	0.6611	0.0661	−0.0661	0.0661	0.4644	0.3611	0.4339	0.437
−1	1	1	1	0.3307	0.6693	−0.0669	0.0669	0.0699	0.3974	0.428	0.5009	0.448
−1	−1	1	−1	−0.3246	−0.6754	0.0675	0.0675	−0.0675	0.465	0.4956	0.4333	0.456
EPOCH-5												
1	1	1	1	1.3939	−0.3939	−0.0394	−0.0394	−0.0394	0.4256	0.4562	0.393	0.155
1	−1	1	1	0.3634	0.6366	0.0637	−0.0637	0.0637	0.4893	0.3925	0.457	0.405
−1	1	1	1	0.3609	0.6391	−0.0639	0.0639	0.0639	0.4253	0.4654	0.5215	0.408
−1	−1	1	−1	−0.3603	−0.6397	0.064	0.064	−0.064	0.4893	0.5204	0.4575	0.409

Total mean square error after each epoch is given as

Epoch	Total mean square error
Epoch 1	3.02
Epoch 2	1.938
Epoch 3	1.5506
Epoch 4	1.417
Epoch 5	1.377

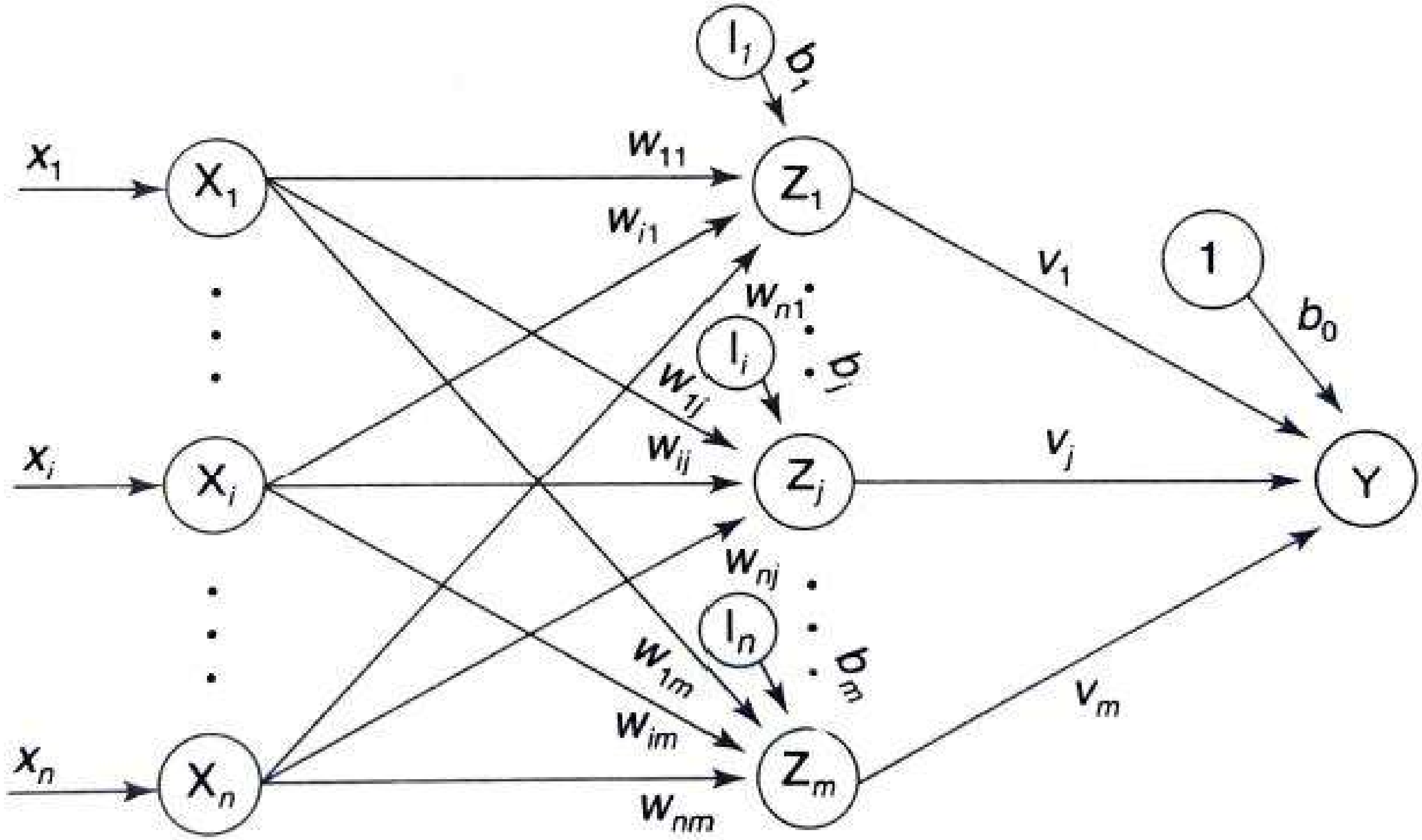
Network architecture of ADALINE



Madaline

- Stands for Multiple Adaptive Linear Neuron
- Developed by Ridgway, Hoff and Glanz
- Combination of Adalines
- Also called multilayered Adalines
- Madaline= i/ps+ Adaline elements+ o/p
- Training process of Madaline is similar to that of Adaline

Architecture



Architecture of Madaline layer.

Training Algorithm

In this training algorithm, only the weights between the hidden layer and the input layer are adjusted, and the weights for the output units are fixed. The weights $v_1, v_2, \ldots, v_m$ and the bias b_0 that enter into output unit Y are determined so that the response of unit Y is 1. Thus, the weights entering Y unit may be taken as

$$v_1 = v_2 = \cdots = v_m = \frac{1}{2}$$

and the bias can be taken as

$$b_0 = \frac{1}{2}$$

The activation for the Adaline (hidden) and Madaline (output) units is given by

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

Step 0: Initialize the weights. The weights entering the output unit are set as above. Set initial small random values for Adaline weights. Also set initial learning rate α.

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.

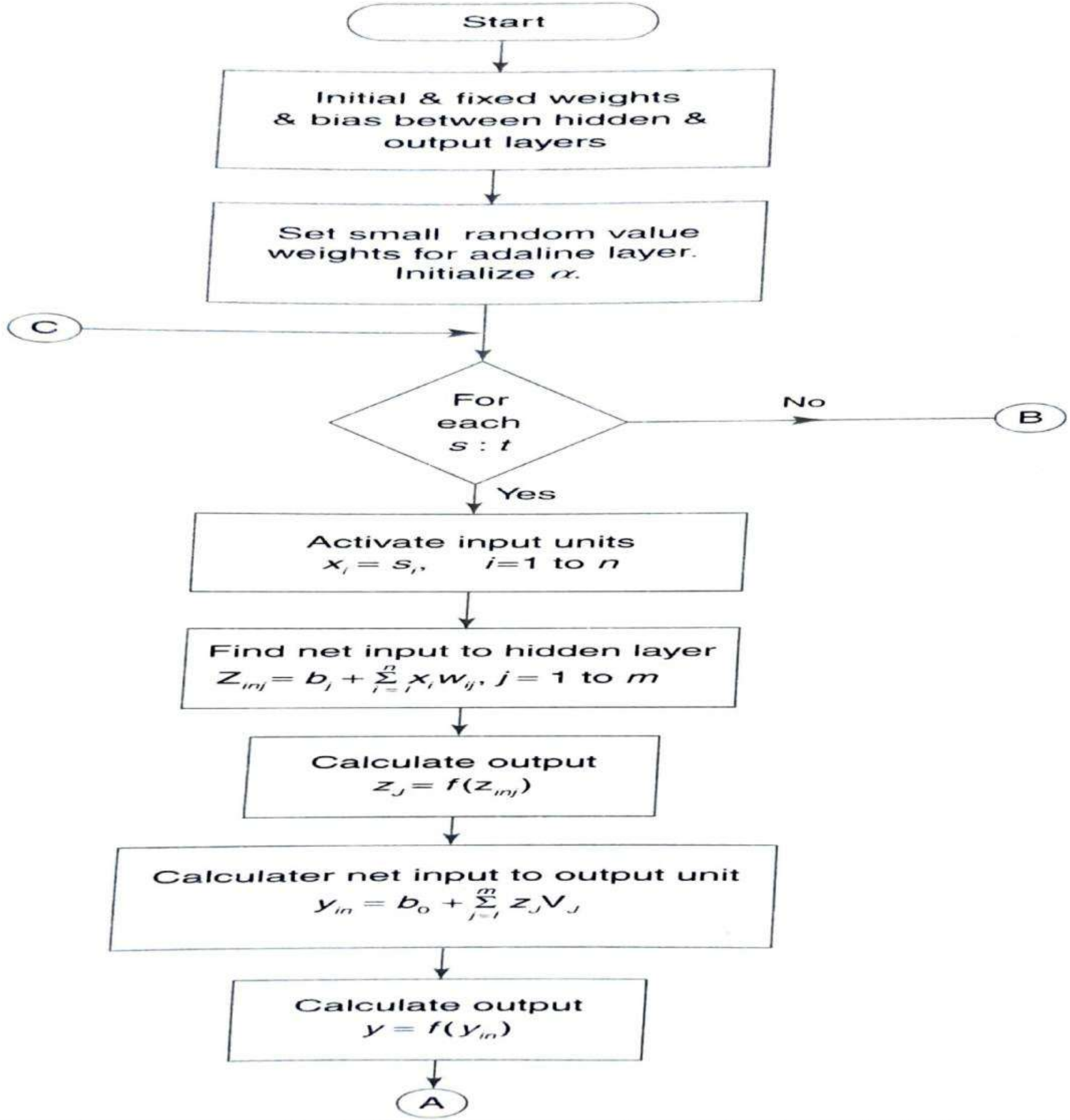
1. If $t = y$, no weight updation is required.
2. If $t \neq y$ and $t = +1$, update weights on z_j, where net input is closest to 0 (zero):

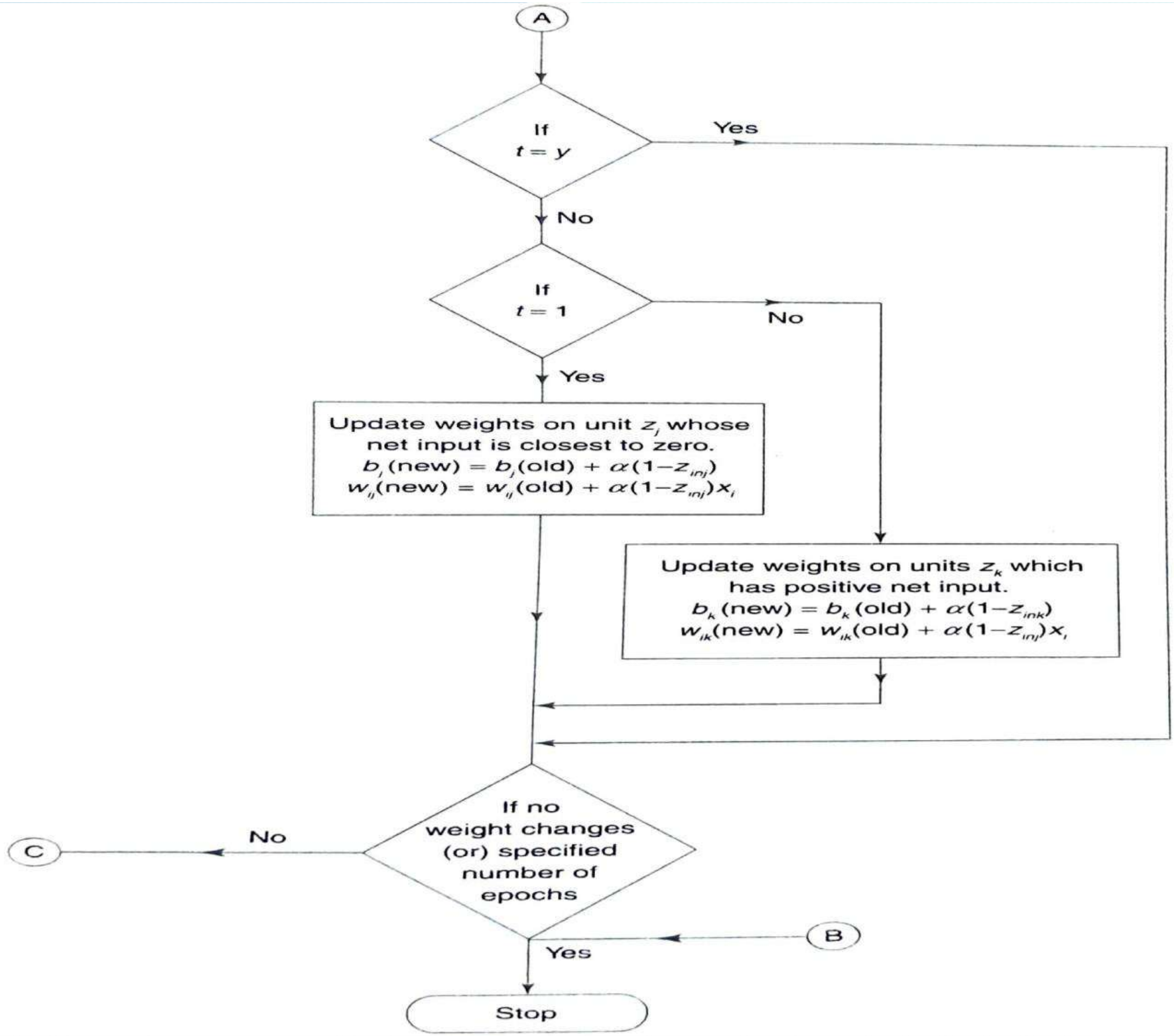
$$b_j(\text{new}) = b_j(\text{old}) + \alpha\,(1 - z_{inj})$$
$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha\,(1 - z_{inj})x_i$$

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 level or if a specified maximum number of iterations of weight updation have been performed then stop, or else continue).


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 $x_i = s_i$, $i=1$ 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.
$b_j(\text{new}) = b_j(\text{old}) + \alpha(1 - z_{inj})$
$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha(1 - z_{inj})x_i$
Update weights on units z_k which has positive net input.
$b_k(\text{new}) = b_k(\text{old}) + \alpha(1 - z_{ink})$
$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(1 - z_{inj})x_i$
If no weight changes (or) specified number of epochs
No
C
Yes
B
Stop

Back Propagation Network

Topics in BPN

- Introduction
- Theory
- Architecture
- Flowchart for training process
- Training algorithm
- Testing Algorithm
- Learning factors of BPN

Introduction

- Most common network in real-time applications
- Multilayer feed forward network
- Error is propagated backward from output unit to hidden unit
- Uses continuous differentiable activation function
- Learning rule is Gradient –descent method

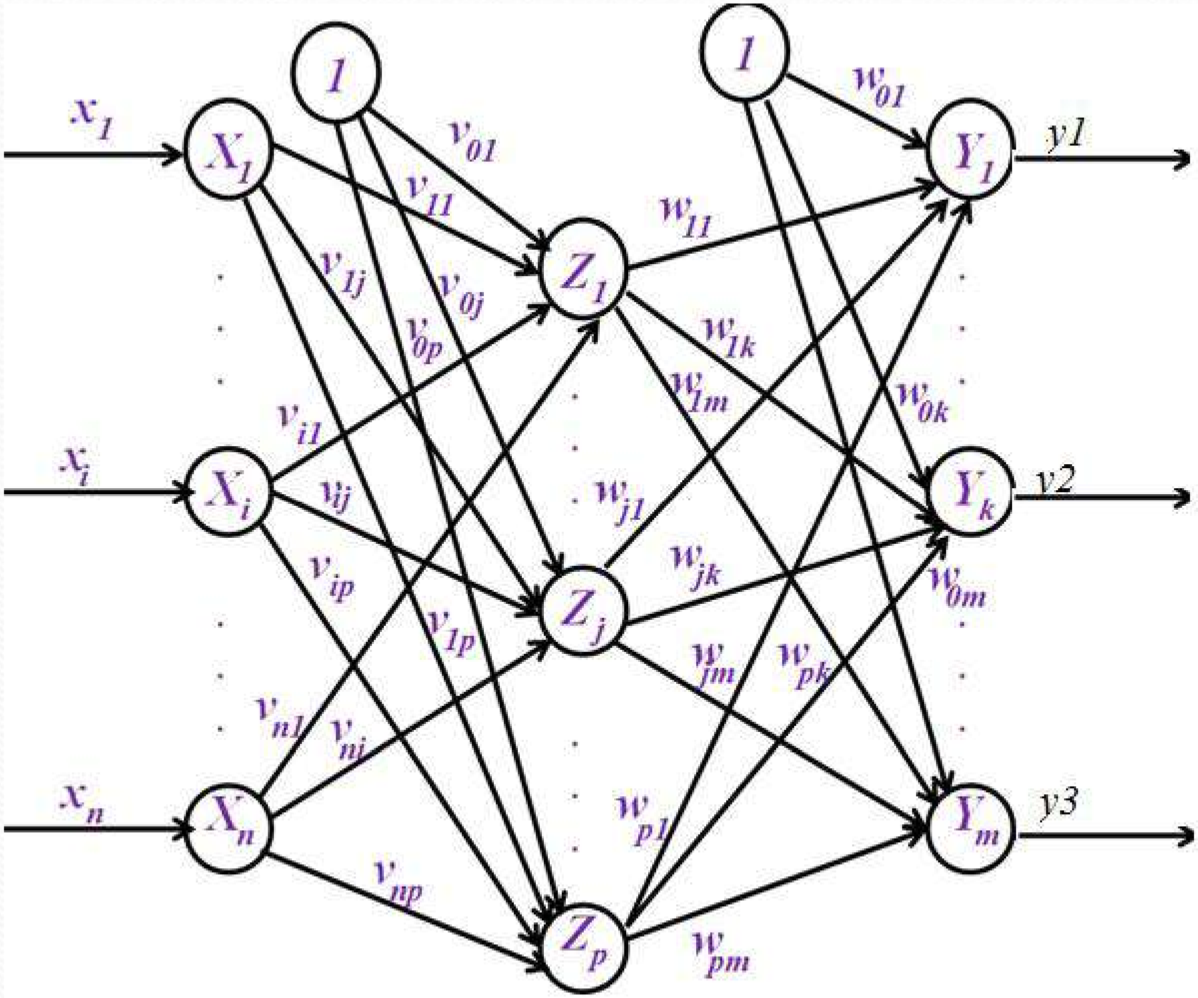
Back Propagation Network -Theory

- This learning algorithm is applied to multilayer feed forward networks consisting of processing elements with continuous differentiable activation functions.
- The networks associated with back propagation learning algorithm are called back propagation networks(*BPNs*).
- Algorithm provides a procedure for changing the weights to classify the given input patterns correctly.
- It uses *gradient descent method* .
- This is a method where the error is propagated back to the hidden unit.

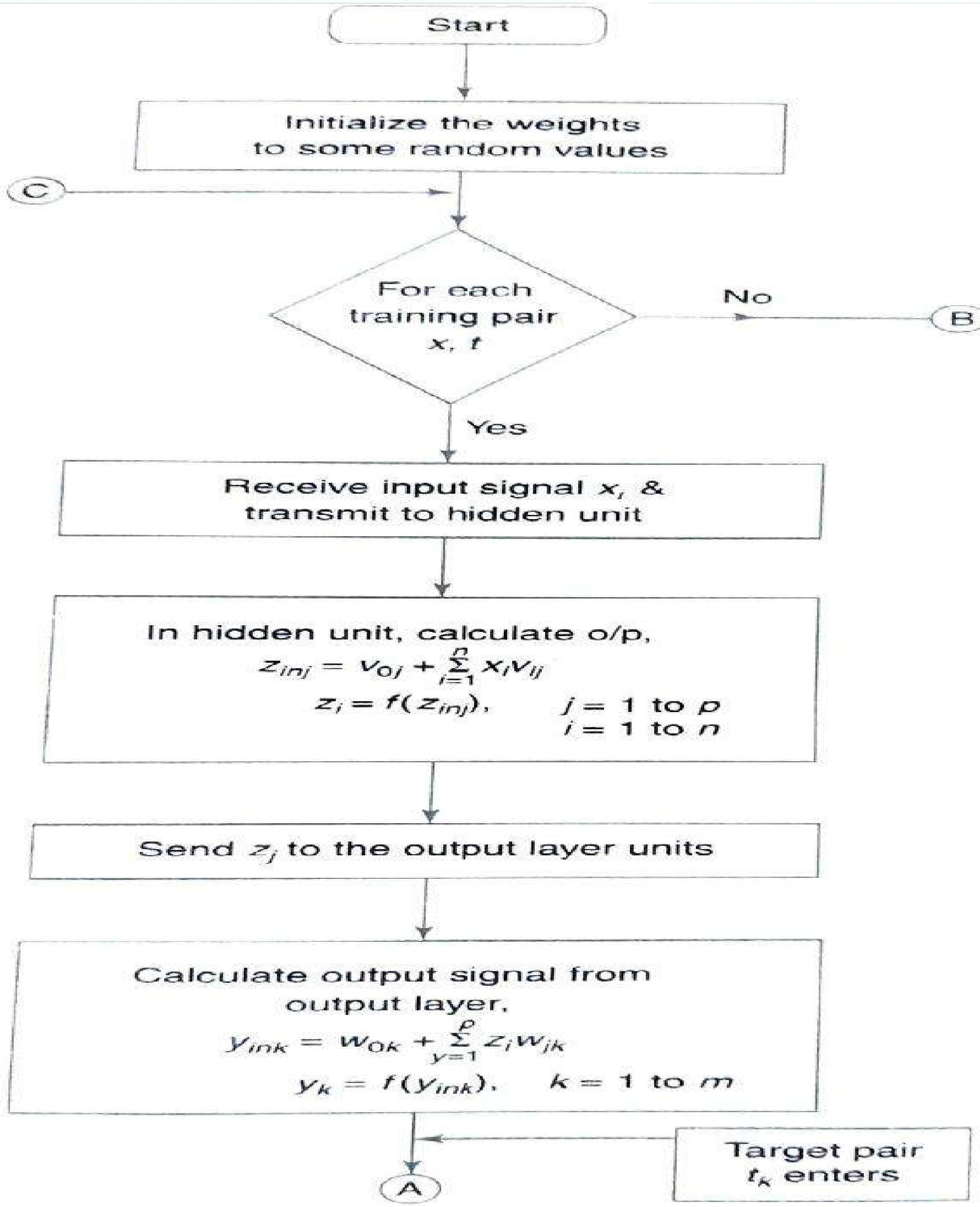
- Generalization is one of the major advantage of BPN-ability of the model to respond to new data/ unknown data and make accurate predictions
- Complexity in training of the network increases as the no of hidden layers increases
- Training of BPN is done in 3 stages/ phases

1. Feed Forward of the input pattern (from i/p to hidden)
2. Back propagation of errors (from o/p to hidden)
3. Weight and bias updating

Architecture



Flow chart


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,
$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$
$z_j = f(z_{inj})$, $j = 1$ to p
$i = 1$ to n
Send z_j to the output layer units
Calculate output signal from output layer,
$y_{ink} = w_{0k} + \sum_{y=1}^{p} z_j w_{jk}$
$y_k = f(y_{ink})$, $k = 1$ 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)

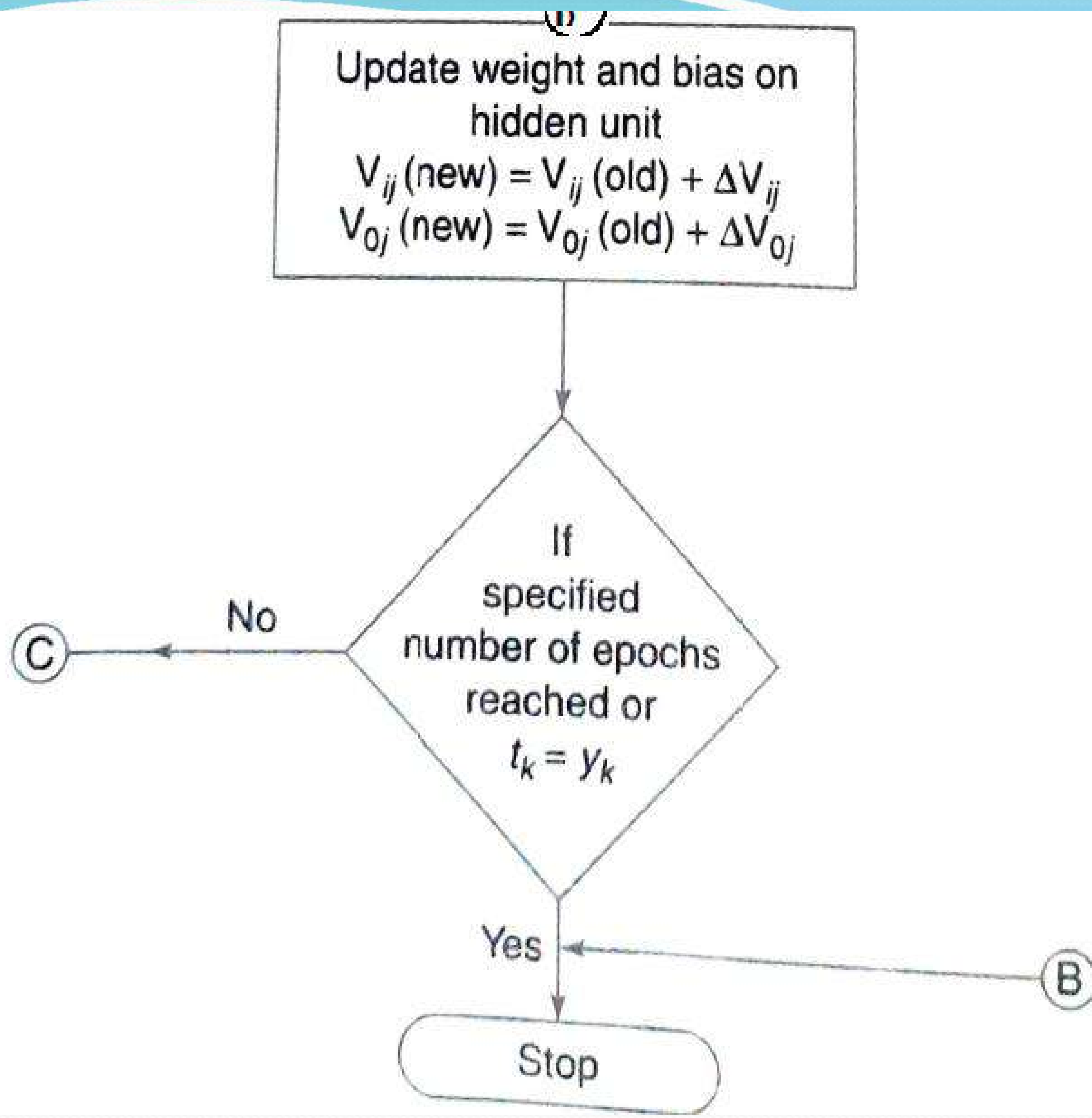
Find weight & bias correction term
$\Delta w_{jk} = \alpha \delta_k z_j,\ \Delta w_{0k} = \alpha \delta_k$

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

D


Update weight and bias on hidden unit
$V_{ij}(\text{new}) = V_{ij}(\text{old}) + \Delta V_{ij}$
$V_{0j}(\text{new}) = V_{0j}(\text{old}) + \Delta V_{0j}$
If specified number of epochs reached or $t_k = y_k$
No
C
Yes
B
Stop

Training Algorithm

- *Step 0*: Initialize weights and learning rate.
- *Step 1*: Perform $Steps 2-9$ when stopping condition is false.
- *Step 2*: Perform $Steps 3-8$ for each training pair.

Feed–forward phase(Phase I)

- *Step 3*: Each input unit receives input signal x_i and sends it to the hidden unit $(i=1 \text{ to } n)$.
- *Step 4*: Each hidden unit z_j $(j=1 \text{ to } p)$ sums its weighted input signals to calculate net input:

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

Calculate output of the hidden unit by applying activation function,

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

Send zj to output unit

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

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

and apply the activation function to compute output signal:

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

Back–propagation of error(Phase II)

- *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:

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

Then update the weights and bias:

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

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:

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

Then update the weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i$$
$$\Delta v_{oj} = \alpha \delta_j$$

Weight and bias updation(Phase III)

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

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

Each output unit z_j *(j=1 to p)* updates the bias and weights:

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

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

Testing Algorithm of BPN

- *Step 0*: Initialize the weights. The weights are taken from the training algorithm.
- *Step 1*: Perform $Steps 2-4$ for each input vector.
- *Step 2*: Set the activation of input unit for x_i *(i=1 to n)*.
- *Step 3*: Calculate the net input to hidden unit x and its output. For *j=1 to p*,

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

- *Step 4*: Now compute the output of the output layer unit. For *k=1 to m*,

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

Use sigmoidal activation functions for calculating the output.

Sigmoidal activation functions

1 *Binary sigmoid function*

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

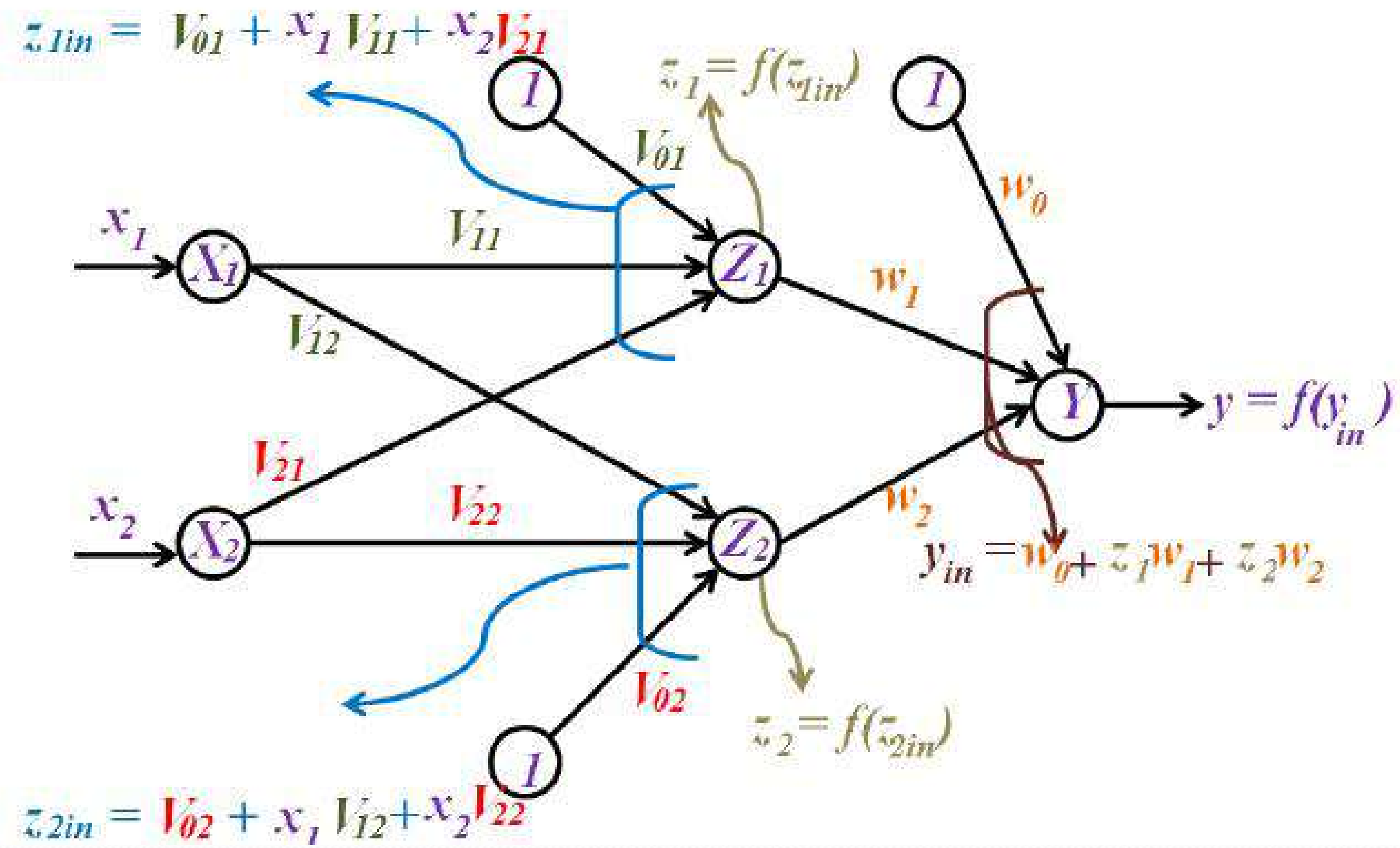
$$f'(x) = f(x)[1 - f(x)]$$

2 *Bipolar sigmoid function*

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

$$f'(x) = 0.5[1 + f(x)][1 - f(x)]$$

Summarization of algorithm



- Compute the error, δ_k, here, $k = 1$, only one output neuron.

$$\delta_k = (t_k - y_k)f'(y_{ink})$$
$$\delta_1 = (t - y)f'(y_{in})$$

- changes in weight between hidden and output layer:

$$\Delta w_0 = \alpha\delta_1$$
$$\Delta w_1 = \alpha\delta_1 z_1$$
$$\Delta w_2 = \alpha\delta_1 z_2$$

- Compute the error portion δ_j between input and hidden layer, j=1,2(ie, z_{in1}, z_{in2}):

$$\delta_j = \delta_{inj} f'(z_{inj})$$
$$\delta_{inj} = \sum_{k=1}^{w_{jk}}$$

Here, k=1(only one output neuron)

$$\delta_{inj} = \delta_1 w_{j1}$$
$$\delta_{in1} = \delta_1 w_{11}, \delta_{in2} = \delta_1 w_{21}$$
$$w_{11} = w_1, w_{21} = w_2$$

- Error,

$$\delta_1 = \delta_{in1} f'(z_{in1})$$
$$\delta_2 = \delta_{in2} f'(z_{in2})$$

- Change in weights between input and hidden layer:

$$\Delta v_{11} = \alpha \delta_1 x_1$$
$$\Delta v_{21} = \alpha \delta_1 x_2$$
$$\Delta v_{01} = \alpha \delta_1$$
$$\Delta v_{12} = \alpha \delta_2 x_1$$
$$\Delta v_{22} = \alpha \delta_2 x_2$$
$$\Delta v_{02} = \alpha \delta_2$$

- Compute the final weights of the network:

$$v_{11}(new) = v_{11}(old) + \Delta v_{11}$$
$$v_{12}(new) = v_{12}(old) + \Delta v_{12}$$
$$v_{21}(new) = v_{21}(old) + \Delta v_{21}$$
$$v_{22}(new) = v_{22}(old) + \Delta v_{22}$$
$$w_{1}(new) = w_{1}(old) + \Delta w_{1}$$
$$w_{2}(new) = w_{2}(old) + \Delta w_{2}$$
$$w_{0}(new) = w_{0}(old) + \Delta w_{0}$$
$$v_{01}(new) = v_{01}(old) + \Delta v_{01}$$
$$v_{02}(new) = v_{02}(old) + \Delta v_{02}$$

Learning factors of BPN

- *Initial Weights*
  - Initialized at small random values.
  - The choice of initial weight determines how fast the network converges.
  - One method of choosing the weight w_{ij} is choosing it in the range,

$$[\frac{-3}{\sqrt{o_i}}, \frac{3}{\sqrt{o_i}}]$$

    where o_i is the number of processing elements j that feed–forward to processing element i.
  - *Nyugen– Widrow initialization:* Based on the geometric analysis of the response of hidden neurons to a single input.
  - Random initialization of weights connecting the input neurons to the hidden units is obtained by,

$$v_{ij}(new) = \gamma \frac{v_{ij}(old)}{||v_j(old)||}$$

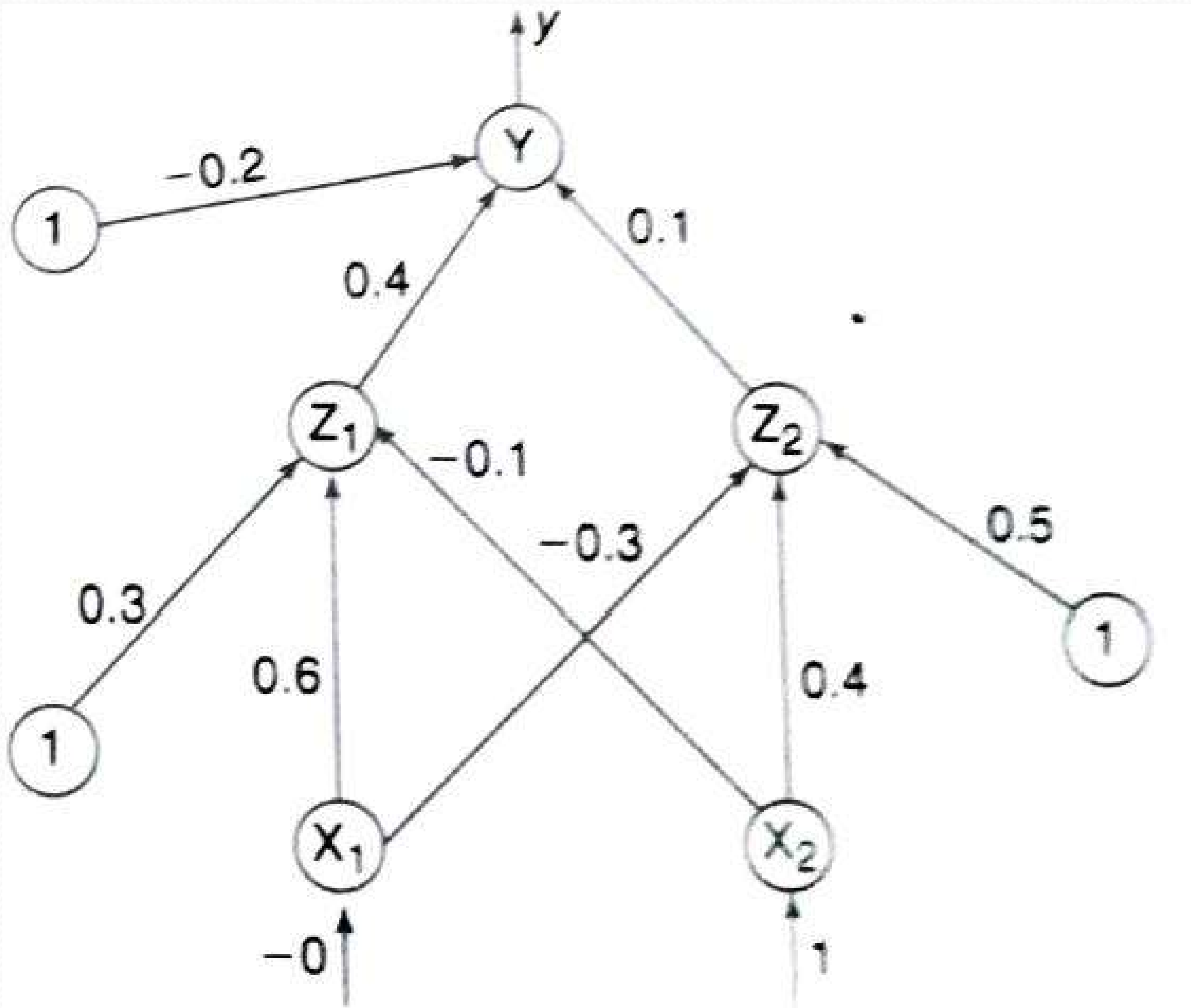
$$\gamma = 0.7(P)^{1/n}$$

- *Learning rate, α*
  - The range of α from 10^{-3} to 1.0 has been used successfully.
- *Momentum factor, η*
  - $\eta \in [0, 1]$ and the value of 0.9 is often used for the momentum factor.
  - Weight updation formulas used here are,

$$w_{jk}(t+1) = w_{jk}(t) + \alpha\delta_k z_j + \eta[w_{jk}(t) - w_{jk}(t-1)] \text{ and}$$
$$v_{ij}(t+1) = v_{ij}(t) + \alpha\delta_j x_i + \eta[v_{ij}(t) - v_{ij(t-1)}]$$

- *Generalization*
- *Number of training data*
- *Number of hidden layer nodes*

Problem: Using BPN find the new weight of the network shown. It is presented with the i/p pattern [0,1] and target o/p is 1. Use learning rate α= 0.25 and binary sigmoidal activation function



- Initial weights are $[v_{11}\ v_{21}\ v_{01}]= [0.6 \quad -0.1 \quad 0.3]$ and

  $[v_{12}\ v_{22}\ v_{02}]= [-0.3 \quad 0.4 \quad 0.5]$ and

  $[w_1\ w_2\ w_{01}= [0.4 \quad 0.1 \quad -0.2]$

Learning rate $\alpha= 0.25$

Activation function is binary sigmoidal function

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

Given output sample $[x1,x2\ ]= [0,1]$ and target $t=1$

- Calculate the net input: For z_1 layer

$$
\begin{aligned}
z_{in1} &= v_{01} + x_1 v_{11} + x_2 v_{21} \\
&= 0.3 + 0 \times 0.6 + 1 \times -0.1 = 0.2
\end{aligned}
$$

For z_2 layer

$$
\begin{aligned}
z_{in2} &= v_{02} + x_1 v_{12} + x_2 v_{22} \\
&= 0.5 + 0 \times -0.3 + 1 \times 0.4 = 0.9
\end{aligned}
$$

Applying activation to calculate the output, we obtain

$$z_1 = f(z_{in1}) = \frac{1}{1 + e^{-z_{in1}}} = \frac{1}{1 + e^{-0.2}} = 0.5498$$

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

- Calculate the net input entering the output layer. For y layer

$$\begin{aligned} y_{in} &= w_0 + z_1 w_1 + z_2 w_2 \\ &= -0.2 + 0.5498 \times 0.4 + 0.7109 \times 0.1 \\ &= 0.09101 \end{aligned}$$

Applying activations to calculate the output, we obtain

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

- Compute the error portion δ_k:

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

Now

$$f'(y_{in}) = f(y_{in})[1 - f(y_{in})] = 0.5227[1 - 0.5227]$$
$$f'(y_{in}) = 0.2495$$

This implies

$$\delta_1 = (1 - 0.5227)\ (0.2495) = 0.1191$$

Find the changes in weights between hidden and output layer:

$$\Delta w_1 = \alpha \delta_1 z_1 = 0.25 \times 0.1191 \times 0.5498$$
$$= 0.0164$$

$$\Delta w_2 = \alpha \delta_1 z_2 = 0.25 \times 0.1191 \times 0.7109$$
$$= 0.02117$$

$$\Delta w_0 = \alpha \delta_1 = 0.25 \times 0.1191 = 0.02978$$

- Compute the error portion δ_j between input and hidden layer ($j = 1$ to 2):

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

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

$$\delta_{inj} = \delta_1 w_{j1} \quad [\because \text{ only one output neuron}]$$

$$\Rightarrow \delta_{in1} = \delta_1 w_{11} = 0.1191 \times 0.4 = 0.04764$$

$$\Rightarrow \delta_{in2} = \delta_1 w_{21} = 0.1191 \times 0.1 = 0.01191$$

Error, $\delta_1 = \delta_{in1} f'(z_{in1})$

$$f'(z_{in1}) = f(z_{in1})[1 - f(z_{in1})]$$
$$= 0.5498[1 - 0.5498] = 0.2475$$
$$\delta_1 = \delta_{in1} f'(z_{in1})$$
$$= 0.04764 \times 0.2475 = 0.0118$$

Error, $\delta_2 = \delta_{in2} f'(z_{in2})$

$$f'(z_{in2}) = f(z_{in2})[1 - f(z_{in2})]$$
$$= 0.7109[1 - 0.7109] = 0.2055$$
$$\delta_2 = \delta_{in2} f'(z_{in2})$$
$$= 0.01191 \times 0.2055 = 0.00245$$

Now find the changes in weights between input and hidden layer:

$$\Delta v_{11} = \alpha \delta_1 x_1 = 0.25 \times 0.0118 \times 0 = 0$$

$$\Delta v_{21} = \alpha \delta_1 x_2 = 0.25 \times 0.0118 \times 1 = 0.00295$$

$$\Delta v_{01} = \alpha \delta_1 = 0.25 \times 0.0118 = 0.00295$$

$$\Delta v_{12} = \alpha \delta_2 x_1 = 0.25 \times 0.00245 \times 0 = 0$$

$$\Delta v_{22} = \alpha \delta_2 x_2 = 0.25 \times 0.00245 \times 1 = 0.0006125$$

$$\Delta v_{02} = \alpha \delta_2 = 0.25 \times 0.00245 = 0.0006125$$

- Compute the final weights of the network:

$$v_{11}(\text{new}) = v_{11}(\text{old}) + \Delta v_{11} = 0.6 + 0 = 0.6$$

$$v_{12}(\text{new}) = v_{12}(\text{old}) + \Delta v_{12} = -0.3 + 0 = -0.3$$

$$v_{21}(\text{new}) = v_{21}(\text{old}) + \Delta v_{21}$$
$$= -0.1 + 0.00295 = -0.09705$$

$$v_{22}(\text{new}) = v_{22}(\text{old}) + \Delta v_{22}$$
$$= 0.4 + 0.0006125 = 0.4006125$$

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0.4 + 0.0164$$
$$= 0.4164$$

$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0.1 + 0.02117$$
$$= 0.12117$$

$$v_{01}(\text{new}) = v_{01}(\text{old}) + \Delta v_{01} = 0.3 + 0.00295$$
$$= 0.30295$$

$$v_{02}(\text{new}) = v_{02}(\text{old}) + \Delta v_{02}$$
$$= 0.5 + 0.0006125 = 0.5006125$$

$$w_0(\text{new}) = w_0(\text{old}) + \Delta w_0 = -0.2 + 0.02978$$
$$= -0.17022$$

APPLICATIONS OF BACKPROPAGATION NETWORK

- Load forecasting problems in power systems.
- Image processing.
- Fault diagnosis and fault detection.
- Gesture recognition, speech recognition.
- Signature verification.
- Bioinformatics.
- Structural engineering design (civil).