Example 3.8 Generate XOR function using McCulloch-Pitts neuron by writing an M-file.

Solution The truth table for the XOR function is,

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

The MATLAB program is given by,

Program

```
%XOR function using McCulloch-Pitts neuron
clear;
clc;
%Getting weights and threshold value
disp('Enter weights');
w11=input('Weight w11=');
w12=input('weight w12=');
w21=input('Weight w21=');
w22=input('weight w22=');
v1=input('weight v1=');
v2=input('weight v2=');
disp('Enter Threshold Value');
theta=input('theta=');
x1=[0 0 1 1];
x2=[0 1 0 1];
z=[0 1 1 0];
con=1;
while con
  zin1=x1*w11+x2*w21;
  zin2=x1*w21+x2*w22;
  for i=1:4
     if zin1(i)>=theta
        y1(i)=1;
     else
        y1(i)=0;
     end
```

```
        if zin2(i)>=theta
            y2(i)=1;
        else
         y2(i)=0;
       end
    end
    yin=y1*v1+y2*v2;
    for i=1:4
       if yin(i)>=theta;
          y(i)=1;
       else
          y(i)=0;
       end
    end
    disp('Output of Net');
    disp(y);
    if y==z
       con=0;
    else
       disp('Net is not learning enter another set of weights and Threshold value');
       w11=input('Weight w11=');
       w12=input('weight w12=');
       w21=input('Weight w21=');
       w22=input('weight w22=');
       v1=input('weight v1=');
       v2=input('weight v2=');
       theta=input('theta=');
    end
  end
  disp('McCulloch-Pitts Net for XOR function');
  disp('Weights of Neuron Z1');
  disp(w11);
  disp(w21);
  disp('weights of Neuron Z2');
  disp(w12);
  disp(w22);
  disp('weights of Neuron Y');
  disp(v1);
  disp(v2);
  disp('Threshold value');
  disp(theta);
```

4.1.1 Hebbian Learning Rule

The earliest and simplest learning rule for a neural net is generally known as the Hebb rule. Hebbian learning rule suggested by Hebb in 1949. Hebb's basic idea is that if a unit U_j receives an input from a unit U_i and both unite are highly active (positive) , then the weight W_{ij} (from unit i to unit j) should be strengthened(increase), otherwise the weight decrease.

This idea is formulated as:-

$$\Delta w_{ij} = \zeta\ x_i y_j$$

Where ζ is the learning rate $\zeta = \alpha = 1$,

Δw is the weight change

$$w(new) = w(old) + xy$$

$$\therefore\ w(new) = w(old) + \Delta w$$

Algorithm (Hebbian learning Rule)

Step 0: Initialize all weights

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

Step 1: for each I/p training vector target o/p

Pair. S : t do steps 2- 4.

Step 2 : Set activations for I/P units:

$$w_i = s_i\ \ (i = 1 \text{ to } n)$$

Step 3 : set activation for O/P unit :

$$y = t$$

Step 4 : Adjust the weights for

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

Adjust the bias:

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

Note that the bias is adjusted exactly like a weight from a "unit" whose output signal is always 1.

Ex 4:

A Hebb net for the AND function: binary input and targets

Input		Target	
1	1	1	1
1	0	1	0
0	1	1	0
0	0	1	0

$\Delta w_1 = x_1 y, \quad \Delta w_2 = x_2 y, \quad \Delta b = y$ Initial weights = 0, $w_1 = 0$, $w_2 = 0$, $w_3 = 0$

	x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 0
1	1	1	1	1	1	1	1	1	1	1
2	1	0	1	0	0	0	0	1	1	1
3	0	1	1	0	0	0	0	1	1	1
4	0	0	1	0	0	0	0	1	1	1

The first input pattern shows that the response will be correct presenting the second, third, and fourth training i/p shows that because the target value is 0, no learning occurs. Thus, using binary target values prevents the net from learning only pattern for which the target is "off".

The AND function can be solved if we modify its representation to express the inputs as well as the targets in bipolar form. Bipolar representation of the inputs and targets allows modifications of a weight when the input unit and the target value are both "on" at the same time and when they are both "off" at the same time and all units will learn whenever there is an error in the output. The Hebb net for the AND function: bipolar inputs and targets are:

$$\Delta w_1 = x_1 * y$$
$$= 1 * 1 = 1$$
$$w_1(new) = w_1(old) + \Delta w_1$$
$$= 0 + 1 = 1$$

x_1	x_2	b	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Presenting the first input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 0
1	1	1	1	1	1	1	1	1	1

Presenting the second input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 1	w_2 1	b 1
1	-1	1	-1	-1	1	-1	0	2	0

Presenting the third input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 2	b 0
-1	1	1	-1	1	-1	-1	1	1	-1

Presenting the fourth input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 1	w_2 1	b -1
-1	-1	1	-1	1	1	-1	2	2	-2

The first iteration will be:-

Input			target	Weight change			weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 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

Second Method

$W_{ij} = \sum X_i Y_j$ ***or*** $[W]=[X]^T[Y]$

Ex. 5

What would the weights be if Hebbian learning is applied to the data shown in the following table? Assume that the weights are all zero at the start.

p	x_1	x_2	y
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

With weights that you've just found, what output values are produce with a threshold of 1, using hyperbolic activation function.

	x_1	x_2	y	Δw_1	Δw	w_1 0	w_2 0
1	0	0	1	0	0	0	0
2	0	1	1	0	1	0	1
3	1	0	0	0	0	0	1
4	1	1	1	1	1	1	2

$$p=0,\quad w_1=0,\quad w_1=0$$

$$p=1,\quad w_1=0+x_1*y=0$$
$$w_2=0+x_2*y=0$$

$$p=2,\quad w_1=0+0*1=0$$
$$w_2=0+1*1=1$$

$$p=3,\quad w_1=0+1*0=0$$
$$w_2=1+0*0=1$$

$$p=4,\quad w_1=0+1*1=1$$
$$w_2=1+1*1=2$$

$$\therefore\quad w_1=1\ ,\ w_2=2$$

$$net=\sum_{i=1}^{4}X_i+W_i$$

$$p=1,\quad net=x_1*w_1+x_2*w_2$$
$$=0*1+0*2=0$$

$$p=2,\quad net=0*1+1*2=2$$

$$p=3,\quad net=1*1+0*2=1$$

$$p=4,\quad net=1*1+1*2=3$$

$$output=F(net+\theta)=\frac{e^{(net+\theta)}-e^{-(net+\theta)}}{e^{(net+\theta)}+e^{-(net+\theta)}}$$

p	x_1	x_2	net	y
1	0	0	0	0
2	0	1	2	1
3	1	0	1	0
4	1	1	3	1