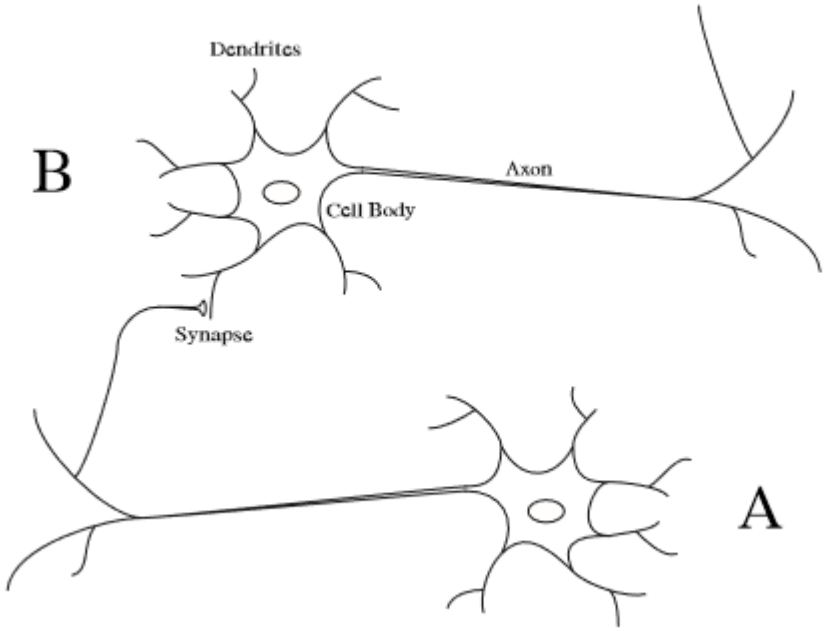
Hebbian Learning

Instructor: Dr. Mohammad Rashedur Rahman

Hebb's Postulate

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

D. O. Hebb, 1949



Extended Hebb Rule

- Hebb proposed that learning occurs by modification of synapse strengths (weights) in a manner such that if two interconnected neurons are both “on” at the same time, then the weight between those neurons should be increased.
- The original statement only talks about neurons firing at the same time and does not say about reinforcing neurons that do not fire at the same time.
- However a stronger form of learning occurs if we also increase the weights of the neurons that are “off” at the same time.
- We call this as “Extended Hebb Rule” [McClelland and Rumelhart, 1988] and it has an improved computational power and refer as Hebb rule.

Learning Algorithm in Hebb Net

- We refer to a single layer (feedforward) neural net trained using the extended Hebb rule as **Hebb net.**
- Since we are considering a single layer net, one of the interconnected neurons will be an input unit and one and one an output unit.
- The linear associator is an example of a type of neural network called an *associative memory*. The task of an associative memory is to learn Q pairs of prototype input/output vectors:

  { {**p**1, **t**1} {**p**2, **t**2} ….. {**p**q, tq*} }*
- In other words, if the network receives an input $p=p_q$ then it should producean output $a=t_q$, for . In addition, if the input is changed slightly (i.e., $p=p_q+c$) then the output should only be changed slightly (i.e., $a=t_q+d$).

Mathematically Interpretation of Hebb rule

- First, let's rephrase the postulate: If two neurons on either side of a synapse are activated simultaneously, the strength of the synapse will increase. We can easily notice that that the connection (synapse) between input and output is the weight .
- Therefore Hebb's postulate would imply that if a positive p_j produces a positive a_i then the weight w_{ij} should increase. This suggests that one mathematical interpretation of the postulate could be

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\, f_i(a_{iq}) g_j(p_{jq}) , \qquad (7.4)$$

where p_{jq} is the jth element of the qth input vector $\mathbf{p}_q$; a_{iq} is the ith element of the network output when the qth input vector is presented to the network; and α is a positive constant, called the learning rate. This equation says that the change in the weight w_{ij} is proportional to a product of functions of the activities on either side of the synapse. For this chapter we will simplify Eq. (7.4) to the following form

$$w_{ij}^{new} = w_{ij}^{old} + \alpha a_{iq} p_{jq} . \qquad (7.5)$$

Mathematical Interpretation of Hebb Rule (cont..)

- Note that this expression actually extends Hebb's postulate beyond its strict interpretation.

- The change in the weight is proportional to a product of the activity on either side of the synapse. Therefore, not only do we increase the weight when both p_j and a_i are positive, but we also increase the weight when they are both negative.

- In addition, this implementation of the Hebb rule will decrease the weight whenever p_j and a_i have opposite sign.

Mathematical Interpretation of Hebb Rule (cont..)

- The Hebb rule defined in Eq. (7.5) is an *unsupervised* learning rule. It does not require any information concerning the target output.
- In this chapter we are interested in using the Hebb rule for supervised learning, in which the target output is known for each input vector. For the *supervised* Hebb rule we substitute the target output for the actual output. In this way, we are telling the algorithm what the network *should* do, rather than what it is currently doing.
- The resulting equation is

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq}p_{jq}, \tag{7.6}$$

where t_{iq} is the ith element of the qth target vector $\mathbf{t}_q$. (We have set the learning rate α to one, for simplicity.)

Notice that Eq. (7.6) can be written in vector notation:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q\mathbf{p}_q^T. \tag{7.7}$$

Example of Hebb Net for Simple AND (Input are binary(1,0) and output is binary (1,0)

Step 0. Initialize all weights:

$w_i = 0 \quad (i = 1 \text{ to } n).$

Step 1. For each input training vector and target output pair, s : *t*, do steps 2–4.

Step 2. Set activations for input units:

$x_i = s_i \quad (i = 1 \text{ to } n).$

Step 3. Set activation for output unit:

$y = t.$

Step 4. Adjust the weights for

$w_i(\text{new}) = w_i(\text{old}) + x_i y \quad (i = 1 \text{ to } n).$

Adjust the bias:

$b(\text{new}) = b(\text{old}) + y.$

Logic functions

Example 2.5 A Hebb net for the AND function: binary inputs and targets

INPUT			TARGET
(x_1	x_2	1)	
(1	1	1)	1
(1	0	1)	0
(0	1	1)	0
(0	0	1)	0

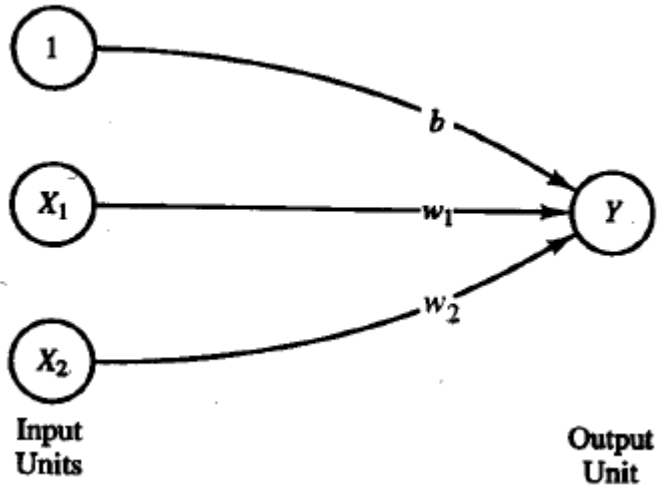
For each training input: target, the weight change is the product of the input vector and the target value, i.e.,

$$\Delta w_1 = x_1 t, \quad \Delta w_2 = x_2 t, \quad \Delta b = t.$$

The new weights are the sum of the previous weights and the weight change. Only one iteration through the training vectors is required. The weight updates for the first input are as follows:

INPUT			TARGET	WEIGHT CHANGES			WEIGHTS		
(x_1	x_2	1)		(Δw_1	Δw_2	Δb)	(w_1	w_2	b)
							(0	0	0)
(1	1	1)	1	(1	1	1)	(1	1	1)

Hebb Net Learning Example



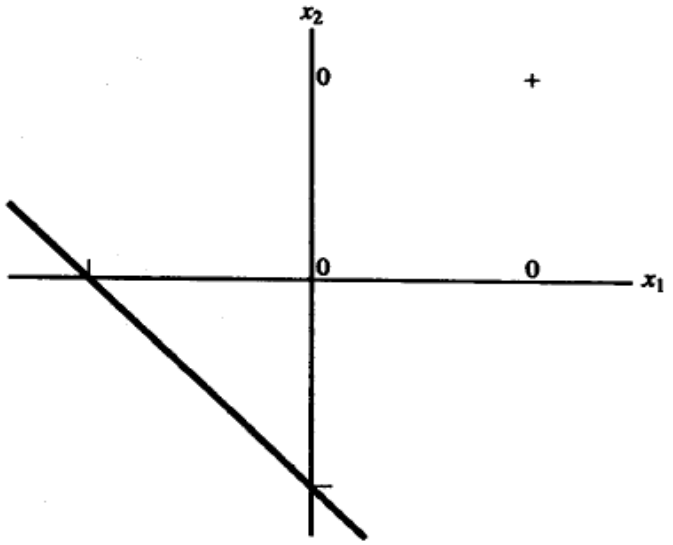


Figure 2.7 Decision boundary for binary AND function using Hebb rule after first training pair.

The boundary between the values of x_1 and x_2 for which the net gives a positive response and the values for which it gives a negative response is the separating line

$$b + x_1 w_1 + x_2 w_2 = 0,$$

or (assuming that $w_2 \neq 0$),

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

Now the separating line becomes x2= -x1-1

The graph, presented in Figure 2.7, shows that the response of the net will now be correct for the first input pattern. Presenting the second, third, and fourth training inputs shows that 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":

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
(1 0 1)	0	(0 0 0)	(1 1 1)
(0 1 1)	0	(0 0 0)	(1 1 1)
(0 0 1)	0	(0 0 0)	(1 1 1)

Hebb Net Learning for AND with Bipolar Signal (1,-1)

INPUT			TARGET
$(x_1$	x_2	1)	
(1	1	1)	1
(1	−1	1)	−1
(−1	1	1)	−1
(−1	−1	1)	−1

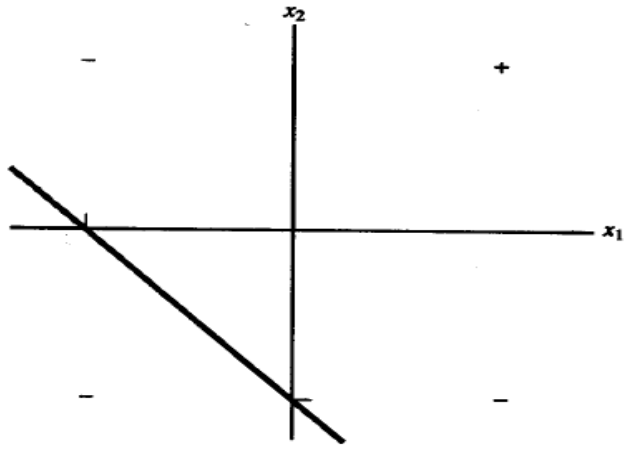


Figure 2.9 Decision boundary for the AND function using Hebb rule after first training pair (bipolar inputs and targets).

Presenting the first input, including a value of 1 for the third component, yields the following:

INPUT			TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)		$(\Delta w_1$	Δw_2	$\Delta b)$	$(w_1$	w_2	$b)$
							(0	0	0)
(1	1	1)	1	(1	1	1)	(1	1	1)

The separating line becomes

$$x_2 = -x_1 - 1.$$

Hebb Net Learning for AND with Bipolar Signal (1,-1) (Cont..)

Presenting second Input will give

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
			(1 1 1)
(1 −1 1)	−1	(−1 1 −1)	(0 2 0)

The separating line becomes

$$x_2 = 0.$$

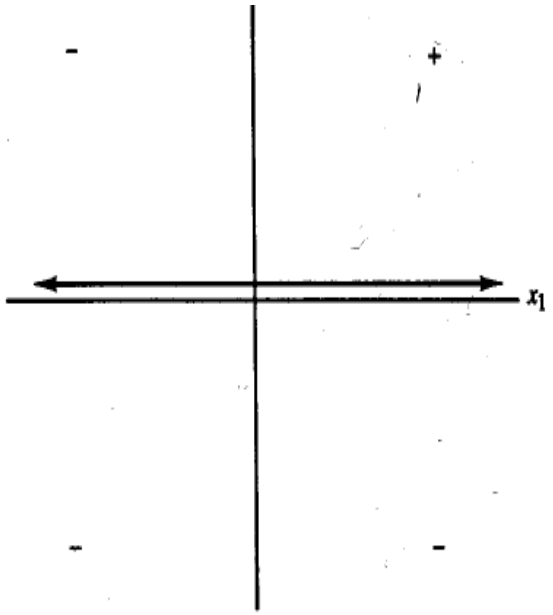


Figure 2.10 Decision boundary for bipolar AND function using Hebb rule after second training pattern (boundary is x_1-axis).

Hebb Net Learning for AND with Bipolar Signal (1,-1) (Cont..)

Presenting Third Input will give

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
			(0 2 0)
(−1 1 1)	−1	(1 −1 −1)	(1 1 −1)

The separating line becomes

$$x_2 = -x_1 + 1.$$

Presenting Last Input will give

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
			(1 1 −1)
(−1 −1 1)	−1	(1 1 −1)	(2 2 −2)

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

$$x_2 = -x_1 + 1.$$

$b + w_1 x_1 + w_2 x_2 = 0$

$-2 + 2x_1 + 2x_2 = 0$

$x_2 = -x_1 + 1$

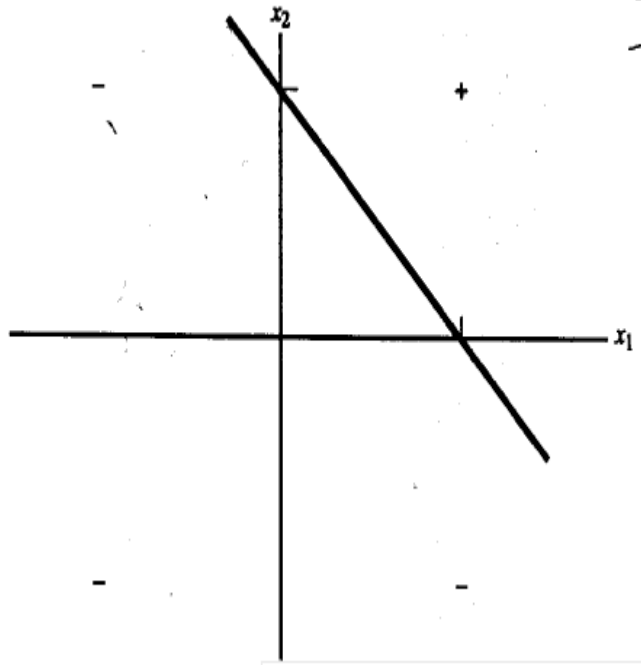
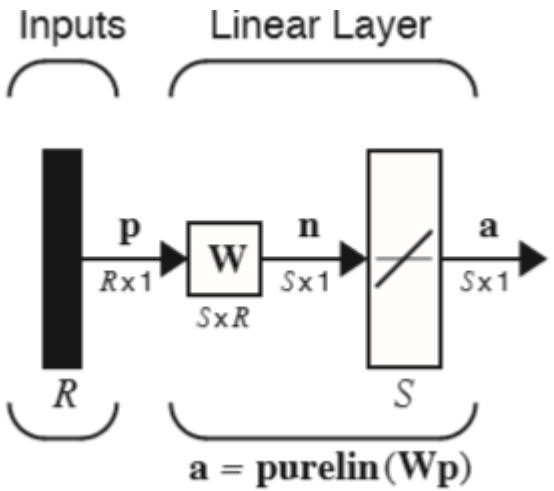


Figure 2.11 Decision boundary for bipolar AND function using Hebb rule after third training pattern.

Hebb Net is Like Linear Associator



$$\mathbf{a} = \mathbf{W}\mathbf{p} \qquad a_i = \sum_{j=1}^{R} w_{ij} p_j$$

Training Set:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

If we assume that the weight matrix is initialized to zero and then each of the Q input/output pairs are applied once to Eq. (7.7), we can write

$$\mathbf{W} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_Q\mathbf{p}_Q^T = \sum_{q=1}^{Q} \mathbf{t}_q\mathbf{p}_q^T. \tag{7.8}$$

This can be represented in matrix form:

$$\mathbf{W} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix} \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} = \mathbf{T}\mathbf{P}^T, \tag{7.9}$$

where

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_Q \end{bmatrix}. \tag{7.10}$$

Performance Analysis of Hebb Net

Performance Analysis

Let's analyze the performance of Hebbian learning for the linear associator. First consider the case where the $\mathbf{p}_q$ vectors are orthonormal (orthogonal and unit length). If $\mathbf{p}_k$ is input to the network, then the network output can be computed

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \left(\sum_{q=1}^{Q} \mathbf{t}_q \mathbf{p}_q^T \right) \mathbf{p}_k = \sum_{q=1}^{Q} \mathbf{t}_q (\mathbf{p}_q^T \mathbf{p}_k) . \tag{7.11}$$

Since the $\mathbf{p}_q$ are orthonormal,

$$\begin{aligned} (\mathbf{p}_q^T \mathbf{p}_k) &= 1 \qquad q = k \\ &= 0 \qquad q \neq k. \end{aligned} \tag{7.12}$$

Therefore Eq. (7.11) can be rewritten

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k . \tag{7.13}$$

The output of the network is equal to the target output. This shows that, if the input prototype vectors are orthonormal, the Hebb rule will produce the correct output for each input.

An example of Orthonormal Input Pattern

As an example, suppose that the prototype input/output vectors are

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix} 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}. \tag{7.15}$$

(Check that the two input vectors are orthonormal.)

The weight matrix would be

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & -0.5 & 0.5 & -0.5 \\ 0.5 & 0.5 & -0.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}. \tag{7.16}$$

If we test this weight matrix on the two prototype inputs we find

$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \tag{7.17}$$

Apple Banana Example (Not Orthonormal)

Banana Apple Normalized Prototype Patterns

$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \quad \left\{ \mathbf{p}_1 = \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_1 = [-1] \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_2 = [1] \right\}$$

Weight Matrix (Hebb Rule):

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -0.5774 & 0.5774 & -0.5774 \\ 0.5774 & 0.5774 & -0.5774 \end{bmatrix} = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix}$$

Tests:

Banana $\quad \mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} -0.6668 \end{bmatrix}$

Apple $\quad \mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.5774 \\ 0.5774 \\ \end{bmatrix} = \begin{bmatrix} 0.6668 \end{bmatrix}$

Pseudo-Inverse Rule

Performance Index: $\mathbf{W}\mathbf{p}_q = \mathbf{t}_q \qquad q = 1, 2, \ldots, Q$

$$F(\mathbf{W}) = \sum_{q=1}^{Q} \|\mathbf{t}_q - \mathbf{W}\mathbf{p}_q\|^2$$

Matrix Form:

$$\mathbf{W}\mathbf{P} = \mathbf{T}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_Q \end{bmatrix}$$

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{W}\mathbf{P}\|^2 = \|\mathbf{E}\|^2$$

Pseudo-Inverse Rule (Cont..)

$$\mathbf{WP} = \mathbf{T}$$

Minimize:

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{WP}\|^2 = \|\mathbf{E}\|^2$$

If an inverse exists for $\mathbf{P}$, $F(\mathbf{W})$ can be made zero:

$$\mathbf{W} = \mathbf{TP}^{-1}$$

When an inverse does not exist $F(\mathbf{W})$ can be minimized using the pseudoinverse:

$$\mathbf{W} = \mathbf{TP}^{+}$$

$$\mathbf{P}^{+} = (\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T}$$

Apple- Banana Example with Pesudoinverse

$$\left\{\mathbf{p}_1 = \begin{bmatrix}-1\\1\\-1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}-1\end{bmatrix}\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\1\\-1\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\end{bmatrix}\right\} \qquad \mathbf{W} = \mathbf{T}\mathbf{P}^+ = \begin{bmatrix}-1 & 1\end{bmatrix}\left(\begin{bmatrix}-1 & 1\\1 & 1\\-1 & -1\end{bmatrix}\right)^+$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T = \begin{bmatrix}3 & 1\\1 & 3\end{bmatrix}^{-1}\begin{bmatrix}-1 & 1 & -1\\1 & 1 & -1\end{bmatrix} = \begin{bmatrix}-0.5 & 0.25 & -0.25\\0.5 & 0.25 & -0.25\end{bmatrix}$$

$$\mathbf{W} = \mathbf{T}\mathbf{P}^+ = \begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}-0.5 & 0.25 & -0.25\\0.5 & 0.25 & -0.25\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}$$

$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}-1\\1\\-1\end{bmatrix} = \begin{bmatrix}-1\end{bmatrix} \qquad \mathbf{W}\mathbf{p}_2 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\1\\-1\end{bmatrix} = \begin{bmatrix}1\end{bmatrix}$$

Application of Hebb Rule in Pattern Recognition

- Now let's see how we might use the Hebb rule on a practical, although greatly oversimplified, pattern recognition problem.

- For this problem we will use a special type of associative memory — the *autoassociative* memory.

- In an *autoassociative memory* the desired output vector is equal to the input vector (i.e.t_q=p_q,).

- We will use an autoassociative memory to store a set of patterns and then to recall these patterns, even when corrupted patterns are provided as input.

Application of Hebb Rule in Pattern Recognition (Cont..)

- The patterns we want to store are shown to the left. (Since we are designing an autoassociative memory, these patterns represent the input vectors and the targets.)
- They represent the digits {0, 1, 2} displayed in a 6X5 grid. We need to convert these digits to vectors, which will become the prototype patterns for our network

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

- Each white square will be represented by a “-1”, and each dark square will be represented by a “1”. Then, to create the input vectors, we will scan each 6X5 grid one column at a time. For example, the first prototype pattern will be

$$\mathbf{p}_1 = [-1\ 1\ 1\ 1\ 1\ -1\ 1\ -1\ -1\ -1\ -1\ 1\ 1\ -1\ \ldots\ 1\ -1]^T$$

Application of Hebb Rule in Pattern Recognition (Cont..)

- The vector p1 corresponds to the digit “0”, p2 to the digit “1”, and p3 to the digit “2”.
- Using the Hebb rule, the weight matrix is computed

$$\mathbf{W} = \mathbf{p}_1\mathbf{p}_1^T + \mathbf{p}_2\mathbf{p}_2^T + \mathbf{p}_3\mathbf{p}_3^T.$$

- Because there are only two allowable values for the elements of the prototype vectors, we will modify the linear associator so that its output elements can only take on values of “-1” or “1”.
- We can do this by replacing the linear transfer function with a symmetrical hard limit transfer function.
- The resulting network is displayed in next slide

Application of Hebb Rule in Pattern Recognition (Cont..)