Al-Mustaqbal University

College of Engineering

Department of Biomedical Engineering

Stage: 5th Subject : Neural Eng.

2025-2026

Lecture ( ):

6. Neural Network Learning:

One of the questions will raised from the above examples "How do we determine the weight matrix and bias for perceptron networks with many inputs, where it is impossible to visualize the decision boundaries?". The answer is to build an algorithm for training perceptron networks, so that they can learn to solve classification problems.

1. Types of Learning:

A procedure for modifying the weights and biases of a network. (This procedure may also be referred to as a ***training algorithm***). The purpose of the learning rule is to train the network to perform some tasks. There are many types of neural network learning rules. They fall into two broad categories: ***supervised learning*** and ***unsupervised learning***.

1. Supervised Learning:

The learning rule is provided with a set of examples (the training set) of proper network behavior:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \{\mathbf{p}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{p}_q, \mathbf{t}_q\}$$

where $\mathbf{p}_q$ is an input to the network and $\mathbf{t}_q$ is the corresponding correct (target) output. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets.

2. Unsupervised Learning:

The weights and biases are modified in response to network inputs only. There are no target outputs available. At first glance this might seem to be impractical. How can you train a network if you don't know what it is supposed to do? Most of these algorithms perform some kind of clustering operation. They learn to categorize the input patterns into a finite number of classes.

2. Learning Rules:

1. Hebbian Learning Rules:

The earliest and simplest learning rule for neural network is generally known as the Hebb rule. The Hebb (1949) learning rule is based on the assumption that if two neighbor neurons must be activated and deactivated at the same time, then the weight connecting these neurons should increase. For neurons operating in the opposite phase, the weight between them should decrease (i.e. the weight increase if both p_j and a_i are positive or negative, the weight decrease whenever p_j and a_i have opposite sign). The weight update can be written as:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\, a_{iq}\, p_{jq} \qquad (6.1)$$

$$b_i^{new} = b_i^{old} + \alpha\, a_{iq} \qquad (6.2)$$

Where p_{jq} is the j^{th} element of the q^{th} input vector $\mathbf{p}_q$, a_{iq} is the i^{th} element of the network output when the q^{th} input vector is presented to the network, and α is a positive constant ($0 < \alpha \leq 1$), called the learning rate.

Note: The choice of the value of learning rate is important when we implement a neural network. A large learning rate corresponds to rapid learning but might also result in oscillations.

The Hebb rule defined in Eq.'s 6.1 and 6.2 is an ***unsupervised*** learning rule. It does not require any information concerning the target output.

The ***supervised*** Hebb rule substitute the target (desired) output with the neuron output. The weight update can be written as:

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq}\, p_{jq} \qquad (6.3)$$

$$b_i^{new} = b_i^{old} + t_i \qquad (6.4)$$

Where t_{iq} is the i^{th} element of the q^{th} target vector $\mathbf{t}_q$. ($\alpha = 1$ for simplicity).

Equation 6.3 and 6.4 can be written in vector notation:

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \mathbf{t}_q \, \mathbf{p}_q \quad (6.5)$$

$$\mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{t}_q \quad (6.6)$$

1. The Algorithm:

Step 1: Initialize all weights and bias : $w_{ij} = 0$, $b_i = 0$

Step 2: For each q input training vector and target output pair $(\mathbf{p}, \mathbf{t})$, do steps 3 – 4.

Step 3: Adjust the weights for

$$w_{ij}^{new} = w_{ij}^{old} + t_i \, p_j$$

Step 4: Adjust the bias:

$$b_i^{new} = b_i^{old} + t_i$$

2. Example:

A Hebb net for AND function: binary case

Table 9: The AND Logical Operation (Binary)

Input		Target
p_1	p_2	p_1 AND p_2
1	1	1
0	1	0
1	0	0
0	0	0

Solution:

To use Hebb rule, initial weights are:

$$w_{1,1} = 0 \quad , w_{1,2} = 0 \quad \text{and} \quad b = 0$$

The weight change for training (input, target) are:

$$w_{1,1} = t \, p_1 \quad , \; w_{1,2} = t \, p_2 \quad \text{and} \quad b = t$$

The weight updates for the first input pair are as follows:

Input			Target	Weights		
p_1	p_2	b	p_1 AND p_2	$w_{1,1}$	$w_{1,2}$	b
1	1	1	1	1	1	1

Recall equation $p_2 = -\frac{b}{w_{1,2}} - \frac{w_{1,1}\, p_1}{w_{1,2}}$

So, the separating line becomes $p_2 = -p_1 - 1$

Figure 33 shows that the response of the net after first training pair.

The following Table shows weight update after second, third and fourth training inputs pairs:

Input			Target	Weights		
p_1	p_2	b	p_1 AND p_2	$w_{1,1}$	$w_{1,2}$	b
0	1	1	0	1	1	1
1	0	1	0	1	1	1
0	0	1	0	1	1	1

Because the target value is 0, no learning occurs. Thus, using binary target values prevents the net from learning any pattern for which the target is "off".

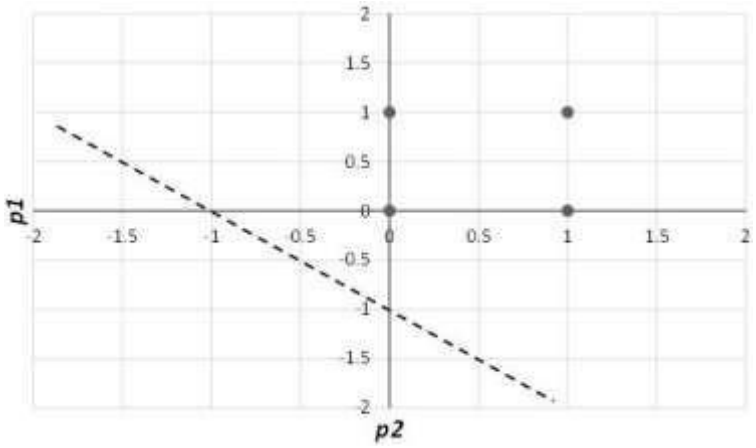


Figure 33: Decision boundary for binary AND function using Hebb rule after first training pair

The choice of training patterns can play a significant role in determining which problems can be solved using the Hebb rule. The next example shows that the AND function can be solved if we modify its representation to bipolar form.

6.2.1.3 Example:

A Hebb net for AND function: bipolar case

Table 10: The AND Logical Operation (Bipolar)

Input		Target
p_1	p_2	p_1 AND p_2
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1

Solution:

Take, initial weights are:

$$w_{1,1} = 0 \quad , w_{1,2} = 0 \quad \text{and} \quad b = 0$$

The weight change for training (input, target) are:

$$w_{1,1} = t\, p_1 \quad , \quad w_{1,2} = t\, p_2 \quad \text{and} \quad b = t$$

Presenting the first input pair, we get

Input			Target	Weights		
p_1	p_2	b	p_1 AND p_2	$w_{1,1}$	$w_{1,2}$	b
1	1	1	1	1	1	1

So, the separating line becomes

$$p_2 = -p_1 - 1$$

Figure 34 shows that the response of the net after will now be correct for the first training pair.

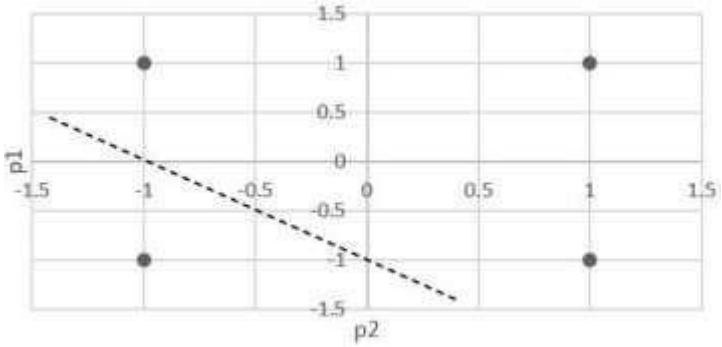


Figure 34: Decision boundary for bipolar AND function using Hebb rule after first training pair

Presenting the second input pair, we get

Input			Target	Weights		
p_1	p_2	b	p_1 AND p_2	$w_{1,1}$	$w_{1,2}$	b
1	−1	1	−1	0	2	0

So, the separating line becomes

$$p_2 = 0$$

Figure 35 shows that the response of the net after second training pair.

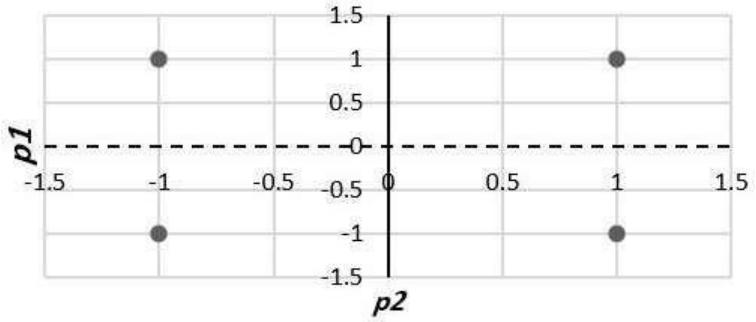


Figure 35: Decision boundary for bipolar AND using Hebb rule after second training pair

Presenting the third input pair, we get

Input			Target	Weights		
p_1	p_2	b	p_1 AND p_2	$w_{1,1}$	$w_{1,2}$	b
−1	1	1	−1	1	1	−1

So, the separating line becomes

$$p_2 = -p_1 + 1$$

Figure 36 shows that the response of the net after third training pair.

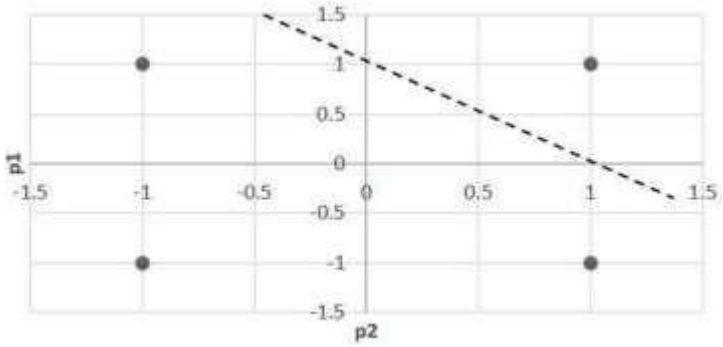


Figure 36: Decision boundary for bipolar AND function using Hebb rule after third training pair

Presenting the fourth input pair, we get

Input			Target	Weights		
p_1	p_2	b	p_1 AND p_2	$w_{1,1}$	$w_{1,2}$	b
−1	−1	1	−1	2	2	−2

Even though the weights have changed, the separating line is still

$$p_2 = -p_1 + 1$$

So the graph of the decision regions remains as in Figure 36.

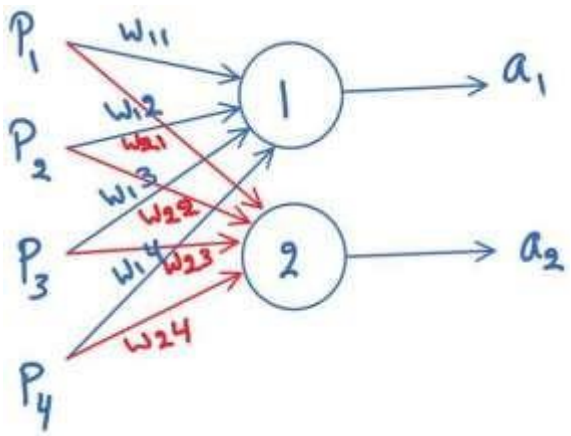
6.2.1.4 Example:

Let $\mathbf{p}_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix}$, $\mathbf{t}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $\mathbf{p}_2 = \begin{bmatrix} 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{bmatrix}$, $\mathbf{t}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Use Hebbian learning rule to train the neural network. Graph the network. (Hint: Don't use bias)

Solution:

Take, initial weights are:

$$w_{1,i} = 0 \quad , w_{2,i} = 0 \quad i = 1, 2, 3, 4$$



Presenting the first input pattern $(\mathbf{p}_1, \mathbf{t}_1)$

$w_{1,1} = t_1\, p_1 = (1)(0.5) = 0.5$

$w_{1,2} = t_1\, p_2 = (1)(-0.5) = -0.5$

$w_{1,3} = t_1\, p_3 = (1)(0.5) = 0.5$

$w_{1,4} = t_1\, p_4 = (1)(-0.5) = -0.5$

$w_{2,1} = t_2\, p_1 = (-1)(0.5) = -0.5$

$w_{2,2} = t_2\, p_2 = (-1)(-0.5) = 0.5$

$w_{2,3} = t_2\, p_3 = (-1)(0.5) = -0.5$

$w_{2,4} = t_2\, p_4 = (-1)(-0.5) = 0.5$

Now, if we presenting the pattern $(\mathbf{p}_2, \mathbf{t}_2)$, we get wrong output

$a_1 = w_{1,1}p_1 + w_{1,2}p_2 + w_{1,3}p_3 + w_{1,4}p_4$

$= (0.5)(0.5) + (-0.5)(0.5) + (0.5)(-0.5) + (-0.5)(-0.5) = 0$

$a_2 = w_{2,1}p_1 + w_{2,2}p_2 + w_{2,3}p_3 + w_{2,4}p_4$

$= (-0.5)(0.5) + (0.5)(0.5) + (-0.5)(-0.5) + (0.5)(-0.5) = 0$

Which mean we need to train the network again to get the right output. (i.e. repeat the above calculation for the pattern $(\mathbf{p}_2, \mathbf{t}_2)$).

6.2.1.5 Note:

The above calculation can be done using matrix notation

$$\mathbf{W} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_Q\mathbf{p}_Q^T$$

$$\mathbf{W} = [\mathbf{t}_1\ \mathbf{t}_2\ \ldots\ \mathbf{t}_Q] \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} = \mathbf{T}\mathbf{P}^T$$

So, the weights for the example 6.2.1.4 are

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & -0.5 & 0.5 & -0.5 \\ 0.5 & 0.5 & -0.5 & -0.5 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

Now, check

$$\rightarrow \quad \mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad , \quad \mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

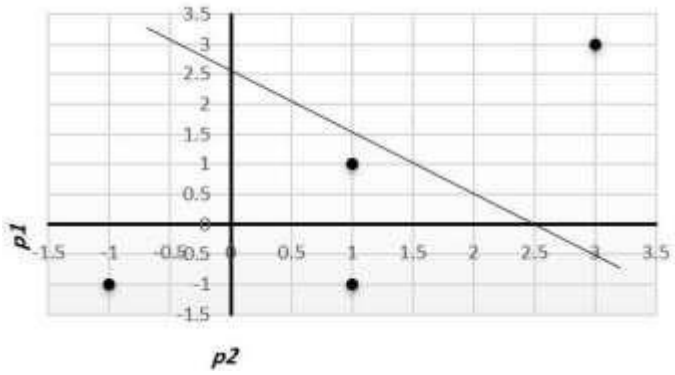
6.2.1.6 Example:

Is the following problem linear separable or not, explain your answer. Apply neural network to solve this problem.

p_1	p_2	t
1	1	0
−1	−1	0
1	−1	0
3	3	1

Solution:

This problem is linear separable because the data can be separate by a line



Multiple – input neuron with threshold function $f(n) = \begin{cases} 0 & n < T \\ 1 & n > T \end{cases}$ can solve this problem as the following:

Method 3 (Hebb rule):

Use Hebb rule with threshold function $f(n) = \begin{cases} 0 & n < 7 \\ 1 & n > 7 \end{cases}$

$$\mathbf{W} = \mathbf{TP}^{T} = [0 \quad 0 \quad 0 \quad 1] \begin{bmatrix} 1 & 1 \\ -1 & -1 \\ 1 & -1 \\ 3 & 3 \end{bmatrix} = [3 \quad 3]$$

Now, check

$$\rightarrow \quad \mathbf{a} = f(\mathbf{WP}) = f\left([3 \quad 3] \begin{bmatrix} 1 & -1 & 1 & 3 \\ 1 & -1 & -1 & 3 \end{bmatrix}\right) = f\left(\begin{bmatrix} 6 \\ -6 \\ 0 \\ 18 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$