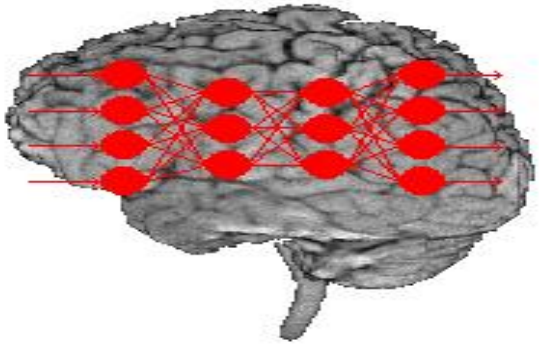
Deep Learning
Lecture-5

Artificial Neural Network

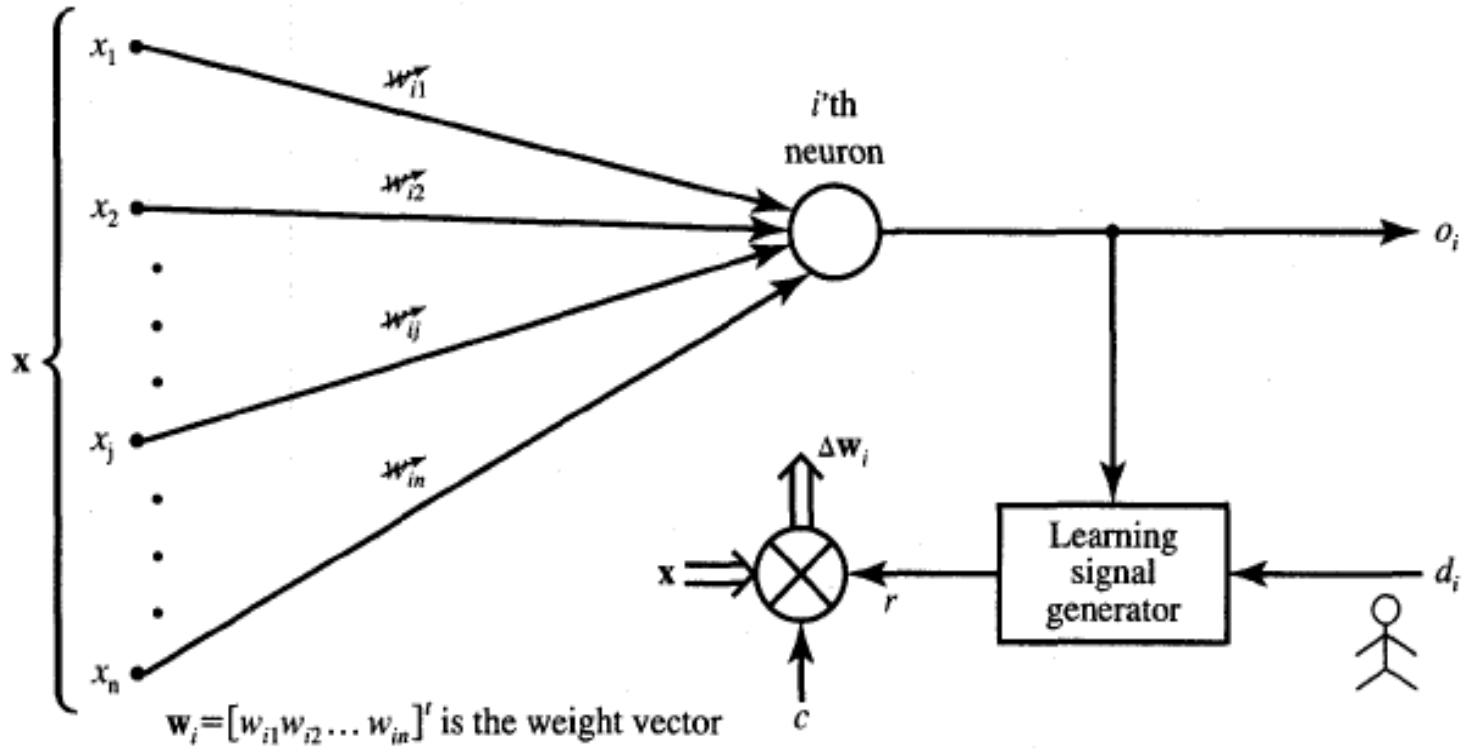


Asst. Lect. Ali Al-khawaja

2025-2026

Class Room

Neural Network Learning rules



$$r = r(\mathbf{w}_i, \mathbf{x}, d_i)$$

$$\Delta \mathbf{w}_i(t) = cr\left[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)\right]\mathbf{x}(t)$$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + cr\left[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)\right]\mathbf{x}(t)$$

c – learning constant

Hebbian Learning Rule

FEED FORWARD UNSUPERVISED LEARNING

- The learning signal is equal to the neuron's output

$$r \triangleq f(\mathbf{w}_i^t \mathbf{x})$$

$$\Delta \mathbf{w}_i = cf(\mathbf{w}_i^t \mathbf{x})\mathbf{x}$$

The single weight w_{ij} is adapted using the following increment:

$$\Delta w_{ij} = cf(\mathbf{w}_i^t \mathbf{x})x_j$$

This can be written briefly as

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

Features of Hebbian Learning

- Feedforward unsupervised learning
- “When an axon of a cell A is near enough to exicite a cell B and repeatedly and persistently takes place in firing it, some growth process or change takes place in one or both cells increasing the efficiency”
- If $o_i x_j$ is positive the results is increase in weight else vice versa

Example-zurada-p.61

This example illustrates Hebbian learning with binary and continuous activation functions of a very simple network. Assume the network shown in Figure 2.22 with the initial weight vector

$$\mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

needs to be trained using the set of three input vectors as below

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

for an arbitrary choice of learning constant $c = 1$. Since the initial weights are of nonzero value, the network has apparently been trained beforehand. Assume first that bipolar binary neurons are used, and thus $f(net) = \text{sgn}(net)$.

Step 1 Input $\mathbf{x}_1$ applied to the network results in activation net^1 as below:

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = 3$$

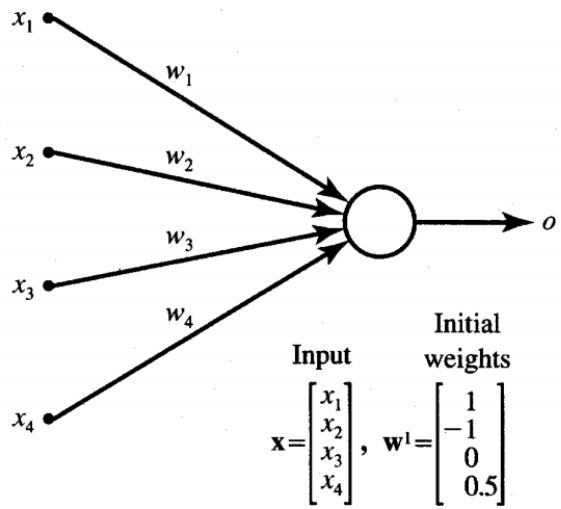


Figure 2.22 Network for training in Examples 2.4 through 2.6.

The updated weights are

$$\mathbf{w}^2 = \mathbf{w}^1 + \text{sgn}\,(net^1)\mathbf{x}_1 = \mathbf{w}^1 + \mathbf{x}_1$$

and plugging numerical values we obtain

$$\mathbf{w}^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix}$$

where the superscript on the right side of the expression denotes the number of the current adjustment step.

Step 2 This learning step is with $\mathbf{x}_2$ as input:

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = \begin{bmatrix} 2 & -3 & 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = -0.25$$

The updated weights are

$$\mathbf{w}^3 = \mathbf{w}^2 + \text{sgn}\,(net^2)\mathbf{x}_2 = \mathbf{w}^2 - \mathbf{x}_2 = \begin{bmatrix} 1 \\ -2.5 \\ 3.5 \\ 2 \end{bmatrix}$$

Step 3 For input $\mathbf{x}_3$, we obtain in this step

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = \begin{bmatrix} 1 & -2.5 & 3.5 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = -3$$

The updated weights are

$$\mathbf{w}^4 = \mathbf{w}^3 + \text{sgn}\,(net^3)\mathbf{x}_3 = \mathbf{w}^3 - \mathbf{x}_3 = \begin{bmatrix} 1 \\ -3.5 \\ 4.5 \\ 0.5 \end{bmatrix}$$

It can be seen that learning with discrete $f(net)$ and $c = 1$ results in adding or subtracting the entire input pattern vectors to and from the weight vector, respectively. In the case of a continuous $f(net)$, the weight incrementing/decrementing vector is scaled down to a fractional value of the input pattern.

- For the same inputs for bipolar continuous activation function the final updated weight is given by

$$f(net^1) = 0.905 \qquad f(net^2) = -0.077$$

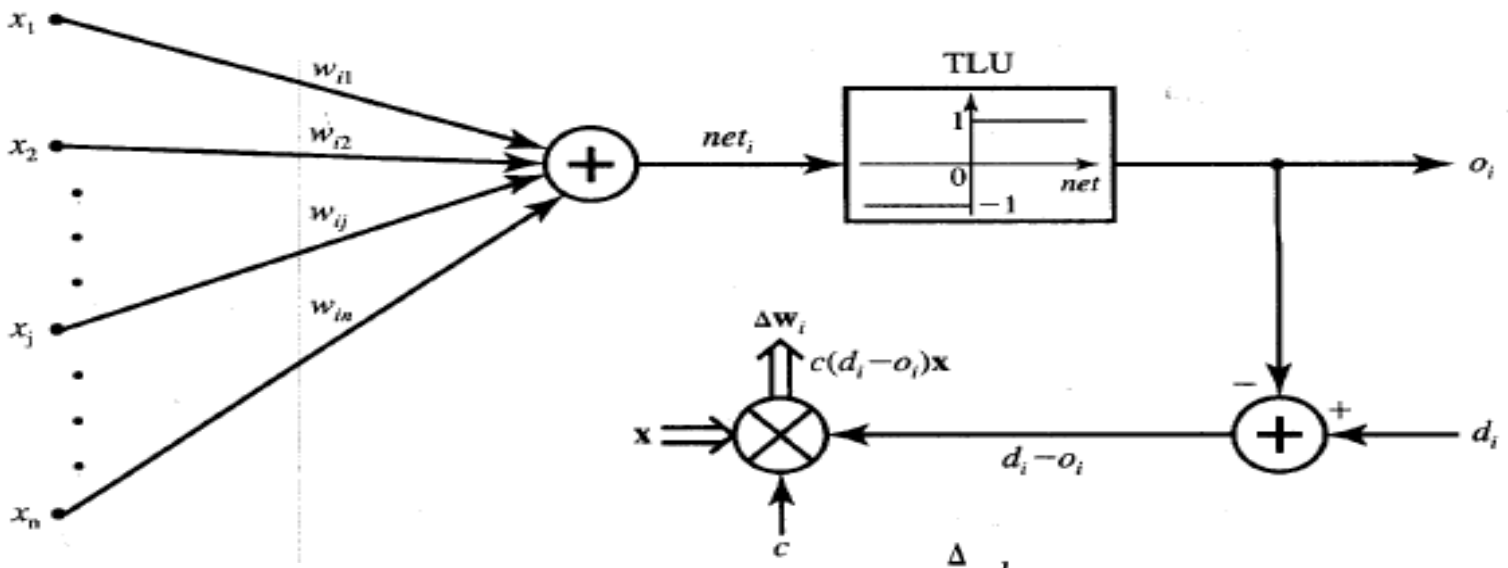
$$\mathbf{w}^2 = \begin{bmatrix} 1.905 \\ -2.81 \\ 1.357 \\ 0.5 \end{bmatrix} \qquad \mathbf{w}^3 = \begin{bmatrix} 1.828 \\ -2.772 \\ 1.512 \\ 0.616 \end{bmatrix}$$

$$f(net^3) = -0.932$$

$$\mathbf{w}^4 = \begin{bmatrix} 1.828 \\ -3.70 \\ 2.44 \\ -0.783 \end{bmatrix}$$

Perceptron Learning rule

- Learning signal is the difference between the desired and actual neuron's response
- Learning is supervised



$$r \triangleq d_i - o_i$$

$$o_i = \text{sgn}\,(\mathbf{w}_i^t \mathbf{x})$$

$$\Delta \mathbf{w}_i = c\,[d_i - \text{sgn}\,(\mathbf{w}_i^t \mathbf{x})]\,\mathbf{x}$$

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

$$\mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

$c = 0.1.$

$d_1 = -1$, $d_2 = -1$, and $d_3 = 1$.

$$\mathbf{w}^4 = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

Example p.65

This example illustrates the perceptron learning rule of the network shown in Figure 2.23. The set of input training vectors is as follows:

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

and the initial weight vector $\mathbf{w}^1$ is assumed identical as in Example 2.4. The learning constant is assumed to be $c = 0.1$. The teacher's desired responses for $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ are $d_1 = -1$, $d_2 = -1$, and $d_3 = 1$, respectively. The learning according to the perceptron learning rule progresses as follows.

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

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

Correction in this step is necessary since $d_1 \neq \text{sgn}\,(2.5)$. We thus obtain 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:

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

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:

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

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

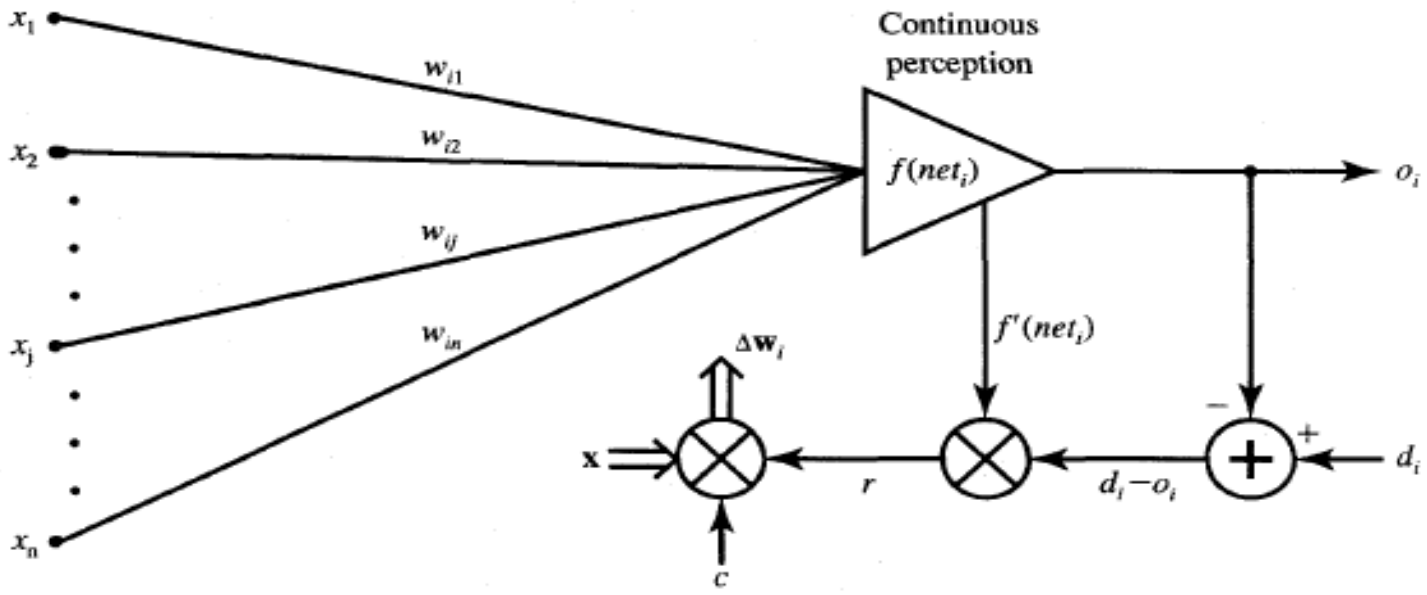
$$\mathbf{w}^4 = \mathbf{w}^3 + 0.1(1 + 1)\mathbf{x}_3$$

or

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

Delta Learning Rule

- Only valid for continuous activation function
- Used in supervised training mode
- Learning signal for this rule is called delta
- The aim of the delta rule is to minimize the error over all training patterns



Delta Learning Rule Contd.

$$r \triangleq [d_i - f(\mathbf{w}_i^t \mathbf{x})] f'(\mathbf{w}_i^t \mathbf{x})$$

Learning rule is derived from the condition of least squared error.

Calculating the gradient vector with respect to **wi**

$$E \triangleq \frac{1}{2}(d_i - o_i)^2$$

$$\nabla E = -(d_i - o_i) f'(\mathbf{w}_i^t \mathbf{x}) \mathbf{x}$$

The components of the gradient vector are

$$\frac{\partial E}{\partial w_{ij}} = -(d_i - o_i) f'(\mathbf{w}_i^t \mathbf{x}) x_j, \quad \text{for } j = 1, 2, \ldots, n$$

Minimization of error requires the weight changes to be in the negative gradient direction

$$\Delta \mathbf{w}_i = -\eta \nabla E$$

$$\Delta \mathbf{w}_i = \eta (d_i - o_i) f'(net_i) \mathbf{x}$$

Widrow-Hoff learning Rule

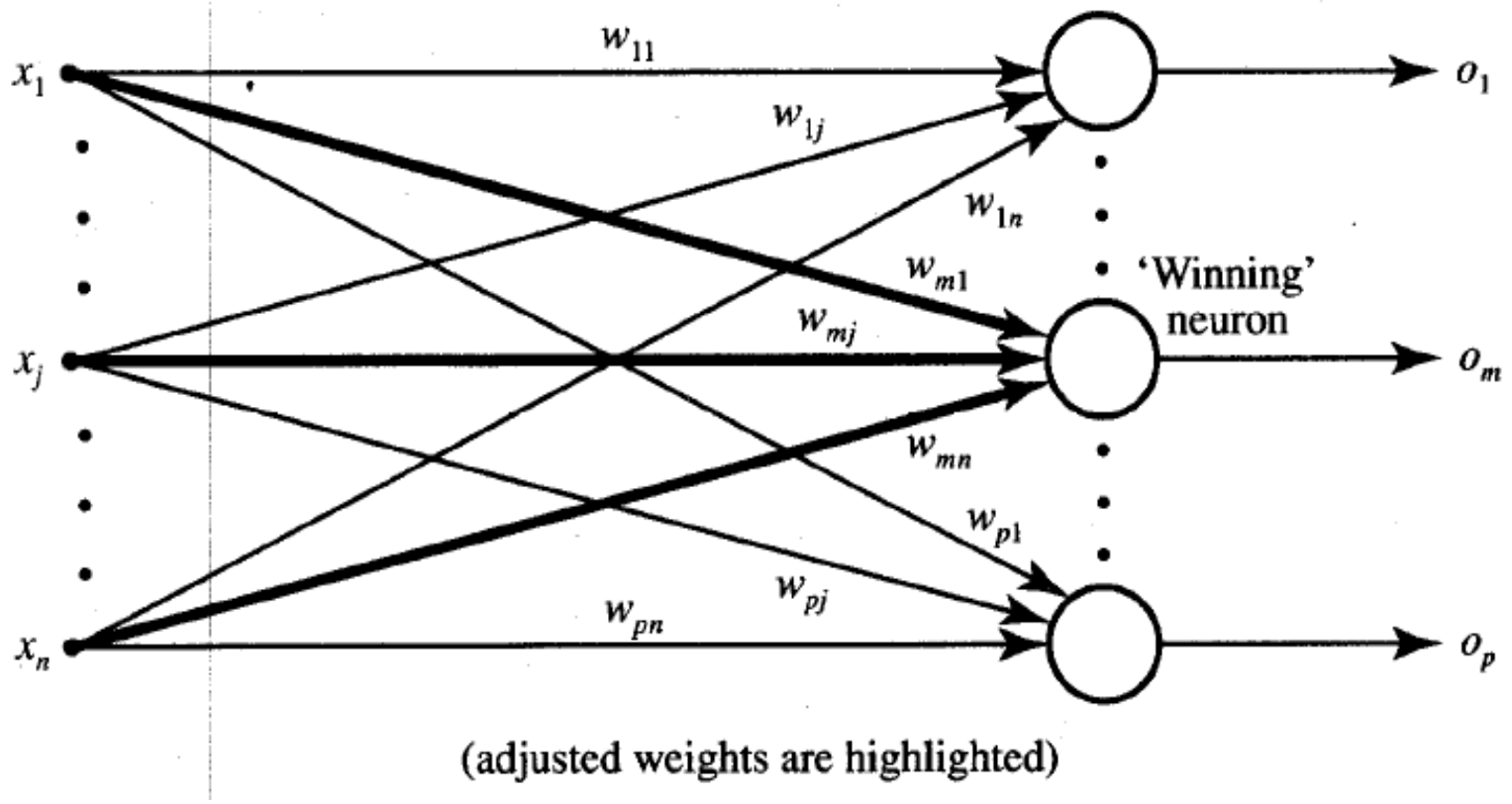
- Also called as least mean square learning rule
- Introduced by Widrow(1962), used in supervised learning
- Independent of the activation function
- Special case of delta learning rule wherein activation function is an identity function ie *f(net)=net*
- Minimizes the squared error between the desired output value d_i and net_i

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

The weight vector increment under this learning rule is

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

Winner-Take-All learning rules



Winner-Take-All Learning rule Contd…

- Can be explained for a layer of neurons
- Example of competitive learning and used for unsupervised network training
- Learning is based on the premise that one of the neurons in the layer has a maximum response due to the input x
- This neuron is declared the winner with a weight

$$\mathbf{w}_m = \begin{bmatrix} w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}^t$$

Its increment is computed as follows

$$\Delta \mathbf{w}_m = \alpha(\mathbf{x} - \mathbf{w}_m)$$

The winner selection is based on the following criterion of maximum

activation among all p neurons participating in a competition:

$$\mathbf{w}_m^t \mathbf{x} = \max_{i=1,2,\ldots,p} (\mathbf{w}_i^t \mathbf{x})$$

Summary of learning rules

Summary of learning rules and their properties.

Learning rule	Single weight adjustment Δw_{ij}	Initial weights	Learning	Neuron characteristics	Neuron / Layer
Hebbian	$co_i x_j$ $j = 1, 2, \ldots, n$	0	U	Any	Neuron
Perceptron	$c\left[d_i - \text{sgn}\left(\mathbf{w}_i^t \mathbf{x}\right)\right] x_j$ $j = 1, 2, \ldots, n$	Any	S	Binary bipolar, or Binary unipolar*	Neuron
Delta	$c(d_i - o_i) f'(net_i) x_j$ $j = 1, 2, \ldots, n$	Any	S	Continuous	Neuron
Widrow-Hoff	$c(d_i - \mathbf{w}_i^t \mathbf{x}) x_j$ $j = 1, 2, \ldots, n$	Any	S	Any	Neuron
Winner-take-all	$\Delta w_{mj} = \alpha(x_j - w_{mj})$ m-winning neuron number $j = 1, 2, \ldots, n$	Random Normalized	U	Continuous	Layer of p neurons

Thank you…

Any questions??

يرجى مسح رمز الاستجابة السريعة QR Code لتعبئة نموذج التغذية الراجعة حول المحاضرة. ملاحظاتكم مهمة لتحسين المحاضرات القادمة.

My google site