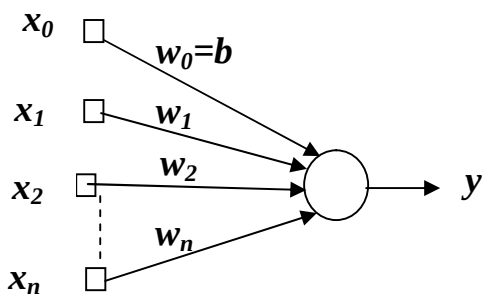
Single-layer patterns classification's learning rules

1- Hebb net (Hebb rule)

It is the easiest and simplest learning rule. This rule had been improved and extended in 1988.

- we shall refer to a single-layer (feedforward) neural net trained using the extended Hebb rule as a **Hebb net.**
- If data are represented in bipolar form, it is easy to express the desired weight update



$$w_i(new) = w_i(old) + x_i y$$

$$b(new) = b(old) + y$$

in general,

$$\Delta W = XY$$

and

$$W(new) = W(old) + \Delta W$$

Example-4 : a Hebb net for AND function, using binary input and binary target (we see later that the resulting separating line and results are incorrect).

Sol:

$\Delta w_1 = x_1 t$

$\Delta w_2 = x_2 t$

$\Delta b = 1 * t$

Only one iteration through the training vector is required

Input			Target
x_1	x_2	x_0	t
1	1	1	1
1	0	1	0
0	1	1	0
0	0	1	0

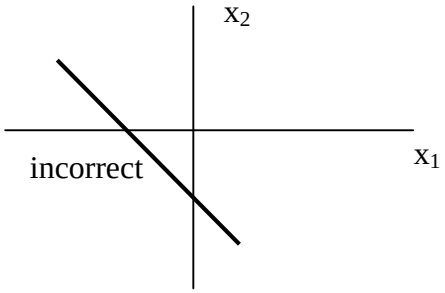
Input			Target	Weight change			Weight		
x_1	x_2	1	t	**Δw_1**	**Δw_2**	**Δb**	**w_1**	**w_2**	b
							0	0	0
1	1	1	1	1	1	1	1	1	1
1	0	1	0	0	0	0	1	1	1
0	1	1	0	0	0	0	1	1	1
0	0	1	0	0	0	0	1	1	1

In the last three steps because the target is zero no learning occurs.

The separating line :

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

$x_2 = - x_1 - 1$



Example-5 : a Hebb net for AND function, using bipolar inputs and targets.

Sol:

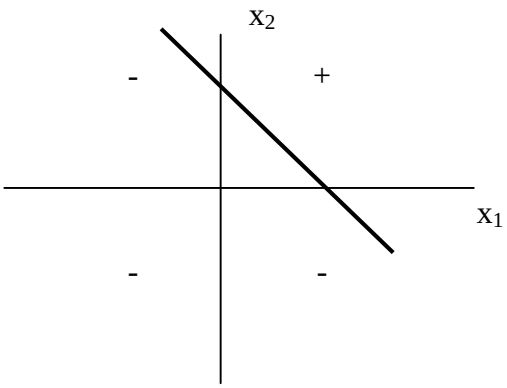
Input			Target
x_1	x_2	x_0	t
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Input			Target	Weight change			Weight		
x_1	x_2	1	t	**Δw_1**	**Δw_2**	**Δb**	**w_1**	**w_2**	b
							0	0	0
1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	0	2	0
-1	1	1	-1	1	-1	-1	1	1	-1
-1	-1	1	-1	1	1	-1	2	2	-2

The separating line

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

$$x_2 = - x_1 + 1$$



2- Perceptron

The perceptron rule is more powerful learning rule than the Hebb rule.

- The goal of the net is to classify each input pattern as **belonging** or **not belonging** to a particular class.
- Belonging is signified by a response +1.
- Not belonging is signified by a response -1.
- The output of the Perceptron is:

$$y = f(net)$$

$$net = b + \sum_i x_i w_i$$

$$y = f(net) = \begin{cases} 1 & if \quad net > \theta \\ 0 & if \quad -\theta \le net \le \theta \\ -1 & if \quad net < -\theta \end{cases}$$

$$w_i(new) = w_i(old) + \alpha t x_i$$

where, ***t*** is the target value (+1 or -1),

α is the learning rate, in general ($0 < \alpha \le 1$)

Note-1: in Perceptron learning rule, if y ≠ target then the weight must be updated:

$$w_i(new) = w_i(old) + \alpha t x_i$$

$$b_i(new) = b_i(old) + \alpha t$$

when y = target, then

$$w_i(new) = w_i(old)$$

$$b_i(new) = b_i(old)$$

Note-2: the result of the learning is two separable lines called "the line bounding the inequality"

- the first line separate the region of positive response from the region of zero response.

$$w_1x_1 + w_2x_2 + b > \theta$$

- the second line separate the region of zero response from the region of negative response.

$$w_1x_1 + w_2x_2 + b < -\theta$$

- if the value of $\theta = 0$, then there will be one separable line (separate the positive region from the negative region).

Example-6: A perceptron for the AND function, bipolar inputs and targets. The training process for the bipolar input α = 1. The threshold and initial weights = 0

Sol:

$$y = f(net)$$

$$net = b + \sum_i x_i w_i$$

for $y \neq t$,

$$w_i(new) = w_i(old) + \alpha t x_i \rightarrow \Delta w_i = \alpha t x_i$$

$$b_i(new) = b_i(old) + \alpha t \rightarrow \Delta b_i = \alpha t$$

for $y = target$,

$$w_i(new) = w_i(old) \rightarrow \Delta w_i = 0$$

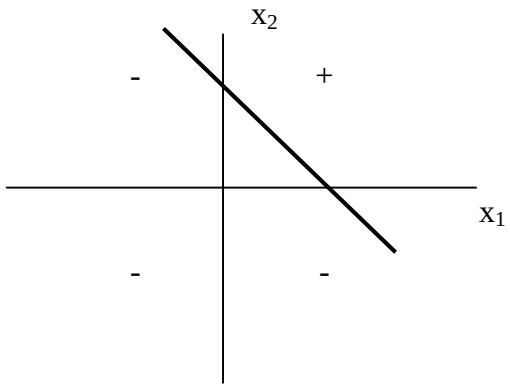
$$b_i(new) = b_i(old) \rightarrow \Delta b_i = 0$$

Input				*net*		*f(net)*		*target*		*Weight change*			*weight*		
x_1	x_2	1				y		t		Δw_1	Δw_2	Δb	w_1	w_2	b
1st. epoch													*0*	*0*	*0*
1	*1*	*1*		*0*		*0*		*1*		*1*	*1*	*1*	*1*	*1*	*1*
1	*-1*	*1*		*1*		*1*		*-1*		*-1*	*1*	*-1*	*0*	*2*	*0*
-1	*1*	*1*		*2*		*1*		*-1*		*1*	*-1*	*-1*	*1*	*1*	*-1*
-1	*-1*	*1*		*-3*		*-1*		*-1*		*0*	*0*	*0*	*1*	*1*	*-1*
2nd epoch															
1	*1*	*1*		*1*		*1*		*1*		*0*	*0*	*0*	*1*	*1*	*-1*
1	*-1*	*1*		*-1*		*-1*		*-1*		*0*	*0*	*0*	*1*	*1*	*-1*
-1	*1*	*1*		*-1*		*-1*		*-1*		*0*	*0*	*0*	*1*	*1*	*-1*
-1	*-1*	*1*		*-3*		*-1*		*-1*		*0*	*0*	*0*	*1*	*1*	*-1*

Since all **Δw's** are 0 in 2nd. epoch, the system was fully trained after the first epoch.

The separable line is

$$x_2 + x_1 - 1 = 0$$



Example: A Perceptron to classify letters from different fonts (character recognition)

Consider the 21 input patterns in Figure 1 as examples of A or not-A. In other words, we train the perceptron to classify each of these vectors as belonging, or not belonging, to the class A. In that case, the target value for each pattern is either 1 or – 1.

We could, of course, use the same vectors as examples of B or not-B and train the net in a similar manner.

Note: that is because we are using a single-layer net, the weights for the output unit signifying A do not have any interaction with the weights for the output unit signifying B.

Therefore, we can solve these two problems at the same time, by allowing a column of weights for each output unit.

Our net would have 63 input units and 2 output units. The first output unit would correspond to "A or not-A", the second unit to "B or not-B." Continuing this idea, we can identify 7 output units, one for each of the 7 categories into which we wish to classify our input.

The architecture of such a net is shown in Figure 2.

For this example, each input vector is a 63-tuple representing a letter expressed as a pattern on a 7 x 9 grid of pixels. There are seven components to the output vector, each representing a letter: A, B, C, D, E, K, or J. For ease of reading, we show the target output pattern indicating that

the input was an "A" as (A . . . . . .), a "B" a (. B . . . . .), etc.

The training input patterns and target responses must be converted to an appropriate form for the neural net to process. A **bipolar** representation has better computational characteristics than does a binary representation.

The input patterns may be converted to bipolar vectors.

The target output pattern (A . . . . . .) becomes the bipolar vector (1, -1, -1, -1, -1, -1, -1), and the target pattern (. B . . . . .) is represented by the bipolar vector (- 1, 1, - 1, -1, -1, -1, -1).

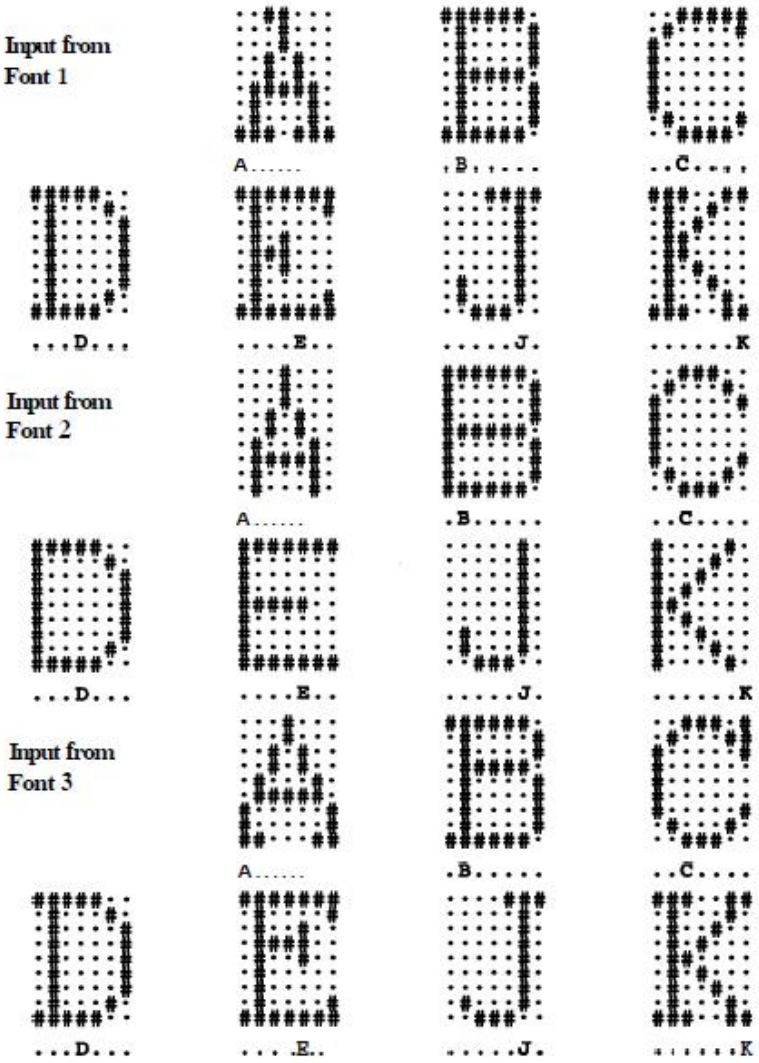


Figure 1 Training input and target output patterns.

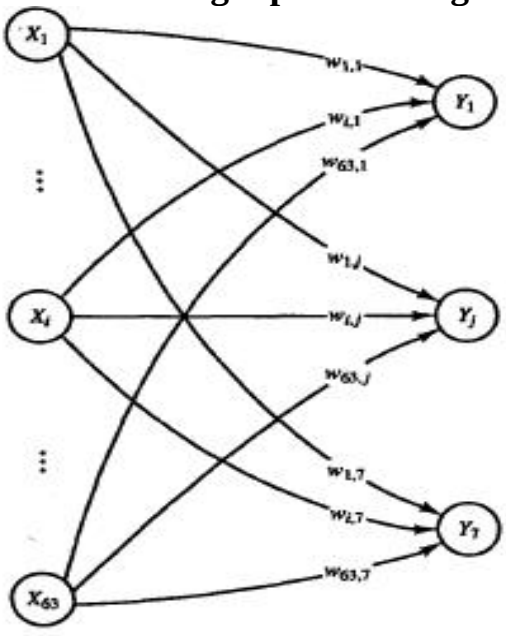


Figure 2 Perceptron to classify into seven categories.

A modified training algorithm for several output categories (threshold = 0,
learning rate = 1, bipolar training pairs) is as follows:

Step 0. Initialize weights and biases
(0 or small random values).

Step 1. While stopping condition is false, do Steps 1-6.

Step 2. For each bipolar training pair s : t, do Steps 3-5.

Step 3. Set activation of each input unit, $i = 1, \ldots n$: ***(n=63)***

$$x_i = s_i$$

Step 4. Compute activation of each output unit,

$j=1, \ldots ,m$

$$net_j = b_j + \sum_i w_{ij} x_i$$

$$y_j = \begin{cases} 1 & \text{if } net_j > \theta \\ 0 & \text{if } -\theta \le net_j \le \theta \\ -1 & \text{if } net_j < -\theta \end{cases}$$

Step 5. Update biases and weights, $j = 1, \ldots , m$; ***(m=7)***

$i = 1, \ldots , n$:

If $t_j \neq y_j$ then

$$b(new)_j = b(old)_j + t_j$$

$$w(new)_{ij} = w(old)_{ij} + t_j x_i$$

Else, biases and weights remain unchanged.

Step 6. Test for stopping condition:
If no weight changes occurred in Step 2, stop; otherwise, continue.

After training, the net correctly classifies each of the training vectors.