Perceptron Learning Rule

For the perceptron learning rule, the learning signal is the difference between the desired and actual neuron's response (Rosenblatt 1958). This example illustrates the perceptron learning rule of the network shown in Figure**.** The set of input training vectors is as follows where:

$$d1= -1,\ d2 = -1,\ d3 = 1,\ C = 0.1,\ W^t = [1\ -1\ 0\ 0.5]$$

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

Step 1 Input is $\mathbf{x}_1$, desired output is d_1:

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = [1 \quad -1 \quad 0 \quad 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

Correction in this step is necessary since $d_1 \neq \text{sgn}(2.5)$. We thus obtai updated weight vector

$$\mathbf{w}^2 = \mathbf{w}^1 + 0.1(-1 - 1)\mathbf{x}_1$$

Plugging in numerical values we obtain

$$\mathbf{w}^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} - 0.2 \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

Step 2 Input is $\mathbf{x}_2$, desired output is d_2. For the present weight vector $\mathbf{w}^2$ we compute the activation value net^2 as follows:

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = [0 \quad 1.5 \quad -0.5 \quad -1] \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} = -1.6$$

Correction is not performed in this step since $d_2 = \text{sgn}(-1.6)$

Step 3 Input is $\mathbf{x}_3$, desired output is d_3, present weight vector is $\mathbf{w}^3$. Computing net^3 we obtain:

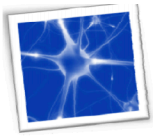
$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = [-1 \quad 1 \quad 0.5 \quad -1] \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} = -2.1$$

Correction is necessary in this step since $d_3 \neq \text{sgn}(-2.1)$. The updated weight values are

$$\mathbf{w}^4 = \mathbf{w}^3 + 0.1(1 + 1)\mathbf{x}_3$$

or

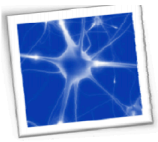
$$\mathbf{w}^4 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + 0.2 \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$



Example

- Consider the 2-dimensional training set $C_1 \cup C_2$,
- $C_1 = \{(1, 1, 1), (1, 1, -1), (1, 0, -1)\}$ with class label 1
- $C_2 = \{(1, -1, -1), (1, -1, 1), (1, 0, 1)\}$ with class label 0
- Train a perceptron on $C_1 \cup C_2$
- First pass is as follows

Input	Weight	Desired	Actual	Update?	New weight
(1, 1, 1)	(1, 0, 0)	1	1	No	(1, 0, 0)
(1, 1, -1)	(1, 0, 0)	1	1	No	(1, 0, 0)
(1,0, -1)	(1, 0, 0)	1	1	No	(1, 0, 0)
(1,-1, -1)	(1, 0, 0)	0	1	Yes	(0, 1, 1)
(1,-1, 1)	(0, 1, 1)	0	0	No	(0, 1, 1)
(1, 0, 1)	(0, 1, 1)	0	1	Yes	(-1, 1, 0)

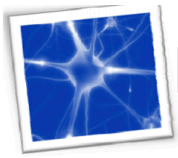


Example

C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}

Fill out this table sequentially (Second pass):

Input	Weight	Desired	Actual	Update?	New weight
(1, 1, 1)	(-1, 1, 0)	1	0	Yes	(0, 2, 1)
(1, 1, -1)	(0, 2, 1)	1	1	No	(0, 2, 1)
(1,0, -1)	(0, 2, 1)	1	0	Yes	(1, 2, 0)
(1,-1, -1)	(1, 2, 0)	0	0	No	(1, 2, 0)
(1,-1, 1)	(1, 2, 0)	0	0	No	(1, 2, 0)
(1, 0, 1)	(1, 2, 0)	0	1	Yes	(0, 2, -1)



Example

C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
Fill out this table sequentially (Third pass):

Input	Weight	Desired	Actual	Update?	New weight
(1, 1, 1)	(0, 2, -1)	1	1	No	(0, 2, -1)
(1, 1, -1)	(0, 2, -1)	1	1	No	(0, 2, -1)
(1,0, -1)	(0, 2, -1)	1	1	No	(0, 2, -1)
(1,-1, -1)	(0, 2, -1)	0	0	No	(0, 2, -1)
(1,-1, 1)	(0, 2, -1)	0	0	No	(0, 2, -1)
(1, 0, 1)	(0, 2, -1)	0	0	No	(0, 2, -1)

At epoch 3 no weight changes.
$\Rightarrow$ stop execution of algorithm.
Final weight vector: (0, 2, -1).
$\Rightarrow$ decision hyperplane is $2x_1 - x_2 = 0$.

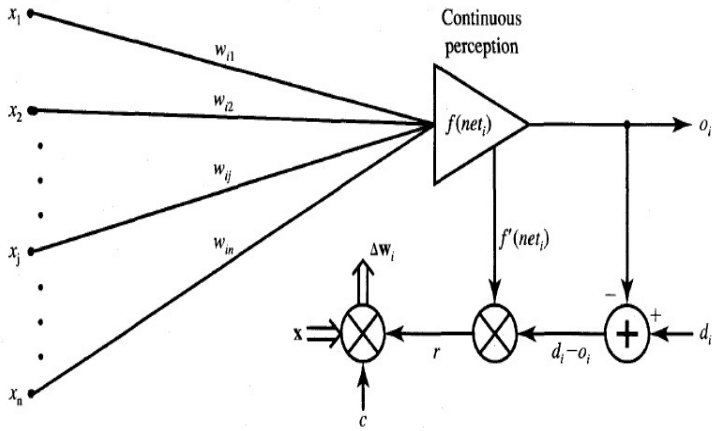
Delta Learning Rule

The delta learning rule is only valid for continuous activation functions as defined before, and in the supervised training mode. The adjustment of learning weight:

Xj

$$\Delta w_{ij} = [d_i - f(\mathbf{w}_i^t \mathbf{x})] f'(\mathbf{w}_i^t \mathbf{x})$$

for this rule is called delta and is defined as follows



This example discusses the delta learning rule as applied to the network shown in Figure. Training input vectors, desired responses, and initial weights are identical to those in Example. The delta learning requires

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

that the value f'(net) be computed in each step.

$$f'(\text{net}) = \frac{1}{2}(1 - o^2)$$

Example/

d1= -1, d2 = -1, d3 = 1, C = 0.1, $\lambda = 1$

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

Step 1 Input is vector $\mathbf{x}_1$, initial weight vector is $\mathbf{w}^1$:

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = 2.5$$
$$o^1 = f(net^1) = 0.848$$
$$f'(net^1) = \frac{1}{2}[1 - (o^1)^2] = 0.140$$
$$\mathbf{w}^2 = c(d_1 - o^1)f'(net^1)\mathbf{x}_1 + \mathbf{w}^1$$
$$= [0.974 \quad -0.948 \quad 0 \quad 0.526]^t$$

Step 2 Input is vector $\mathbf{x}_2$, weight vector is $\mathbf{w}^2$:

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = -1.948$$
$$o^2 = f(net^2) = -0.75$$
$$f'(net^2) = \frac{1}{2}[1 - (o^2)^2] = 0.218$$
$$\mathbf{w}^3 = c(d_2 - o^2)f'(net^2)\mathbf{x}_2 + \mathbf{w}^2$$
$$= [0.974 \quad -0.956 \quad 0.002 \quad 0.531]^t$$

Step 3 Input is x_3, weight vector is $\mathbf{w}^3$:

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = -2.46$$
$$o^3 = f(net^3) = -0.842$$
$$f'(net^3) = \frac{1}{2}[1 - (o^3)^2] = 0.145$$
$$\mathbf{w}^4 = c(d_3 - o^3)f'(net^3)\mathbf{x}_3 + \mathbf{w}^3$$
$$= [0.947 \quad -0.929 \quad 0.016 \quad 0.505]^t$$

Widrow-Hoff Learning Rule

The Widrow-Hoff learning rule (Widrow 1962) or Adaline is applicable for the supervised training of neural networks. It is independent of the activation function of neurons used since it minimizes the squared error between the desired output value d, and the neuron's activation value, net = $w^t x$. The learning signal for this rule is defined as follows:

$$r \triangleq d_i - \mathbf{w}_i^t\mathbf{x}$$

The weight vector increment under this learning rule is

$$\Delta\mathbf{w}_i = c(d_i - \mathbf{w}_i^t\mathbf{x})\mathbf{x}$$

Correlation Learning Rule

By substituting r = di into the general learning rule we obtain the correlation learning rule. The adjustments for the weight vector and the single weights, respectively are

$$\Delta \mathbf{w}_i = cd_i\mathbf{x}$$

$$\Delta w_{ij} = cd_i x_j, \quad \text{for } j = 1, 2, \ldots, n$$

Multi-layer ANN and Backpropagation Learning Rule

All the aforementioned learning rules work with single neuron ANN but don't work with multilayer ANN. A multilayer perceptron (shown in Fig. below) is a feedforward neural network with one or more hidden layers. Typically, the network consists of an input layer of source neurons, at least one middle or hidden layer of computational neurons, and an output layer of computational neurons. The input signals are propagated in a forward direction on a layer-by-layer basis.

Each layer in a multilayer neural network has its own specific function. The input layer accepts input signals from the outside world and redistributes these signals to all neurons in the hidden layer. Actually, the input layer rarely includes computing neurons, and thus does not process input patterns. The output layer accepts output signals, or in other words a stimulus pattern, from the hidden layer and establishes the output pattern of the entire network.

Neurons in the hidden layer detect the features; the weights of the neurons represent the features hidden in the input patterns. These features are then used by the output layer in determining the output pattern. With one hidden layer, we can represent any continuous function of the input signals, and with two hidden layers even discontinuous functions can be represented.

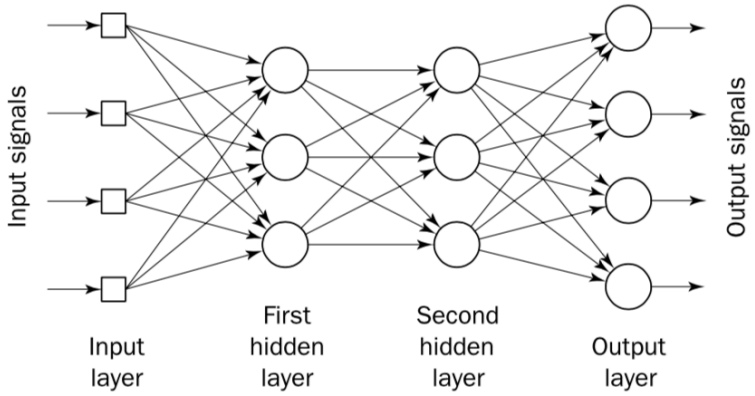


Figure: Multilayer perceptron with two hidden layers

The determination of the error is a recursive process which start with the o/p units and the error is back propagated to the I/p units. Therefore the rule is called error Back propagation (EBP) or simply Back Propagation (BP). The weight is changed exactly in the same form of the standard DR

$$\Delta w_{ij} = \xi\ \delta_j\ x_i$$

$$\Rightarrow \quad w_{ij}(t+1) = w_{ij}(t)\ +\ \xi\ \delta_j\ x_i$$

There are two other equations that specify the error signal. If a unite is an o/p unit, the error signal is given by:-

$$\delta = (d_j - y_j)\ f_j(\text{net}\ \ j)$$

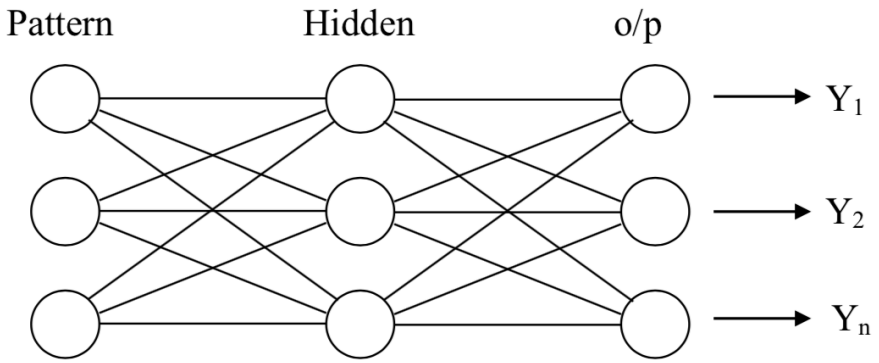
$$\text{Where} \quad \text{net}\ \ j\ = \sum w_{ij}\ x_i + \theta$$

The GDR minimize the squares of the differences between the actual and the desired o/p values summed over the o/p unit and all pairs of I/p and o/p vectors. The rule minimize the overall error $E = \sum E_p$ by implementing a gradient descent in E: - where, $E_p = 1/2 \sum_j (d_j - y_j)^2$.

The BP consists of two phases:-

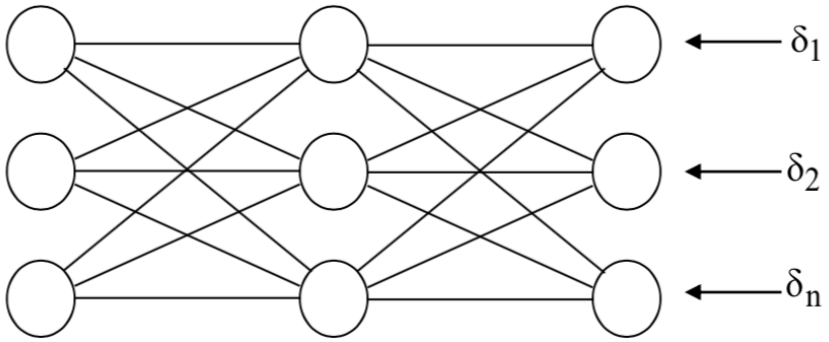
1- Forward Propagation:-

During the forward phase, the I/p is presented and propagated towards the o/p.



2- Backward Propagation:-

During the backward phase, the ***errors*** are formed at the o/p and propagated towards the I/p



3- Compute the error in the hidden layer.

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

$$f' = y(1-y)$$

Equation is can rewrite as:-

$$\delta_j = y(1-y)(d_j - y_j)$$

The error signal for hidden units for which there is no specified target (desired o/p) is determined recursively in terms of the error signals of the units to which it directly connects and the weights of those connections:-

That is

$$\delta_j = f'(net_j)\sum_k \delta_k w_{ik}$$

Or

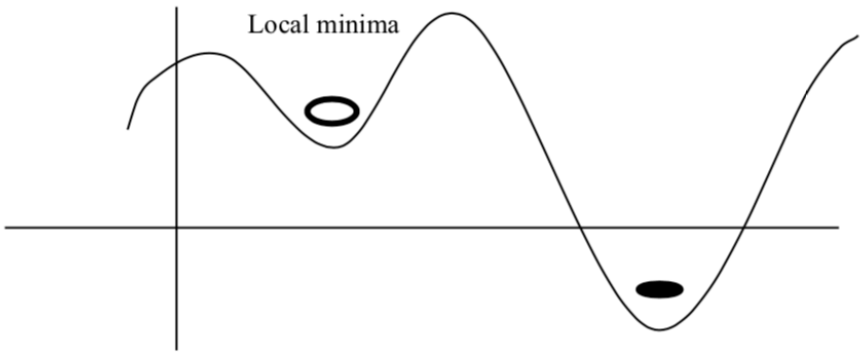
$$\delta_j = y_j(1-y_j)\sum_k \delta_k w_{ik}$$

B.P learning is implemented when hidden units are embedded between input and output units.

Convergence

A quantitative measure of the learning is the :Root Mean Square (RMS) error which is calculated to reflect the "degree" of learning.

Generally, an RMS bellow (0.1) indicates that the net has learned its training set. Note that the net does not provide a yes /no response that is "correct" or "incorrect" since the net get closer to the target value incrementally with each step. It is possible to define a cut off point when the nets o/p is said to match the target values.



- Convergence is not always easy to achieve because sometimes the net gets stuck in a "Local minima" and stops learning algorithm.
- Convergence can be represented intuitively in terms of walking about mountains.

Momentum term

The choice of the learning rate plays important role in the stability of the process. It is possible to choose a learning rate as large as possible without leading to oscillations. This offers the most rapid learning. One way to increase the learning rate without leading to oscillations is to modify the GDR to include momentum term.

This can be achieved by the following rule:-

$$W_{ij}(t+1) = W_{ij}(t) + \zeta\delta_j x_i + \propto (W_{ij}(t) - W_{ij}(t-1))$$

Where $\propto (0 < \propto < 1)$ is a constant which determines the effect of the past weight changes on the current direction of movement in weight space.

A "global minima" unfortunately it is possible to encounter a local minima, avally that is not the lowest possible in the entire terrain. The net does not leave a local minima by the standard BP algorithm and special techniques should be used to get out of a local minima such as:-

1- Change the learning rate or the momentum term.
2- Change the no. of hidden units (10%).
3- Add small random value to the weights.
4- Start the learning again with different initial weights.

3.1.3.1 Back Propagation Training Algorithm

Training a network by back propagation involves three stages:-
1-the feed forward of the input training pattern
2-the back propagation of the associated error
3-the adjustment of the weights
let n = number of input units in input layer,
let p = number of hidden units in hidden layer
let m = number of output units in output layer
let V_{ij} be the weights between i/p layer and the hidden layer,
let W_{ij} be the weights between hidden layer and the output layer,
we refer to the i/p units as X_i , i=1, 2, ….,n. and we refer to the hidden units as Z_j , j=1,….,p. and we refer to the o/p units as y_k, k=1,….., m.
δ_{1j} is the error in hidden layer,
δ_{2k} is the error in output layer,
ζ is the learning rate

$\propto$ is the momentum coefficient (learning coefficient, $0.0 < \propto < 1.0$,

y_k is the o/p of the net (o/p layer),

Z_j is the o/p of the hidden layer,

X_i is the o/p of the i/p layer.

η is the learning coefficient.

The algorithm is as following :-

Step 0 : initialize weights (set to small random value).

Step 1 : while stopping condition is false do steps 2-9

Step 2: for each training pair, do steps 3-8

Feed forward :-

Step 3:- Each i/p unit (X_i) receives i/p signal X_i & broad casts this signal to all units in the layer above (the hidden layer)

Step 4:- Each hidden unit (Z_j) sums its weighted i/p signals,

$$Z - inj = Vaj + \sum_{i=1}^{n} x_i v_{ij} \quad (Vaj \ \text{is abias})$$

and applies its activation function to compute its output signal (the activation function is the binary sigmoid function),

$$Z_j f(Z - inj) = 1 \ / \ (1 + \exp - (Z - inj))$$

and sends this signal to all units in the layer above (the o/p layer).

Step 5:- Each output unit (Yk)sums its weighted i/p signals,

$$y - ink = wok + \sum_{j=1}^{p} Zjwjk \quad (\text{where wok is abias})$$

and applies its activation function to compute its output signal.

$$y_k = f(y - ink) = 1/(1 + \exp -(y - ink))$$

back propagation of error:-

step 6 : Each output unit (y_k , k= 1 tom) receive a target pattern corresponding to the input training pattern, computes its error information term and calculates its weights correction term used to update W_{jk} later,

$$\delta_{2k} = y_k(1-y_k)*(T_k - y_k),$$

where T_k is the target pattern & k=1 to m .

step 7 : Each hidden unit (Z_j, j= 1 top) computes its error information term and calculates its weight correction term used to update Vij later,

$$\delta_{1j} = Zj*(1-Zj)*\sum_{k=1}^{m}\delta 2kWjk$$

Update weights and bias :-

step 8: Each output unit (y_k, k =1 tom) updates its bias and weights:

$$Wjk(new) = \eta*\delta 2k*Zj+\propto *[Wjk(dd)],$$

j= 1 to p

Each hidden unit (Z_j, j= 1 to p) update its bias and weights:

$$Vij(new) = \eta*\delta 1j*Xi+\propto [vij(dd)],$$

I = 1 to n

Step 9 : Test stopping condition.

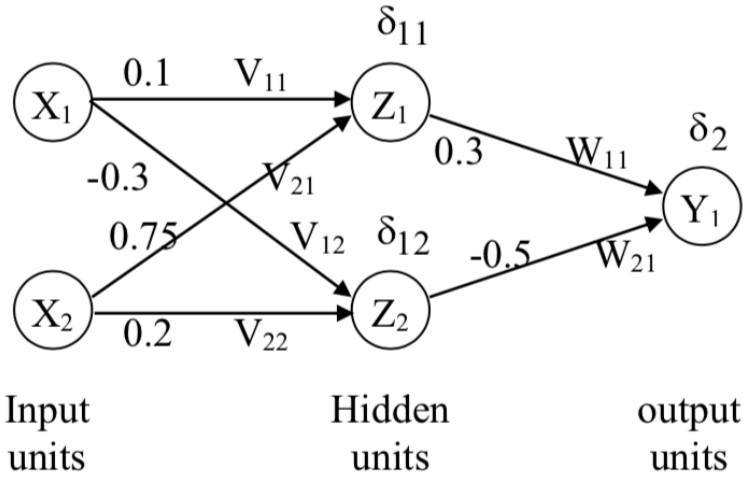
EX6

Suppose you have BP- ANN with 2-input , 2-hiddden , 1-output nodes with sigmoid function and the following matrices weight, trace with 1-iteration.

$$V = \begin{bmatrix} 0.1 & -0.3 \\ 0.75 & 0.2 \end{bmatrix} \qquad w = [0.3 \quad -0.5]$$

Where $\propto = 0.9,\ \eta = 0.45,\ x = (1,0)$, and $T_k = 1$

Solution:-



1-Forword phase :-

$$Z - in1 = X_1V_{11} + X_2V_{21} = 1*0.1 + 0*0.75 = 0.1$$
$$Z - in2 = X_1V_{12} + X_2V_{22} = 1*-0.3 + 0*0.2 = -0.3$$
$$Z_1 = f(Z - in1) = 1/(1 + \exp-(Z - in1)) = 0.5$$
$$Z_2 = f(Z - in2) = 1/(1 + \exp-(Z - in2)) = 0.426$$
$$y - in1 = Z_1W_{11} + Z_2W_{21}$$
$$= 0.5*0.3 + 0.426*(-0.5) = -0.063$$
$$y_1 = f(y - in1) = 1/(1 + \exp-(y - in1) = 0.484$$

2-Backward phase :-

$$\delta_2 k = yk(1-yk)*(Tk-yk)$$

$$\delta_{21} = 0.484(1-0.484)*(1-0.484)0.129$$

$$\delta_{1j} = Z_j*(1-Z_j)*\sum_{k=1}^{m}\delta_{2k}W_{jk}$$

$$\delta_{11} = Z_1(1-Z_1)*(\delta_{21}W_{11})$$
$$= 0.5\,(1-0.5)*(0.129*0.3) = 0.0097$$

$$\delta_{12} = Z_2(1-Z_2)*(\delta_{21}W_{21})$$
$$= 0.426(1-0.426)*(0.129*(-0.5)) = -0.015$$

3-Update weights:-

$$W_{jk}(new) = \eta*\delta_{2k}*Z_j + \propto *\left[W_{jk}(old)\right]$$

$$W_{11} = \eta*\delta_{21}*Z_1 + \propto *\left[W_{11}(old)\right]$$
$$= 0.45*0.129*0.5 + 0.9*0.3 = 0.299$$

$$W_{21} = \eta*\delta_{21}*Z_2 + \propto *\left[W_{21}(old)\right]$$
$$= 0.45*0.129*0.426 + 0.9*-0.5 = -0.4253$$

$$V_{ij}(new) = \eta*\delta_{1j}*X_i + \propto *\left[V_{ij}(old)\right]$$

$$V_{11} = \eta*\delta_{11}*X_1 + \propto *\left[V_{11}(old)\right]$$
$$= 0.45*0.0097*1 + 0.9*0.1 = 0.0944$$

$$V_{12} = \eta*\delta_{12}*X_1 + \propto *\left[V_{12}(old)\right]$$
$$= 0.45*0.0158*1 + 0.9*-0.3 = -0.2771$$

$$V_{21} = \eta*\delta_{11}*X_2 + \propto *\left[V_{21}(old)\right]$$
$$= 0.45*0.0097*0 + 0.9*0.75 = 0.675$$

$$V_{22} = \eta*\delta_{12}*X_2 + \propto *\left[V_{22}(old)\right]$$
$$= 0.45*-0.0158*0 + 0.9*0.2 = 0.18$$

$$\therefore V = \begin{bmatrix} 0.0944 & -0.2771 \\ 0.675 & 0.18 \end{bmatrix} \qquad W = \begin{bmatrix} 0.299 & -0.4253 \end{bmatrix}$$