Chapter 6

Back-Propagation Network 'BPN'

4th Class

INTELLIGENT APPLICATIONS

التطبيقات الذكية

إعدادالمحاضرة :م. ايمان حسين رحيم

أستاذةالمادة الدراسة الصباحية : أ.م.د. ايناس محمد حسين

أستاذةالمادة الدراسة المسائية : م. ايمان حسين رحيم

Back-Propagation Network 'BPN'

6.1 Back-Propagation Network (BPN)

6.2 Back Propagation Training Algorithm

6.3 Example

6.1 Back-Propagation Network (BPN)

• A single-layer neural network has many restrictions. This network can accomplish very limited classes of tasks.

Minsky and Papert (1969) showed that a two layer feed-forward network can overcome many restrictions, but they did not present a solution to the problem as "how to adjust the weights from input to hidden layer”?

• An answer to this question was presented by Rumelhart, Hinton and Williams in 1986. The central idea behind this solution is that the errors for the units of the hidden layer are determined by back-propagating the errors of the units of the output layer.

This method is often called the *Back-propagation learning rule*.

Back-propagation can also be considered as ***a generalization of the delta rule for non-linear activation functions and multi-layer networks.***

• Back-propagation is a systematic method of training multi-layer artificial neural networks.

• A Back-propagation network consists of at least three layers of units:
- an **input layer**,
- at least one intermediate **hidden layer**, and
- an **output layer**.

• Typically, units are connected in a feed-forward fashion with input units fully connected to units in the hidden layer and hidden units fully connected to units in the output layer.

• When a Back-propagation network is cycled, an input pattern is propagated forward to the output units through the intervening input-to-hidden and hidden-to-output weights.

• The output of a Back-propagation network is interpreted as a classification decision.

• With Back-propagation networks, learning occurs ***during a training phase***.

<u>The steps followed during learning are :</u>

–each input pattern in a training set is applied to the input units and then propagated forward.

–the pattern of activation arriving at the output layer is compared with the correct (associated) output pattern to calculate an error signal.

– the error signal for each such target output pattern is then back-propagated from the outputs to the inputs in order to appropriately adjust the weights in each layer of the network.

–after a Back-propagation network has learned the correct classification for a set of inputs; it can be tested on a second set of inputs to see how well it classifies untrained patterns.

• An important consideration in applying Back-propagation learning is how well the network generalizes.

6.2 Back Propagation Training Algorithm

BPN has two phases:

1- **Forward pass** **:** (computes 'functional signal', feed forward propagation of input pattern signals through network)

Present input pattern to input units:

1. Compute values for hidden units (*compute functional signal for hidden units)*

$$u_j(t) = \sum_i v_{ji}(t)\, x_i(t)$$

$$z_j = g(u_j(t))$$

at time t

2. Compute values for output units (*compute functional signal for output units)*

$$a_k(t) = \sum_j w_{kj}(t)\, z_j(t)$$

$$y_k = g(a_k(t))$$

at time t

2- Backward Pass : computes 'error signal', propagates the error backwards through network starting at output units (where the error is the difference between actual and desired output values)

Present Target response to output units

1- Compute error signal for output units ($\triangle i$ (t)) via :

(وهي مقدار الخطأ يستخدم للتعديل على اوزان طبقة المخرجات :$\triangle$)

$$\Delta_i(t) = (d_i(t) - y_i(t))\, g'(a_i(t))$$

$g'(a_i(t))$ = $y_i(1 - y_i)$ *derivative of activation function*

$$\Delta_i(t) = (d_i(t) - y_i(t)) * y_i(1 - y_i)$$

2- Compute error signal for hidden units (δ_i (t)) via:

Note: - **Δ_i's** propagate back to get error terms δ for hidden layers using:

- (وهي مقدار الخطأ يستخدم للتعديل على اوزان الطبقة الخفية :δ)

$$\delta_i(t) = g'(u_i(t)) \sum_k \Delta_k(t)\, w_{ki}$$

$g'(u_i(t))$ = $z_i(1 - z_i)$ *derivative of activation function*

$$\delta_i(t) = \sum_k \Delta_k(t)\, w_{ki} * z_i(1 - z_i)$$

3- Update all weights at same time:

Note: Once weight changes are computed for all units, weights are updated at the same time

- **Weight updates for output unit :** *Weight changes will be:*

$$w_{ij}(t+1) = \propto w_{ij}(t) + \eta \Delta_i(t) z_j(t)$$

- **Weight updates for hidden unit :** *Weight changes will be:*

$$v_{ij}(t+1) = \propto v_{ij}(t) + \eta \delta_i(t) x_j(t)$$

6.3 Example

EX1: Train a back-propagation neural network to learn the following network:
Have input [0 ٠] with target [0], Learning rate η= 1, $\propto$=0.1, using **<u>Sigmoid Function</u>**. The weights matrix

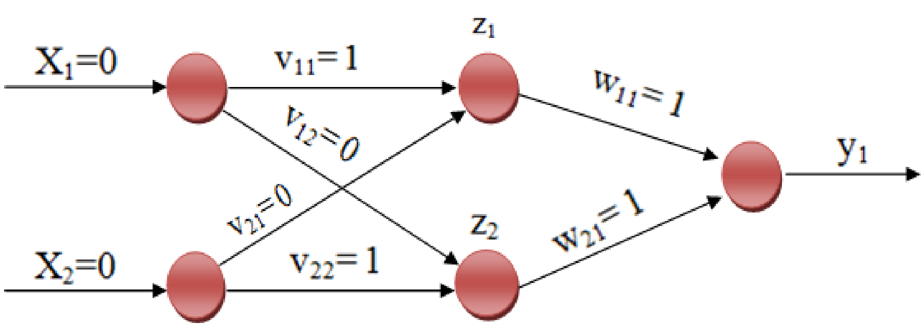
$$V=\begin{Bmatrix} 1 & 0 \\ 0 & 1 \end{Bmatrix} \qquad W=\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

$$X=\begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

توضيح كيفية ترتيب weights matrix حسب الصورة التالية سواء كانت (V) او (W)

$$W=\begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix}$$

Sol:



a/u known as activation
g is the activation function
d_k is the target value
η is the learning rate parameter ($0 < \eta <=1$)
$\propto$ is the momentum coefficient (learning coefficient, $0.0 < \propto < 1.0$

BP has two phases:

**1- Forward pass : **(computes ‘functional signal’, feed forward propagation of input pattern signals through network)

Present input pattern to input units:

1- Compute values for hidden units (*compute functional signal for hidden units)*

$$u_j(t) = \sum_i v_{ji}(t)\, x_i(t)$$

$$z_j = g(u_j(t))$$

at time t

$$u_j(t) = \sum_i v_{ji}(t)\, x_i(t)$$

at time t

$u_j = (\sum V_{ji} * X_i)$

$u_1 = (V_{11}*X_1+V_{21}*X_2)$

$= ((1* 0) + (0*0))$

$= 0$

$u_2 = (V_{12}*X_1+V_{22}*X_2)$

$= ((0* 0) + (1*0))$

$= 0$

$$z_j = g(u_j(t))$$

at time t

$F(u) = z = 1/ (1+e^{-u})$

$Z_1 = g(u_1) = 1/ (1+e^{-u1})$ here was pass through Sigmoid function

$Z_2 = g(u_2) = 1/ (1+e^{-u2})$ here was pass through Sigmoid function

(weighted sum thru activation functions)

$Z_1 = g(u_1) = 1/(1+e^{-0}) = 0.5$

$Z_2 = g(u_2) = 1/(1+e^{-0}) = 0.5$

2- Compute values for output units (compute *functional signal for output units)*

$$a_k(t) = \sum_j w_{kj}(t) z_j(t) \quad \text{at time t}$$

$$y_k = g(a_k(t))$$

$$a_k(t) = \sum_j w_{kj}(t) z_j(t) \quad \text{at time t}$$

$a_k = (\sum w_{kj} * z_j)$

$a_1 = (w_{11}*z_1+w_{21}*z_2)$

$= ((1* 0.5) + (1*0.5))$

$= 1$

$$y_k = g(a_k(t)) \quad \text{a t time t}$$

$F(a) = y = 1/(1+e^{-a})$

$y_1 = g(a_1) = 1/(1+e^{-a1})$ here was pass through Sigmoid function (weighted sum thru activation functions)

$y_1 = g(1) = 1/(1+e^{-1}) = 0.731058578 = 0.73106$

2- <u>Backward Pass</u> : computes 'error signal', propagates the error backwards through network starting at output units (where the error is the difference between actual and desired output values)

<u>Present Target response to output units</u>

1- Compute error signal for output units ($\triangle$i (t)) via :

(وهي مقدار الخطأ يستخدم للتعديل على اوزان طبقة المخرجات :$\triangle$)

$$\Delta_i(t) = (d_i(t) - y_i(t))\, g'(a_i(t))$$

$$g'(a_i(t)) = yi\,(1 - y_i)$$ *derivative of activation function*

$$\Delta_i(t) = (d_i(t) - y_i(t)) \;*\; yi\,(1 - y_i)$$

Target = [0] so $d_1 = 0$

Remind: $\triangle_i = (O_{desired} - O_{actual}) = (d_i - y_i)$

$$\triangle_1 = (d_1 - y_1) * y_1\,(1 - y_1)$$

= (0 – 0.73106) * 0.73106 (1- 0.73106)

= – 0.73106 * 0.73106 * 0.26894

= – 0.143734639724984

= – 0.14373

2- Compute error signal for hidden units (δ_i (t)) via:

Note: - **Δ_i's** propagate back to get error terms δ for hidden layers using:

- (δ: وهي مقدار الخطأ يستخدم للتعديل على اوزان الطبقة الخفية)

$$\delta_i(t) = g'(u_i(t)) \sum_k \Delta_k(t) w_{ki}$$

$$g'(u_i(t)) = z_i(1 - z_i) \quad \textit{derivative of activation function}$$

$$\delta_i(t) = \sum_k \Delta_k(t) w_{ki} * z_i(1 - z_i)$$

$$\delta_1 = (\Delta_1(t) * w_{11}) * z_1(1 - z_1)$$

$$\delta_2 = (\Delta_1(t) * w_{21}) * z_2(1 - z_2)$$

$\delta_1 = (-0.14373 * 1) * 0.5 (1 - 0.5)$

$\delta_1 = (-0.14373) * 0.5 * 0.5$
$= -0.14373 * 0.25$
$= -0.0359325$
$= -0.03593$

$\delta_2 = (-0.14373 * 1) * 0.5 (1 - 0.5)$
$\delta_2 = (-0.14373) * 0.5 * 0.5$
$= -0.14373 * 0.25$
$= -0.0359325$
$= -0.03593$

3- Update all weights at same time:
Note: Once weight changes are computed for all units, weights are updated at the same time

- **Weight updates for output unit :** *Weight changes will be:*

$$w_{ij}(t+1) = \propto w_{ij}(t) + \eta \Delta_i(t) z_j(t)$$

$$W_{ij}(t+1) = \propto W_{ij}(t) + \eta \triangle_i(t) z_j(t)$$

$$W_{ij}\ new = \propto W_{ij}\ old + \eta \triangle_i(t) z_j(t)$$

$$W_{11} = \propto W_{11}\ old + \eta \triangle_1 z_1$$
$$= (0.1 * 1) + (1 * -0.14373 * 0.5)$$
$$= 0.1 + (-0.071865)$$
$$= 0.028135$$

$$W_{21} = \propto W_{21}\ old + \eta \triangle_1 z_2$$
$$= (0.1 * 1) + (1 * -0.14373 * 0.5)$$
$$= 0.1 + (-0.071865)$$
$$= 0.028135$$

- **Weight updates for hidden unit :** *Weight changes will be:*

$$v_{ij}(t+1) = \propto v_{ij}(t) + \eta\delta_i(t)x_j(t)$$

$$v_{ij}(t+1) = \propto v_{ij}(t) + \eta\,\delta_i(t)\,x_j(t)$$

$$v_{ij}\text{ new} = \propto v_{ij}\text{ old} + \eta\,\delta_i(t)\,x_j(t)$$

$$v_{11} = \propto v_{11}\text{ old} + \eta\,\delta_1\,x_1$$
$$= (0.1*1) + (1*-0.03593*0)$$
$$= 0.1$$

$$v_{12} = \propto v_{12}\text{ old} + \eta\,\delta_2\,x_1$$
$$= (0.1*0) + (1*-0.03593*0)$$
$$= 0 + 0$$
$$= 0$$

$$v_{21} = \propto v_{21}\text{ old} + \eta\,\delta_1\,x_2$$
$$= (0.1*0) + (1*-0.03593*0)$$
$$= 0 + 0$$

x_1	x_2	d_1	Z_1	Z_2	y_1	$\triangle_1$	δ_1	δ_2
0	0	0	0.5	0.5	0.73106	– 0.14373	-0.03593	-0.03593

$$= 0$$

$$v_{22} = v_{22}\text{ old} + \eta\,\delta_2\,x_2$$
$$= (0.1*1) + (1*-0.03593*0)$$
$$= 0.1 - 0$$
$$= 0.1$$

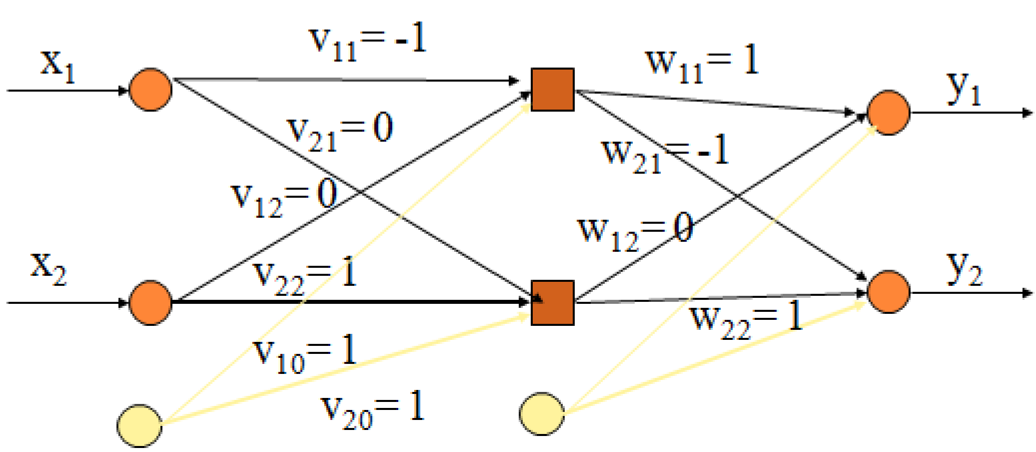
Old weights				New weights			
v_{11}	v_{12}	v_{21}	v_{22}	v_{11}	v_{12}	v_{21}	v_{22}
1	0	0	1	0.1	0	0	0.1

Old weights		New weights	
w_{11}	w_{21}	w_{11}	w_{21}
1	1	0.028135	0.028135

………………………………………………………………………………………..

مثال افتراضي لغرض التعلم

EX2: Train a back-propagation neural network to learn the following network:
Have input [0 1] with target [1 0], that the bias applied to the neuron is 1, Learning rate η= 0.1, $\propto$= 0.2, Use identity activation function (linear function).



Sol:
biases set as extra weights (b=1)
a/u known as activation
g is the activation function
d_k is the target value
η is the learning rate parameter (0 < η <=1)
$\propto$ is the momentum coefficient (learning coefficient, 0.0 < $\propto$ < 1.0

BP has two phases:

1- Forward pass : (computes 'functional signal', feed forward propagation of input pattern signals through network)

Present input pattern to input units:

1- Compute values for hidden units (*compute functional signal for hidden units)*

$$u_j(t) = \sum_i v_{ji}(t)\, x_i(t)$$

$$z_j = g(u_j(t))$$

at time t

$$u_j(t) = \sum_i v_{ji}(t)\, x_i(t)$$

at time t

$u_j = (\sum V_{ji} * X_i) + \mathbf{b}$

$u_1 = (V_{11}*X_1+V_{21}*X_2) +1$

$= ((-1* 0) + (0*1)) +1$

$= 0+0+1$

$= 1$

$u_2 = (V_{12}*X_1+V_{22}*X_2) +\mathbf{b}$

$= ((0* 0) + (1*1)) +1$

$= 1+1$

$= 2$

at time t

$$z_j = g(u_j(t))$$

$$\left.\begin{aligned} Z_1 &= g(u_1) = g(1) = 1 \\ Z_2 &= g(u_2) = g(2) = 2 \end{aligned}\right\}$$

here was pass through linear function $g(u) = u$

(weighted sum thru activation functions)

2. Compute values for output units (*compute functional signal for output units)*

$$a_k(t) = \sum_j w_{kj}(t) z_j(t)$$ at time t

$$y_k = g(a_k(t))$$

$$a_k(t) = \sum_j w_{kj}(t) z_j(t)$$ at time t

$a_k = (\sum w_{kj} * z_j) + b$

$a_1 = (w_{11} * z_1 + w_{21} * z_2) + 1$

$= ((1 * 1) + (0 * 2)) + 1$

$= 1 + 1$

$= 2$

$a_2 = (w_{12} * z_1 + w_{22} * z_2) + b$

$= ((-1 * 1) + (1 * 2)) + 1$

$= (-1 + 2) + 1$

$= 1 + 1$

$= 2$

$$y_k = g(a_k(t))$$ a t time t

$y_1 = g(a_1) = g(2) = 2$
$y_2 = g(a_2) = g(2) = 2$

here was pass through linear function g (a) = a
(weighted sum thru activation functions)

2. Backward Pass : computes 'error signal', propagates the error backwards through network starting at output units (where the error is the difference between actual and desired output values)

Present Target response to output units

1- Compute error signal for output units ($\triangle$i (t)) via :

(وهي مقدار الخطأ يستخدم للتعديل على اوزان طبقة المخرجات :$\triangle$)

$$\Delta_i(t) = (d_i(t) - y_i(t))\, g'(a_i(t))$$

$g'(a_i(t))$ = 1 *derivative of activation function g*

$$Note:\ g'(a) = \frac{dg}{da} = 1da = 1$$

$$\Delta_i(t) = (d_i(t) - y_i(t))$$

Target = [1, 0] so $d_1 = 1$ and $d_2 = 0$

Remind: $\triangle_i = (O_{desired} - O_{actual}) = (d_i - y_i)$

$\triangle_1 = (d_1 - y_1) = 1 - 2 = -1$

$\triangle_2 = (d_2 - y_2) = 0 - 2 = -2$

2- Compute error signal for hidden units (δ_i (t)) via:

Note: - **Δ_i's** propagate back to get error terms δ for hidden layers using:

- (δ: وهي مقدار الخطأ يستخدم للتعديل على اوزان الطبقة الخفية)

$$\delta_i(t) = g'(u_i(t)) \sum_k \Delta_k(t)\, w_{ki}$$

$g'(u_i(t))$ = 1 *derivative of activation function g*

$$Note:\ g'(u) = \frac{dg}{du} = 1du = 1$$

$$\delta_i(t) = \sum_k \Delta_k(t) w_{ki}$$

$\delta_1 = (\Delta_1(t) * w_{11}) + (\Delta_2(t) * w_{12})$

$\delta_1 = ((-1*1) + (-2*-1))$

$= -1+2$

$= 1$

$\delta_2 = (\Delta_1(t) * w_{21}) + (\Delta_2(t) * w_{22})$

$\delta_2 = ((-1*0) + (-2*1))$

$= -2$

3- Update all weights at same time:
Note: Once weight changes are computed for all units, weights are updated at the same time

- **Weight updates for output unit :** *Weight changes will be:*

$$w_{ij}(t+1) = \propto w_{ij}(t) + \eta \Delta_i(t) z_j(t)$$

$W_{ij}(t+1) = \propto W_{ij}(t) + \eta \triangle_i(t) z_j(t)$

W_{ij} new $= \propto W_{ij}$ old $+ \eta \triangle_i(t) z_j(t)$

$W_{11} = \propto W_{11}$ old $+ \eta \triangle_1 z_1$
$= (0.2*1) + (0.1*-1*1)$
$= 0.2 + (-0.1)$
$= 0.2 - 0.1$
$= 0.1$

$W_{12} = \propto W_{12}\text{ old} + \eta \triangle_2 z_1$
$= (0.2*-1) + (0.1*-2*1)$
$= -0.2 - 0.2$
$= -0.4$

$W_{21} = \propto W_{21}\text{ old} + \eta \triangle_1 z_2$
$= (0.2*0) + (0.1*-1*2)$
$= 0 + (-0.2)$
$= -0.2$

$W_{22} = \propto W_{22}\text{ old} + \eta \triangle_2 z_2$
$= (0.2*1) + (0.1*-2*2)$
$= 0.2 - 0.4$
$= -0.2$

- **Weight updates for hidden unit :** *Weight changes will be:*

$$v_{ij}(t+1) = \propto v_{ij}(t) + \eta\delta_i(t)\,x_j(t)$$

$v_{ij}(t+1) = \propto v_{ij}(t) + \eta\, \delta_i(t)\, x_j(t)$

$v_{ij}\text{ new} = \propto v_{ij}\text{ old} + \eta\, \delta_i(t)\, x_j(t)$

$v_{11} = \propto v_{11}\text{ old} + \eta\, \delta_1 x_1$
$= (0.2*-1) + (0.1*1*0)$
$= -0.2 + 0$
$= -0.2$

$v_{12} = \propto v_{12}\text{ old} + \eta\, \delta_2 x_1$
$= (0.2*0) + (0.1*-2*0)$
$= 0 + 0$
$= 0$

$$v_{21} = \propto v_{21}\text{ old} + \eta\, \delta_1\, x_2$$
$$= (0.2*0) + (0.1*1*1)$$
$$= 0 + (0.1)$$
$$= 0.1$$

$$v_{22} = \propto v_{22}\text{ old} + \eta\, \delta_2\, x_2$$
$$= (0.2*1) + (0.1*-2*1)$$
$$= 0.2 - 0.2$$
$$= 0$$

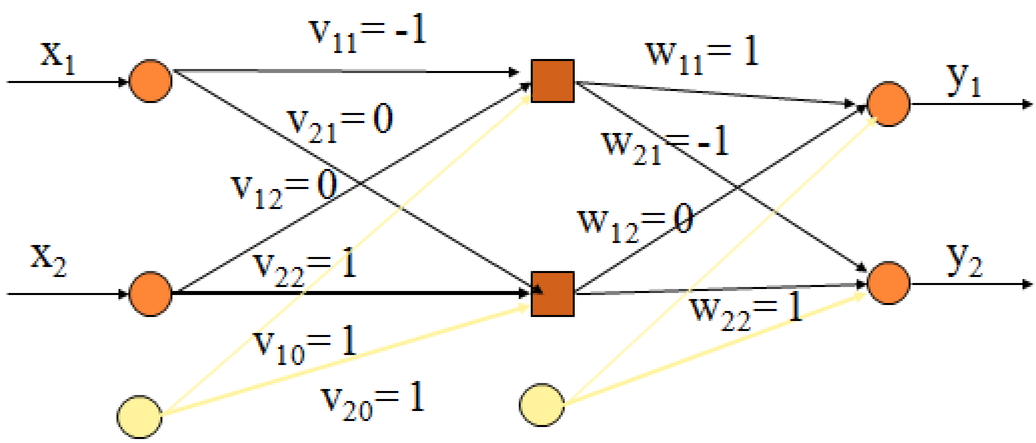
x_1	x_2	d_1	d_2	Z_1	Z_2	y_1	y_2	$\triangle_1$	$\triangle_2$	δ_1	δ_2
0	1	1	0	1	2	2	2	-1	-2	1	-2

Old weights				New weights			
v_{11}	v_{12}	v_{21}	v_{22}	v_{11}	v_{12}	v_{21}	v_{22}
-1	0	0	1	- 0.2	0	0.1	0

Old weights				New weights			
w_{11}	w_{12}	w_{21}	w_{22}	w_{11}	w_{12}	w_{21}	w_{22}
1	-1	0	1	0.1	-0.4	-0.2	-0.2

Homework:

H.W (1) Train a back-propagation neural network to learn the following network:
Have input [0 1] with target [1 0], that the bias applied to the neuron is 1, Learning rate η= 0.1, ∝= 0.2 .Using **Sigmoid Function.**



……………………………………………………………………………………………

H.W (2) 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.

$$V = \begin{Bmatrix} 0.1 & 0.3 \\ 0.75 & 0.2 \end{Bmatrix} \qquad W = \begin{Bmatrix} 0.3 \\ 0.5 \end{Bmatrix}$$

Where ∝ = 0.9, η = 0.45, x = (1, 0), and T = 1

Sol: توضيح للحل

X Pattern =Input Value
a/u known as activation
g is the activation function
d_k = T is the target value
η is the learning rate parameter (0 < η <=1)
∝ is the momentum coefficient (learning coefficient, 0.0 < ∝ < 1.0