Example 1: Write a matlab program to find the weight matrix of an auto associative net to store the vector (1 1 -1 -1). Test the response of the network by presenting the same pattern and recognize whether it is a known vector or unknown vector.

The auto associative net has the same inputs and targets. The MATLAB program for the auto associative net is as follows:

Program

```
clc;
clear;
x = [1 1 -1 -1];
w=zeros (4, 4);
w=x'*x;
yin=x*w;
for i=1:4
    if yin(i)>0
        y(i)=1;
    else
        y(i) = -1;
    end
end
disp ('Weight matrix');
disp (w);
if x == y
    disp ('The vector is a Known Vector');
else
    disp ('The vector is a Unknown Vector');
end
```

Output

```
Weight matrix
1 1 -1 -1
1 1 -1 -1
-1 -1 1 1
-1 -1 1 1

The vector is a known vector.
```
Example 2: Write an M–file to store the vectors \((-1 -1 -1 -1)\) and \((-1 -1 1 1)\) in an auto associative net. Find the weight matrix. Test the net with \((1 1 1 1)\) as input.

The MATLAB program for the auto association problem is as follows:

Program

```matlab
clc;
clear;
x=[-1 -1 -1 -1;-1 -1 1 1];
t=[1 1 1 1];
w=zeros(4, 4);
for i=1:2
    w=w + x(i,1:4)'*x(i,1:4);
end
yin = t*w;
for i=1:4
    if yin(i)>0
        y(i)=1;
    else
        y(i)=-1;
    end
end
disp ('The calculated weight matrix');
disp (w);
if x(1,1:4)==y(1:4) | x(2,1:4)==y(1:4)
    disp ('The vector is a Known Vector');
else
    disp ('The vector is a unknown vector');
end
```

Output

The calculated weight matrix

```
2 2 0 0
2 2 0 0
0 0 2 2
0 0 2 2
```

The vector is an unknown vector.

Example 3: Write an M–file to calculate the weights for the following patterns using hetero associative neural net for mapping four input vectors to two output vectors.

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>t1</th>
<th>t2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Solution

%Hetero associative NN for mapping input vectors to output vectors
clc;
clear;
x=[1 1 0 0; 1 0 1 0; 1 1 1 0; 0 1 1 0];
t=[1 0; 1 0; 0 1; 0 1];
w=zeros (4, 2);
for i=1:4
    w=w+x (i, 1:4)'*t(i,1:2);
end
disp('Weight matrix');
disp(w);

Example 4: Write a MATLAB program to store the vector (1 1 1 0) in bipolar binary form and calculate the weight matrix for Hopfield net.

%The MATLAB program for calculating the weight matrix is as follows
%Discrete Hopfield net
clc;
clear;
x=[1 1 1 0];
w=(2*x'-1)*(2*x-1);
for i=1:4
    w (i, i)=0;
end
disp('Weight matrix');
disp(w);

Example 5: Auto-associative Memories (continuous):

%Auto-associative Memories (continuous)
x1=[-0.3; 0.9; -0.2];
x2=[0.44; -0.7; 0.9];
x3=[0.9; 0.6; 0.8];
Total_M = x1*x1' + x2*x2' + x3*x3';
estimate_x1= Total_M *x1;
estimate_x2= Total_M *x2;
estimate_x3= Total_M *x3;
%Estimates are not perfect because of non-orthogonality of the vectors
%Euclidean distance between x1 and other key vectors
d11 = norm(x1-estimate_x1);
d21 = norm(x2-estimate_x1);
d31 = norm(x3-estimate_x1);
% As expected the response vector estimate_x1 is closest to x1
%Euclidean distance between x2 and other key vectors
d12 = norm(x1-estimate_x2);
d22 = norm(x2-estimate_x2);
d32 = norm(x3-estimate_x2);
% As expected the response vector estimate_x2 is closest to x2
%Euclidean distance between x3 and other key vectors
d13 = norm(x1-estimate_x3);
d23 = norm(x2-estimate_x3);
d33 = norm(x3-estimate_x3);
% As expected the response vector estimate_x3 is closest to x3

- Matlab code for converting continuous data (x1 vector) to binary bipolar data.

```matlab
for i=1:length(x1)
    if x1(i)>0
        x1(i) = 1;
    else
        x1(i) = -1;
    end
end
```

- In Matlab Hopfield networks can be implemented as vector matrix manipulations. To make the pattern vectors as easy as possible to read and write we define them as row vectors.

- We prefer to make the calculations within the interval $[-1, 1]$ (bipolar) as this makes the calculations simpler. It is, however, easier to type in and to visually recognize values in the range $[0, 1]$ (binary). Therefore, it may be better to use this for input and output. Translate a vector from the binary format into the bipolar.

**Example 6:**

```matlab
%Enter three test patterns.
x1b= [0 0 1 0 1 0 0 1];
x2b= [0 0 0 0 0 1 0 0];
x3b= [0 1 1 0 0 1 0 1];
%Translate a vector from the binary format into the bipolar.
x1= [-1 -1 1 1 -1 -1 -1 -1];
x2= [-1 -1 -1 -1 1 1 -1 -1];
x3= [-1 1 1 1 -1 -1 -1 -1];
x1 = 2* x1b -1;
x2 = 2* x2b -1;
x3 = 2* x3b -1;
%Calculate a weight matrix.
w=x1'*x1+x2'*x2+x3'*x3-3*eye (8, 8);
%Check if the network was able to store all three patterns.
```
x1test=sign (w*x1');
x2test=sign (w*x2');
x3test=sign (w*x3');

% Convergence and attractors.
% Can the memory recall the stored patterns from distorted inputs
% patterns? Define a few new patterns which are distorted versions of
% the original ones:
x1d=[1 0 1 0 1 0 0 1];
x2d=[1 1 0 0 1 0 0];
x3d=[1 1 1 0 1 1 0 1];
% x1d has one bit error, x2d and x3d have two bit errors.
x1d=[1 -1 1 -1 1 -1 1 1];
x2d=[1 1 -1 -1 1 -1 -1 -1];
x3d=[1 1 1 -1 1 1 -1 1];

Hopfield neural networks using Matlab Neural Network Tool Box

- Hopfield neural networks can be simulated by using the Neural Network Tool Box. The architecture is shown below.

- **net = newhop (T)** takes one input argument:
  - T - R x Q matrix of Q target vectors. (Values must be +1 or -1) and returns a new Hopfield recurrent neural network with stable points at the vectors in T.
  - Hopfield networks consist of a single layer with the dotprod weight function, netsum net input function, and the satlins transfer function.

**Example 7:**
- Consider the following design example. Suppose that we want to design a network with two stable points in a three-dimensional space T.
  
  \[
  T = [-1 -1 1; 1 -1 1]';
  \]
- We can execute the design with:
  
  **net = newhop (T);**
- To check that the network is stable at these points use them as **initial layer delay conditions**. If the network is stable we would expect that the outputs...
\( Y \) will be the same. (Since Hopfield networks have no inputs, the second argument to \( \text{sim} \) is \( Q = 2 \) when using matrix notation).

\[
\begin{align*}
\text{Ai} &= \mathbf{T}; \\
[Y, Pf, Af] &= \text{sim}(\text{net}, 2, [\ ], \text{Ai}); \\
\% Y
\end{align*}
\]

- To see if the network can correct a corrupted vector, run the following code, which simulates the Hopfield network for five time steps. (Since Hopfield networks have no inputs, the second argument to \( \text{sim} \) is \( \{Q TS\} = [1 5] \) when using cell array notation.)

\[
\begin{align*}
\text{Ai} &= \{[-0.9; -0.8; 0.7]\}; \\
[Y, Pf, Af] &= \text{sim}(\text{net}, \{1 5\}, \{\}, \text{Ai}); \\
\% Y\{1\}
\end{align*}
\]

- If you run the above code, \( Y\{1\} \) will equal \( \mathbf{T}(\cdot, 1) \) if the network has managed to convert the corrupted vector \( \text{Ai} \) to the nearest target vector.

**Description of \text{sim} function**

**Purpose:** simulate a neural network.

**Syntax:**

\[
\begin{align*}
[Y, Pf, Af, E, \text{perf}] &= \text{sim}(\text{net}, P, \Pi, \text{Ai}, T) \\
[Y, Pf, Af, E, \text{perf}] &= \text{sim}(\text{net}, \{Q TS\}, \Pi, \text{Ai}, T) \\
[Y, Pf, Af, E, \text{perf}] &= \text{sim}(\text{net}, Q, \Pi, \text{Ai}, T)
\end{align*}
\]

**Description** \( \text{sim} \) simulates neural networks.

\( [Y, Pf, Af, E, \text{perf}] = \text{sim}(\text{net}, P, \Pi, \text{Ai}, T) \) takes,

\( \text{net} \) - Network.
\( P \) - Network inputs.
\( \Pi \) - Initial input delay conditions, default = zeros.
\( \text{Ai} \) - Initial layer delay conditions, default = zeros.
\( T \) - Network targets, default = zeros.

and returns,

\( Y \) - Network outputs.
\( Pf \) - Final input delay conditions.
\( Af \) - Final layer delay conditions.
\( E \) - Network errors.
\( \text{perf} \) - Network performance.

Note that arguments \( \Pi, \text{Ai}, Pf, \) and \( Af \) are optional and need only be used for networks that have input or layer delays.

\( \text{sim}\)'s signal arguments can have two formats: **cell array or matrix.**