Solution Manual to

Aritificial Neural Networks

(B. Yegnanarayana, Prentice Hall of India Pvt Ltd, New Delhi, 1999)

B. Yegnanarayana and S. Ramesh

Dept. of Computer Science and Engineering

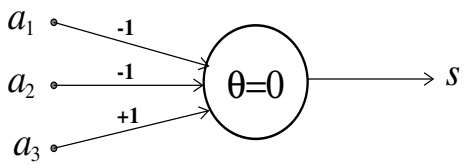
Indian Institute of Technology Madras

Chennai - 600036

March 2001

CHAPTER 1

Solution to problem 1 (a):



Using the logic function,

$$\begin{aligned} f(x) &= 1,\ x > \theta \\ &= 0,\ x \leq \theta \end{aligned}$$

where $x = \sum_i w_i a_i$, the truth table is obtained by giving all possible combinations of a_1, a_2, a_3. The results are shown in the following table.

Truth table:

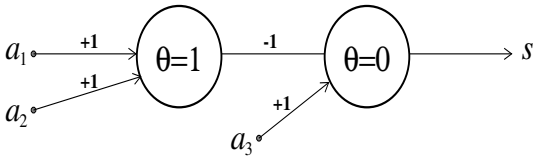
a_1	a_2	a_3	s
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

To derive the logic function, one can use the following Karnaugh map (K-map) representation of the truth table. The logic function is obtained from the map by expressing the terms corresponding to the entries with 1.

a_1 \ a_2a_3	00	01	11	10
0	0	1	0	0
1	0	0	0	0

Logic function $s(a_1, a_2, a_3) = \bar{a}_1 \bar{a}_2 a_3$

(b)



Truth table:

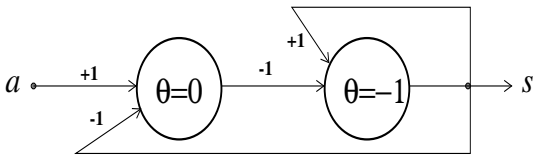
a_1	a_2	a_3	s
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

K-map representation of the truth table is as follows:

a_1 \ a_2a_3	00	01	11	10
0	0	1	1	0
1	0	1	0	0

Logic function $s(a_1, a_2, a_3) = \bar{a}_2 a_3 + \bar{a}_1 a_3$

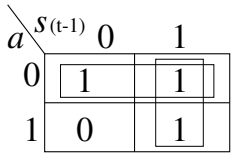
(c)



Truth table:

a	$s(t-1)$	$s(t)$
0	0	1
0	1	1
1	0	0
1	1	1

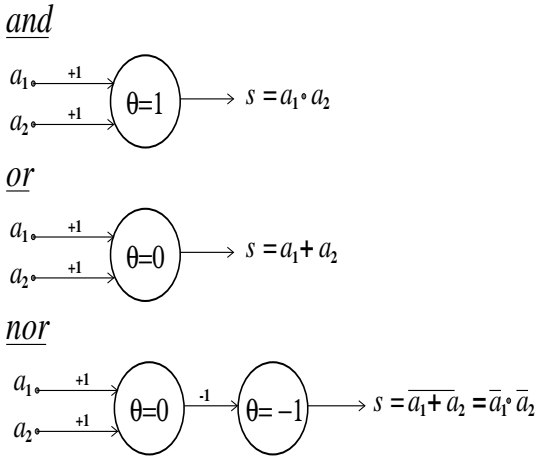
K-map representation of the truth table is as follows:



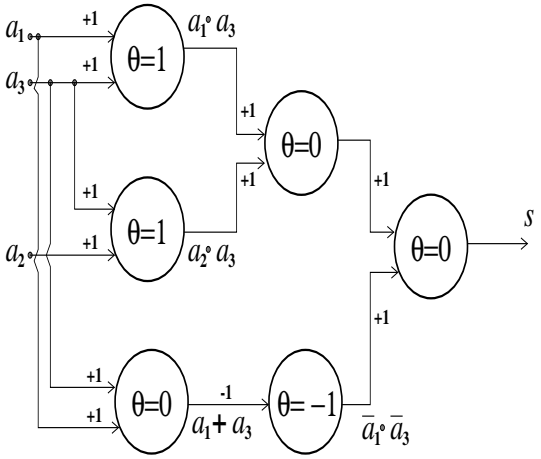
Logic function $s(t) = \bar{a} + s(t-1)$

Solution to problem 2:

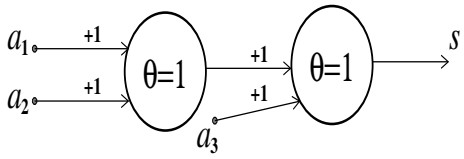
Using the following basic logic networks



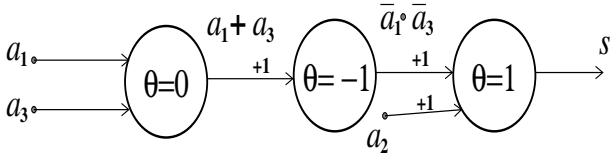
(a) The logic network for $s(a_1, a_2, a_3) = a_1a_3 + a_2a_3 + \bar{a}_1\bar{a}_3$ is given by



(b) The logic network for $s(a_1, a_2, a_3) = a_1 a_2 a_3$ is given by (c)



The logic network for $s(a_1, a_2, a_3) = \bar{a}_1 a_2 \bar{a}_3$ is given by



Solution to problem 3:

By applying the operations of an M-P neuron, the output of the network shown in Fig.P1.2 for the input $[1\,1\,1]^T$ is $s = 0$.

Solution to problem 4:

For the output function: $f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$

Input vectors: $\mathbf{i}_1 = [1\,1\,0\,0]^T$, $\mathbf{i}_2 = [1\,0\,0\,1]^T$, $\mathbf{i}_3 = [0\,0\,1\,1]^T$, $\mathbf{i}_4 = [0\,1\,1\,0]^T$

Choose the learning parameter $\eta = 1$

Let the initial weight vector $\boldsymbol{w}(0) = [1\,0\,0\,0]^T$

(a)

For the output function shown in Fig.P1.3b.

Output of the first input vector $s = f(\boldsymbol{w}^T(0)\mathbf{i}_1) = 1$

Weight update $\boldsymbol{w}(1) = \boldsymbol{w}(0) + s\mathbf{i}_1 = [1\,0\,0\,0]^T + [1\,1\,0\,0]^T = [2\,1\,0\,0]^T$

Output of the second input vector $s = f(\boldsymbol{w}^T(1)\mathbf{i}_2) = 1$

Weight update $\boldsymbol{w}(2) = \boldsymbol{w}(1) + s\mathbf{i}_2 = [2\,1\,0\,0]^T + [1\,0\,0\,1]^T = [3\,1\,0\,1]^T$

Output of the third input vector $s = f(\boldsymbol{w}^T(2)\mathbf{i}_3) = 1$

Weight update $\boldsymbol{w}(3) = \boldsymbol{w}(2) + s\mathbf{i}_3 = [3\,1\,0\,1]^T + [0\,0\,1\,1]^T = [3\,1\,1\,2]^T$

Output of the fourth input vector $s = f(\boldsymbol{w}^T(3)\mathbf{i}_4) = 1$

Weight update $\boldsymbol{w}(4) = \boldsymbol{w}(3) + s\mathbf{i}_4 = [3\,1\,1\,2]^T + [0\,1\,1\,0]^T = [3\,2\,2\,2]^T$

(b)

For the sigmoid output function shown in Fig.P1.3c $f(x) = \dfrac{1}{1+e^{-x}}$

Output for the first input vector $s = f(\boldsymbol{w}^T(0)\mathbf{i}_1) = f(1) = 0.731$

Weight update

$$\begin{aligned}\boldsymbol{w}(1) &= \boldsymbol{w}(0) + s\mathbf{i}_1 \\ &= [1\ 0\ 0\ 0]^T + 0.73\ [1\ 1\ 0\ 0]^T \\ &= [1\ 0\ 0\ 0]^T + [0.73 \quad 0.73 \quad 0 \quad 0]^T \\ &= [1.73 \quad 0.73 \quad 0 \quad 0]^T\end{aligned}$$

Output for the second input vector $s = f(\boldsymbol{w}^T(1)\mathbf{i}_2) = f(1.73) = 0.85$

Weight update

$$\begin{aligned}\boldsymbol{w}(2) &= \boldsymbol{w}(1) + s\mathbf{i}_2 \\ &= [1.73 \quad 0.73 \quad 0 \quad 0]^T + 0.85\ [1\ 0\ 0\ 1]^T \\ &= [1.73 \quad 0.73 \quad 0 \quad 0]^T + [0.85 \quad 0 \quad 0 \quad 0.85]^T \\ &= [2.58 \quad 0.73 \quad 0 \quad 0.85]^T\end{aligned}$$

Output for the third input vector $s = f(\boldsymbol{w}^T(2)\mathbf{i}_3) = f(0.85) = 0.70$

Weight update

$$\begin{aligned}\boldsymbol{w}(3) &= \boldsymbol{w}(2) + s\mathbf{i}_3 \\ &= [2.58 \quad 0.73 \quad 0 \quad 0.85]^T + 0.7\ [0\ 0\ 1\ 1]^T \\ &= [2.58 \quad 0.73 \quad 0 \quad 0.85]^T + [0 \quad 0 \quad 0.7 \quad 0.7]^T \\ &= [2.58 \quad 0.73 \quad 0.7 \quad 1.55]^T\end{aligned}$$

Output for the fourth input vector $s = f(\boldsymbol{w}^T(3)\mathbf{i}_4) = f(1.43) = 0.81$

Weight update

$$\begin{aligned}\boldsymbol{w}(4) &= \boldsymbol{w}(3) + s\mathbf{i}_4 \\ &= [2.58 \quad 0.73 \quad 0.7 \quad 1.55]^T + 0.81\ [0\ 1\ 1\ 0]^T \\ &= [2.58 \quad 0.73 \quad 0.7 \quad 1.55]^T + [0 \quad 0.81 \quad 0.81 \quad 0]^T \\ &= [2.58 \quad 1.54 \quad 1.51 \quad 1.55]^T\end{aligned}$$

The examples show that the successive weights accumulate the weighted sum of the input vectors. Also the significance of non-zero (arbitrary) initial weight vector can be seen. Note that with zero initial weight vector there will be no change in the weights for the first output function (Fig.P1.3b).

Solution to problem 5 (a):

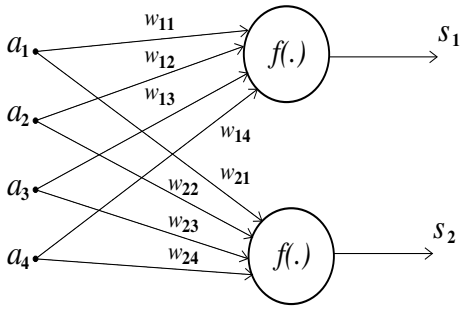
Input: $[1100]^T$ $[1001]^T$ $[0011]^T$ $[0110]^T$

Output: $[11]^T$ $[10]^T$ $[01]^T$ $[00]^T$

Perceptron law applicable for binary output function: Weight update

$\Delta w_{ij} = \eta(b_i - s_i)a_j,$

where $s_i = f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$



Choose the initial weights:

$\boldsymbol{w}_1 = [w_{11}\ w_{12}\ w_{13}\ w_{14}]^T = [0.8 \quad 0.9 \quad 0.6 \quad 0.6]^T$

$\theta_1 = 0.5$

$\boldsymbol{w}_2 = [w_{21}\ w_{22}\ w_{23}\ w_{24}]^T = [0.9 \quad 0.7 \quad 0.6 \quad 0.8]^T$

$\theta_2 = 0.5$

The weights and thresholds after 1000 iterations for different learning rate parameter:

η	w_{11}	w_{12}	w_{13}	w_{14}	θ_1	w_{21}	w_{22}	w_{23}	w_{24}	θ_2
0.3	1.1	0.6	0.0	0.6	0.8	0.6	0.4	0.3	0.5	1.1
0.1	0.8	0.6	0.1	0.4	1	0.5	0.4	0.4	0.5	1.1
0.04	0.8	0.66	0.2	0.44	0.9	0.5	0.46	0.48	0.52	1.02
0.01	0.8	0.66	0.22	0.46	0.88	0.5	0.49	0.5	0.51	1.0

The Matlab program used to implement this question is given below:

```
function[ ]=prob1_5a()
%    Given Inputs and Desired Outputs
%    The value '-1' at the end of each row of inputs will act as Bias input
     Inputs = [1 1 0 0 -1;1 0 0 1 -1;0 0 1 1 -1;0 1 1 0 -1];
     des_output = [1 1;1 0;0 1;0 0];
     max_iteration = 1000;
%    From eta value 1 to 0.1 in steps of 0.1
     for eta = 1 : -0.1 : 0.1
%       Next line can be uncommented to run program with random weights
%       weights = randn(2,5);
%       Next line can be commented while using random weights
        weights = [.8 .9 .6 .6 .5;.9 .7 .6 .8 .5]; % Initial weights
        for iteration = 1 : max_iteration
           for i = 1 : 4
              for k = 1 : 2
                 wted_sum = 0;
                 for j = 1 : 5
                    wted_sum = wted_sum + Inputs(i,j) * weights(k,j);
                 end
                 calc_output(k) = wted_sum;
              end
              wt_change=eta*(des_output(i,1)-sgn(calc_output(1)))*Inputs(i,:);
              weights(1,:) = weights(1,:) + wt_change;
              wt_change=eta*(des_output(i,2)-sgn(calc_output(2)))*Inputs(i,:);
              weights(2,:) = weights(2,:) + wt_change;
           end
        end
   end
```

(b)

Delta learning law for $f(x) = \dfrac{1}{1+e^{-x}}$

Weight update $\Delta w_{ij} = \eta(b_i - f(\boldsymbol{w}^T\boldsymbol{a}))\dot{f}(\boldsymbol{w}^T\boldsymbol{a})a_j$

The network is trained using the same initial weight and bias values used in part (a).

Weights after 1000 iterations for different learning rate parameter:

η	w_{11}	w_{12}	w_{13}	w_{14}	θ_1	w_{21}	w_{22}	w_{23}	w_{24}	θ_2
0.3	3.9164	0.5149	-2.9916	0.5098	0.9753	0.4977	0.4955	0.4977	0.5000	1.0046
0.1	3.2949	0.5174	-2.3702	0.5073	0.9753	0.4982	0.4942	0.4957	0.4997	1.0061
0.04	2.7382	0.5241	-1.8135	0.5006	0.9753	0.5035	0.4830	0.4847	0.5052	1.0118
0.01	1.8260	0.5603	-0.8980	0.4677	0.9721	0.5428	0.4354	0.4372	0.5446	1.0200

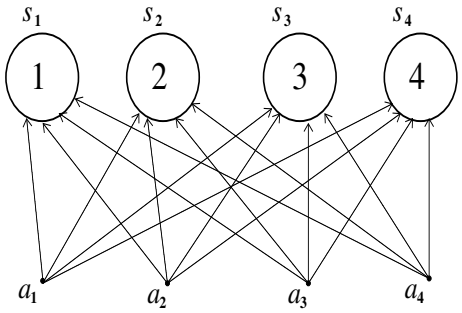
The Matlab program used to implement this question is given below. In this program $sigmoid$ and $derv\_sigmoid$ are function calls which implements sigmoidal function and first derivative of sigmoidal function.

```
function[ ]=prob1_5b()
%    Given Inputs and Desired Outputs
%    The value '-1' at the end of each row of inputs will act as Bias input
Inputs = [1 1 0 0 -1;1 0 0 1 -1;0 0 1 1 -1;0 1 1 0 -1];
des_output = [1 1;1 0;0 1;0 0];
output_error = 10;
max_iteration = 1000;
%    From eta value 1 to 0.1 in steps of 0.1
   for eta = 1 :-0.1 :0.1
%     Next line can be uncommented to run program with random weights
%     weights = randn(2,5);
%     Next line can be commented while using random weights
      weights = [.8 .9 .6 .6 .5;.9 .7 .6 .8 .5]; % Initial weights
      for iteration = 1 : max_iteration
         for i = 1 : 4
            for k = 1 : 2
```

```
            wted_sum = 0;
            for j = 1 : 5
               wted_sum = wted_sum + Inputs(i,j) * weights(k,j);
            end
            calc_output(k) = wted_sum;
         end
         wt_change = eta * (des_output(i,1) - sigmoid(calc_output(1))) *...
                         derv_sigmoid(calc_output(1)) * Inputs(i,:);
         weights(1,:) = weights(1,:) + wt_change;

         wt_change = eta * (des_output(i,2) - sigmoid(calc_output(2))) *...
                         derv_sigmoid(calc_output(1)) * Inputs(i,:);
         weights(2,:) = weights(2,:) + wt_change;
      end
   end
end
```

Discussion: The weights will not be the same for different values of η.

Solution to problem 6:



A program is written to implement the instar learning law. Weight vectors are normalized at each iteration. Choosing $\eta = 0.6$ and

$$\text{Initial weight vector} = \begin{bmatrix} 0.4 & 0.2 & 0.5 & 0.7 \\ 0.9 & 0.7 & 0.4 & 0.8 \\ 0.2 & 0.6 & 0.9 & 0.8 \\ 0.7 & 0.9 & 0.8 & 0.4 \end{bmatrix}$$

Final weight vector is obtained as:
$$\begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5774 & 0.5774 & 0 & 0.5774 \\ 0.5774 & 0.5774 & 0.5774 & 0 \\ 0.7071 & 0.7071 & 0 & 0 \end{bmatrix}$$

Grouping is done by collecting the input patterns for which each of the output units give maximum output. The input $[0\,0\,0\,0]$ is arbitrarily assigned to Group IV. Note that the maximum Hamming distance between any pair in a group is 2. The resulting outputs of 16 binary input vectors are

input	s_1	s_2	s_3	s_4
0000	0	0	0	<u>0</u>
0001	0.5	<u>0.5774</u>	0	0
0010	0.5	0	<u>0.5774</u>	0
0011	<u>1.0</u>	0.5774	0.5774	0
0100	0.5	0.5774	0.5774	<u>0.7071</u>
0101	1.0	<u>1.1548</u>	0.5774	0.7071
0110	1.0	0.5774	<u>1.1548</u>	0.7071
0111	<u>1.5</u>	1.1548	1.1548	0.7071
1000	0.5	0.5774	0.5774	<u>0.7071</u>
1001	1.0	<u>1.1548</u>	0.5774	0.7071
1010	1.0	0.5774	<u>1.1548</u>	0.7071
1011	<u>1.5</u>	1.1548	1.1548	0.7071
1100	1.0	1.1548	1.1548	<u>1.4142</u>
1101	1.5	<u>1.7322</u>	1.1548	1.4142
1110	1.5	1.1548	<u>1.7322</u>	1.4142
1111	<u>2.0</u>	1.7322	1.7322	1.4142

Group I	Group II	Group III	Group IV
0011	0001	0010	0100
0111	0101	0110	1000
1011	1001	1010	1100
1111	1101	1110	0000

The Matlab program used to implement this question is given below:

```
function[ ]=prob1_6()
Eta = 0.6;
Weights = [0.4 0.2 0.5 0.7; 0.9 0.7 0.4 0.8; 0.2 0.6 0.9 0.8 ; 0.7 0.9 0.8 0.4];
MaxIteration = 2000;
Inp = [0 0 0 0;0 0 0 1;0 0 1 0;0 0 1 1;0 1 0 0;0 1 0 1;0 1 1 0;0 1 1 1;...
       1 0 0 0;1 0 0 1;1 0 1 0;1 0 1 1;1 1 0 0;1 1 0 1;1 1 1 0;1 1 1 1];
for j = 1 : 16
   for i = 1 : MaxIteration
      for k = 1 : 4
         WtedSum(k) = Inp(j,:) * Weights(k,:)';
      end
      [val c] = max(WtedSum);
      Wt_Adj = Eta * ( Inp(j,:) - Weights(c,:));
      Weights(c,:) = Weights(c,:) + Wt_Adj ;
      Weights(c,:) = normalize(w(c,:));
   end
end
```

CHAPTER 2

Solution to problem 1:

Consider the additive autoassociative model

$$\dot{x}_i(t) = -A_i x_i(t) + \sum_{j=1}^{N} f_j(x_j(t)) w_{ij} + B_i I_i \tag{1}$$

where $f_j(.)$ is the output function of the j^{th} unit. For

$$\begin{aligned} A_i &= \frac{1}{R_i} = \text{membrane conductance}, \\ w_{ij} &= \frac{1}{R_{ij}} \end{aligned}$$

and for the linear output function

$$f_j(x_j(t)) = x_j(t), \qquad \forall j = 1, 2, 3.....N$$

Equation (1) becomes

$$(1) \Rightarrow \dot{x}_i(t) = \frac{-x_i(t)}{R_i} + \sum_{j=1}^{N} \frac{x_j(t)}{R_{ij}} + B_i I_i \tag{2}$$

The Perkel-model is given by

$$\begin{aligned} \dot{x}_i(t) &= \frac{-x_i(t)}{R_i} + \sum_{j=1}^{N} \frac{x_j(t) - x_i(t)}{R_{ij}} + B_i I_i \\ &= \frac{-x_i(t)}{R_i} + \sum_{j=1}^{N} \frac{x_j(t)}{R_{ij}} - \sum_{j=1}^{N} \frac{x_i(t)}{R_{ij}} + B_i I_i \\ &= -x_i(t) \left[\frac{1}{R_i} + \sum_{j=1}^{N} \frac{1}{R_{ij}} \right] + \sum_{j=1}^{N} \frac{x_j(t)}{R_{ij}} + B_i I_i \\ \Rightarrow \dot{x}_i(t) &= \frac{-x_i(t)}{R_i'} + \sum_{j=1}^{N} \frac{x_j(t)}{R_{ij}} + B_i I_i \qquad (3) \\ & \quad \text{where} \quad \frac{1}{R_i'} = \frac{1}{R_i} + \sum_{j=1}^{N} \frac{1}{R_{ij}} \end{aligned}$$

Equation (3) is same as Equation (2), and hence the Perkel-model is a special case of the additive autoassociative model.

Solution to problem 2:

A shunting activation model with on-center off-surround configuration is given by

$$\dot{x}_i(t) = -A_i x_i(t) + [B_i - x_i(t)]\, I_i - x_i(t) \sum_{j \neq i} I_j$$

where A_i, B_i and I_i are all positive.

Suppose that $x_i(t) > B_i$. We show by contradiction that this is not true.

For steady state activation value, $\dot{x}_i(t) = 0$

$$\begin{aligned}
&\Rightarrow \quad 0 = -A_i x_i(t) + [B_i - x_i(t)]\, I_i - x_i(t) \sum_{j \neq i} I_j \\
&\Rightarrow \quad [B_i - x_i(t)]\, I_i = A_i x_i(t) + x_i(t) \sum_{j \neq i} I_j \\
&\Rightarrow \quad B_i - x_i(t) > 0, \text{ since RHS is positive} \\
&\Rightarrow \quad x_i(t) < B_i, \text{ which is contradiction.}
\end{aligned}$$

Hence, $x_i(t) < B_i$, for all t.

Solution to problem 3:

General form of a shunting model with excitatory feedback from the same unit and inhibitory feedback from the other units is given by

$$\dot{x}_i(t) = -A_i x_i(t) + (B_i - C_i x_i(t))\, [I_i + f_i(x_i(t))] - (E_i + D_i x_i(t)) \left[J_i + \sum_{j \neq i} f_j(x_j(t)) w_{ij} \right]$$

For steady state activation value, $\dot{x}_i(t) = 0$

$$\Rightarrow 0 = -A_i x_i(t) + (B_i - C_i x_i(t))\, [I_i + f_i(x_i(t))] - (E_i + D_i x_i(t)) \left[J_i + \sum_{j \neq i} f_j(x_j(t)) w_{ij} \right]$$

$$\Rightarrow A_i x_i(t) = (B_i - C_i x_i(t))\, [I_i + f_i(x_i(t))] - (E_i + D_i x_i(t)) \left[J_i + \sum_{j \neq i} f_j(x_j(t)) w_{ij} \right]$$

If $x_i(t) > \frac{B_i}{C_i}$, then the first term on the RHS becomes negative, since $f(x)$=1, $\forall x > 0$ and $I_i > 0$. The contribution due to the second term on the RHS is also negative. Hence the RHS is negative. But since we assumed that $x_i(t) > \frac{B_i}{C_i}$, there is a contradiction.

$$\text{Hence} \quad x_i(t) \leq \frac{B_i}{C_i}.$$

Solution to problem 4:

General form of a shunting model with excitatory feedback from the same unit and inhibitory feedback from the other units is given by,

$$\dot{x}_i(t) = -A_i x_i(t) + (B_i - C_i x_i(t))\left[I_i + f_i(x_i(t))\right] - (E_i + D_i x_i(t))\left[J_i + \sum_{j \neq i} f_j(x_j(t)) w_{ij}\right]$$

For steady state activation value $\dot{x}_i(t) = 0$

$$\Rightarrow 0 = -A_i x_i(t) + (B_i - C_i x_i(t))\left[I_i + f_i(x_i(t))\right] - (E_i + D_i x_i(t))\left[J_i + \sum_{j \neq i} f_j(x_j(t)) w_{ij}\right]$$

$$\Rightarrow A_i x_i(t) = (B_i - C_i x_i(t))\left[I_i + f_i(x_i(t))\right] - (E_i + D_i x_i(t))\left[J_i + \sum_{j \neq i} f_j(x_j(t)) w_{ij}\right]$$

If $x_i(t) < \frac{-E_i}{D_i}$, then the contribution of the first term on the RHS is positive, since $f(x) \geq 0$, for all x. Since the contribution due to the second term on the RHS is also positive, the RHS is positive.

But we assumed that $x_i(t) < \dfrac{-E_i}{D_i}$, which is negative. Thus, there is a contradiction.

Hence $x_i(t) \geq \dfrac{-E_i}{D_i}$.

Solution to problem 5:

General form of a shunting model with excitatory feedback from the same unit and inhibitory feedback from other units is given as

$$\dot{x}_i(t) = -A_i x_i(t) + (B_i - C_i x_i(t))\left[I_i + f_i(x_i(t))\right] - (E_i + D_i x_i(t))\left[J_i + \sum_{j \neq i} f_j(x_j(t)) w_{ij}\right]$$

The contribution of the excitatory term (2^{nd} term) is zero for

$$x_i(t) = \frac{B_i}{C_i}.$$

Likewise, the contribution of the inhibitory term (3^{rd} term) will be zero when

$$x_i(t) = -\frac{E_i}{D_i}.$$

This is equivalent to shunting effect of an electrical circuit. Hence this model is called 'shunting' model.

Solution to problem 6:

(a) $\dot{x}(t) = 0$, for all i.

For $\dot{x}_i(t) = 0$, the resulting activation value corresponds to steady state activation value without any transient effects due to membrane conductance and capacitance.

(b) $\dot{V}(\mathbf{x}) \leq 0$.

If the Lyapunov function is interpreted as energy function, then the condition that $\dot{V}(\mathbf{x}) \leq 0$ means that any change in energy due to change in the state of the network results in a state corresponding to a lower or equal energy. This demonstrates the existence of stable states.

(c) $\dot{V}(\mathbf{x}) = 0$

The states corresponding to this condition are minimum or maximum energy states. Only the minimum energy states are also equilibrium or stable states.

(d) $\dot{w}_{ij} \neq 0$, for all i, j.

Learning takes place when the weights are changing, and hence this corresponds to synaptic dynamics.

Solution to problem 7:

$$\text{General form:} \quad \dot{x}_i(t) = -\, a_i(x_i) \left[b_i(x_i) - \sum_{k=1}^{N} c_{ik} d_k(x_k) \right] \qquad i = 1, 2, ...N \qquad (1)$$

(a) The activation model in Eq.(2.10) is

$$\begin{aligned} \dot{x}_i(t) &= -A_i x_i(t) + \sum_{j=1}^{N} f_j(x_j(t)) w_{ij} + B_i I_i \\ \Rightarrow x_i(t) &= -A_i \left\{ \left[x_i(t) - \frac{B_i I_i}{A_i} \right] - \sum_{k=1}^{N} \left[\frac{-f_k(x_k(t))}{A_i} \right] w_{ik} \right\} \end{aligned} \qquad (2)$$

Comparing equation (2) with the general form given in equation (1)

$$\begin{aligned} a_i(x_i) &= A_i, \\ b_i(x_i) &= x_i(t) - \frac{B_i I_i}{A_i}, \qquad c_{ik} = w_{ik} \\ \text{and} \quad d_k(x_k) &= \frac{-f_k(x_k(t))}{A_i} \end{aligned}$$

(b) The activation model in Eq.(2.21) is

$$\dot{x}_i(t) = -A_i x_i(t) + (B_i - C_i x_i(t))\left[I_i + f_i(x_i(t))\right] - (E_i + D_i x_i(t))\left[J_i + \sum_{j\neq i} f_j(x_j(t))w_{ij}\right]$$

$$\Rightarrow \dot{x}_i(t) = -A_i x_i(t) + (B_i - C_i x_i(t))\left[I_i + f_i(x_i(t))\right] - (E_i + D_i x_i(t))J_i$$
$$-(E_i + D_i x_i(t)) \sum_{k=1,k\neq i}^{N} f_k(x_k(t))w_{ik}$$

$$\Rightarrow \dot{x}_i(t) = (E_i + D_i x_i(t))\left[\frac{-A_i x_i(t) + (B_i - C_i x_i(t))(I_i + f(x_i(t)))}{(E_i + D_i x_i(t))} - J_i\right]$$
$$-(E_i + D_i x_i(t)) \sum_{k=1,k\neq i}^{N} f_k(x_k(t))w_{ik}$$

$$\Rightarrow x_i(t) = (E_i + D_i x_i(t))\left\{\left[\frac{-A_i x_i(t) + (B_i - C_i x_i(t))(I_i + f(x_k(t)))}{(E_i + D_i x_i(t))} - J_i\right] - \sum_{k=1,k\neq i}^{N} f_k(x_k(t))w_{ik}\right\} \tag{3}$$

Comparing equation (3) with general form given in equation (1)

$$\begin{aligned} a_i(x_i) &= E_i + D_i x_i(t) \\ b_i(x_i) &= \frac{-A_i x_i(t) + (B_i - C_i x_i(t))(I_i + f_i(x_i))}{(E_i + D_i x_i(t))} - J_i \\ c_{ik} &= w_{ik} \\ \text{and} \quad d_k(x_k) &= f_k(x_k) \end{aligned}$$

Solution to problem 8:

Consider the R-L-C series circuit

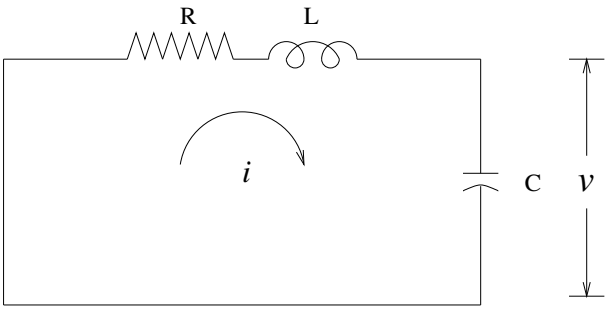
$$\text{Energy :} \quad E(t) = \frac{1}{2}Cv^2(t) + \frac{1}{2}Li^2(t)$$
$$\frac{dE}{dt} = Cv(t)\frac{dv(t)}{dt} + Li(t)\frac{di(t)}{dt} \tag{1}$$

Applying Kirchoff's voltage law to the circuit

$$Ri(t) + L\frac{di(t)}{dt} + \frac{1}{C}\int i(t)dt = 0$$

$$\text{But,} \quad i(t) = C\frac{dv(t)}{dt} \tag{2}$$

$$\therefore \quad Ri(t) + L\frac{di(t)}{dt} + v(t) = 0$$



$$\Rightarrow \frac{di(t)}{dt} = -\frac{v(t)}{L} - \frac{R}{L}i(t) \tag{3}$$

Substituting equations (2) and (3) in equation (1)

$$\begin{aligned}
(1) \Rightarrow \frac{dE}{dt} &= Cv(t)\frac{dv(t)}{dt} + Li(t)\frac{di(t)}{dt} \\
&= Cv(t)\frac{i(t)}{C} + Li(t)\left[-\frac{v(t)}{L} - \frac{R}{L}i(t)\right] \\
&= v(t)i(t) - v(t)i(t) - Ri^2(t) \\
\frac{dE}{dt} &= -Ri^2(t) \\
\Rightarrow \frac{dE}{dt} &\leq 0
\end{aligned}$$

Any function E is said to be Lyapunov function if

(a) It is continuous

(b) $\dot{E}(t) < 0$, which indicates that energy is decreasing with time.

Since energy E of an electric circuit satisfies the above conditions, Lyapunov function represents some form of energy of an electric circuit.

Solution to problem 9:

Using

$$\begin{aligned}
\dot{x}_i(t) &= -a_i(x_i)\left[b_i(x_i) - \sum_k c_{ik}d_k(x_k)\right] \\
\dot{c}_{ik} &= -c_{ik} + d_i(x_i)d_k(x_k) \\
\text{and} \quad V(\mathbf{x}) &= \sum_{i=1}^{N}\int_0^{x_i} b_i(\xi_i)\dot{d}_i(\xi_i)d\xi_i - \frac{1}{2}\sum_{i,k} c_{ik}d_i(x_i)d_k(x_k) + \frac{1}{4}\sum_{i,k} c_{ik}^2 \\
\dot{V} &= \sum_{j=1}^{N}\left[\frac{\partial V}{\partial x_j}\frac{\partial x_j}{\partial t} + \sum_k \frac{\partial V}{\partial c_{jk}}.\frac{\partial c_{jk}}{\partial t}\right] \\
&= \sum_{j=1}^{N}\left[b_j(x_j)\dot{d}_j(x_j) - \sum_k c_{jk}\dot{d}_j(x_j)d_k(x_k)\right]\dot{x}_j + \sum_{j=1}^{N}\left[-\frac{1}{2}\sum_k d_j(x_j)d_k(x_k) + \frac{1}{2}\sum_k c_{jk}\right]\dot{c}_{jk}
\end{aligned}$$

$$= \sum_{j=1}^{N} \dot{d}_j(x_j) \left[b_j(x_j) - \sum_k c_{jk} d_k(x_k) \right] \dot{x}_j - \frac{1}{2} \sum_{j,k} \left[-c_{jk} + d_j(x_j) d_k(x_k) \right] \dot{c}_{jk}$$

Substituting for $\dot{x}_j$ and $\dot{c}_{jk}$, we get

$$\begin{aligned} \dot{V}(\mathbf{x}) &= -\sum_{j=1}^{N} a_j(x_j) \dot{d}_j(x_j) [b_j(x_j) - \sum_k c_{jk} d_k(x_k)]^2 - \frac{1}{2} \sum_{j,k} (\dot{c}_{jk})^2 \\ \therefore \ \dot{V}(\mathbf{x}) &\leq 0, \text{ for } a_j(x_j) > 0 \text{ and } \dot{d}_j(x_j) \geq 0 \end{aligned}$$

Solution to problem 10:

$$\begin{aligned} \dot{x}_i &= -a_i(x_i) \left[b_i(x_i) - \sum_j c_{ij} d_j(y_j) \right] \\ \dot{y}_j &= -a_j(y_j) \left[b_j(y_j) - \sum_i c_{ji} d_i(x_i) \right] \\ \dot{c}_{ij} &= -c_{ij} + d_i(x_i) d_j(y_j) \end{aligned}$$

Lyapunov function:

$$\begin{aligned} V(\mathbf{x}, \mathbf{y}) &= \sum_i \int_0^{x_i} b_i(\xi_i) \dot{d}_i(\xi_i) d\xi_i + \sum_j \int_0^{y_j} b_j(\xi_j) \dot{d}_j(\xi_j) d\xi_j - \sum_{i,j} c_{ij} d_i(x_i) d_j(y_j) + \frac{1}{2} \sum_{i,j} c_{ij}^2 \\ \dot{V}(\mathbf{x}, \mathbf{y}) &= \sum_i \frac{\partial V}{\partial x_i} \frac{\partial x_i}{\partial t} + \sum_j \frac{\partial V}{\partial y_j} \frac{\partial y_j}{\partial t} + \sum_{i,j} \frac{\partial V}{\partial c_{ij}} \frac{\partial c_{ij}}{\partial t} \\ &= \sum_i b_i(x_i) \dot{d}_i(x_i) \dot{x}_i + \sum_j b_j(y_j) \dot{d}_j(y_j) \dot{y}_j - \sum_i \dot{d}_i(x_i) \dot{x}_i \sum_j c_{ij} d_j(y_j) \\ &\quad - \sum_j \dot{d}_j(y_j) \dot{y}_j \sum_i c_{ji} d_i(x_i) - \sum_{i,j} \dot{c}_{ij} d_i(x_i) d_j(y_j) + \sum_{i,j} c_{ij} \dot{c}_{ij} \end{aligned}$$

where the symmetry property $c_{ij} = c_{ji}$ is used in the fourth term on the RHS. Rearranging the terms, we get

$$\begin{aligned} \dot{V}(\mathbf{x}, \mathbf{y}) &= \sum_i \dot{d}_i(x_i) \left[b_i(x_i) - \sum_j c_{ij} d_j(y_j) \right] \dot{x}_i + \sum_j \dot{d}_j(y_j) \left[b_j(y_j) - \sum_i c_{ji} d_i(x_i) \right] \dot{y}_j \\ &\quad - \sum_{i,j} \dot{c}_{ij} \left[-c_{ij} + d_i(x_i) d_j(y_j) \right] \end{aligned}$$

Substituting the expressions for $\dot{x}_i, \dot{y}_j$ and $\dot{c}_{ij}$ in above equation.

$$\Rightarrow \dot{V}(\mathbf{x}, \mathbf{y}) = -\sum_i \dot{d}_i(x_i) \left[b_i(x_i) - \sum_j c_{ij} d_j(y_j) \right]^2 a_i(x_i) - \sum_j \dot{d}_j(y_j) \left[b_j(y_j) - \sum_i c_{ji} d_i(x_i) \right]^2 a_j(y_j) - \sum_{i,j} (\dot{c}_{ij})^2$$

In the final expression of $\dot{V}(\mathbf{x}, \mathbf{y})$, if $a_i(x_i) > 0, d_j(y_j) > 0,$ and $\dot{d}_i(x_i) > 0$, then $\dot{V}(\mathbf{x}, \mathbf{y}) \leq 0$.

Solution to problem 11:

Consider a stochastic unit with bipolar output function.

Let the actual output be s_i. Then, the average value (or Expectation $E(\cdot)$)of output is given by,

$$\begin{aligned} <s_i> &= E(s_i) = \sum s_i P(s_i) \\ <s_i> &= 1 \times P(s_i = 1|x_i) + (-1) \times P(s_i = -1|x_i) \end{aligned}$$

Since $P(s_i = 1|x_i) + P(s_i = -1|x_i) = 1$

$$\begin{aligned} P(s_i = -1|x_i) &= 1 - P(s_i = 1|x_i) \\ &= 1 - \frac{1}{1 + e^{-2\lambda x_i}} \\ \therefore \ <s_i> &= \frac{1}{1 + e^{-2\lambda x_i}} - 1 + \frac{1}{1 + e^{-2\lambda x_i}} = \frac{1 - e^{-2\lambda x_i}}{1 + e^{-2\lambda x_i}} \\ \therefore \ <s_i> &= tanh(\lambda x_i) \end{aligned}$$

If the learning of the stochastic unit is based on gradient descent on the error between desired (d_i) and average ($<s_i>$) outputs, then

$$\begin{aligned} \triangle w_{ij} &= -\eta \frac{\partial e(m)}{\partial w_{ij}} \\ \text{where } e(m) &= \frac{1}{2} \left[d_i - <s_i>\right]^2 \\ \therefore \ \frac{\partial e(m)}{\partial w_{ij}} &= -[d_i - <s_i>] \ \frac{\partial \left[<s_i>\right]}{\partial w_{ij}} \\ \text{hence, } \triangle w_{ij} &= \eta \left[d_i - <s_i>\right] \ \frac{\partial <s_i>}{\partial w_{ij}} \\ \triangle w_{ij} &= \eta \left[d_i - tanh(\lambda x_i)\right] \ \frac{\partial (tanh(\lambda x_i))}{\partial w_{ij}} \\ &= \eta \left[d_i - f(\lambda x_i)\right] \dot{f}(\lambda x_i) \lambda a_j, \text{ since } x_i = \sum_i w_{ij} a_j \end{aligned}$$

The above result can be interpreted as learning law obtained using delta-learning for a deterministic unit with hyperbolic tangent as the output function, i.e., $f(\lambda x_i) = tanh(\lambda x_i)$.

Solution to problem 12:

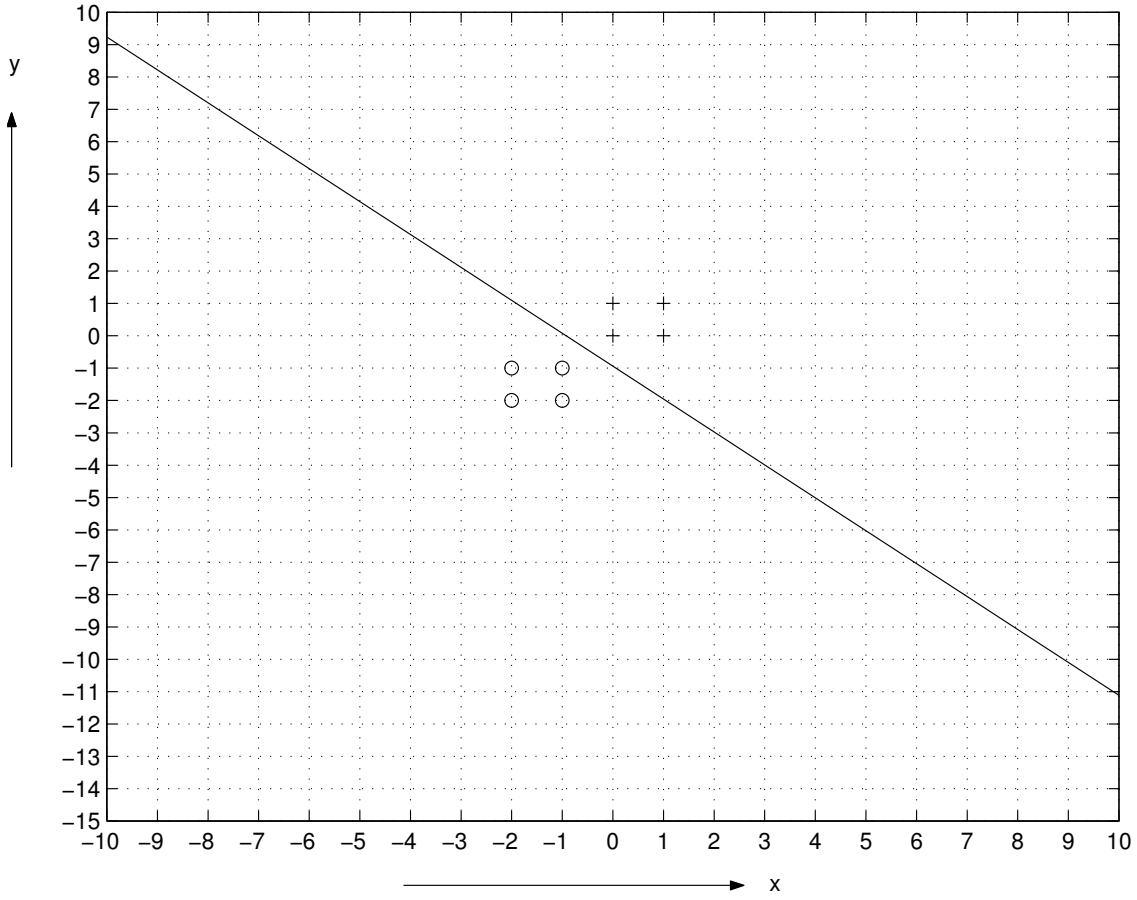
A program is written to implement reinforcement learning.

Choose the initial weights as (0.02, 0.01) and threshold as zero.

After 1000 iterations the weights obtained are (2.4977, 2.4963), and the threshold is -2.4093.

Hence the decision surface becomes a straight line given by:

$$2.4977\ x + 2.4963\ y = \text{-}2.4093$$



The Matlab program used to implement this question is given below:

```
function[ ]=prob2_12()
EtaPlus = 0.1;
EtaMinus = 0.01;
Input1 = [0 0 -1;1 0 -1;0 1 -1;1 1 -1];
Input2 = [-1 -1 -1;-1 -2 -1;-2 -1 -1;-2 -2 -1];
Output1 = [1 1 1 1];
Output2 = [-1 -1 -1 -1];
MaxIteration = 10000;
Weights = [0.02 0.01 0];
for j = 1 : MaxIteration
```

```
   for i = 1 : 4
      x = Input1(i,:) * Weights';
      CalcOutput = sign(x);
      CalcOutputAvg = tanh(x);
      if Output1(i) == CalcOutput
         WtChange = EtaPlus * (Output1(i) - CalcOutputAvg) *...
                    (1 - CalcOutputAvg * CalcOutputAvg) * Input1(i,:);
      else
         WtChange = EtaMinus * (-Output1(i) - CalcOutputAvg) *...
                    (1 - CalcOutputAvg * CalcOutputAvg) * Input1(i,:);
      end
      Weights = Weights + WtChange;
      x = Input2(i,:) * Weights';
      CalcOutput = sign(x);
      CalcOutputAvg = tanh(x);
      if Output2(i) == CalcOutput
         WtChange = EtaPlus * (Output2(i) - CalcOutputAvg) *...
                    (1 - CalcOutputAvg * CalcOutputAvg) * Input2(i,:);
      else
         WtChange = EtaMinus * (-Output2(i) - CalcOutputAvg) *...
                    (1 - CalcOutputAvg * CalcOutputAvg) * Input2(i,:);
      end
      Weights = Weights + WtChange;
   end
end
```

Solution to problem 1(a):

Counting the number of weights for the case $\mathbf{a} \in \{0,1\}^M$

For the binary case the lower limit is $i_p > i_{p-1}$, because the case of $i_p = i_{p-1}$ results in terms which have powers of a_{i_p} like $a_{i_p}^2$, $a_{i_p}^3$, etc. But $a_{i_p}^l = a_{i_p}$ for any integer l, since $a_{i_p} = 0 \,\text{or}\, 1$. So those terms must have been considered in the lower order terms. For the second term this is equivalent to counting the terms in the upper triangular matrix of the MxM matrix excluding the terms in the diagonal.

Number of threshold values	=	1	=	${}_M\mathbf{C}_0$	=	$\binom{M}{0}$
Number of weights in the 1^{st} term	=	M	=	${}_M\mathbf{C}_1$	=	$\binom{M}{1}$
Number of weights in the 2^{nd} term	=	Number of combinations of two variables a_{i_1} and a_{i_2} without duplication or repetition	=	${}_M\mathbf{C}_2$	=	$\binom{M}{2}$
Number of weights in the 3^{rd} term	=	Number of combinations of three variables a_{i_1}, a_{i_2} and a_{i_3} without duplication or repetition	=	${}_M\mathbf{C}_3$	=	$\binom{M}{3}$

In general, the number of weights in the i^{th} term $= {}_M\mathbf{C}_i = \binom{M}{i}$

Therefore the total number of weights for a p^{th} order threshold function is

$$r = \sum_{i=0}^{p} \binom{M}{i} \quad \text{for } \mathbf{a} \in \{0,1\}^M$$

(b) Counting the number of weights for the case $\mathbf{a} \in \mathcal{R}^{\mathcal{M}}$

Let $\Phi_p(M)$ denote the number of weights in the p^{th} term. Then

$$\begin{aligned}
\Phi_p(M) &= \text{number of weights in } \left(\sum_{i_1=1}^{M} \sum_{i_2=i_1}^{M} \cdots \sum_{i_p=i_{p-1}}^{M} w_{i_1 i_2 \ldots i_p} a_{i_1} a_{i_2} \ldots a_{i_p} \right) \\
&= \text{number of weights in } \left(\sum_{i_2=i_1}^{M} \sum_{i_3=i_2}^{M} \cdots \sum_{i_p=i_{p-1}}^{M} w_{1 i_2 \ldots i_p} a_1 a_{i_2} \ldots a_{i_p} \right) \\
&\quad + \text{number of weights in } \left(\sum_{i_1=2}^{M} \sum_{i_2=i_1}^{M} \cdots \sum_{i_p=i_{p-1}}^{M} w_{i_1 i_2 \ldots i_p} a_{i_1} a_{i_2} \ldots a_{i_p} \right) \qquad (1)
\end{aligned}$$

In the second term on the RHS the summation for i_1 is for $M-1$ terms, as the first index i_1 starts from 2. Therefore

$$\Phi_p(M) = \Phi_{p-1}(M) + \Phi_p(M-1) \qquad (2)$$

Likewise,
$$\begin{aligned}\Phi_p(M-1) &= \Phi_{p-1}(M-1)+\Phi_p(M-2)\\ &\cdots \quad \cdots \quad \cdots\\ \Phi_p(M-M+2) &= \Phi_{p-1}(M-M+2)+\Phi_p(1)\end{aligned}$$

But, $\Phi_p(1) = $ number of weights in $\left(\sum_{i_1=1}^{1}\sum_{i_2=i_1}^{1}\cdots\sum_{i_p=i_{p-1}}^{1} w_{i_1 i_2 \ldots i_p} a_1 a_{i_2}\ldots a_{i_p}\right)$

$= 1.$

This is valid for all p, including p=0, which is the constant term.

Hence, $\Phi_p(1) = \Phi_{p-1}(1) = 1$

Therefore,
$$\Phi_p(M) = \sum_{i=1}^{M}\Phi_{p-1}(i) \tag{3}$$

Let p=0.

Then, $\Phi_0(M) = 1,$ is a constant term, and is valid for all M. Hence $\Phi_o(i) = 1$.

From Equation(3),

$$\begin{aligned}\Phi_1(M) &= \sum_{i=1}^{M}\Phi_0(i)\\ &= M = \binom{M}{1} = \binom{M+1-1}{1}\\ \Phi_2(M) &= \sum_{i=1}^{M}\Phi_1(i)\\ &= \sum_{i=1}^{M} i, \text{ since } \Phi_1(i) = i\\ &= \frac{M(M+1)}{2} = \binom{M+1}{2} = \binom{M+2-1}{2}\\ \Phi_3(M) &= \sum_{i=1}^{M}\Phi_2(i)\\ &= \sum_{i=1}^{M}\binom{i+1}{2}, \text{ since } \Phi_2(i) = \binom{i+1}{2}\end{aligned}$$

We will show that, $\Phi_3(M) = \binom{M+2}{3} = \binom{M+3-1}{3}$

To prove this, and in general to prove the following for all p,

$$\Phi_p(M) = \sum_{i=1}^{M}\Phi_{p-1}(i) = \binom{M+p-1}{p} \tag{4}$$

We need to show that,

$$\binom{M+p-1}{p} = \sum_{i=1}^{M}\binom{i+p-1-1}{p-1} \tag{5}$$

Let us consider the Pascal's identity,

$$\binom{M+p}{p} = \binom{M+p-1}{p} + \binom{M+p-1}{p-1} \tag{6}$$

This is equivalent to stating that the number of combinations of p variables from $(M+p)$ variables is equal to sum of the number of combinations of p variables in which a given variable is not present and the number of combinations of p variables in which a given variable is present. Applying the Pascal's identity to the first term, we get

$$\binom{M+p}{p} = \binom{M-2+p}{p} + \binom{M-1+p-1}{p-1} + \binom{M+p-1}{p-1} \tag{7}$$

Continuing this for $(M-1)$ times, we get

$$\begin{aligned}
\binom{M+p}{p} &= \binom{M-M+p}{p} + \binom{M-(M-1)+p-1}{p-1} + \binom{M-(M-2)+p-1}{p-1} \\
&\quad \cdots\cdots + \binom{M-1+p-1}{p-1} + \binom{M+p-1}{p} \\
&= \binom{p}{p} + \binom{1+p-1}{p-1} + \binom{2+p-1}{p-1} + \cdots + \binom{M+p-1}{p-1} \\
&= \sum_{i=0}^{M} \binom{i+p-1}{p-1}, \text{ since } \binom{p}{p} = \binom{p-1}{p-1} = 1
\end{aligned}$$

Replacing M by $M-1$,

$$\binom{M-1+p}{p} = \binom{M+p-1}{p} = \sum_{i=0}^{M-1} \binom{i+p-1}{p-1} = \sum_{j=1}^{M} \binom{j+p-1-1}{p-1} \tag{8}$$

which is the desired result in Equation(5).

Thus the number of weights in the i^{th} term is

$$\Phi_i(M) = \binom{M+i-1}{i} \tag{9}$$

The total number of weights in all the $(p+1)$ terms including the constant term is given by

$$r = \sum_{i=0}^{p} \Phi_i(M) = \sum_{i=0}^{p} \binom{M+i-1}{i} \tag{10}$$

Again using the Pascal's identity in Equation(6), namely,

$$\binom{M+p}{p} = \binom{M+p-1}{p} + \binom{M+p-1}{p-1} \tag{11}$$

and applying the identity again to the second term this time, we get

$$\binom{M+p}{p} = \binom{M+p-1}{p} + \binom{M+p-2}{p-1} + \binom{M-p-2}{p-2} \tag{12}$$

Continuing this expansion until $(p-p)$, we get

$$\binom{M+p}{p} = \sum_{i=0}^{p} \binom{M+i-1}{i} \tag{13}$$

Hence the summation in Equation(10) is given by

$$r = \binom{M+p}{p} \tag{14}$$

Solution to problem 2:

From the solution to Problem 1 above, we have the number of weights for a polynomial threshold function of order p, which is given by,

$$r = \sum_{i=0}^{p} \binom{M}{i}, \quad \text{for } \mathbf{a} \in \{0,1\}^M$$

For $p = M$, the number of weights (using the Binomial expansion for $(1+1)^M$) is given by

$$r = \sum_{i=0}^{M} \binom{M}{i} = (1+1)^M = 2^M.$$

Solution to problem 3:

(See Hassoun p.17 for solution) Consider a set of N points in general position in $\mathcal{R}^M$. Let $C(N, M)$ be the number of linearly separable dichotomies of the N points. Now consider a new point such that the set (X') of $(N+1)$ points is in general position. Some of the linear dichotomies of the set X can be achieved by hyperplanes which pass through the new point. Let the number of such dichotomies be D. For each of the D linear dichotomies, there will be two new dichotomies obtained by enclosing an infinitesimally small region around the new point on *either side* of the plane. Thus there will be one additional new linear dichotomy for each old one. Thus the number

$$C(N+1, M) = C(N, M) - D + 2D = C(N, M) + D$$

Here D is the number of dichotomies of X that has the dividing hyperplane drawn through the new point. But this number is $C(N, M-1)$, because constraining the hyperplane to go through a particular point makes the problem effectively $(M-1)$ dimensional. Therefore applying the equation

$$C(N+1, M) = C(N, M) + C(N, M-1)$$

recursively for N, N-1, N-2, N-3, .. , 1, we get

$$C(N,M) = \sum_{i=0}^{M-1} \binom{N-1}{i} C(1, M-i)$$

For only one point in M-dimensional space, there are two dichotomies, since the point can be on either side of the hyperplane. Hence, $C(1, M-i) = 2$.

$$\text{Therefore,} \quad C(N,M) = 2\sum_{i=0}^{M} \binom{N-1}{i}, \quad \text{for } N > M+1,$$

Since $\binom{n}{m} = 0$, for $m > n$, for $N \leq M+1$,

$$\begin{aligned} C(N,M) &= 2\sum_{i=0}^{N-1} \binom{N-1}{i} \\ &= 2 * 2^{N-1} = 2^N \end{aligned}$$

Solution to problem 4:

The number of different weights for a p^{th} order polynomial threshold function is given by

$$r = \binom{M+p}{p}$$

(See the solution for the Problem 1). This can be viewed as a linear threshold function in $\mathcal{R}^{r-1}$. Therefore the number of Boolean functions of M-variables that can be realized by an M-input p-order polynomial threshold function is equivalent to an r-input linearly separable dichotomies of N points.

Hence the number of functions are less than $C(N, r-1)$. (See the solution for Problem 3 above)

Solution to problem 5:

From Eq.(4.11)

$$\begin{aligned} S &= (W - BA^+)AA^T(W - BA^+)^T + B(I - A^+A)B^T \\ &= WAA^TW^T - WAA^T(A^+)^TB^T - BA^+AA^TW^T + BA^+AA^T(A^+)^TB^T + BB^T - BA^+AB^T \end{aligned}$$

Using the matrix identities $AA^T(A^+)^T = A$, $A^+AA^T = A^T$

$$\begin{aligned} S &= WAA^TW^T - WAB^T - BA^TW^T + BA^+AB^T + BB^T - BA^+AB^T \\ &= WAA^TW^T - WAB^T - BA^TW^T + BB^T \\ &= BB^T - WAB^T + WAA^TW^T - BA^TW^T \\ &= B(B^T - A^TW^T) - WA(B^T - A^TW^T) \end{aligned}$$

$$
\begin{aligned}
&= B(B^T - (WA)^T) - WA(B^T - (WA)^T) \\
&= B(B - WA)^T - WA(B - WA)^T \\
S &= (B - WA)(B - WA)^T
\end{aligned}
$$

Solution to problem 6:

Starting from Eq.(4.12),

$$E_{min} = \frac{1}{L} tr[B(I - A^+A)B^T] \tag{15}$$

Since $A = \sum_{i=1}^{r} \lambda_i^{\frac{1}{2}} \mathbf{p}_i \mathbf{q}_i^T$, $A^+ = \sum_{i=1}^{r} \lambda_i^{-\frac{1}{2}} \mathbf{q}_i \mathbf{p}_i^T$

where $\mathbf{p}_i^T \mathbf{p}_j = \begin{cases} 1 & , \quad i = j \\ 0 & , \quad i \neq j \end{cases}$ and $\mathbf{q}_i^T \mathbf{q}_j = \begin{cases} 1 & , \quad i = j \\ 0 & , \quad i \neq j \end{cases}$

$$
\begin{aligned}
\therefore I - A^+A &= I - \left(\sum_{i=1}^{r} \lambda_i^{-\frac{1}{2}} \mathbf{q}_i \mathbf{p}_i^T \right) \left(\sum_{j=1}^{r} \lambda_j^{\frac{1}{2}} \mathbf{p}_j \mathbf{q}_j^T \right) \\
&= I - \sum_{i=1}^{r} \mathbf{q}_i \mathbf{q}_i^T
\end{aligned}
$$

Since I is an $L \times L$ identity matrix, I can be written as $I = \sum_{i=1}^{L} \mathbf{q}_i \mathbf{q}_i^T$

$$\therefore I - A^T A = \sum_{i=1}^{L} \mathbf{q}_i \mathbf{q}_i^T - \sum_{i=1}^{r} \mathbf{q}_i \mathbf{q}_i^T = \sum_{i=r+1}^{L} \mathbf{q}_i \mathbf{q}_i^T$$

$$\therefore \text{ Equation (1)} \Rightarrow E_{min} = \frac{1}{L} tr \left(\sum_{i=r+1}^{L} B \mathbf{q}_i \mathbf{q}_i^T B^T \right)$$

$$= \frac{1}{L} \sum_{i=r+1}^{L} \|B \mathbf{q}_i\|^2$$

Solution to problem 7:

$$A = \left[\begin{array}{ccc} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{array} \right] = \left[\begin{array}{ccc} 1/6 & 1/2 & -5/6 \\ -5/6 & 1/2 & 1/6 \\ -1/6 & -1/2 & -1/6 \\ 1/2 & 1/2 & 1/2 \end{array} \right]$$

$$B = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Since A is an orthogonal matrix

$W = BA^T$ will give minimum error

$$W = BA^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1/6 & -5/6 & -1/6 & 1/2 \\ 1/2 & 1/2 & -1/2 & 1/2 \\ -5/6 & 1/6 & -1/6 & 1/2 \end{bmatrix}$$

$$\Rightarrow \quad W = \begin{bmatrix} 1/6 & -5/6 & -1/6 & 1/2 \\ 1/2 & 1/2 & -1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 & 1/2 \end{bmatrix}$$

Solution to problem 8:

A program is written to design a classifier for the 2-class problem using perceptron learning law.

$C_1 = [\ -2 \quad 2\ ]^T,\ [\ -2 \quad 1.5\ ]^T,\ [\ -2 \quad 0\ ]^T,\ [\ 1 \quad 0\ ]^T,\ [\ 3 \quad 0\ ]^T$

$C_2 = [\ 1 \quad 3\ ]^T,\ [\ 3 \quad 3\ ]^T,\ [\ 1 \quad 2\ ]^T,\ [\ 3 \quad 2\ ]^T,\ [\ 10 \quad 0\ ]^T$

Choose the initial weights $w_1 = -0.1106$, $w_2 = 0.2309$ and bias $\theta = 0.5839$.

Final weights $w_1 = -3.1106$, $w_2 = -4.7691$ and bias $\theta = 11.5839$.

The Matlab program used to implement this question is given below:

```
function[ ] = prob4_8()
Inp1 = [ -2 -2 -2  1 3  1 3 1 3 10; 2 1.5 0 0 0 3 3 2 2 0];
DesOutp = [1 1 1 1 1 0 0 0 0 0];
% Next 4 lines is to get the range of input values
min1 = min(Inp1(1,:));
max1 = max(Inp1(1,:));
min2 = min(Inp1(2,:));
max2 = max(Inp1(2,:));
net = newp([min1 max1;min2 max2],1);
net.iw{1} = [-0.1106 0.2309];
```

```
net.b{1}  = 0.5839;
net = train(net,Inp1,DesOutp);
% Uncomment following line to test the network
% ObtOutput = sim(net,Inp1);
```

Solution to problem 9:

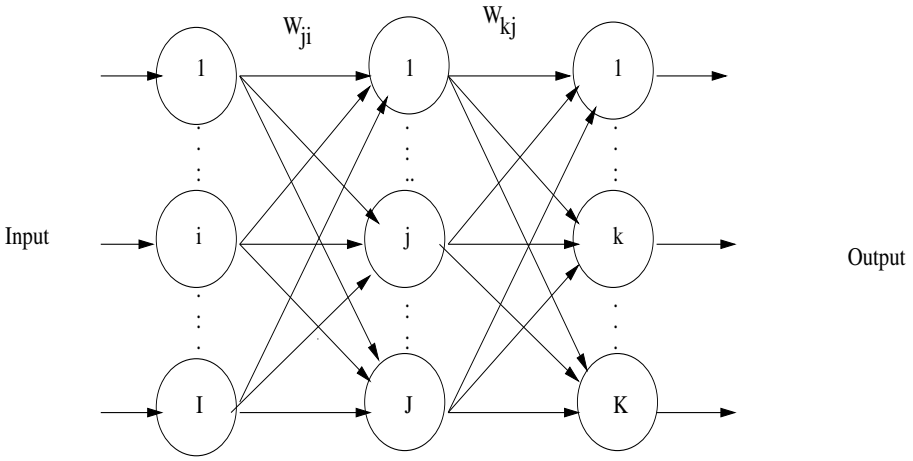
Let I: Number of units in input layer

J: Number of units in the hidden layer

K: Number of units in the output layer

A program is written to implement MLFFNN. The initial and final weight vectors and bias values are shown for J=8, i.e., Number of units in the hidden layer = 8.

The network structure is shown in the figure below.



Network Structure

The Matlab commands used to implement this problem are:

- *newff* - To create a feed-forward backpropagation network as per the structure passed as a parameter.
- *train*- Trains a network for the given inputs and desired outputs.
- *net.iw*{*1*} - To view the weights between input layer and hidden layer. Here *net* is the network object
- *net.lw*{*LayerNo*} - To view the weights between the $LayerNo$ and $LayerNo + 1$

- *net.b{LayerNo}* - To view the bias value of the nodes in the given *LayerNo*

(a) Parity Problem:

Given I=4 & K=1. Let J=8. Using the Matlab command *newff([zeros(4,1) ones(4,1)],[8 1],{'logsig','logsig'})* the feed-forward network has been created.

The initial weight vectors and bias generated are:

$$[w_{ji}] = \begin{bmatrix} 2.6366 & -7.3643 & -6.1926 & -7.0221 & 1.6455 & -6.7024 & -2.4650 & 2.3807 \\ 5.2867 & 2.9301 & -5.3255 & -1.0342 & -1.1013 & -2.2232 & -6.4476 & 3.1811 \\ 4.8393 & 3.4600 & -0.2768 & 1.2426 & -7.1741 & -0.7460 & 4.0184 & 7.1148 \\ -5.5118 & -3.7295 & 4.6812 & -6.0643 & 5.7716 & 6.1870 & -4.9904 & -4.7214 \end{bmatrix}^T$$

$$[\theta_j] = \begin{bmatrix} -8.3345 & 5.7154 & 5.5750 & 7.1117 & 1.1018 & -0.2758 & 1.5788 & 0.7314 \end{bmatrix}^T$$

$$[w_{kj}] = \begin{bmatrix} -2.4339 & -2.2792 & 0.7612 & -1.6710 & 1.8589 & -1.7628 & -1.7796 & 2.6720 \end{bmatrix}$$

$$[\theta_k] = \begin{bmatrix} 2.3172 \end{bmatrix}$$

Using the Matlab command *train(net,InputData,DesiredOutput);*, the network has been trained.

Final weight vectors and bias are:

$$[w_{ji}] = \begin{bmatrix} 9.0484 & 9.1108 & 7.5750 & -12.4670 \\ -6.8453 & 7.2264 & 6.6316 & -11.1612 \\ -4.8452 & -4.8318 & -4.1937 & 7.4128 \\ -10.2269 & -6.0397 & 9.2958 & -7.5102 \\ 1.0832 & -0.5717 & -8.6589 & 4.2438 \\ -7.8428 & -11.5315 & -7.5355 & 11.8005 \\ -9.0840 & -9.1161 & 9.3922 & -11.6694 \\ 0.0043 & 1.3187 & 7.8108 & -6.0530 \end{bmatrix}$$

$$[\theta_j] = \begin{bmatrix} -11.5261 & 5.0699 & 5.9604 & 11.9211 & 1.0940 & 2.3545 & 5.2659 & -2.4693 \end{bmatrix}^T$$

$$[w_{kj}] = \begin{bmatrix} -30.5388 & -11.3753 & 5.2873 & 36.0788 & 1.2476 & -36.1048 & -30.0336 & 7.4365 \end{bmatrix}$$

$$[\theta_k] = \begin{bmatrix} 11.7801 \end{bmatrix}$$

The Matlab program used to implement this parity problem is given below:

```
function[]=prob4_9a()
%Next set of loop is to generate the input and output data
% according to the given condition
index=0;
for i=0:1
for j=0:1
for k=0:1
for l=0:1
    index=index+1;
    p(index,:)=[i j k l];
    if mod(i+j+k+l,2)==0
       t(index,1)=0;
    else
       t(index,1)=1;
    end
end
end
end
end
net=newff([zeros(4,1) ones(4,1)],[8 1],{'logsig' 'logsig'});
'Inital Weights'
net.iw{1}
net.b{1}
net.lw{2}
net.b{2,1}
net=train(net,p',t');
'Final Weights'
net.iw{1}
net.b{1}
net.lw{2}
net.b{2}
```

```
obtoutput=sim(net,p');
[obtoutput' t]
```

(b) Encoding Problem:

Given I=8 & K=8. Let J=8. Using the Matlab command *newff([zeros(8,1) ones(8,1)],[8 8],{'logsig', 'logsig'})* the feed-forward network has been created.

The initial weight vectors and bias generated are:

$$[w_{ji}] = \begin{bmatrix} 0.0507 & -1.7943 & -4.0682 & -0.2235 & 1.8284 & 2.9493 & -2.3311 & 3.9298 \\ -0.4412 & -1.3263 & 2.0552 & 5.0464 & 0.8668 & 3.2507 & 2.5517 & -1.8076 \\ 0.4425 & 0.0322 & 2.3940 & -0.5229 & -0.4188 & 4.8055 & 0.2357 & -4.8183 \\ 4.0020 & 2.0084 & -1.9757 & 2.7552 & -0.4948 & 0.5242 & 3.9167 & 2.3102 \\ -1.2578 & -1.5403 & -1.8932 & 2.6140 & -3.2768 & -3.8611 & 1.6958 & 3.1382 \\ -1.0666 & -4.2126 & 2.2644 & -3.6249 & -0.6139 & 1.0447 & -2.9540 & -2.3228 \\ -2.4911 & -0.7346 & 3.6769 & -1.3743 & -1.7319 & 4.0959 & -0.6523 & -3.2228 \\ -0.7149 & -0.2687 & 3.9301 & 0.1678 & -1.5963 & 3.8568 & -2.6500 & 3.4983 \end{bmatrix}$$

$$[\theta_j] = \begin{bmatrix} -3.8017 & -2.5042 & -2.6312 & -7.0418 & 1.6719 & 4.1867 & -1.3767 & -6.7427 \end{bmatrix}^T$$

$$[w_{kj}] = \begin{bmatrix} -3.2634 & 0.5487 & 0.5899 & 0.8659 & -3.8634 & 2.3572 & -4.2681 & 1.4118 \\ 3.4959 & -1.4330 & -2.9983 & 3.5619 & -1.8698 & 1.3174 & 2.5329 & -2.2676 \\ 2.0480 & -1.2693 & 0.2243 & -2.1429 & 5.0683 & -3.7445 & 1.2291 & 1.0361 \\ 2.7964 & 2.9582 & -3.0800 & 3.1274 & -2.3165 & -3.2541 & 0.4844 & 0.8440 \\ -2.6013 & -1.1437 & 2.4151 & -3.5649 & -0.0142 & -4.3408 & -2.2903 & 1.4272 \\ -0.3952 & -3.7516 & -1.0995 & -3.8253 & -1.8437 & -1.8639 & 2.8337 & -2.7865 \\ -3.7863 & -2.7129 & 2.9228 & -2.4014 & 1.5614 & 2.8626 & -2.1401 & 1.2341 \\ 2.2046 & -2.8286 & -2.8466 & 2.7193 & 2.8756 & 3.0482 & 1.5925 & -2.0700 \end{bmatrix}$$

$$[\theta_k] = \begin{bmatrix} 6.4418 & -3.7634 & -2.7808 & -1.2986 & 4.5377 & 4.8098 & -1.3638 & 1.2836 \end{bmatrix}^T$$

Using the Matlab command *train(net,InputData,DesiredOutput)*, the network has been trained. Final weight vectors and bias obtained as:

$$[w_{ji}] = \begin{bmatrix} 10.1223 & -8.4986 & -14.2486 & -7.7608 & 9.6174 & 13.7030 & -16.5067 & 12.723 \\ 5.8326 & -2.0376 & 2.1571 & 21.1133 & 8.2204 & 8.1280 & -4.3462 & -5.663 \\ -0.0613 & -5.0624 & -14.7595 & -4.9729 & -8.1639 & 8.7469 & -2.0848 & -1.779 \\ 4.8438 & 12.3737 & 8.7852 & 7.8906 & 4.8332 & 4.7385 & 0.7131 & 4.369 \\ -8.2452 & -6.9411 & -11.4763 & 21.6004 & -7.2109 & -15.1490 & 12.5705 & -8.081 \\ -4.6513 & -7.9700 & -5.1002 & -6.1559 & -0.9193 & -4.4831 & 3.5827 & -6.353 \\ -5.2098 & -6.7294 & -9.2239 & -6.9853 & -4.3285 & -4.1775 & -3.1231 & -10.500 \\ 13.8486 & 8.2652 & 10.7165 & 10.3678 & 8.1866 & 12.9745 & -39.1789 & 11.971 \end{bmatrix}$$

$$[\theta_j] = \begin{bmatrix} -7.6522 & -20.1691 & 19.6221 & -39.9443 & 10.0033 & 22.3619 & 41.2483 & -30.9464 \end{bmatrix}^T$$

$$[w_{kj}] = \begin{bmatrix} 27.6968 & -15.3895 & -11.7509 & -9.7931 & 39.8741 & -26.6494 & 35.9503 & -13.8570 \\ 35.3433 & -0.8484 & -7.5553 & 56.0481 & -9.3960 & -9.2411 & -11.7716 & -45.3909 \\ 34.4781 & 9.4061 & -63.8607 & 38.6314 & -20.8057 & 2.1879 & -28.6527 & -68.7798 \\ 34.9628 & 51.3528 & -21.2929 & 20.1797 & -15.6599 & -11.1555 & -17.2597 & -48.8204 \\ 75.2662 & 13.1115 & -62.4540 & 23.8475 & -0.3578 & -21.0610 & -3.3486 & -74.1051 \\ 35.4224 & 15.5662 & -19.2480 & 21.6163 & -55.5779 & -15.5047 & -17.5111 & -49.4509 \\ 6.1147 & 15.8459 & -3.7786 & 13.0918 & -20.4837 & -6.2984 & -27.0099 & -63.1195 \\ 36.0052 & 7.0209 & -4.9993 & 15.3743 & -7.4040 & -0.8992 & -54.6761 & -43.2427 \end{bmatrix}$$

$$[\theta_k] = \begin{bmatrix} -80.1136 & -27.2003 & 3.9875 & -39.8990 & -19.7775 & -5.0562 & 10.1603 & 3.3472 \end{bmatrix}^T$$

The Matlab program used to implement this encoding problem is given below:

```
function[]=prob4_9b()
%Next set of loop is to generate the input and output data
%       according to the given condition
index=0;
for i1=0:1
for i2=0:1
for i3=0:1
for i4=0:1
for i5=0:1
for i6=0:1
for i7=0:1
```

```
for i8=0:1
    index=index+1;
    p(index,[1:8])=[i1 i2 i3 i4 i5 i6 i7 i8];
    if i1+i2+i3+i4+i5+i6+i7+i8 == 7
       t(index,[1:8])=[i1 i2 i3 i4 i5 i6 i7 i8];
    else
       t(index,[1:8])=[0 0 0 0 0 0 0 0];
    end
end
end
end
end
end
end
end
end
end

net=newff([zeros(8,1) ones(8,1)],[8 8],{'logsig' 'logsig'});
'Inital Weights'
net.iw{1,1}
net.b{1,1}
net.lw{2,1}
net.b{2,1}
net=train(net,p',t');
'Final Weights'
net.iw{1,1}
net.b{1,1}
net.lw{2,1}
net.b{2,1}

obtoutput=sim(net,p');
[obtoutput' t]
```

(c) Symmetry Problem:

Given I=4 & K=1. Let J=8. Using the Matlab command *newff([zeros(4,1) ones(4,1)],[8 1],{'logsig', 'logsig'})* the feed-forward network has been created.

The initial weight vectors and bias generated are:

$$[w_{ji}] = \begin{bmatrix} 4.1588 & -6.0130 & 1.4310 & 5.7619 \\ 5.6198 & -3.5419 & 2.9982 & 5.9652 \\ 3.9112 & 7.4225 & 2.3064 & -3.6040 \\ -0.9093 & 7.9989 & 4.4172 & 2.0923 \\ -6.7523 & -2.7001 & 0.1642 & -5.9824 \\ 5.0773 & 6.0270 & 3.3872 & -3.8888 \\ -6.6910 & -0.4427 & 0.3315 & 6.6049 \\ -2.3979 & 2.6687 & 4.6455 & 7.3653 \end{bmatrix}$$

$$[\theta_j] = [-7.3784 \quad -8.8842 \quad -7.0362 \quad -6.1269 \quad 6.9625 \quad -3.2832 \quad -3.2649 \quad -10.8498]^T$$

$$[w_{kj}] = [-2.9075 \quad 0.3681 \quad -0.2716 \quad 2.4207 \quad -1.3022 \quad -2.6002 \quad -0.1399 \quad 2.8916]$$

$$[\theta_k] = [0.7705]$$

Using the Matlab command *train(net,InputData,DesiredOutput)*, the network has been trained.

Final weight vectors and bias obtained as:

$$[w_{ji}] = \begin{bmatrix} 2.8755 & -6.9475 & 1.6345 & 7.3199 \\ 7.2787 & -3.1214 & 2.6843 & 7.0560 \\ 3.5924 & 5.8214 & 6.3810 & -5.4891 \\ -7.5421 & 8.3914 & 6.0910 & 4.7326 \\ -6.0804 & 2.2781 & 1.0564 & -4.2814 \\ 8.1040 & 11.2027 & 10.2467 & -5.2202 \\ -9.0412 & -1.9118 & 1.9592 & 11.4373 \\ -7.4228 & 5.2733 & 8.1011 & 5.0943 \end{bmatrix}$$

$$[\theta_j] = [-4.7503 \quad -11.6566 \quad -8.9390 \quad -10.5504 \quad 14.0611 \quad -4.5828 \quad -6.3212 \quad -9.7566]^T$$

$$[w_{kj}] = [-3.6496 \quad 8.8979 \quad 2.4695 \quad 16.5842 \quad 7.3778 \quad -31.1276 \quad -27.8023 \quad 10.6797]$$

$$[\theta_k] = [7.9283]$$

The Matlab program used to implement this symmetry problem is given below:

```
function[]=prob4_9c()
%Next set of loop is to generate the input and output data
%       according to the given condition
index=0;
for i1=0:1
for i2=0:1
for i3=0:1
for i4=0:1
    index=index+1;
    p(index,[1:4])=[i1 i2 i3 i4];
    if (i2==i3) & (i1==i4)
       t(index,1)=1;
    else
       t(index,1)=0;
    end
end
end
end
end

net=newff([zeros(4,1) ones(4,1)],[8 1],{'logsig' 'logsig'});
'Inital Weights'
net.iw{1,1}
net.b{1,1}
net.lw{2,1}
net.b{2,1}
whos
net=train(net,p',t');
'Final Weights'
net.iw{1,1}
net.b{1,1}
```

```
net.lw{2,1}

net.b{2,1}


obtoutput=sim(net,p');

[obtoutput' t]
```

(d) Addition Problem:

Given I=4 & K=3. Let J=8. Using the Matlab command *newff([zeros(4,1) ones(4,1)],[8 3],{'logsig', 'logsig'})* the feed-forward network has been created.

The initial weight vectors and bias generated are:

$$[w_{ji}] = \begin{bmatrix} -8.3389 & 2.7583 & 0.6842 & -3.3295 \\ 4.8730 & -4.3625 & 1.8826 & 6.5098 \\ 3.7696 & 3.1777 & 5.7846 & 5.5615 \\ 1.1897 & 2.0730 & 3.5034 & -8.4091 \\ -3.7319 & 2.3255 & 5.4940 & -6.2594 \\ 3.6792 & -3.6178 & -5.5656 & -5.5766 \\ -5.1833 & 2.9890 & -4.1503 & -5.9727 \\ 4.7663 & -6.1955 & 5.2444 & 0.3054 \end{bmatrix}$$

$$[\theta_j] = \begin{bmatrix} 8.8220 & -7.8150 & -11.1649 & 0.1488 & 0.4132 & 7.5586 & 2.7951 & 2.6487 \end{bmatrix}^T$$

$$[w_{kj}] = \begin{bmatrix} -0.3955 & 0.0871 & -1.1392 & -1.7606 & 0.0272 & 3.9510 & 3.8630 & 2.4854 \\ 1.4514 & -3.0406 & -1.5968 & 0.9870 & 3.1977 & -2.7636 & -2.4543 & 1.5842 \\ 0.5993 & -2.2288 & 1.9681 & 3.5352 & 2.3833 & 2.2678 & -2.7455 & 1.1017 \end{bmatrix}$$

$$[\theta_k] = \begin{bmatrix} -0.3470 & 1.3175 & -0.2284 \end{bmatrix}^T$$

Using the Matlab command *train(net,InputData,DesiredOutput)*, the network has been trained.

Final Weight vectors and bias are:

$$[w_{ji}] = \begin{bmatrix} -16.8892 & 11.1326 & -9.5247 & 10.2352 \\ 22.3682 & 10.1930 & 2.6427 & 10.2755 \\ 6.8267 & -7.2200 & 14.1068 & -7.1359 \\ -1.5267 & -0.8954 & -0.3866 & -11.1632 \\ -20.5533 & 12.1179 & 23.5114 & 12.7483 \\ 7.4285 & 0.3278 & -1.8379 & 0.8716 \\ -12.4729 & 11.7390 & -11.7580 & 11.5802 \\ -2.1727 & -16.0076 & 2.3349 & -15.9719 \end{bmatrix}$$

$$[\theta_j] = \begin{bmatrix} 10.2724 & -18.5465 & -6.2919 & -7.5222 & 2.9071 & 14.8274 & -19.7947 & 6.5336 \end{bmatrix}^T$$

$$[w_{kj}] = \begin{bmatrix} -9.9711 & 38.4616 & 3.8052 & -4.9673 & 36.2453 & -22.2780 & -28.2698 & -10.9251 \\ 51.3615 & 0.9396 & 53.1133 & -1.8173 & -57.8594 & -9.4806 & 34.8494 & -3.2516 \\ -13.4066 & -60.8498 & 49.6446 & -2.7591 & -59.1352 & 44.7703 & -17.5658 & -87.4505 \end{bmatrix}$$

$$[\theta_k] = \begin{bmatrix} -24.7889 & -0.9618 & 43.2366 \end{bmatrix}^T$$

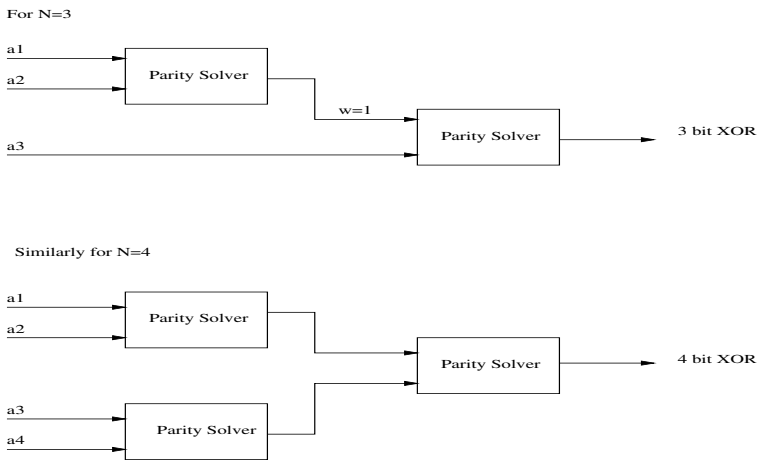
The Matlab program used to implement this addition problem is given below:

```
function[]=prob4_9d()
%Next set of lines is to generate the input and output data
%       according to the given condition
p=[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1; ...
   0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1; 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1];
t=[0 0 0 0 0 0 0 1 0 0 1 1 0 1 1 1; 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 1; ...
   0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0];

net=newff([zeros(4,1) ones(4,1)],[8 3],{'logsig' 'logsig'});
'Initla Weights'
net.iw{1,1}
net.b{1,1}
net.lw{2,1}
net.b{2,1}
net=train(net,p,t);
'Final Weights'
net.iw{1,1}
```

```
net.b{1,1}
net.lw{2,1}
net.b{2,1}

obtoutput=sim(net,p);
[obtoutput' t']
```

Solution to problem 10:

The XOR problem for ($N > 2$) variables can be looked at as a repeated parity operation on pairs of inputs. Thus if we want to solve for $N = 3$, we have to consider the parity problem for $N = 2$, then this output and another input have to be used for another parity solver.



3-bit XOR and 4-bit XOR

To obtain a 2-bit parity solver for the above solution, we need a 3 layer FFNN as the problem is not linearly separable. The problem is solved using a 3-layer FFNN with 3 hidden nodes.

The initial weights and bias values are:

$$
\begin{aligned}
[w_{ji}] &= \begin{bmatrix} 6.5963 & 7.1111 \\ 9.6182 & -1.2531 \\ 9.6994 & -0.0334 \end{bmatrix} \\
[\theta_j] &= [-11.7035 \quad -4.1826 \quad 0.0167]^T \\
[w_{kj}] &= [-4.3629 \quad 2.1887 \quad -2.7450]^T \\
[\theta_k] &= [2.4596]
\end{aligned}
$$

The final weights and bias values are:

$$
\begin{aligned}
[w_{ji}] &= \begin{bmatrix} 7.6604 & 7.4399 \\ 10.6659 & -5.9485 \\ 9.8579 & -11.0767 \end{bmatrix} \\
[\theta_j] &= [-13.4770 \quad -5.9044 \quad 5.5014]^T \\
[w_{kj}] &= [-11.1443 \quad 26.6857 \quad -25.4300]^T \\
[\theta_k] &= [12.3703]
\end{aligned}
$$

The Matlab program used to implement this question is given below:

```
function[ ] = prob4_10()
% Input and Output used for training
Input1 = [0 0;0 1;1 0;1 1];
Output1 = [0;1;1;0];
% Creating the network
net = newff([zeros(2,1) ones(2,1)],[3 1],{'logsig' 'logsig'});
% Following codes are to display initial weights
'Inital Weights'
net.iw{1}
net.b{1}
net.lw{2}
net.b{2}
% Training the network
net=train(net,Input1',Output1');
% Following codes are to display final weights (i.e. after training)
net.iw{1}
net.b{1}
net.lw{2}
net.b{2}
TestInput = [ 0     0     0     0; 0     0     0     1;...
              0     0     1     0; 0     0     1     1;...
              0     1     0     0; 0     1     0     1;...
              0     1     1     0; 0     1     1     1;...
```

```
              1     0     0     0; 1     0     0     1;...
              1     0     1     0; 1     0     1     1;...
              1     1     0     0; 1     1     0     1;...
              1     1     1     0; 1     1     1     1 ];
% output1 and output2 contains the output of intermediate XOR's
% TestOutput contains overall output of given 4 input bits
for i = 1 : 16
      output1(i,1) = sim(net,TestInput(i,[1:2])');
      output2(i,1) = sim(net,TestInput(i,[3:4])');
      TestOutput(i,1) = sim(net,[output1(i,1) output2(i,1)]');
end
```

Solution to problem 11:

Class C_1 : $[-2\ 2]^T$, $[-2\ 3]^T$, $[-1\ 1]^T$, $[-1\ 4]^T$, $[0\ 0]^T$, $[0\ 1]^T$, $[0\ 2]^T$, $[0\ 3]^T$ and $[1\ 1]^T$

Class C_2 : $[1\ 0]^T$, $[2\ 1]^T$, $[3\ -1]^T$, $[3\ 1]^T$, $[3\ 2]^T$, $[4\ -2]^T$, $[4\ 1]^T$, $[5\ -1]^T$ and $[5\ 0]^T$

Perceptron Learning:

A program is written to implement the perception learning law. The final weight vector and bias are found to be

$$\begin{aligned} \mathbf{w} &= [-20 \quad 15]^T \\ \text{bias} \quad \theta &= [7] \end{aligned}$$

Matlab program used to solve this problem is

```
function[ ]= prob4_11a()
Inp1=[ -2 -2 -1 -1 0 0 0 0 1 1 2  3 3 3  4 4  5 5; ...
        2  3  1  4 0 1 2 3 1 0 1 -1 1 2 -2 1 -1 0];
DesOutput=[1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 ];
min1=min(Inp1(1,:)); max1=max(Inp1(1,:));
min2=min(Inp1(2,:)); max2=max(Inp1(2,:));
net=newp([min1 max1;min2 max2],1);
net.trainParam.epochs=5000;
net=train(net,Inp1,DesOutput);
```

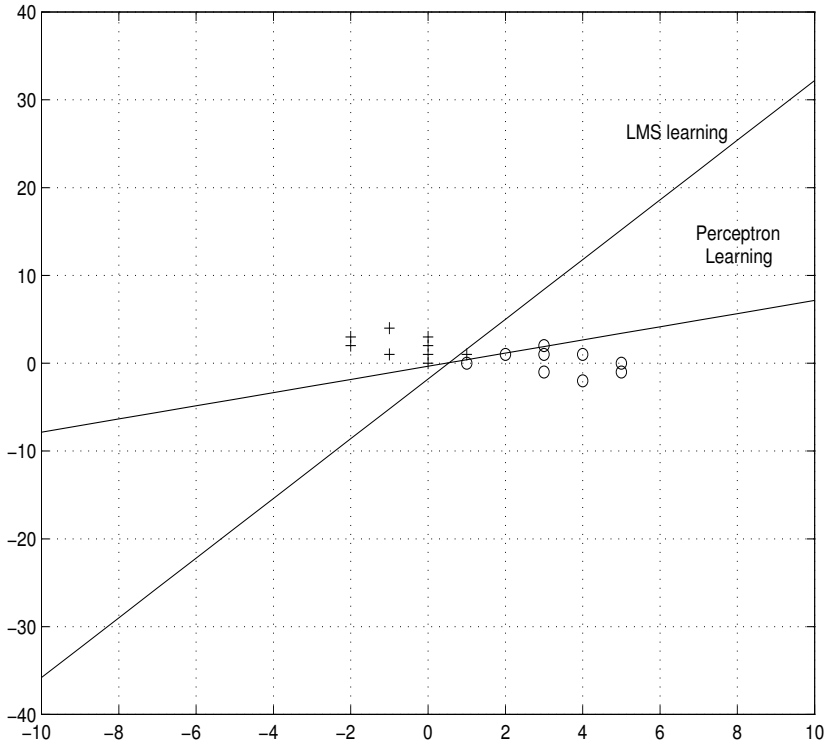
```
% Uncomment next line to test the network
% ObtOutput=sim(net,Inp1);
```

LMS Learning:

A program is written to implement LMS learning law. The final weight vector and bias are given by

$$\begin{aligned} \mathbf{w} &= [-0.05 \quad 0.17]^T \\ \text{bias} \quad \theta &= [0.09] \end{aligned}$$

The decision boundaries obtained by LMS and perceptron learning law is shown in following figure:



Matlab program used to solve the problem(4.11b) is

```
function[ ] = prob4_11b()
Inp1=[ -2 -2 -1 -1 0 0 0 0 1 1 2  3 3 3  4 4  5 5; ...
        2  3  1  4 0 1 2 3 1 0 1 -1 1 2 -1 1 -1 0];
DesOutput=[1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 ];
min1=min(Inp1(1,:)); max1=max(Inp1(1,:));
```

```
min2=min(Inp1(2,:)); max2=max(Inp1(2,:));
net=newlin([min1 max1;min2 max2],1);
net.iw{1}=[0 0];
net.b{1}=0;
net=adapt(net,Inp1,DesOutput);
% Uncomment next line to test the network
% ObtOutput=sim(net,Inp1);
```

Solution to problem 12:

The input data was generated as per the given constraints. Mean square error of the input set for different values of hidden units, learning rate parameter and momentum term are given in following tables.

(a) For momentum term, $\mu = 0.1$

$\eta \rightarrow$ no. of hidden units $\downarrow$	0.01	0.1	0.9
2	0.0248	0.0250	0.0250
4	0.0240	0.0242	0.0249
6	0.0242	0.0246	0.0242

(b) For momentum term, $\mu = 0.5$

$\eta \rightarrow$ no. of hidden units $\downarrow$	0.01	0.1	0.9
2	0.0249	0.0250	0.0249
4	0.0248	0.0249	0.0248
6	0.0247	0.0239	0.0245

Conclusion:

The networks with

$\mu = 0.1$, Number of hidden units = 4, $\eta = 0.01$ and

$\mu = 0.5$, Number of hidden units = 6, $\eta = 0.1$

work better in the sense that each one of them gives the lowest error for a given momentum value.

The Matlab program used to implement this problem is:

```
function [] = prob4_12()
load InputTrain.mat %Input samples for training
load DesOutputTrain.mat %Desired outputs for training
load InputTest.mat %Test input samples
load DesOutputTest.mat %Desired outputs for test data
%Prepare the Input1 in 2xR format
max1=max(InputTrain(1,:));
min1=min(InputTrain(1,:));
max2=max(InputTrain(2,:));
min2=min(InputTrain(2,:));
HiddenUnits=[2 4 6];
Eta=[.01 .1 .9];
Momentum=[.1 .5];
for h=1:3
    for e=1:3
        for m=1:2
            net=newff([min1 max1;min2 max2],[HiddenUnits(h) 1], ...
                     {'logsig','logsig'},'trainlm','learngdm');
            net.trainParam.lr=Eta(e);
            net.trainParam.mc=Momentum(m);
            net.trainParam.epochs=100;
            net.trainParam.goal=.0001;
            net=train(net,InputTrain,DesOutputTrain);
            Y=sim(net,InputTest);
            error(h,e,m)=norm(Y-DesOutputTest);
        end
    end
end
```

Solution to problem 13:

The data used in Problem 12 is used here to find the classification performance of the Bayesian decision. Results of Bayesian decision and the best network in Problem 12 are given in following

table for different experiments consisting of different sets of data. This indicates that the neural network gives performance close to that of Bayesian classification only in a few cases.

Experiment	Success percentage of	
	Bayesian classification	Neural network
1	61	54
2	68	58
3	59	56
4	63	61
5	61	57
6	72	59
7	64	64
8	65	61
9	62	61
10	64	57

CHAPTER 5

Solution to problem 1:

Using Eq.(4.19)

$$E(W) = \frac{1}{L}\Big[\sum_{i=s+1}^{L} \|A\mathbf{q}_i\|^2 + L\sigma^2 \sum_{i=1}^{s} \lambda_i^{-1} \|A\mathbf{q}_i\|^2\Big]$$

To minimize the error $E(W)$ in the presence of noise in the input vectors, choose $W = A\hat{A}^+$,

$$\begin{aligned} \text{where} \qquad \hat{A}^+ &= \sum_{i=1}^{s} \lambda_i^{-\frac{1}{2}} \mathbf{q}_i \mathbf{p}_i^T \\ \text{and} \qquad A &= \sum_{i=1}^{r} \lambda_i^{\frac{1}{2}} \mathbf{p}_i \mathbf{q}_i^T \end{aligned}$$

In the expression of $\hat{A}^+$, 's' is given by

$$\frac{L\sigma^2}{\lambda_s} < 1 < \frac{L\sigma^2}{\lambda_{s+1}}$$

The resulting error for the optimal choice of $W = A\hat{A}^+$ is

$$\begin{aligned} E_{min} &= \frac{1}{L}\Bigg\{\sum_{i=s+1}^{L}\Bigg[\left(\sum_{j=1}^{r} \lambda_j^{\frac{1}{2}} \mathbf{p}_j \mathbf{q}_j^T\right)\mathbf{q}_i\Bigg]^2 + L\sigma^2 \sum_{i=1}^{s} \lambda_i^{-1}\Bigg[\left(\sum_{j=1}^{r} \lambda_j^{\frac{1}{2}} \mathbf{p}_j \mathbf{q}_j^T\right)\mathbf{q}_i\Bigg]^2\Bigg\} \\ &\quad \text{since } \mathbf{q}_j^T \mathbf{q}_i = \begin{cases} 0 & \text{if } j \neq i \\ 1 & \text{if } j = i \end{cases} \qquad (1) \\ E_{min} &= \frac{1}{L}\left[\sum_{i=s+1}^{r} \lambda_i + L\sigma^2 \sum_{i=1}^{s} \lambda_i^{-1}(\lambda_i)\right] \\ &= \frac{1}{L}\left[\sum_{i=s+1}^{r} \lambda_i + L\sigma^2 s\right] \end{aligned}$$

Solution to problem 2:

Using Eq.(5.30)

$$\Delta V = (s_k^{old} - s_k^{new})\Big[\sum_i w_{ki} s_i^{old} - \theta_k\Big]$$

For Hopfield analysis of a feedback network with binary $\{0, 1\}$ output function.

The update rule for each k is as follows

$$\text{Case } A \;:\; \text{if } \sum_i w_{ki} s_i^{old} - \theta_k > 0, \text{ then } s_k^{new} = 1$$

$$\text{Case } B \;:\; \text{if } \sum_i w_{ki} s_i^{old} - \theta_k < 0, \text{ then } s_k^{new} = 0$$

$$\text{Case } C \;:\; \text{if } \sum_i w_{ki} s_i^{old} - \theta_k = 0, \text{ then } s_k^{new} = s_k^{old}$$

For Case A, if $s_k^{old} = 1$, then $\Delta V = 0$, and
if $s_k^{old} = 0$, then $\Delta V < 0$.

For Case B, if $s_k^{old} = 1$, then $\Delta V < 0$, and
if $s_k^{old} = 0$, then $\Delta V = 0$.

For Case C, irrespective of s_k^{old}, $\Delta V = 0$.

Thus we have $\Delta V \leq 0$.

Solution to problem 3:

Consider a 3-unit model with bipolar $\{-1, +1\}$ units.

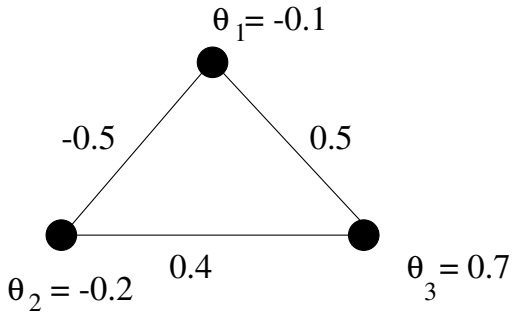
In this case, $s_i(t+1) = \begin{cases} +1, & \sum_j w_{ij} s_j(t) > \theta_i \\ -1, & \sum_j w_{ij} s_j(t) \leq \theta_i \end{cases}$

Choose weights and bias as:

$w_{12} = -0.5$; $\theta_1 = -0.1$

$w_{23} = 0.4$; $\theta_2 = -0.2$

$w_{31} = 0.5$; $\theta_3 = 0.7$



$$\text{Energy, } V(s_1, s_2, s_3) = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i s_i \theta_i$$

$$\Rightarrow V(s_1, s_2, s_3) = -(w_{12} s_1 s_2 + w_{23} s_2 s_3 + w_{31} s_3 s_1) + \theta_1 s_1 + \theta_2 s_2 + \theta_3 s_3$$

The energies at different states are obtained as

$$
\begin{aligned}
V(-1\ \ -1\ \ -1) &= -0.8 \\
V(-1\ \ -1\ \ \ 1) &= \ \ 2.4 \\
V(\ \ 1\ \ \ 1\ \ -1) &= \ \ 0.4 \\
V(-1\ \ \ 1\ \ -1) &= -1.4 \\
V(-1\ \ \ 1\ \ \ 1) &= \ \ 0.2 \\
V(\ \ 1\ \ \ 1\ \ \ 1) &= \ \ 0 \\
V(\ \ 1\ \ -1\ \ -1) &= -1.0 \\
V(\ \ 1\ \ -1\ \ \ 1) &= \ \ 0.2
\end{aligned}
$$

The state transition diagram is shown in the following figure:

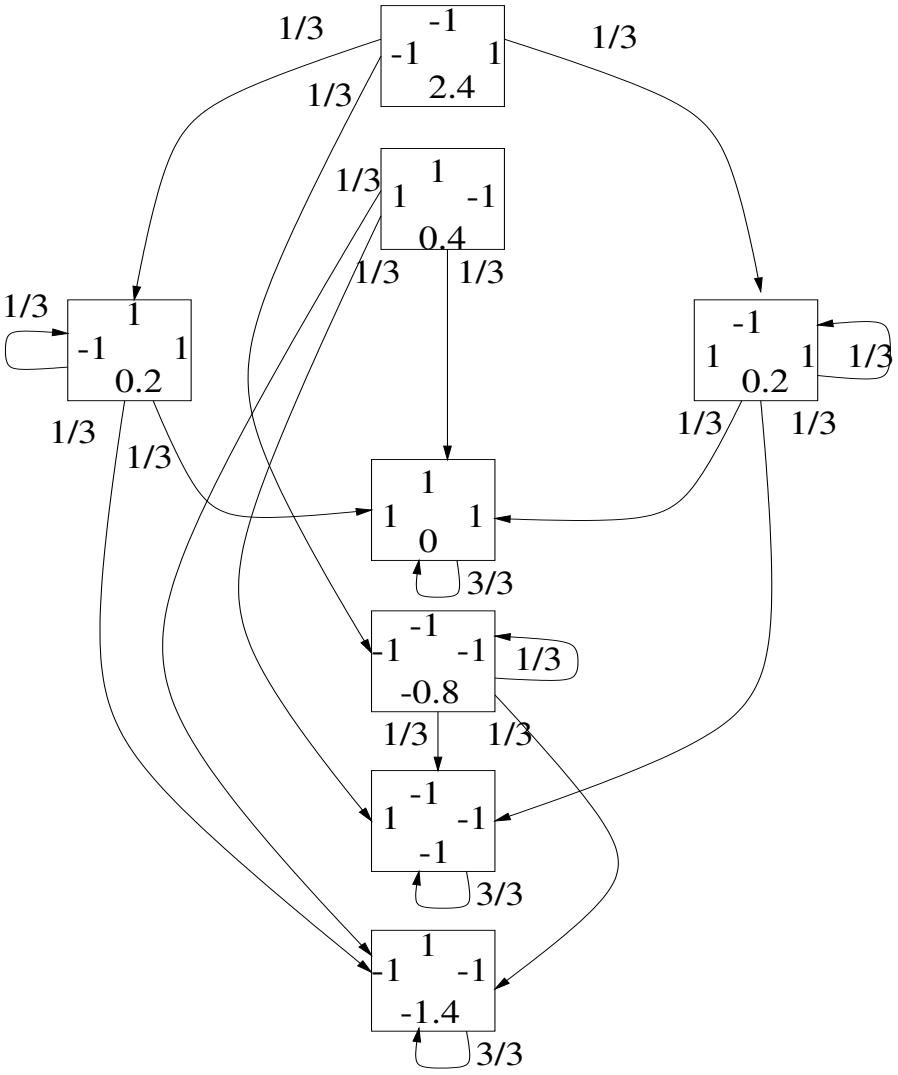


Figure: State transition diagram

Solution to problem 4:

$$\text{Using Eq.(5.65)} \quad \Rightarrow \quad F = -T\, log(\sum_{\alpha} e^{-E_\alpha/T}) \;=\; -T\, log Z$$

$$\text{Using Eq.(5.69)} \quad \Rightarrow \quad \frac{\partial E(\mathbf{s})}{\partial \theta_i} = s_i$$

where we consider the energy terms with the threshold terms θ_i. That is

$$\begin{aligned}
E_\alpha &= E(\mathbf{s}) = -\frac{1}{2}\sum_i \sum_j w_{ij} s_i s_j + \sum_i \theta_i s_i \\
\text{Therefore, } \frac{\partial F}{\partial \theta_i} &= -\frac{T}{Z}\frac{\partial Z}{\partial \theta_i} \\
&= -\frac{T}{Z}\,\frac{\partial}{\partial \theta_i}\left(\sum_\alpha e^{-E_\alpha/T}\right) \\
&= -\frac{T}{Z}\left(-\frac{1}{T}\right)\sum_\alpha e^{-E_\alpha/T}\frac{\partial E_\alpha}{\partial \theta_i} \\
&= \frac{1}{Z}\sum_\alpha s_i e^{-E_\alpha/T} \\
&= \sum_\alpha s_i \frac{e^{-E_\alpha/T}}{Z} \\
&= \sum_\alpha s_i P(\mathbf{s}_\alpha) \;= <s_i>
\end{aligned}$$

Solution to problem 5:

From Eq.(5.71) we have

$$p_{ij}^{-} \quad = \quad \frac{-\partial F}{\partial w_{ij}}$$

To show that

$$p_{ij}^{+} \quad = \quad -\frac{\partial \bar{F}^a}{\partial w_{ij}}$$

We know from Eq.(5.110),

$$\bar{F}^a \quad = \quad \sum_\alpha p^+(V_a) F^a$$

But since by definition (see Eq.(5.108),

$$Z^a_{clamped} \quad = \quad \sum_b e^{-E_{ab}/T} = e^{-F^a/T}$$

we have,

$$\begin{aligned}
F^a &= -T\, log\left(\sum_b e^{-E_{ab}/T}\right) \\
-\frac{\partial F^a}{\partial w_{ij}} &= \frac{T}{\sum_n e^{-E_{an}/T}}\sum_b \left(-\frac{1}{T}\right) e^{-E_{ab}/T}(-s_i^{ab} s_j^{ab}) \\
&= \sum_b \frac{e^{-E_{ab}/T}}{\sum_n e^{-E_{an}/T}} s_i^{ab} s_j^{ab}
\end{aligned}$$

where n is a dummy summation index.

But

$$\frac{e^{-E_{ab}/T}}{\sum_n e^{-E_{an}/T}} = P(H_b|V_a)$$

Therefore,

$$-\frac{\partial F^a}{\partial w_{ij}} = \sum_b P(H_b|V_a) s_i^{ab} s_j^{ab}$$

Hence the average value of $-\frac{\partial F^a}{\partial w_{ij}}$ is given by,

$$\begin{aligned} -\frac{\partial \bar{F}^a}{\partial w_{ij}} &= \sum_a P^+(V_a) \sum_b P(H_b|V_a) s_i^{ab} s_j^{ab} \\ &= \sum_{ab} P^+(V_a \wedge H_b) s_i^{ab} s_j^{ab} \;=\; p_{ij}^+ \end{aligned}$$

Solution to problem 6:

(See Hertz, 1991, p. 277, 278, 172 for details of this solution)

The free energy is given in Eq.(5.65) as $F = -T\, logZ$

If $P(\mathbf{s}_\alpha)$ is the probability of state $\mathbf{s}_\alpha$ at thermal equilibrium, then the average energy is given in Eq.(5.67) as

$$<E>= \sum_\alpha P(\mathbf{s}_\alpha) E_\alpha$$

The difference between the average energy and free energy is given by

$$\begin{aligned} <E> - F &= \sum_\alpha P(\mathbf{s}_\alpha) E_\alpha + T\, logZ \\ &= -T \sum_\alpha P(\mathbf{s}_\alpha)\, log\, \frac{e^{-E_\alpha/T}}{Z} = -T \sum_\alpha P(\mathbf{s}_\alpha)\, log P(\mathbf{s}_\alpha) \\ &= TH, \quad \text{where } H \text{ is the entropy of the system} \end{aligned}$$

In the mean-field approximation the units are treated as independent. Hence the entropy H is given by

$$H = -\sum_{i=1}^{N} [P(s_i = +1)\, log P(s_i = +1) \;+\; P(s_i = -1)\, log\, P(s_i = -1)]$$

where $P(s_i = \pm 1)$ are probabilities of the state of the bipolar unit i being ± 1.

Obviously $P(s_i = +1) + P(s_i = -1) = 1$, and the mean value of the i^{th} unit is

$$\begin{aligned} <s_i> &= (+1)P(s_i = +1) \;+\; (-1)P(s_i = -1) \\ &= P(s_i = +1) - P(s_i = -1) \\ \text{Hence} \quad P(s_i = \pm 1) &= \frac{1 \pm <s_i>}{2} \end{aligned}$$

In the mean-field approximation, since the average of each unit is considered independently, we have $<s_i s_j>=<s_i><s_j>$.

Therefore the mean-field free energy F_{mf} in terms of average energy and entropy is given by

$$\begin{aligned} F_{mf} &= <E> - TH \\ &= -\frac{1}{2}\sum_{ij} w_{ij} <s_i><s_j> \\ &\quad + T \sum_i [P(s_i=+1)\, logP(s_i=+1) \;+\; P(s_i=-1)\, logP(s_j=-1)] \end{aligned}$$

Therefore, $-\dfrac{\partial F_{mf}}{\partial w_{ij}} = <s_i><s_j>= p_{ij}^{-}$

A similar expression can be obtained for the mean-field clamped free energy $\bar{F}^a_{mf}$, by substituting the clamped values (± 1) of the state for $<s_i>$ at the visible units. Therefore

$$-\frac{\partial \bar{F}^a_{mf}}{\partial w_{ij}} = <s_i^a><s_j^a>= p_{ij}^{+} \text{ (See solution to problem 5)}$$

where a indicates the states of clamped network

$$\therefore \quad \triangle w_{ij} = \frac{\eta}{T}\left[<s_i^a><s_j^a> \;-\; <s_i><s_j>\right]$$

Solution to problem 7:

Using Eq.(5.86)

The error function $G = \sum_a P^+(V_a) \log \dfrac{P^+(V_a)}{P^-(V_a)}$

Slope of $\log x$ is $\frac{1}{x}$. Therefore for $x > 1$, slope of $\log\, x$ is < 1, and for $x < 1$, slope of $\log\, x$ is > 1

Therefore as shown in the figure, $\log\, x < x - 1$ for $x > 1$, $\log\, x = x - 1$ for $x = 1$, and $\log\, x < x - 1$ for $x < 1$. Hence $\log\, x \leq x - 1$ for all $x > 0$.

$$\text{Therefore } G = \sum_a P^+(V_a) \log \frac{P^+(V_a)}{P^-(V_a)} = -\sum_a P^+(V_a) \log \frac{P^-(V_a)}{P^+(V_a)} \tag{1}$$

$$\begin{aligned} \sum_a P^+(V_a) \log \frac{P^-(V_a)}{P^+(V_a)} &\leq \sum_a P^+(V_a)\left[\frac{P^-(V_a)}{P^+(V_a)} - 1\right] \quad (\text{using} \quad \log x \leq x-1) \\ &\leq \sum_a P^-(V_a) - \sum_a P^+(V_a) \\ &\leq 0 \\ \text{Therefore } G &\geq 0 \end{aligned}$$

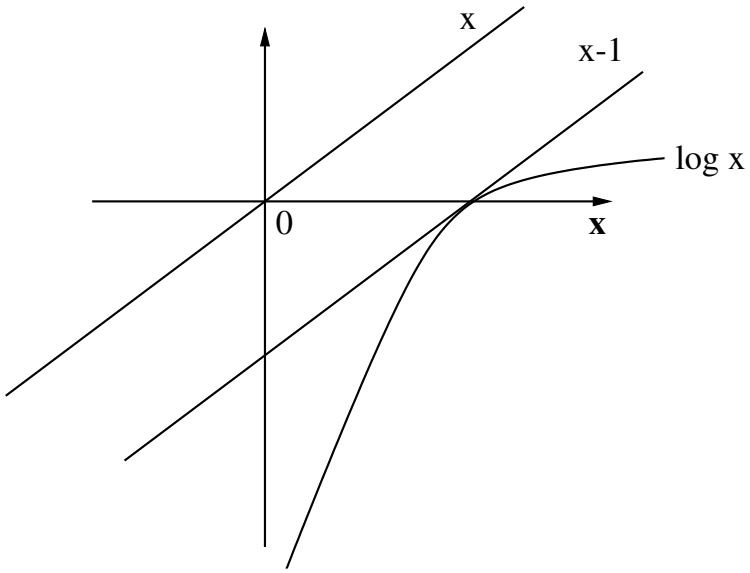
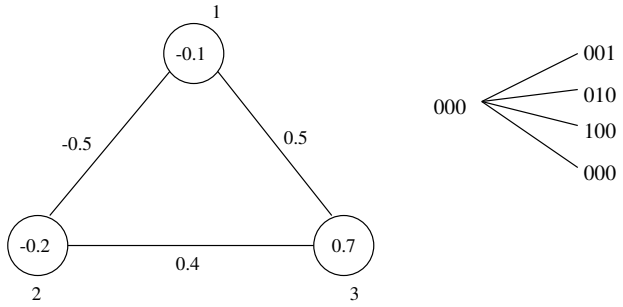


Figure: Plots of x, x-1 and log x

Solution to problem 8:

For the 3-unit network computing state transition probabilities $p(i/j)$ at T=1.0 for the state [0 0 0]

Assuming only one unit is allowed to change at a time.



For T=1.0, the values of $P(1|x)$ for units 1, 2, 3 are 0.525, 0.55, 0.332, respectively, and the values of $P(0|x)$ for units 1, 2, 3 are 0.475, 0.45, 0.67, respectively.

A change of state from 000 to 100 occurs when first unit fires.

Since such a unit can be chosen with a probability of 1/3, and the probability of unit 1 firing is 0.525. Therefore

the transition probability from 000 to 100 is 0.525/3 = 0.175

the transition probability from 000 to 010 is 0.550/3 = 0.183

and from 000 to 001 is 0.330/3 = 0.11

The probability of self transition is 1 - 0.175 - 0.183 - 0.11 = 0.532

Similarly computing the state transition probabilities for all the states, we get the following matrix.

STATE	000	001	010	011	100	101	110	111
000	0.532	0.11	0.183	0	0.175	0	0	0
001	0.1033	0.4514	0	0.2152	0	0.23	0	0
010	0.1501	0	0.5143	0.1419	0	0	0.1338	0
011	0	0.1181	0.1915	0.4989	0	0	0	0.1915
100	0.1583	0	0	0	0.5331	0.1667	0.1419	0
101	0	0.1033	0	0	0.1667	0.555	0	0.175
110	0	0	0.1966	0	0.1915	0	0.4094	0.1996
111	0	0	0	0.1419	0	0.1583	0.1338	0.6056

Solution to problem 9:

For a 5-unit Feedback network

$$W = \begin{bmatrix} 0 & 1 & -1 & -1 & -3 \\ 1 & 0 & 1 & 1 & -1 \\ -1 & 1 & 0 & 3 & 1 \\ -1 & 1 & 3 & 0 & 1 \\ -3 & -1 & 1 & 1 & 0 \end{bmatrix}$$

Given, $\theta_i = 0$ & $w_{ii} = 0$, $\forall\ i = 1, 2, \cdots, 5$. Therefore

$$\begin{aligned} V = -\frac{1}{2}\sum w_{ij}s_is_j = \ - \ (& w_{12}s_1s_2 + w_{13}s_1s_3 + w_{14}s_1s_4 + w_{15}s_1s_5 \\ & + w_{23}s_2s_3 + w_{24}s_2s_4 + w_{25}s_2s_5 \\ & + w_{34}s_3s_4 + w_{35}s_3s_5 \\ & + w_{45}s_4s_5) \end{aligned}$$

V(-1 1 1 1 1) = -$\Big($ -1 + 1 + 1 + 3 + 1 + 1 - 1 + 3 + 1 + 1 $\Big)$ = -10

Similarly, V(-1 -1 1 -1 -1) = 6.

Solution to problem 10:

Let $\mathbf{s} = [s_1\ s_2\ s_3]^T$ represent state vector. Then for the weights matrix $W = [w_{ij}]$, the energy of

the feedback network is given by

$$V = -\frac{1}{2}\mathbf{s}^T W \mathbf{s}$$

Therefore, gradient of V is given by

$$\begin{aligned}
\nabla V &= -W\mathbf{s} \\
&= -\begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} -s_2 - s_3 \\ -s_1 + s_3 \\ -s_1 + s_2 \end{bmatrix}
\end{aligned}$$

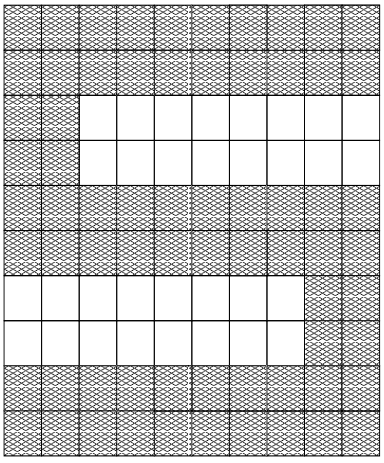
The Hessian matrix is given by

$$\begin{aligned}
\nabla^2 V &= -W \\
&= -\begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix}
\end{aligned}$$

Solution to problem 11:

Consider the representation of each of the ten digits $(0, 1, \cdots, 9)$ by a matrix of 10x10 elements, where each element is either 1 or -1.

For example for the digit 5.



Digit 1 is represented is by a 10x10 matrix. This can be generated from the figure by assuming black pixels as +1 and white pixels as -1. Similarly the other digits are also represented by 10x10 matrix.

In this Hopfield network, number of units = 100

Each unit correspond to one of the 100 pixels of each digit.

Weights are obtained using

$$w_{ij} = \begin{cases} \frac{1}{N}\sum_{l=1}^{L} a_{li}a_{lj}, & \text{for } i \neq j \\ 0, & \text{for } i = j \end{cases}$$

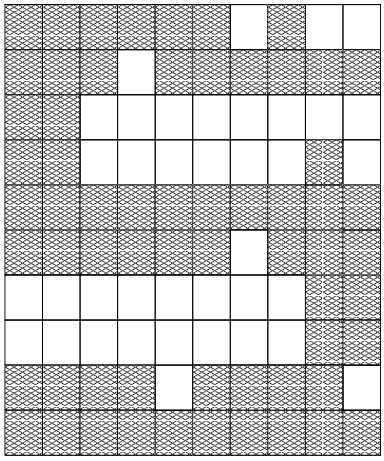
Here L=10, and a_{li} refers to the i^{th} pixel of l^{th} pattern.

Now 10% of the elements in the input array (i.e., 10 elements) were flipped, and given as input to the network. The output is iterated until there is no change in the state of the network using the following equation.

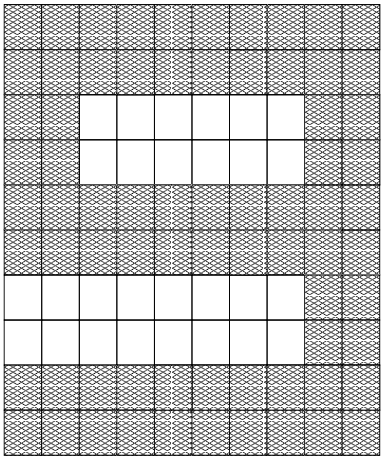
$$s_i(t+1) = sgn\left[\sum_{j=1}^{N} w_{ij}s_j(t)\right], i = 1, \ldots N$$

The performance of the Hopfield network can be studied with an example. The following figures show the noisy pattern of digit '5' and the reconstructed pattern. In this case the reconstructed pattern is digit '9' as it is closer to the given noisy version of the digit '5'.

Noisy pattern



Reconstructed pattern



Solution to problem 12:

(a) FeedForward Neural Network

(i) Linear units: $C \leq M$(dimensionality of the input)

In the case of FFNN with linear units the number of patterns associated can be limited by the dimension of the input pattern.

(ii) Non-linear units: $C = 2M$ for large M

Although the number of linearly separable functions for a 2-class problem(with M inputs and 2 outputs) is 2^{2^M}, the maximum capacity of a FFNN with non-linear units is $C = 2M$, for large M. (See Hertz *et al.*, 1991, pp.111-114).

(b) Feedback Neural Network with N units: $C = 0.138\ N$ for large N.

The maximum capacity of a feedback neural network is a fraction of the dimension of the input pattern, i.e., the number of patterns it will store is less than the dimension of the input vector.

(c) Hamming Network: $C = 2^{pM}$, $p \leq 1$

The p-class Hamming Network will have p output neurons. The capacity of the Hamming Network to store patterns is higher than the Hopfield network. The reason for this is that the number of connections increase linearly for Hamming network as the number of bits in the input pattern increase. On the other hand, the number of connections increase quadratically with the number of bits for the Hopfield network.

CHAPTER 6

Solution to problem 1:

From Eq.(6.12), weight update for a single unit instar with Hebbian learning

$$\Delta w_i = \eta y x_i$$

where $y = \sum_j w_j x_j$. Taking the expectation to compute the average

$$\begin{aligned} E[\Delta w_i] &= E[\eta \sum_j w_j x_j x_i] = \eta \sum_j w_j E[x_j x_i] \\ \text{Therefore} \quad E[\Delta \mathbf{w}] &= \eta R \mathbf{w}, \end{aligned}$$

where $R = E[\mathbf{x}\mathbf{x}^T]$ is the autocorrelation matrix. R is positive semidefinite, and hence R will have only positive eigenvalues. The weight vector converges if R$\mathbf{w}$=$\mathbf{0}$. This means that the resulting weight vector is an eigenvector with zero eigenvalue, which is not a stable situation, since R has positive eigenvalues.

Any fluctuation of the weight vector with a component along an eigenvector of R will result in a weight vector which would grow eventually, and the component along the eigenvector with the largest eigenvalue will dominate (see p. 208 for details).

i.e., $E[\Delta \mathbf{w}] = \eta R \mathbf{w}_0 = \eta \lambda_{max} \mathbf{w}_0$

Therefore, the average change of the weight will be dominated by the component along the eigenvector with maximum eigenvalue. Therefore, the norm of $\mathbf{w}$ will increase without bound (See p.208 for details).

Solution to problem 2:

For $f(x) = x^n$, $x > 0$ and $n > 1$

$= 0, x \leq 0,$

from Eq.(6.27) we have,

$$x\dot{X}_i = BxX_i \sum_k X_k[g(xX_i) - g(xX_k)] + BI_i - BX_i \sum_j I_j$$

where $g(y) = \dfrac{f(y)}{y} = \dfrac{y^n}{y} = y^{n-1}$

In the steady state $\dot{X}_i = 0$

If the input is removed, $I_i = 0, \forall i$

$$\therefore \qquad BxX_i \sum_k X_k[x^{n-1}X_i^{n-1} - x^{n-1}X_k^{n-1}] = 0$$

$$\Rightarrow \quad \sum_k X_k[X_i^{n-1} - X_k^{n-1}] = 0$$

$$\Rightarrow \quad X_i^{n-1} = \frac{\sum\limits_k X_k^n}{\sum\limits_k X_k} = \sum_k X_k^n, \qquad [\because \sum_k X_k = 1]$$

$$\Rightarrow \quad X_i^{n-1} = X_i^n + \sum_{k \neq i} X_k^n$$

Since this is to be satisfied for all n, the only non-zero solution for X_i is $X_i = 1$ and $X_k = 0 \quad \forall \quad k \neq i$

Therefore, when the input is zero, in the steady state only one of the units is activated to the maximum value.

Consider Eq.(6.24),

$$\dot{x} = -Ax + (B - x)\left[\sum_k f(x_k) + \sum_j I_j\right]$$

Imposing the above mentioned condition namely, $\dot{x} = 0$ and $f(x) = x^n$, we get

$$-Ax + (B - x)\sum_k x_k^n = 0$$

$$\Rightarrow x = \frac{B\sum\limits_k x_k^n}{A + \sum\limits_k x_k^n} = \frac{B}{1 + (A/\sum\limits_k x_k^n)}$$

This shows that total activation value, x, is bounded by B, since A and B are positive.

Solution to problem 3:

Oja's learning rule (Eq.(6.20)):

$$\begin{aligned}
\Delta w_i &= \eta y x_i - \eta y^2 w_i \\
<\Delta w_i> &= \eta <y x_i - y^2 w_i> \\
&= \eta <\sum_j w_j x_j x_i - \sum_{j,k} w_j x_j w_k x_k w_i>, \text{ since } y = \sum_j w_j x_j \\
<\Delta w_i> &= \eta \left(\sum_j R_{ij} w_j - \left[\sum_{j,k} w_j R_{jk} w_k\right] w_i\right) \qquad (1)
\end{aligned}$$

Weight vector converges if $<\Delta w_i>= 0$

$$\Rightarrow <\Delta \mathbf{w}>= R\mathbf{w} - (\mathbf{w}^T R\mathbf{w})\mathbf{w} = 0 \tag{2}$$

Thus at equilibrium,

$$R\mathbf{w} = \lambda \mathbf{w} \tag{3}$$

where $\lambda = \mathbf{w}^T R\mathbf{w} = \mathbf{w}^T \lambda \mathbf{w} = \lambda |\mathbf{w}|^2$

Equation (3) shows clearly that at equilibrium $\mathbf{w}$ must be an eigenvector of R, and the norm of the eigenvector $|\mathbf{w}|^2$ =1.

This can also be interpreted that Oja's rule minimizes the variance $\mathbf{w}^T R\mathbf{w}$ subjected to the constraint that $\sum_j w_j^2 = |\mathbf{w}|^2 = \mathbf{w}^T\mathbf{w} = 1$, since the cost function can be written as $-\frac{1}{2}\mathbf{w}^T R\mathbf{w} + \frac{\lambda}{2}(1 - \mathbf{w}^T\mathbf{w})^2$.

Thus the weight vector in the Oja's rule aligns with the eigenvector of the autocorrelation matrix. That this eigenvector corresponds to the largest eigenvalue is proved in [Hertz 1991, p. 203].

Any normalized eigenvector satisfies Equation(3), but only the eigenvector corresponding to the largest eigenvalue λ_{max} is stable. To prove this, consider the eigenvector $\mathbf{q}_i$ of R. Let us consider a weight vector that is a small deviation $\boldsymbol{\epsilon}$ from $\mathbf{q}_i$. That is $\mathbf{w} = \mathbf{q}_i + \boldsymbol{\epsilon}$.

Then from Equation(2)

$$\begin{aligned} <\triangle \boldsymbol{\epsilon}> \; = \; <\triangle \mathbf{w}> \;\; &= \;\; R(\mathbf{q}_i + \boldsymbol{\epsilon}) - \left[(\mathbf{q}_i^T + \boldsymbol{\epsilon}^T)\; R\; (\mathbf{q}_i + \boldsymbol{\epsilon})\right][\mathbf{q}_i + \boldsymbol{\epsilon}] \\ &= \;\; R\boldsymbol{\epsilon} - 2\lambda_i(\boldsymbol{\epsilon}^T\mathbf{q}_i)\mathbf{q}_i - \lambda_i\boldsymbol{\epsilon} + o(|\boldsymbol{\epsilon}|^2) \end{aligned}$$

where $o(|\boldsymbol{\epsilon}|^2)$ are higher order terms in $|\boldsymbol{\epsilon}|$. Taking the component of $<\triangle\boldsymbol{\epsilon}>$ along another eigenvector $\mathbf{q}_j$, we get

$$\begin{aligned} \mathbf{q}_j^T <\triangle \boldsymbol{\epsilon}> \;\; &= \;\; [\lambda_j - \lambda_i - 2\lambda_i\delta_{ij}]\, \mathbf{q}_j^T\boldsymbol{\epsilon} \\ &\text{where } \delta_{ij} = \begin{cases} 1, & \text{for } i = j \\ 0, & \text{for } i \neq j \end{cases} \end{aligned}$$

Note that we have used Equation(3) and its transpose in the above simplification. From the above it is obvious that if $\lambda_j > \lambda_i$ then the component of $\boldsymbol{\epsilon}$ projected onto $\mathbf{q}_j$ will grow unbounded. Hence the solution is unstable if $\lambda_j > \lambda_i$. If λ_i is the largest eigenvalue λ_{max}, then $\lambda_j - \lambda_i$ will be negative. Then the projected component will fluctuate, and thus the solution is stable. Hence the stable solution is the eigenvector having the largest eigenvalue.

Solution to problem 4:

$$R = \begin{bmatrix} 5 & 3 & 0 \\ 3 & 5 & 3 \\ 0 & 3 & 5 \end{bmatrix}$$

Using the Matlab command *eig*, which returns eigenvalues and eigenvectors of the input matrix,

The eigenvalues are $\begin{bmatrix} 9.2426 \\ 5 \\ 0.7574 \end{bmatrix}$

Eigenvector of 9.2426 is $\begin{bmatrix} 0.5 & 0.7071 & 0.5 \end{bmatrix}^T$

Eigenvector of 5.0 is $\begin{bmatrix} -0.7071 & 0 & 0.7071 \end{bmatrix}^T$

Eigenvector of 0.7574 is $\begin{bmatrix} -0.5 & 0.7071 & -0.5 \end{bmatrix}^T$

$\therefore$ Eigenvector corresponding to the first Principal component is $\begin{bmatrix} 0.5 & 0.7071 & 0.05 \end{bmatrix}^T$

Solution to problem 5:

$$R_{xy} = \begin{bmatrix} 5 & 3 & 0 & 0 \\ 2 & 4 & 2 & 0 \\ 0 & 2 & 5 & 1 \end{bmatrix}$$

Apply SVD to R_{xy} , i.e., $R_{xy} = USV^T$

Using the Matlab command *svd*, which returns the eigenvalues, right eigenvector and left eigenvector,

$$U = \begin{bmatrix} 0.6214 & -0.6318 & 0.4634 \\ 0.6148 & 0.0265 & -0.7883 \\ 0.4857 & 0.7747 & 0.4048 \end{bmatrix}$$

$$S = \begin{bmatrix} 7.7754 & 0 & 0 & 0 \\ 0 & 5.0716 & 0 & 0 \\ 0 & 0 & 1.3495 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.5577 & -0.6124 & 0.5488 & -0.1127 \\ 0.6810 & -0.0473 & -0.7062 & 0.1879 \\ 0.4705 & 0.7742 & 0.3318 & -0.2631 \\ 0.0625 & 0.1528 & 0.3 & 0.9396 \end{bmatrix}$$

The left eigenvector corresponding to the first principal component is

$$\begin{bmatrix} 0.6214 & 0.6148 & 0.4857 \end{bmatrix}^T$$

The right eigenvector corresponding to first principal component is

$$\begin{bmatrix} 0.5577 & 0.6810 & 0.4705 & 0.0625 \end{bmatrix}^T$$

Solution to problem 6:

$$\Delta w_i = \eta(a_i x + \alpha a_i + \beta x + \gamma),$$

where $x = \sum_{j=1}^{M} w_j a_j + \theta$.

$$\begin{aligned} \frac{<\Delta w_i>}{\eta} &= E[a_i x] + \alpha E[a_i] + \beta E[x] + \gamma \qquad (1) \\ E[x] &= E[\sum_j w_j a_j + \theta] = E[\sum_j w_j a_j] + \theta = \sum_j w_j \bar{a} + \theta \\ E[a_i] &= \bar{a} \end{aligned}$$

$$\begin{aligned} E[a_i x] &= E[a_i(\sum_j w_j a_j + \theta)] = E[\sum_j w_j a_j a_i] + \theta E[a_i] \\ &= \sum_j w_j E[a_j a_i] + \theta \bar{a} \\ &= \sum_i w_j \bar{a}.\bar{a} + \sum_j w_j C_{ij} + \theta \bar{a} \end{aligned}$$

since $E[a_i a_j] = E[(\bar{a}+\epsilon_i)(\bar{a}+\epsilon_j)] = \bar{a}.\bar{a} + E[\epsilon_i \epsilon_j] = \bar{a}.\bar{a} + C_{ij}, E[\epsilon_i] = E[\epsilon_j] = 0$ and $C_{ij} = E[\epsilon_i \epsilon_j]$.

$$\begin{aligned} \text{Equation(1)} \quad &\Rightarrow \quad \sum_j w_j \bar{a}.\bar{a} + \sum_j w_j C_{ij} + \theta\bar{a} + \alpha\bar{a} + \beta \sum_j w_j \bar{a} + \beta\theta + \gamma = \frac{<\Delta w_i>}{\eta} \\ &\Rightarrow \quad \sum_j w_j C_{ij} + [-\bar{a}(\bar{a}+\beta)] \left[\frac{\theta\bar{a} + \alpha\bar{a} + \beta\theta + \gamma}{-\bar{a}(\bar{a}+\beta)} - \sum_j w_j \right] = \frac{<\Delta w_i>}{\eta} \end{aligned}$$

Let $\lambda = -\bar{a}(\bar{a}+\beta), \qquad \mu = \dfrac{\theta\bar{a} + \alpha\bar{a} + \beta\theta + \gamma}{-\bar{a}(\bar{a}+\beta)}$

$$\therefore \qquad <\Delta w_i>= \eta \left[\sum_j C_{ij} w_j + \lambda[\mu - \sum_j w_j] \right]$$

Solution to problem 7:

From the cost function

$$\begin{aligned} E &= -\frac{1}{2}\mathbf{w}^T C \mathbf{w} + \frac{\lambda}{2}(\mu - \sum_j w_j)^2, \text{ where } C = [C_{ij}] \\ &= -\frac{1}{2}\sum_i \sum_j C_{ij} w_i w_j + \frac{\lambda}{2}(\mu - \sum_j w_j)^2 \qquad (1) \\ \therefore \Delta w_i &= -\eta \frac{\partial E}{\partial w_i} = \eta \left[\sum_j C_{ij} w_j + \lambda(\mu - \sum_j w_j) \right] \end{aligned}$$

which is Linsker rule.

The expression for E consists of output variance and the constraint term $\sum_j w_j = \mu$. Linsker's rule tries to maximize the variance by adjusting the weights according to the negative gradient of the variance subjected to the constraint $\sum_j w_j = \mu$.

Solution to problem 8: (See Haussoun, 1995, p. 96-97, Hertz et al, 1991, p.212-213)

The average of Linsker rule at the component level is

$$<\triangle w_i>= \eta \left[\sum_j C_{ij} w_j + \lambda(\mu - \sum_j w_j) \right] \qquad (1)$$

The average of the Linsker rule at the vector level is

$$<\triangle \mathbf{w}>= \eta \left[C\mathbf{w} + \lambda(\mu - \sum_j w_j)\mathbf{1} \right] \qquad (2)$$

where $\mathbf{1}$ is the M-dimensional column vector of ones, and C is the MxM covariance matrix of input data.

At equilibrium, $<\triangle \mathbf{w}>= 0$. Then we must have

$$C\mathbf{w} = \mathbf{r} = \text{constant vector},$$

which means that the eigenvectors of C should all have the same constant value. This is a trivial solution.

Since the Linsker rule minimizes $-\frac{1}{2}\mathbf{w}^T C \mathbf{w} + \frac{\lambda}{2}(\mu - \sum_j w_j)^2$, for large λ, the minimum occurs when $\sum_j w_j = \mu$. This can happen if the weights are pinned at one of the boundary values,

either at w_- with $<\triangle w_i> \leq 0$, or at w_+ with $<\triangle w_i> \geq 0$ with $w_+ = -w_- = \mu$.

Thus the final state will have $(M+1)/2$ values of w_+ and the rest w_-, if M is odd. For M even, $\frac{M}{2}$ weights will be w_+ and $\frac{M}{2} - 1$ weights will be w_- and one of the component weights will be zero. Thus all, or all but one, of the weights components must be at a boundary value.

Any weight vector with two or more components not at a boundary will be unstable as per Equation(1) or Equation(2). To see this, let us consider the contrary situation, by considering a perturbation of weight $\mathbf{w}' = \mathbf{w} + \boldsymbol{\epsilon}$, away from the equilibrium point. In particular, choose $\boldsymbol{\epsilon}$ such that $\sum_j \epsilon_j = 0$, with $\epsilon_i = 0$, if w_i is at the boundary. Then (2) gives $<\triangle \boldsymbol{\epsilon}> = \eta C \boldsymbol{\epsilon}$, which after m iterations gives $\boldsymbol{\epsilon} = \boldsymbol{\epsilon}_0 + m\eta C \boldsymbol{\epsilon}_0$, where $\boldsymbol{\epsilon}_0$ is some initial value.

Since C is positive definitive, $|\boldsymbol{\epsilon}|$ grows with m. Thus the chosen point $\mathbf{w}'$ is not a stable or equilibrium point.

Therefore all the weights are pinned to the boundary values.

Solution to problem 9:

$$
\begin{aligned}
y_i &= \sum_{j=1}^{M} w_{ij} a_j \\
\Delta w_{ij} &= \eta\, y_i \left[a_j - \sum_{k=1}^{i} y_k w_{kj} \right] \\
\frac{<\Delta w_{ij}>}{\eta} &= E[y_i a_j] - \sum_{k=1}^{i} E[y_k y_i] w_{kj} \qquad (1) \\
E[y_i a_j] &= E[\sum_k w_{ik} a_k a_j] = \sum_k w_{ik} R_{kj} \\
E[y_k y_i] &= E[\sum_r w_{kr} a_r \sum_p w_{ip} a_p] \\
&= \sum_r \sum_p w_{kr} w_{ip} E[a_r a_p] = \sum_{rp} w_{kr} R_{rp} w_{ip} \\
(1) \Rightarrow \frac{<\Delta w_{ij}>}{\eta} &= \sum_k w_{ik} R_{kj} - \sum_{k=1}^{i} \left[\sum_{rp} w_{kr} R_{rp} w_{ip} \right] w_{kj}
\end{aligned}
$$

Combining all the components w_{ij} into the vector $\mathbf{w_i}$, we get

$$
\frac{<\triangle \mathbf{w_i}>}{\eta} = R\mathbf{w_i} - \sum_{k=1}^{i-1} \left[\mathbf{w_k}^T R \mathbf{w_i} \right] \mathbf{w_k} - (\mathbf{w}_i^T R \mathbf{w}_i) \mathbf{w}_i
$$

where the term $k = i$ is separated from the summation.

Assume that the first $(i-1)$ weights correspond to the first $(i-1)$ eigenvectors of R. Then $\mathbf{w}_k^T R \mathbf{w}_i$ is the component of the vector $R\mathbf{w}_i$ on the eigenvector $\mathbf{w}_k$, since $\mathbf{w}_k$ is a unit vector. Therefore the projection of $R\mathbf{w}_i$ on the eigenvector $\mathbf{w}_k$ is $(\mathbf{w}_k^T R \mathbf{w}_i)\mathbf{w}_k$. Summing this for all the first $(i-1)$ eigenvectors gives the projection of $R\mathbf{w}_i$ on the subspace spanned by the first

$(i-1)$ eigenvectors. Hence $R\mathbf{w}_i - \sum_{k=1}^{i-1}(\mathbf{w}_k^T R\mathbf{w}_i)\mathbf{w}_k$ is the error vector orthogonal to the subspace spanned by the first $(i-1)$ eigenvectors and hence in the subspace orthogonal to first $(i-1)$ eigenvectors. Now, if $\mathbf{w}_i$ is not in this orthogonal subspace, then this error vector will be zero.

$$\text{Hence} \quad \frac{<\triangle \mathbf{w}_i>}{\eta} = -(\mathbf{w}_i R\mathbf{w}_i)\mathbf{w}_i$$

This is like decay term, since $\mathbf{w}_i^T R\mathbf{w}_i$ is a positive scalar. Hence $<\triangle \mathbf{w}_i> \rightarrow 0$. That means $\mathbf{w}_i$ relaxes to the subspace of $(i-1)$ eigenvectors. But if $\mathbf{w}_i$ is in the orthogonal subspace, then the resulting learning law is like Oja's 1-unit rule for the unit i. Therefore the learning leads to the maximum eigenvector in the orthogonal subspace. Thus $\mathbf{w}_i$ is the i^{th} eigenvalue of R.

Solution to problem 10:

$$\begin{aligned} y &= \sum_{i=1}^{M}(w_i + \sum_{j=1}^{M} w_{ij}x_j)x_i = \sum_{i=1}^{M} w_i x_i + \sum_{ij} w_{ij}x_j x_i \\ &= (w_1x_1 + w_2x_2 + \cdots + w_M x_M) + (w_{11}x_1x_1 + w_{12}x_1x_2 + \cdots + w_{MM}x_M x_M) \end{aligned}$$

$$= \begin{bmatrix} w_1 & w_2 \cdots w_M & w_{11} & w_{12} \cdots w_{MM} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \\ x_1x_1 \\ x_1x_2 \\ \vdots \\ x_2x_1 \\ \vdots \\ x_Mx_M \end{bmatrix}$$

$$\therefore y = \mathbf{w}^T\mathbf{z} = \mathbf{z}^T\mathbf{w}$$

Therefore $\mathbf{z}$ is interpreted as input vector and $\mathbf{w}$ is interpreted as weight vector. Therefore

$$\begin{aligned} \Delta\mathbf{w} &= \eta\mathbf{z}y \\ &= \eta\mathbf{z}(\mathbf{z}^T\mathbf{w}) \\ <\Delta\mathbf{w}> &= E[\eta\,\mathbf{z}\mathbf{z}^T\mathbf{w}] = \eta\mathbf{w}E[\mathbf{z}\mathbf{z}^T] \\ &= \eta\mathbf{w} <\mathbf{z}\mathbf{z}^T> \end{aligned}$$

This equation is similar to that of plain Hebbian learning, except the $<\mathbf{x}\mathbf{x}^T>$ is replaced by $<\mathbf{z}\mathbf{z}^T>$.

Therefore, the principal component resulting from this network corresponds to the principal eigenvector of the matrix $<\mathbf{z}\mathbf{z}^T>$.

Solution to problem 11:

Malsburg learning

$$\begin{aligned}
\Delta w_{kj} &= \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right], \forall j \\
w_{kj}(n+1) &= w_{kj}(n) + \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj}(n) \right] \\
\sum_j w_{kj}(n+1) &= \sum_j w_{kj}(n)[1-\eta] + \frac{\eta}{\sum_i x_i} \sum_j x_j \\
&= (1-\eta) \sum_j w_{kj}(n) + \eta
\end{aligned}$$

Since the initial weights are normalized, $\sum_j w_{kj}(n) = 1$, we get

$$\sum_j w_{kj}(n+1) = 1 - \eta + \eta = 1$$

Solution to problem 12:

A set of random points are selected using the four Gaussian distributions in the problem is shown in Table 1 and the cluster centers formed by Matlab program for different η values are shown in Table 2. The Matlab program used for the execution of this problem is:

```
function[]=prob6_12(eta,cycles)
data1=twoD_gauss_v1(2,2,-5,-5); data2=twoD_gauss_v1(1.732,1.732,-5,5);
data3=twoD_gauss_v1(1.414,1.414,5,-2); data4=twoD_gauss_v1(1.414,1.414,5,5);
Inp1=[data1;data2;data3;data4]';
min1=min(Inp1(1,:)); max1=max(Inp1(1,:));
min2=min(Inp1(2,:)); max2=max(Inp1(2,:));
net=newc([min1 max1;min2 max2],4,eta);
net.trainParam.epochs=cycles;
net=train(net,Inp1);
```

Table 1: A set of random points

Data 1		Data 2		Data 3		Data 4	
$Mean = [-5 \quad -5]$ $\sigma = 4$		$Mean = [-5 \quad 5]$ $\sigma = 3$		$Mean = [5 \quad -2]$ $\sigma = 2$		$Mean = [5 \quad 5]$ $\sigma = 2$	
x	y	x	y	x	y	x	y
-6	-5	-6	6	4	-2	4	5
-6	-4	-5	5	4	-1	4	6
-6	-3	-5	6	4	0	4	7
-5	-6	-5	7	5	-3	5	4
-5	-5	-4	4	5	-2	5	5
-5	-4	-4	5	5	-1	5	6
-5	-3	-4	6	5	0	5	7
-5	-2	-4	7	5	1	5	8
-4	-6	-4	8	6	-3	6	4
-4	-5	-3	5	6	-2	6	5

Table 2: Cluster centers of problem 12

η	CLUSTER CENTERS							
	Data 1		Data 2		Data 3		Data 4	
	x	y	x	y	x	y	x	y
0.5	-2.75	-5.05	-4.41	6.93	4.95	-1.07	6.65	6.83
0.2	-3.88	-4.35	-4.35	5.66	5.94	-0.95	6.23	6.70
0.1	-3.44	-3.77	-3.09	5.52	6.12	-0.79	5.95	5.63
0.05	-4.23	-2.9	-4.19	5.84	4.93	-1.2	6.40	6.36
0.01	-4.12	-3.99	-4.04	5.82	5.90	-1.07	6.11	6.24

Solution to problem 13:

For the data in problem 12, the cluster centers obtained using the LVQ1 algorithm are shown in Table 3.

The Matlab program used for the execution of this problem is:

```
function[]=prob6_13(eta)
data1=twoDgauss(2,2,-5,-5); [r1 c1]=size(data1);
var1=repmat(1,[1 r1]);
data2=twoDgauss(1.732,1.732,-5,5); [r2 c2]=size(data2);
var2=repmat(2,[1 r2]);
data3=twoDgauss(1.414,1.414,5,-2); [r3 c3]=size(data3);
var3=repmat(3,[1 r3]);
data4=twoDgauss(1.414,1.414,5,5); [r4 c4]=size(data4);
var4=repmat(4,[1 r4]);
P=[data1;data2;data3;data4]'; C=[var1 var2 var3 var4];
T=ind2vec(C);
[W1,W2]=initlvq(P,4,T);
tp=[20 500 eta];
[W1,W2]=trainlvq(W1,W2,P,T,tp);
```

Table 3: Cluster centers of problem 13

η	CLUSTER CENTERS							
	Data 1		Data 2		Data 3		Data 4	
	x	y	x	y	x	y	x	y
0.5	-5.0	-4.52	-3.92	6.81	6.01	-0.08	5.97	6.81
0.2	-4.4	-3.69	-4.81	6.02	6.04	-0.84	6.45	6.52
0.1	-3.59	-3.88	-2.31	4.32	3.20	-0.39	6.81	4.9
0.05	-3.72	-3.37	-3.74	6.37	5.83	-0.88	5.60	5.73
0.01	-3.12	-3.29	-2.42	4.24	3.98	-0.50	4.62	4.29

Solution to problem 14:

The input data is generated by selecting the points at random from an annular ring.

(a) The units in the SOM are arranged in a 2-dimensional plane

The feature map is displayed by the weight values of the connections leading to each unit after training. The weights for each unit is a point in the (w_1, w_2) space, and the points corresponding to adjacent units in the output layer are joined in the figure.
Initially the weights are set to random values. As training proceeds, the weights are modified to span the space in the (w_1, w_2) plane. The shape of the spanned space is dictated by the shape of the region from which the input points are selected at random. The feature maps obtained for different iterations are shown in the Figs. 1, 2, and 3.

After 1000 iterations.

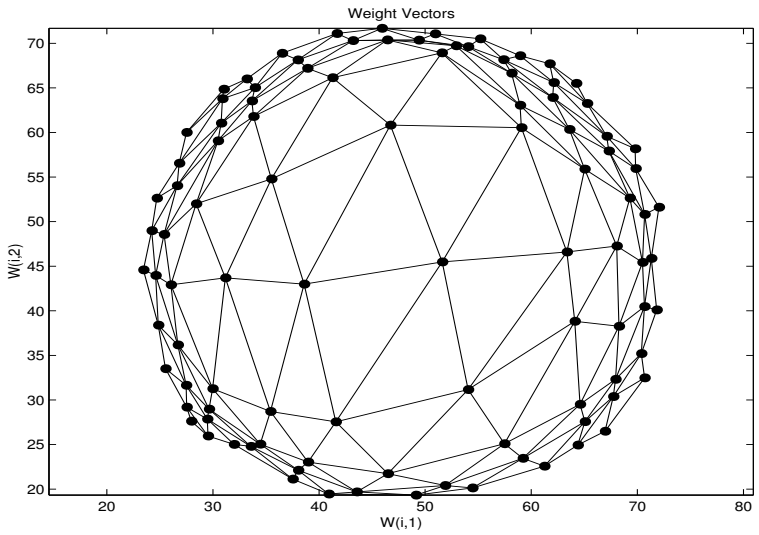


Figure 1: The feature map obtained after 1000 iterations

After 2000 iterations.

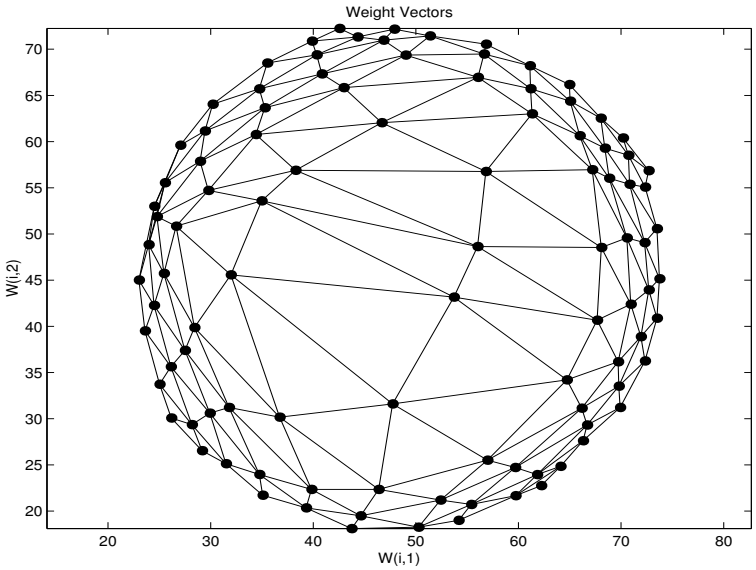


Figure 2: The feature map obtained after 2000 iterations

After 3000 iterations.

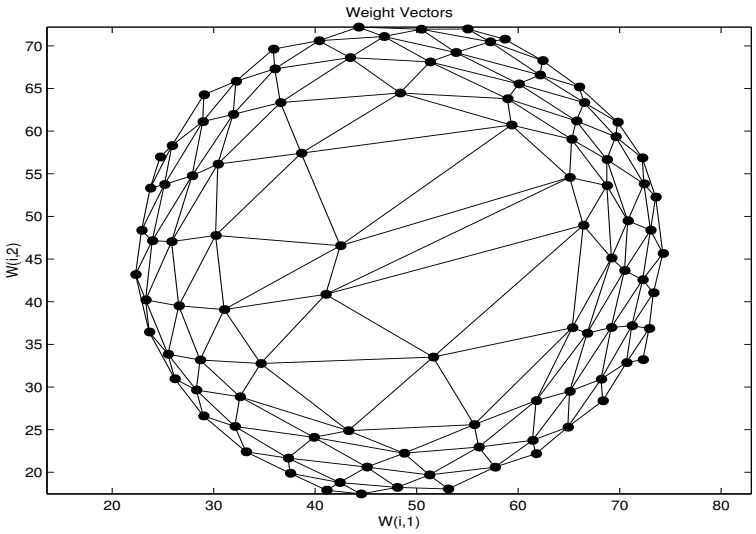


Figure 3: The feature map obtained after 3000 iterations

Comment: The feature map displays concentration of points in the region of annular ring as the number of iterations increase.

(b) The units in the SOM are arranged to form a 1-D layer. In this case, the feature maps obtained for different iterations are shown in the Figs.4, 5, and 6.

After 1000 iterations.

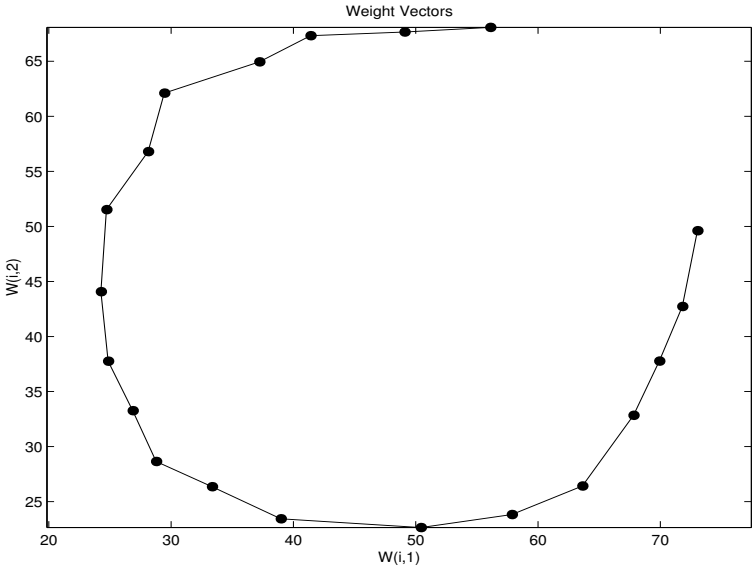


Figure 4: The feature map obtained after 1000 iterations

After 2000 iterations.

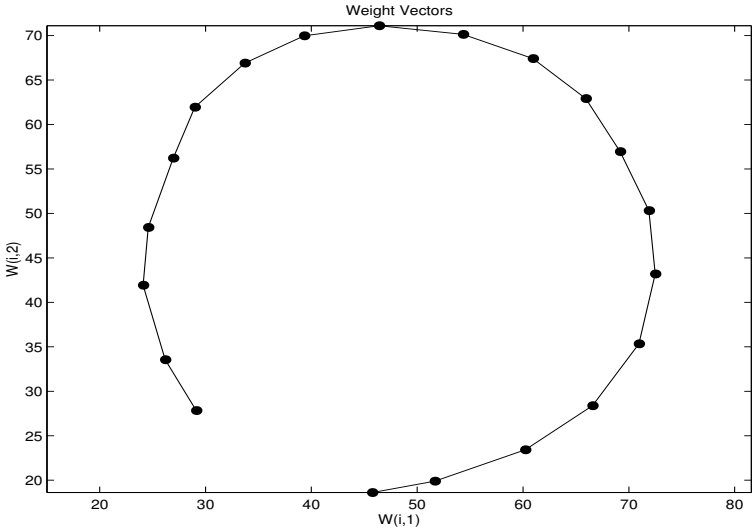


Figure 5: The feature map obtained after 2000 iterations

After 3000 iterations.

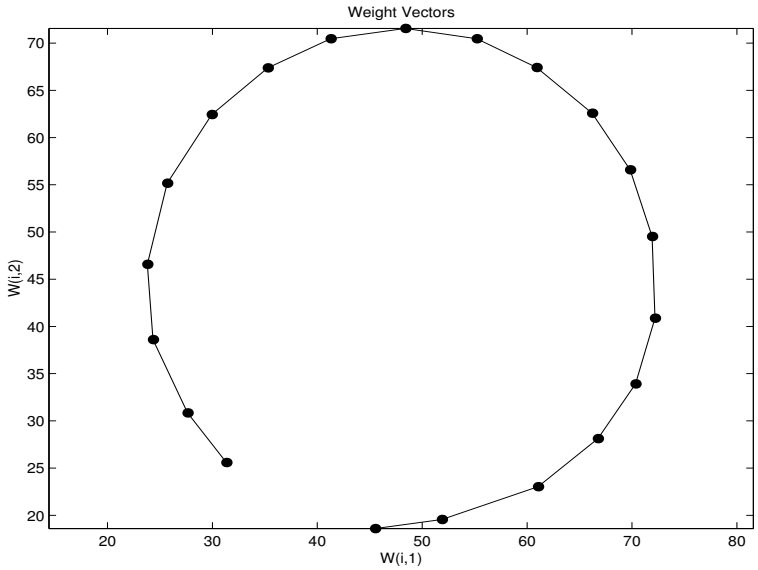


Figure 6: The feature map obtained after 3000 iterations

Comment: Above figures illustrates that, as the number of iterations increases, the weights are organized according to the shape of the annular ring, formed by the two concentric circles. And for 3000 iterations, it almost organized the weight vectors in a ring shape. The Matlab program used for the execution of this problem is:

```
function[] = prob6_14()
load InputData.mat
net=newsom([0 1;0 1],[10 10]);
%Uncomment next line for som of 1-D layer
net=newsom([0 1;0 1],[20 1]);
for i=1000:1000:3000
    net.trainParam.epochs=i;
    net=train(net,InputData);
    plotsom(net.iw{1,1},net.layers{1}.distances);
end
```

CHAPTER 7

Solution to problem 1:

The BAM weight matrix from the first layer to the second layer is given by:

$$W = \sum_{l=1}^{L} \mathbf{a}_l \, \mathbf{b}_l^T$$

The activation equations for the binary case are as follows:

$$b_j(m+1) \;\; = \;\; \begin{cases} 1, & \text{if } y_j > 0 \\ b_j(m), & \text{if } y_j = 0 \\ 0, & \text{if } y_j < 0 \end{cases}$$

where $y_j = \sum_{i=1}^{M} w_{ji} \, a_i(m)$, and

$$a_i(m+1) \;\; = \;\; \begin{cases} 1, & \text{if } x_i > 0 \\ a_i(m), & \text{if } x_i = 0 \\ 0, & \text{if } x_i < 0 \end{cases}$$

where $x_i = \sum_{j=1}^{N} w_{ij} \, b_j(m)$.

$$\text{Lyapunov Energy function } V(a,b) \;\; = \;\; -\sum_{i=1}^{M} \sum_{j=1}^{N} w_{ij} \, b_j \, a_i$$

The change in energy due to change in $\triangle a_i$ in a_i is given by

$$\triangle V_{a_i} = -\left[\sum_{j=1}^{N} w_{ij} \, b_j \right] \triangle a_i, \;\; i = 1, 2, 3, \cdots M$$

Likewise, the change in energy due to change in $\triangle b_j$ in b_j is given by

$$\triangle V_{b_j} = -\left[\sum_{i=1}^{M} w_{ji} \, a_i \right] \triangle b_j, \;\; j = 1, 2, 3, \cdots N$$

For binary units

$$\triangle a_i = \begin{cases} 0 \text{ or } 1 & \text{if } x_i > 0 \\ 0 & \text{if } x_i = 0 \\ -1 \text{ or } 0 & \text{if } x_i < 0 \end{cases}$$

$$\triangle b_j = \begin{cases} 0 \text{ or } 1 & \text{if } y_j > 0 \\ 0 & \text{if } y_j = 0 \\ -1 \text{ or } 0 & \text{if } y_j < 0 \end{cases}$$

From these relations

We get $\triangle V_{a_i} \leq 0$, for $i = 1, 2, \ldots M$,

$$\triangle V_{b_j} \leq 0, \text{ for } j = 1, 2, \cdots N$$

Therefore BAM is unconditionally stable for any binary units.

Solution to problem 2:

Consider a BAM with M units in the first layer and N units in the second layer with bidirectional links. The weight matrix from the first layer to the second layer is given by

$$U = [\mathbf{u}_1\, \mathbf{u}_2 \cdots \mathbf{u}_N]^T = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1M} \\ u_{21} & u_{22} & \cdots & u_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ u_{N1} & u_{N2} & \cdots & u_{NM} \end{bmatrix} = [u_{ij}]_{N\text{x}M}$$

The weight matrix from the second layer to the first layer is given by

$$V = [\mathbf{v}_1\, \mathbf{v}_2 \cdots \mathbf{v}_N]^T = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1N} \\ v_{21} & v_{22} & \cdots & v_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ v_{M1} & v_{M2} & \cdots & v_{MN} \end{bmatrix} = [v_{ij}]_{M\text{x}N}$$

The overall weight matrix is given by

$$W = \begin{bmatrix} 0 & 0 & \cdots & 0 & u_{11} & u_{12} & \cdots & u_{1M} \\ 0 & 0 & \cdots & 0 & u_{21} & u_{22} & \cdots & u_{2M} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & u_{N1} & u_{N2} & \cdots & u_{NM} \\ & & & & & & & \\ v_{11} & v_{12} & \cdots & v_{1N} & 0 & 0 & \cdots & 0 \\ v_{21} & v_{22} & \cdots & v_{2N} & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{M1} & v_{M2} & \cdots & v_{MN} & 0 & 0 & \cdots & 0 \end{bmatrix}_{(M+N)\text{x}(M+N)}$$

Since the links are bidirectional, $v_{ij} = u_{ji}$ for $i = 1, 2, \cdots, M$ and $j = 1, 2, \cdots, N$. Since there

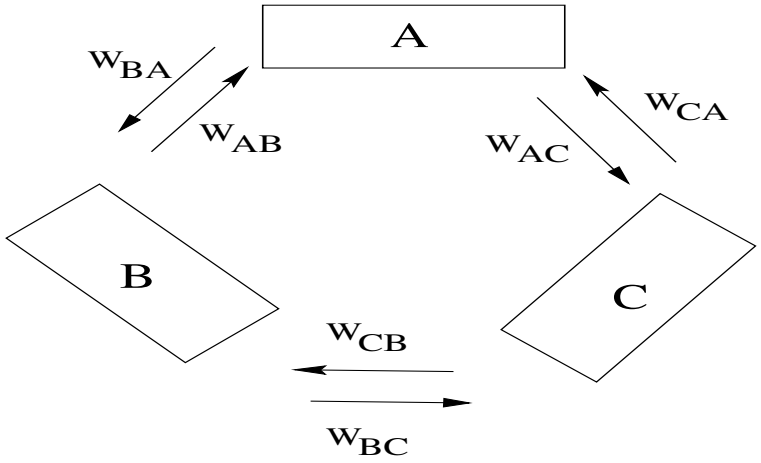
exists no self loops and the weights are symmetric, we can visualize this network as Hopfield type. Hence, it is stable for asynchronous update.

Solution to problem 3:

Consider the following three vectors for a tridirectional associative memory:

$$\mathbf{a} = [1 \ \ -1 \ \ -1 \ \ 1 \ \ -1]^T \ , \ \mathbf{b} = [1 \ \ 1 \ \ -1 \ \ -1]^T \ , \ \mathbf{c} = [1 \ \ -1 \ \ 1]^T$$

The weight matrices of the following tridirectional associative memory are



$$W_{BA} = W_{AB}^T = \mathbf{b}\,\mathbf{a}^T = \begin{bmatrix} 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & 1 \end{bmatrix}$$

$$W_{CB} = W_{BC}^T = \mathbf{c}\,\mathbf{b}^T = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \end{bmatrix}$$

$$W_{AC} = W_{CA}^T = \mathbf{a}\,\mathbf{c}^T = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix}$$

For recall, the initial noisy vectors $\mathbf{a}^{(0)}, \mathbf{b}^{(0)}, \mathbf{c}^{(0)}$ at layers A, B and C are used in the following manner. Note that each network between the layers is bidirectional and each layer gets input from both the adjacent networks.

Let the initial noisy vectors be:

$$\mathbf{a}^{(0)} = [-1 \quad -1 \quad 1 \quad 1 \quad 1]^T \ , \ \mathbf{b}^{(0)} = [1 \quad -1 \quad 1 \quad -1]^T \ , \ \mathbf{c}^{(0)} = [1 \quad 1 \quad 1]^T$$

$$\begin{aligned} \mathbf{a}^{(\mathbf{1})} &= f[W_{AB}\,\mathbf{b}^{(0)} + W_{AC}\,\mathbf{c}^{(0)}] \\ \mathbf{b}^{(\mathbf{1})} &= f[W_{BC}\,\mathbf{c}^{(0)} + W_{BA}\,\mathbf{a}^{(0)}] \\ \mathbf{c}^{(\mathbf{1})} &= f[W_{CA}\,\mathbf{a}^{(0)} + W_{CB}\,\mathbf{b}^{(0)}] \end{aligned}$$

Where $f(\cdot)$ is the output function applied to each component of the vector. That is, for each component x, if $x > 0$, then the output $f(x) = 1$, and if $x \leq 0$, then the output $f(x) = -1$.

For this noisy input, the tridirectional associative memory restores the stored pattern in a single iteration.

For the bidirectional associative memory, consider the first two vectors $\mathbf{a}$ and $\mathbf{b}$. The BAM weight matrix is

$$W_{BA} = W_{AB}^T = \mathbf{b}\,\mathbf{a}^T$$

For recall, use

$$\begin{aligned} \mathbf{a}^{(1)} &= f[W_{AB}\,\mathbf{b}^{(0)}] \\ \mathbf{b}^{(1)} &= f[W_{BA}\,\mathbf{a}^{(1)}] \end{aligned}$$

For the same noisy input, BAM requires two iterations to restore the stored pattern.

Hence the pattern recall is superior in a tridirectional associative memory compared to bidirectional associative memory.

Solution to problem 4:

The cost function for $\lambda \neq 0$ is given by (Eq.7.21)

$$E(F) = \frac{1}{2}\sum_{l=1}^{L}\left(d_l - \sum_{i=1}^{H} w_i \phi(\mathbf{a}_l;\boldsymbol{\mu}_i)\right)^2 + \frac{1}{2}\lambda \parallel \mathbf{P}F(\mathbf{a}) \parallel^2 \qquad (1)$$

The first term on RHS of Equation (1) may be expressed as squared Euclidean norm $\| \mathbf{d}-\Phi\mathbf{w} \|^2$, where the vector $\mathbf{d}$ and matrix Φ are given by

$$\begin{aligned}
\mathbf{d} &= [\ d_1 \quad d_2 \quad \cdots \quad d_L\ ]^T, \text{ and} \\
\Phi &= \begin{bmatrix}
\phi(\mathbf{a}_1;\boldsymbol{\mu}_1) & \phi(\mathbf{a}_1;\boldsymbol{\mu}_2) & \cdots & \phi(\mathbf{a}_1;\boldsymbol{\mu}_H) \\
\phi(\mathbf{a}_2;\boldsymbol{\mu}_1) & \phi(\mathbf{a}_2;\boldsymbol{\mu}_2) & \cdots & \phi(\mathbf{a}_2;\boldsymbol{\mu}_H) \\
\vdots & \vdots & \ddots & \vdots \\
\phi(\mathbf{a}_L;\boldsymbol{\mu}_1) & \phi(\mathbf{a}_L;\boldsymbol{\mu}_2) & \cdots & \phi(\mathbf{a}_L;\boldsymbol{\mu}_H)
\end{bmatrix}
\end{aligned}$$

Since $F(\mathbf{a}) = \sum_{i=1}^{H} w_i\, \phi(\mathbf{a};\boldsymbol{\mu}_i)$, the approximating $\mathbf{F}$ is a linear combination of the Green's function for the linear differential operator $\mathbf{P}$. Therefore, we may express the second term on the RHS of Equation (1) as (See Haykin, p 257),

$$\| \mathbf{P}F(\mathbf{a}) \|^2 = (\mathbf{PF}, \mathbf{PF})_H = (F, \mathbf{P}^*\mathbf{P}F)_H,$$

where $(\cdot,\cdot)_H$ refers to the inner product in Hilbert space, and $\mathbf{P}^*$ is the adjoint of the differential operator $\mathbf{P}$.

$$\begin{aligned}
\text{Therefore, } \| \mathbf{P}F(\mathbf{a}) \|^2 &= \left[\sum_{i=1}^{H} w_i\phi(\mathbf{a};\boldsymbol{\mu}_i), \mathbf{P}^*\mathbf{P}\sum_{j=1}^{H} w_j\phi(\mathbf{a};\boldsymbol{\mu}_j)\right]_H \\
&= \left[\sum_{i=1}^{H} w_i\phi(\mathbf{a};\boldsymbol{\mu}_i), \sum_{j=1}^{H} w_j\mathbf{P}^*\mathbf{P}\phi(\mathbf{a};\boldsymbol{\mu}_j)\right]_H \\
&= \left[\sum_{i=1}^{H} w_i\phi(\mathbf{a};\boldsymbol{\mu}_i), \sum_{j=1}^{H} w_j\delta(\mathbf{a};\boldsymbol{\mu}_j)\right]_H \text{ (from Eq.7.27)} \\
&= \sum_{j=1}^{H}\sum_{i=1}^{H} w_j w_i \phi(\boldsymbol{\mu}_j, \boldsymbol{\mu}_i) \\
&= \mathbf{w}^T\Phi_o\mathbf{w}
\end{aligned}$$

where $\mathbf{w}$ is the weight vector of the output layer, and Φ_o is defined by

$$\Phi_o = \begin{bmatrix}
\phi(\boldsymbol{\mu}_1;\boldsymbol{\mu}_1) & \phi(\boldsymbol{\mu}_1;\boldsymbol{\mu}_2) & \cdots & \phi(\boldsymbol{\mu}_1;\boldsymbol{\mu}_H) \\
\phi(\boldsymbol{\mu}_2;\boldsymbol{\mu}_1) & \phi(\boldsymbol{\mu}_2;\boldsymbol{\mu}_2) & \cdots & \phi(\boldsymbol{\mu}_2;\boldsymbol{\mu}_H) \\
\vdots & \vdots & \vdots & \vdots \\
\phi(\boldsymbol{\mu}_L;\boldsymbol{\mu}_1) & \phi(\boldsymbol{\mu}_L;\boldsymbol{\mu}_2) & \cdots & \phi(\boldsymbol{\mu}_L;\boldsymbol{\mu}_H)
\end{bmatrix}$$

Finding the function F, that minimizes the cost function of Equation (1), is equivalent to finding the weight vector $\mathbf{w} \in \mathcal{R}^{\mathcal{M}}$ that minimizes the function,

$$E(\mathbf{w}) = E_1(\mathbf{w}) + \lambda E_2(\mathbf{w})$$

where λ is a regularization parameter and

$$E_1(\mathbf{w}) = \frac{1}{2} \parallel \mathbf{d} - \Phi\mathbf{w} \parallel^2 = \frac{1}{2}(\mathbf{d} - \Phi\mathbf{w})^T(\mathbf{d} - \Phi\mathbf{w}) \quad (2)$$

$$E_2(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi_o\mathbf{w} \quad (3)$$

A necessary condition for an extremum at $\mathbf{w}$ is that the Frechet differential (See Haykin, 1994, p.247) of $E(\mathbf{w})$ satisfies the condition

$$dE(\mathbf{w}, \mathbf{h}) = dE_1(\mathbf{w}, \mathbf{h}) + \lambda\, dE_2(\mathbf{w}, \mathbf{h}) = 0 \quad (4)$$

where $dE(\mathbf{w}, h) = \dfrac{d}{d\beta}E(\mathbf{w} + \beta\mathbf{h})|_{\beta=0}$, for any vector $\mathbf{h}$
From Equations (2) & (3) we find that

$$dE_1(\mathbf{w}, \mathbf{h}) = -\mathbf{h}^T\Phi^T(\mathbf{d} - \Phi\mathbf{w}) \quad (5)$$

$$dE_2(\mathbf{w}, \mathbf{h}) = \mathbf{h}^T\Phi_o\mathbf{w} \quad (6)$$

Substituting Equations(5) and (6) in Equation (4)

$$-\mathbf{h}^T\Phi^T(\mathbf{d} - \Phi\mathbf{w}) + \lambda\mathbf{h}^T\Phi_o\mathbf{w} = 0$$

For this to be true for all $\mathbf{h}$, $\Phi^T(\mathbf{d} - \Phi\mathbf{w}) + \lambda\Phi_o\mathbf{w} = 0$. Hence,

$$\mathbf{w} = (\Phi^T\Phi + \lambda\Phi_o)^{-1}\Phi^T\mathbf{d}$$

Solution to problem 5:

In this problem we explore the operation of a MLFNN with one hidden layer, trained using backpropagation algorithm, for the following functions:

(a) $f(x) = logx$, for $1 \leq x \leq 10$

(b) $f(x) = e^{-x}$, for $1 \leq x \leq 10$

The network is trained with $\eta = 0.3$ and α=0.7, using 5000 epochs each epoch consisting of 75 points in the chosen interval. Ten different network configurations are trained to learn the mapping. The networks are tested with 25 points. The results for both the problems are shown in the following table.

No. of hidden units	Average % error at the network output	
	$\log x$	$\exp(-x)$
2	0.2711	0.8641
3	0.0902	0.1918
4	0.2796	0.2645
5	0.2570	0.2222
7	0.2269	0.1666
10	0.1626	0.2759
15	0.2558	0.1565
20	0.1997	0.0736
30	0.4010	0.1360
100	0.4112	0.2764

The Matlab program used for this problem is:

```
function[]=prob7_5()
%Replace the function log10 with exponential function
%        for solving next part of the problem
index=0;
for i=1:.2:10
index=index+1;
TrainInputValue(index,1)=i;
TrainOutputValue(index,1)=log10(i);
end
index=0;
for i=1:.1:10
index=index+1;
TestInputValue(index,1)=i;
TestOutputValue(index,1)=log10(i);
end
MinValue=min(TrainInputValue);
MaxValue=max(TrainInputValue);
HID=[2 4 5 8 10 15 25 30 50 100];
```

```
for i=1:10
    hidnodes=HID(i);
    net=newff([MinValue MaxValue],[hidnodes 1],{'logsig','logsig'});
    net.trainParam.lr=0.3;
    net.trainParam.mc=0.7;
    net.trainParam.epochs=1000;
    net.trainParam.goal=0.001;
    net=train(net,TrainInputValue',TrainOutputValue');
    ObtOutput=sim(net,TestInputValue');
end
```

Solution to problem 6:

In this problem, we explore the performance of ART network for clustering 5-bit binary vectors. Consider the ART network shown in Fig.7.10. The results obtained for difference vigilance parameter (ρ) values are shown below:

$\rho = 0$ and $\rho = 1$ are the two extreme cases of this problem. In the former case all the 5-bit binary vectors are grouped under one cluster (having maximum Hamming distance of 5), and in the latter case each 5-bit binary vector form its own group (having maximum Hamming distance of 0, since only one vector is present in every group).

For $\rho = 0.5$, the vectors are grouped into 10 clusters. They are

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

For $\rho = 0.1$, the vectors are grouped into 5 clusters. They are

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The Matlab program used to implement this problem is given below. The vigilance parameter has to be passed as the parameter to obtain output.

```
function[ ]=prob7_6(VigilanceParameter)
%Provide the VigilanceParameter in the range of 0 to 1
index=0;
for one=0:1
for two=0:1
for three=0:1
for four=0:1
for five=0:1
    if index~=0
```

```
        data(index,:)=[one two three four five];
    end
    index=index+1;
end
end
end
end
end
weights_w=rand(5,5);
weights_v=ones(5,5);
epochs=1000;
for i=1:epochs
    for j=1:31
        F2_winner_matrix=weights_w'*data(j,:)';
        WinningPrototype=F2_winner_matrix;
        flag=1;
        index=0;
        while flag == 1
              [temp_value F2_winner]=max(F2_winner_matrix);
              SimilarityMeasure=weights_v(F2_winner,:)/sum(data(j,:));
              if SimilarityMeasure >= VigilanceParameter
                 for m=1:5
                     weights_v(F2_winner,m)=weights_v(F2_winner,m)*data(j,m);
                 end
                 for k=1:5
                     weights_w(:,F2_winner)=(weights_v(F2_winner,k)./ ...
                                     (.5+sum(weights_v(F2_winner,:))))';
                 end
                 flag=0;
             else
                 if index>5
                    weights_k(F2_winner,:)=data(j,:);
                    flag=0;
```

```
                end
                index=index+1;
                F2_winner_matrix(F2_winner)=min(F2_winner_matrix)-10;
            end
        end
    end
end
for i=1:31
    pattern1=data(i,:)';
    pattern1=weights_w'*pattern1;
    pattern1=weights_v'*pattern1;
end
```

Solution to problem 7:

The input data is generated to study the performance of RBFNN for the two class problem from the two Gaussian distributions with specified means and variances given in problem 4.12. Percentage success of RBFNN is given in the following table for different values of λ and for different number (H) of clusters.

$\lambda \rightarrow$	0.01	0.1	100
H$\downarrow$			
10	68	60	60
20	69	46	60

The percentage of success is higher when λ is small, and the number of hidden nodes is large. The Matlab program used to implement this problem is given below:

```
function[ ] = prob7_7()
%The following command is to load the input values generated as per
%              the conditions given in question
load prob7.mat
HiddenNodes=[10 20];
Lambda=[0.01 1 100];
```

```
for loop1=1:2
for loop2=1:3
RdmPts=round((rand(HiddenNodes(loop1),1)*99)+1);
for i=1:HiddenNodes(loop1)
    centers(i,:)=Input1(RdmPts(i),:);
end
MaxDist=0;
for i=1:HiddenNodes(loop1)
    for j=1:HiddenNodes(loop1)
        Phio(i,j)=norm(centers(i,:)-centers(j,:));
        if Phio(i,j) > MaxDist
           MaxDist = Phio(i,j);
        end
    end
end
GaussVariance = MaxDist*MaxDist/HiddenNodes(loop1);
[r c]=size(Input1);
for i=1:r
    for j=1:HiddenNodes(loop1)
        Phi(i,j)=exp(-norm(Input1(i,:)-centers(j,:))/GaussVariance);
    end
end
for i=1:r
    if DesOutput1(i,1) == 1
       DesiredOutput(i,:) = [1 -1];
    else
       DesiredOutput(i,:) = [-1 1];
    end
end
Weights=pinv((Phi' * Phi) + (Lambda(loop2) * Phio))*Phi'*DesiredOutput;
% Testing the RBF network
Success=0;
for i=1:r
```

```
    PresentInput=Input1(i,:);
    for j=1:HiddenNodes(loop1)
        HiddenOutput(j,1)=exp(-norm(PresentInput-centers(j,:))/GaussVariance);
    end
    PresentOutput(i,:)=HiddenOutput'*Weights;
    if (PresentOutput(i,1) > PresentOutput(i,2)) & (DesOutput1(i) == 1)
       Success=Success+1;
    elseif (PresentOutput(i,1) < PresentOutput(i,2)) & (DesOutput1(i) == 0)
       Success=Success+1;
    end
end
end %End of loop1
end %End of loop2
```

Solution to problem 8:

(a) Feedforward Neural Network (Pattern Association)

Initial weights and bias values are:

$$W = \begin{bmatrix} -0.5517 & 3.4792 & 3.3978 & 3.3342 & -1.9634 & -1.9820 & -1.2222 & -0.0346 \\ 3.2045 & 0.1867 & -3.5647 & 0.0209 & -2.3031 & 0.3093 & 2.6716 & 2.9667 \\ -0.2898 & -2.5340 & 1.5448 & 1.7851 & -2.6125 & -2.9752 & 3.0138 & 2.7408 \\ -1.3631 & 2.8844 & -2.0195 & -1.1915 & 3.0534 & 3.3161 & 1.5678 & 2.4282 \end{bmatrix}$$

$$\boldsymbol{\theta} = \begin{bmatrix} 1.1012 & -2.8558 & -1.4464 & -7.6677 \end{bmatrix}^T$$

Final weights and bias values are:

$$W = \begin{bmatrix} 1.7780 & 3.5121 & 5.7275 & 3.3672 & 0.3334 & -1.9491 & 1.1075 & -0.0017 \\ 5.4509 & -6.6443 & -1.3183 & -6.8101 & 6.7744 & -6.5217 & 4.9181 & -3.8643 \\ 2.1781 & -2.1934 & 4.0127 & 2.1256 & -0.4852 & -2.6346 & 5.4816 & 3.0814 \\ 1.6128 & -2.8357 & 0.9565 & -6.9116 & 11.7494 & -2.4040 & 4.5437 & -3.2919 \end{bmatrix}$$

$$\boldsymbol{\theta} = \begin{bmatrix} 3.4310 & -0.6094 & 1.0214 & -4.6917 \end{bmatrix}^T$$

Retrieved output for the input [1 1 0 1 1 0 0 1] is $\begin{bmatrix} 1.0000 & 0.0033 & 0.9968 & 0.0125 \end{bmatrix}^T$

(b) Feedback Neural Network (Pattern Storage)

The weight matrix of the network is given by,

$$W = \begin{bmatrix} 2 & 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 & -2 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 & -2 & 2 & 0 & 2 \\ 0 & -2 & 0 & -2 & 2 & -2 & 0 & -2 \\ 0 & 2 & 0 & 2 & -2 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 & -2 & 2 & 0 & 2 \end{bmatrix}$$

$$\boldsymbol{\theta} = \begin{bmatrix} -3 & -3 & -3 & -3 & 3 & -3 & -3 & -3 \end{bmatrix}^T$$

Retrieved output for the input $\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}^T$ is $\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$

(c) Hamming Network (Pattern Storage)

Weights and threshold values are:

$$W = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 & -0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & 0.5 & -0.5 & 0.5 & -0.5 & 0.5 & -0.5 \end{bmatrix}$$

and

$$\boldsymbol{\theta} = \begin{bmatrix} 4 & 4 \end{bmatrix}^T$$

The network output for the test pattern $\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$ is

$$\begin{bmatrix} 0.3985 \\ 0 \end{bmatrix}$$

which signifies that the pattern $\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$ is nearer to the given test pattern.

(d) Bidirectional Associative Memory

Weight matrix from the first layer to the second layer is given by,

$$W_{AB} = \begin{bmatrix} 2 & 0 & 2 & 0 \\ 0 & -2 & 0 & -2 \\ 2 & 0 & 2 & 0 \\ 0 & -2 & 0 & -2 \\ 0 & 2 & 0 & 2 \\ 0 & -2 & 0 & -2 \\ 2 & 0 & 2 & 0 \\ 0 & -2 & 0 & -2 \end{bmatrix}$$

Weight matrix from the second layer to the first layer is given by,

$$W_{BA} = W_{AB}^T = \begin{bmatrix} 2 & 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & 0 & -2 & 2 & -2 & 0 & -2 \\ 2 & 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & 0 & -2 & 2 & -2 & 0 & -2 \end{bmatrix}$$

Patterns retrieved for the given test pattern $\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$

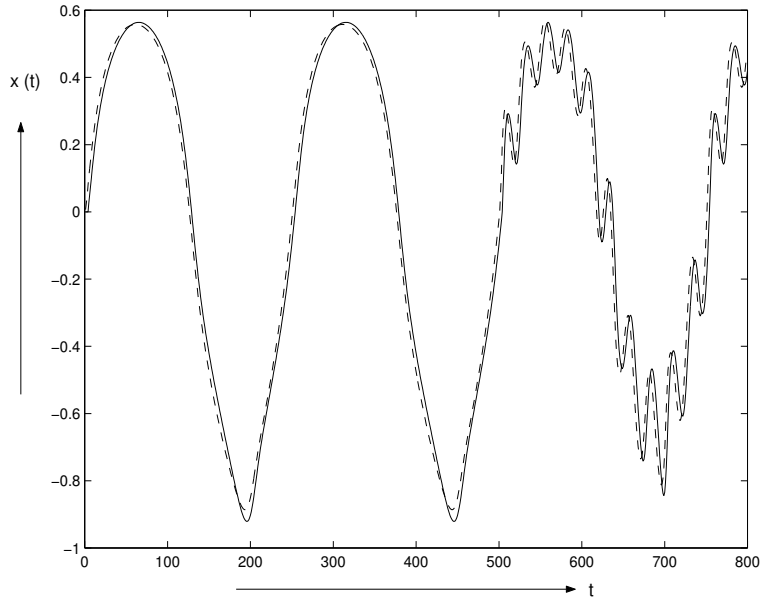
in the first layer is $\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}^T$,

in the second layer is $\begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}^T$

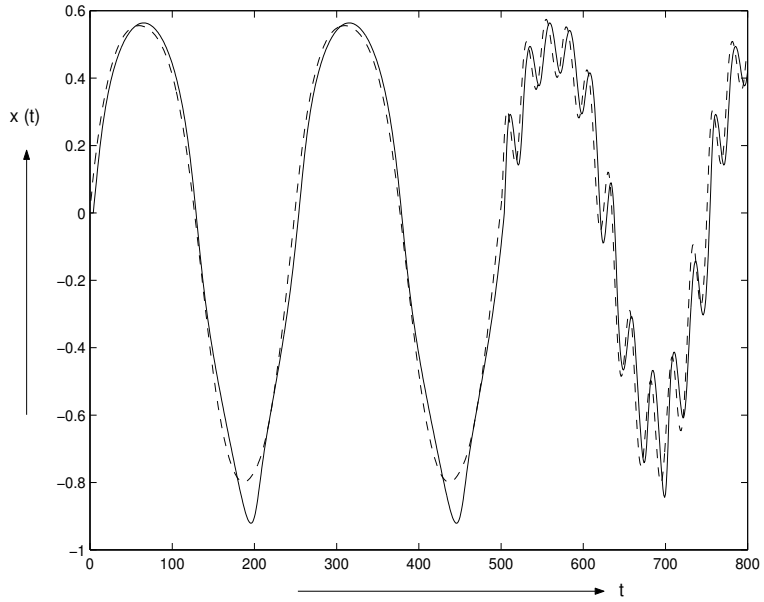
Solution to problem 9:

A MLFFNN is trained (using the Matlab function *newff* and *train*) with $x(t)$,$x(t\text{-}1)$,$x(t\text{-}2)$,$u(t)$, and $u(t\text{-}1)$ as inputs, where $u(t)$ is generated using samples uniformly distributed in the range [-1,+1]. In the figure shown below, solid line corresponds to the calculated output $x(t)$ and dotted line corresponds to the neural network output for the signal input given in the question. The figure shows that the neural network output closely follows the actual output for small η values. Hence it captures the nonlinear dynamical system better with small η values.

(a) For $\eta = 0.1$



(b) For $\eta = 0.3$



CHAPTER 8

Solution to problem 1:

For the Hamming net shown in Fig.8.3, the input to the unit in the upper subnet is given by

$$x_i = \sum_{j=1}^{M} w_{ij} a_{lj} + \frac{M}{2} \tag{1}$$

where the weights are given by $\quad w_{ij} = \frac{a_{ij}}{2}, \quad 1 < j \leq M, \quad 1 < i \leq N$

Substituting in Equation (1)

$$x_i = \frac{1}{2} \sum_{j=1}^{M} a_{ij} a_{lj} + \frac{M}{2} \tag{2}$$

The product $\sum_{j=1}^{M} a_{ij} a_{lj}$ of two bipolar binary vectors can be written as the total number of positions in which the two vectors agree minus the number of positions in which they differ (see Zurada, 1992, p. 392). Note that the number of different bit positions is the HD (Hamming Distance). Also, the number of positions in which two vectors agree is M-HD. Therefore, the equality is written as

$$\begin{aligned} \sum_{j=1}^{M} a_{ij} a_{lj} &= [M - HD(\boldsymbol{a}_l, \boldsymbol{a}_i)] - HD(\boldsymbol{a}_l, \boldsymbol{a}_i) \\ \Rightarrow \frac{1}{2} \sum_{j=1}^{M} a_{ij} a_{lj} &= \frac{M}{2} - HD(\boldsymbol{a}_l, \boldsymbol{a}_i) \end{aligned} \tag{3}$$

Substituting Equation (3) in Equation (2),

$$x_i = M - HD(\boldsymbol{a}_l, \boldsymbol{a}_i) \tag{4}$$

Solution to problem 2:

(a) Finding the hamming distance:

The three prototype vectors are given by

$\boldsymbol{a}_1 = [\ 1 \quad -1 \quad 1 \quad -1 \quad 1 \quad -1\ ]^T$

$\boldsymbol{a}_2 = [\ -1 \quad -1 \quad -1 \quad -1 \quad -1 \quad 1\ ]^T$

$\boldsymbol{a}_3 = [\ 1 \quad 1 \quad 1 \quad -1 \quad -1 \quad 1\ ]^T$

The weight matrix W_H of the Hamming network can be obtained as:

$$W_H = \frac{1}{2}\begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix}$$

The input to the units in the upper subnet is given by

$$\mathbf{net} = \frac{1}{2}\begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix} \boldsymbol{a}_l + \begin{bmatrix} 6/2 \\ 6/2 \\ 6/2 \end{bmatrix}$$

For the input vector $\boldsymbol{a}_1 = [\ 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\ ]^T$ the output of each unit is calculated as follows:

$$\mathbf{net} = \frac{\mathbf{1}}{\mathbf{2}}\begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}$$

$$\mathbf{net} = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}$$

Now computing the Hamming distance using Equation (4)

$$HD_c = \begin{bmatrix} 6 \\ 6 \\ 6 \end{bmatrix} - \mathbf{net} = \begin{bmatrix} 6 \\ 6 \\ 6 \end{bmatrix} - \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 2 \end{bmatrix}$$

Actual Hamming distance $HD_a = \begin{bmatrix} 3 \\ 5 \\ 2 \end{bmatrix}$

Therefore actual Hamming distance is equal to the computed Hamming distance.

(b) For finding the steady state output of upper subnet of the Hamming network:

Weights of upper subnet can be obtained by Eqn.(8.2) choosing ϵ=0.2

Therefore, $V = \begin{bmatrix} 1 & -0.2 & -0.2 \\ -0.2 & 1 & -0.2 \\ -0.2 & -0.2 & 1 \end{bmatrix}$

The input to the upper subnet is the linearly scaled matching scores of the lower subnet, so that the inputs in the range of 0 to n are scaled to the range 0 to 1. Hence the highest steady state output corresponds to the class number to which $\mathbf{a}_l$ is at the smallest HD. This scaling operation is indicated by $\Gamma(\cdot)$. In this case the scaling operation involves division by 6.

$$\mathbf{s}(0) = \mathbf{f}(V\ \mathbf{\Gamma}(\mathbf{net})),\ \mathbf{f}(\mathbf{s}) = \begin{bmatrix} f(s_1) \\ f(s_2) \\ f(s_3) \end{bmatrix},\ \text{where } f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

The recursive formula for computation of $\mathbf{s}(t)$ is

$$\mathbf{s}(t+1) \ = \ \mathbf{f}(V\ \mathbf{s}(t))$$

Step 1:

$$\mathbf{s}(0) \ = \ \mathbf{f}\left(\begin{bmatrix} 1 & -0.2 & -0.2 \\ -0.2 & 1 & -0.2 \\ -0.2 & -0.2 & 1 \end{bmatrix}\begin{bmatrix} 3/6 \\ 1/6 \\ 4/6 \end{bmatrix}\right) = \mathbf{f}\left(\begin{bmatrix} 0.3333 \\ -0.0667 \\ 0.5533 \end{bmatrix}\right)$$

$$\Rightarrow\ \mathbf{s}(0) \ = \ \begin{bmatrix} 0.3333 \\ 0 \\ 0.5533 \end{bmatrix}$$

Step 2:

$$\mathbf{s}(1) \ = \ \mathbf{f}\left(\begin{bmatrix} 1 & -0.2 & -0.2 \\ -0.2 & 1 & -0.2 \\ -0.2 & -0.2 & 1 \end{bmatrix}\begin{bmatrix} 0.3333 \\ 0 \\ 0.5533 \end{bmatrix}\right) = \mathbf{f}\left(\begin{bmatrix} 0.2266 \\ -0.1733 \\ 0.4666 \end{bmatrix}\right)$$

$$\Rightarrow\ \mathbf{s}(1) \ = \ \begin{bmatrix} 0.2266 \\ 0 \\ 0.4666 \end{bmatrix}$$

Step 3:

$$\mathbf{s}(2) = \mathbf{f}\left(\begin{bmatrix} 1 & -0.2 & -0.2 \\ -0.2 & 1 & -0.2 \\ -0.2 & -0.2 & 1 \end{bmatrix}\begin{bmatrix} 0.2266 \\ 0 \\ 0.4666 \end{bmatrix}\right) = \mathbf{f}\left(\begin{bmatrix} 0.1333 \\ -0.1386 \\ 0.4213 \end{bmatrix}\right)$$

$$\Rightarrow \mathbf{s}(2) = \begin{bmatrix} 0.1333 \\ 0 \\ 0.4213 \end{bmatrix}$$

Step 4:

$$\mathbf{s}(3) = \mathbf{f}\left(\begin{bmatrix} 1 & -0.2 & -0.2 \\ -0.2 & 1 & -0.2 \\ -0.2 & -0.2 & 1 \end{bmatrix}\begin{bmatrix} 0.1333 \\ 0 \\ 0.4213 \end{bmatrix}\right) = \mathbf{f}\left(\begin{bmatrix} -0.0299 \\ -0.087 \\ 0.3848 \end{bmatrix}\right)$$

$$\Rightarrow \mathbf{s}(3) = \begin{bmatrix} 0 \\ 0 \\ 0.3848 \end{bmatrix}$$

This terminates the recursion, since for further iterations $\mathbf{s}(t+1) = \mathbf{s}(t)$, for $t > 3$. The result computed after 4 iterations indicates that the vector $\mathbf{a}_l$ is closest to the input vector $\mathbf{a}_3$.

Solution to problem 3:

The capacity of a network is the number of distinct pattern vectors (M) that can be stored in a N-unit network.

The number of connections increase linearly for the Hamming network with the number of bits in the input pattern increases. On the other hand, for the Hopfield Network the number of connections increase quadratically with the number of bits in the input pattern. Thus the capacity of the Hamming network to store the patterns is higher than that of the Hopfield network.

Solution to problem 4:

Graph Bipartition Problem

Weights matrix can be found using the Eq.(8.18). Using the weight matrix, the mean-field approximation will give the desired result. The following code is executed to solve the given problem.

```
function[ ] = prob8_4()
```

```
Weight = [1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 ; ...
          1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 ; ...
          0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 ; ...
          0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 ; ...
          1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 ; ...
          1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 ; ...
          0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 ; ...
          0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 ; ...
          0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 ; ...
          0 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 ; ...
          0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 ; ...
          0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 ; ...
          0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 ; ...
          0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 ; ...
          0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 ; ...
          0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 ];
alpha = .4;
Weight = Weight - alpha;
bias = 16 * alpha / 2;
avg_si1 = (rand(16,1)*2) - 1;
for T = 1 : -.1 : .1
    temp = zeros(16,1);
    while norm(temp-avg_si1) ~= 0
        temp = avg_si1;
        avg_si1 = (Weight*avg_si1)/T;
        avg_si1 = tanh(avg_si1);
end
end
```

The output of the program is

$$\begin{bmatrix} -1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 \end{bmatrix}^T$$

which shows that nodes 3,9,10,11,12,13,14 falls in one partition and remaining nodes falls in other partition.

Solution to problem 5:

Traveling Salesman Problem

Weight matrix can be found using the Eq.(8.24). Using the weight matrix, the following code is executed to solve the problem for some typical distances between the cities and parameters.

```
function[ ]=prob8_5()
N=10; Points=[1 1;3 3;0 4;2 6;1 2;3 5;2 6;1 4;2 5;4 5];
for i=1:N
    for j=1:N
        distance(i,j)=norm(Points(i,:)-Points(j,:));
    end
end
alpha=2.7; beta =0.3; gamma=3.7; indexRow=0;
for i=1:N
    for a=1:N
        indexRow=indexRow+1; indexColumn=0;
        for j=1:N
            for b=1:N
                indexColumn=indexColumn+1;
                Term1 = distance(a,b)*(1-delta(a,b))*(delta(i-1,j)+delta(i+1,j));
                Term2 = alpha*(1-delta(a,b))*delta(i,j);
                Term3 = beta*(1-delta(i,j))*delta(a,b);
                Weights(indexRow,indexColumn) =Term1+Term2+Term3;
            end
        end
    end
end
Threshold=gamma*N;
% Taking a random state
State=rand(N*N,1);
TempState=zeros(N*N,1);
Temperature=1;
while Temperature > 0
```

```
    Temperature=Temperature-.001;
    index=0;
    while(norm(TempState-State) ~= 0)
        TempState=State;
        State=(Weights*State-Threshold);
        State=squash(State,Temperature);
        index=index+1;
    end
end
```

The sample output of the program for $\alpha = 2.7, \beta = 0.3$ and $\gamma = 3.7$ is

$$\begin{bmatrix}
0.8223 & 0.9707 & 0.9197 & 0.9830 & 0.8419 & 0.8231 & 0.8608 & 0.8432 & 0.8604 & 0.8223 \\
0.8223 & 0.9095 & 0.9112 & 0.9211 & 0.8341 & 0.8227 & 0.8454 & 0.8428 & 0.8450 & 0.8223 \\
0.8223 & 0.9038 & 0.9106 & 0.9153 & 0.8336 & 0.8228 & 0.8444 & 0.8429 & 0.8440 & 0.8223 \\
0.8270 & 0.8310 & 0.8757 & 0.8365 & 0.8278 & 0.8272 & 0.8290 & 0.8311 & 0.8287 & 0.8270 \\
0.8223 & 0.9405 & 0.9156 & 0.9525 & 0.8382 & 0.8229 & 0.8534 & 0.8430 & 0.8530 & 0.8223 \\
0.8226 & 0.8598 & 0.9017 & 0.8705 & 0.8279 & 0.8228 & 0.8329 & 0.8421 & 0.8325 & 0.8226 \\
0.8280 & 0.8319 & 0.8542 & 0.8368 & 0.8287 & 0.8281 & 0.8299 & 0.8450 & 0.8296 & 0.8280 \\
0.8223 & 0.8857 & 0.9079 & 0.8969 & 0.8312 & 0.8227 & 0.8396 & 0.8428 & 0.8393 & 0.8223 \\
0.8224 & 0.8507 & 0.9021 & 0.8614 & 0.8265 & 0.8226 & 0.8306 & 0.8424 & 0.8302 & 0.8224 \\
0.8229 & 0.8815 & 0.9038 & 0.8917 & 0.8308 & 0.8231 & 0.8383 & 0.8418 & 0.8381 & 0.8229
\end{bmatrix}$$

Note that the output will differ if we run the program with the same parameters.

The output of the program for $\alpha = 3.0, \beta = 1.2$ and $\gamma = 4.2$ is

$$\begin{bmatrix}
0.5513 & 0.5785 & 0.5680 & 0.5790 & 0.5520 & 0.5514 & 0.5532 & 0.5525 & 0.5532 & 0.5513 \\
0.5513 & 0.5670 & 0.5677 & 0.5674 & 0.5517 & 0.5513 & 0.5524 & 0.5525 & 0.5523 & 0.5513 \\
0.5513 & 0.5678 & 0.5677 & 0.5682 & 0.5517 & 0.5514 & 0.5526 & 0.5525 & 0.5525 & 0.5513 \\
0.5524 & 0.5550 & 0.5655 & 0.5553 & 0.5525 & 0.5524 & 0.5528 & 0.5527 & 0.5527 & 0.5524 \\
0.5513 & 0.5734 & 0.5678 & 0.5739 & 0.5518 & 0.5514 & 0.5529 & 0.5525 & 0.5528 & 0.5513 \\
0.5514 & 0.5583 & 0.5672 & 0.5587 & 0.5516 & 0.5514 & 0.5519 & 0.5525 & 0.5519 & 0.5514 \\
0.5554 & 0.5581 & 0.5572 & 0.5582 & 0.5555 & 0.5555 & 0.5558 & 0.5565 & 0.5558 & 0.5554 \\
0.5513 & 0.5642 & 0.5676 & 0.5646 & 0.5516 & 0.5513 & 0.5523 & 0.5525 & 0.5522 & 0.5513 \\
0.5513 & 0.5577 & 0.5674 & 0.5581 & 0.5515 & 0.5513 & 0.5519 & 0.5525 & 0.5518 & 0.5513 \\
0.5528 & 0.5621 & 0.5643 & 0.5623 & 0.5530 & 0.5528 & 0.5534 & 0.5534 & 0.5534 & 0.5528
\end{bmatrix}$$

where each row represents a city and each column represents a stop in the tour.

The solution found by the network is not good. The system gets struck in a local minimum. The optimum solution for the traveling salesman problem depends critically on the choice of the parameters used for the constraint terms.

Solution to problem 6:

Image Smoothing Problem

Differentiating Eq.(8.26) with respect to v_i and equating to zero (see Hertz, 1991, p. 83), we get

$$\begin{aligned}
\frac{dE}{dv_i} &= \alpha(1-v_i)(s_{i+1}+s_{i-1}-2s_i)+\beta(d_i-s_i)+\gamma=0 \\
&\Rightarrow \alpha(1-v_i)s_{i+1}+\alpha(1-v_i)s_{i-1}-2\alpha(1-v_i)s_i+\beta d_i-\beta s_i+\gamma=0 \\
&\Rightarrow \frac{\alpha(1-v_i)s_{i+1}+\alpha(1-v_i)s_{i-1}+\beta d_i+\gamma}{\beta+2\alpha(1-v_i)}=s_i \qquad (1)
\end{aligned}$$

For the image having discontinuities, an additional unit (referred as HiddenNodes in the following program) is introduced between two adjacent pixel units (referred as VisibleNodes in the following program). These additional units will have bipolar values ± 1, so that $+1$ indicates the presence of discontinuity, and -1 indicates the absence of discontinuity. Present state of the node can be updated using Equation(1). Following program performs the state updation till it becomes stable.

```
function[]=prob8_6()
clear
format compact
alpha=5.5;
beta=.5;
gamma=2;

Pels = [3 4 5 7 9 1 2 4 6 7];
NormPels = Pels / max(Pels);
Pels = randn(19,1);

HiddenNodes = [2 4 6 8 10 12 14 16 18];
VisibleNodes = [1 3 5 7 9 11 13 15 17 19];

for i = 1 : 9
Pels(HiddenNodes(i)) = sign(Pels(HiddenNodes(i)));
```

```
end

for i = 1 : 10
Pels(VisibleNodes(i)) = NormPels(i);
end

OriginalPels = Pels;

TempState = Pels + 1;
index = 0;
rdm_picks = rdm_gen(19);
counter_rdm = 0;
flag = 0;

while norm(TempState([1 3 5 7 9 11 13 15 17 19])-Pels([1 3 5 7 9 11 13 15 17 19])) > 0.01
      | flag == 1

TempState = Pels;
counter_rdm = counter_rdm+1;
r = rdm_picks(counter_rdm);
if counter_rdm == 19
counter_rdm=0;
end
flag=0;
for i=1:9
if r == HiddenNodes(i)
flag=1;
break;
end
end
size(Pels);
% [index norm(TempState([1 3 5 7 9 11 13 15 17 19])-Pels([1 3 5 7 9 11 13 15 17 19])) r fl
if flag==0
```

```
if r~=19
Term1=alpha*(1-Pels(r+1))*Pels(r+2);
Term3=-2*alpha*(1-Pels(r+1))*Pels(r);
else
Term1=0;
Term3=0;
end

if r~=1 & r~=19
Term2=alpha*(1-Pels(r+1))*Pels(r-1);
else
Term2=0;
end
Term4=beta*Pels(r);
Pels(r)=tanh((Term1+Term2+Term3+Term4)/beta);
else
if power(Pels(r-1)-Pels(r+1),2)*alpha/4 > gamma
Pels(r)=1;
else
Pels(r)=-1;
end
end
index=index+1;
end
```

The output state of the program for the hidden nodes are

$$\begin{bmatrix} -1.0000 & -1.0000 & -1.0000 & -1.0000 & 1.0000 & -1.0000 & -1.0000 & -1.0000 & -1.0000 \end{bmatrix}^T$$

which shows that the discontinuity is in between 5th and 6th pixel.

Solution to problem 7:

Weighted Matching Problem

Suppose there are N (even) points in a 2-dimensional space with known distances between each pair of points. Let d_{ij} be the distance between the points i and j. Let n_{ij} be a unit in a Hopfield

network, such that the state $n_{ij} = 1$ indicates that the points are linked, and the state $n_{ij} = 0$ indicates that the points are not linked. The optimization problem involves minimizing the total length of links $L = \sum_{i<j} d_{ij} n_{ij}$ subjected to the constraint that $\sum_j n_{ij} = 1$ for all i. Assuming $n_{ij} = n_{ji}$ and $n_{ii} = 0$, the cost function (see Hertz, 1991, p. 72) to be minimized for maximizing the total length of the links is given by

$$\begin{aligned} H_{ij} &= \sum_{i<j} d_{ij} n_{ij} + \frac{\gamma}{2} \sum_i \left(1 - \sum_k n_{ik}\right)^2 \\ \Rightarrow \Delta H_{ij} &= d_{ij} + \frac{\gamma}{2} \cdot 2 \left(1 - \sum_{k \neq j} n_{ik} - \sum_{k \neq i} n_{jk}\right) (-1) \Delta n_{ij} \\ \Rightarrow \Delta H_{ij} &= \left(d_{ij} - \gamma + \gamma \sum_{k \neq j} n_{ik} + \gamma \sum_{n \neq i} n_{jk}\right) \Delta n_{ij} \end{aligned} \tag{1}$$

Comparing Equation(1) with the Hopfield energy equation, we see that $d_{ij} - \gamma$ is the threshold for unit ij, and the remaining terms dictates the value of weight matrix, i.e., $w_{ij,kl} = -\gamma$, whenever ij has an index in common with kl; otherwise $w_{ij,kl} = 0$.

(a) Deterministic relaxation algorithm

This is the algorithm in which the network is run to determine the state corresponding to the global minimum of the energy function of the network using the dynamics of the network expressed as: Assuming bipolar states for each unit,

$$s_i(t+1) = \text{sgn} \left(\sum_{j \neq i} w_{ij} s_j(t)\right)$$

Following code is executed to solve the problem using deterministic relaxation algorithm.

```
function[]=prob8_7a()
%Test points are
Points=[1 1;3 3;0 4;2 6];
for i=1:4
    for j=1:4
        distance(i,j)=norm(Points(i,:)-Points(j,:));
    end
end
gamma=max(max(distance))/3;
% The nodes are 12 13 14 23 24 34
```

```
Weights=[   0    -gamma -gamma -gamma -gamma    0   ; ...
          -gamma     0   -gamma -gamma   0    -gamma ; ...
          -gamma  -gamma    0     0    -gamma -gamma ; ...
          -gamma  -gamma    0     0    -gamma -gamma ; ...
          -gamma     0   -gamma -gamma   0    -gamma ; ...
             0    -gamma -gamma -gamma -gamma    0   ];
Threshold=[distance(1,2) distance(1,3) distance(1,4)  ...
           distance(2,3) distance(2,4) distance(3,4)]';
Threshold=Threshold-gamma;
% Taking a random state
State=[0 1 1 0 0 1]';
TempState=[0 0 0 0 0 0]';
while(norm(TempState-State) ~= 0)
      TempState=State;
      State=(Weights*State+Threshold)
      State=sgn_m(State)
end
```

The final state of the network is given by

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

which shows points (1,4) and (2,3) can be linked so as to maximize the total length of the links for the given test points.

(b) Stochastic relaxation using simulated annealing

Deterministic relaxation algorithm may lead to a state corresponding to a local minimum of the energy function. In order to reach global minimum, the concept of stochastic unit is used in the activation dynamics of the network. For a stochastic unit the state of the unit is updated using a probability law, which is controlled by a temperature parameter (T). At higher temperature many states are likely to be visited, irrespective of the energies of those states. Thus the local minima of the energy function can be escaped. At $T = 0$, the state with the lowest energy having highest probability can be visited. Hence in simulated annealing, the temperature is gradually reduced so that the state corresponding to global minimum of energy function can be reached, escaping the local minima.

Following code is executed to solve the problem using stochastic relaxation using simulated annealing.

```
function[]=prob8_7b()
%Test points are ...
Points=[1 1;3 3;0 4;2 6];
for i=1:4
    for j=1:4
        distance(i,j)=norm(Points(i,:)-Points(j,:));
    end
end
gamma=max(max(distance))/3;
% The nodes are 12 13 14 23 24 34
Weights=[   0    -gamma -gamma -gamma -gamma    0   ; ...
         -gamma     0   -gamma -gamma   0    -gamma ; ...
         -gamma  -gamma    0     0    -gamma -gamma ; ...
         -gamma  -gamma    0     0    -gamma -gamma ; ...
         -gamma     0   -gamma -gamma   0    -gamma ; ...
            0    -gamma -gamma -gamma -gamma    0   ];
Threshold = [distance(1,2) distance(1,3) distance(1,4) ...
             distance(2,3) distance(2,4) distance(3,4)]';
Threshold=Threshold-gamma;
% Taking a random state
State=rand(1,6)
Temperature=1;
while Temperature>0.1
    PresentAverageState=rand(1,6);
    TempAverageState=rand(1,6);
    Temperature=Temperature-.01
    TempState=rand(1,6);
    while norm(PresentAverageState-TempAverageState) ~= 0
        TempAverageState=PresentAverageState;
        index=1;
        for i=1:100
```

```
            TempState=State;
            State=(Weights*State'+Threshold)';
            State=squash(State,Temperature);
            PresentIteration(index,:)=State;
            index=index+1;
        end
        PartitionFunction=0;
        for i=1:100
            energy=0;
            for j=1:6
                for k=1:6
                    energy=energy+Weights(j,k)*PresentIteration(i,j) ...
                                              *PresentIteration(i,k);
                end
            end
            energy=(-1/2)*energy;
            PartitionFunction=PartitionFunction+exp(-energy/Temperature);
            Energy(i)=energy;
        end
        PresentAverageState=zeros(1,6);
        for i=1:100
            Probability(i)=exp(-Energy(i)/Temperature)/PartitionFunction;
            PresentAverageState=PresentAverageState + ...
                                (PresentIteration(i,:)*Probability(i));
            end
        end
end
```

The final state of the network is given by

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

which shows points (1,4) and (2,3) can be linked so as to maximize the total length of the links for the given test points.

(c) Mean-field annealing

Due to the computation of stationary probabilities for each temperature, implementation of simulated annealing may take long time to come to equilibrium. In order to speed up the process of simulated annealing, the mean-field annealing is used, in which the stochastic update of the binary/bipolar units is replaced by deterministic analog states. Here we iteratively solve the mean-field equations for the averages $< s_i >$.

Following code is executed to solve the problem using mean-field annealing.

```
function[]=prob8_7c()
% Test points are ..
Points=[1 1;3 3;0 4;2 6];
for i=1:4
    for j=1:4
        distance(i,j)=norm(Points(i,:)-Points(j,:));
    end
end
gamma=max(max(distance))/6;
% The nodes are 12 13 14 23 24 34
Weights=[   0    -gamma -gamma -gamma -gamma    0   ; ...
         -gamma     0   -gamma -gamma   0    -gamma ; ...
         -gamma  -gamma    0     0    -gamma -gamma ; ...
         -gamma  -gamma    0     0    -gamma -gamma ; ...
         -gamma     0   -gamma -gamma   0    -gamma ; ...
            0    -gamma -gamma -gamma -gamma    0   ];
Threshold = [distance(1,2) distance(1,3) distance(1,4) ...
             distance(2,3) distance(2,4) distance(3,4)]';
Threshold=Threshold-gamma;
% Taking a random state
State=rand(6,1)
TempState=[0 0 0 0 0 0]';
Temperature=1;
while Temperature > 0
      Temperature=Temperature-.01
```

```
        while (norm(TempState-State) >= 0.00001)
              TempState=State;
              State=(Weights*State+Threshold);
              State=squash(State,Temperature);
        end
end
```

The final state of the network is given by

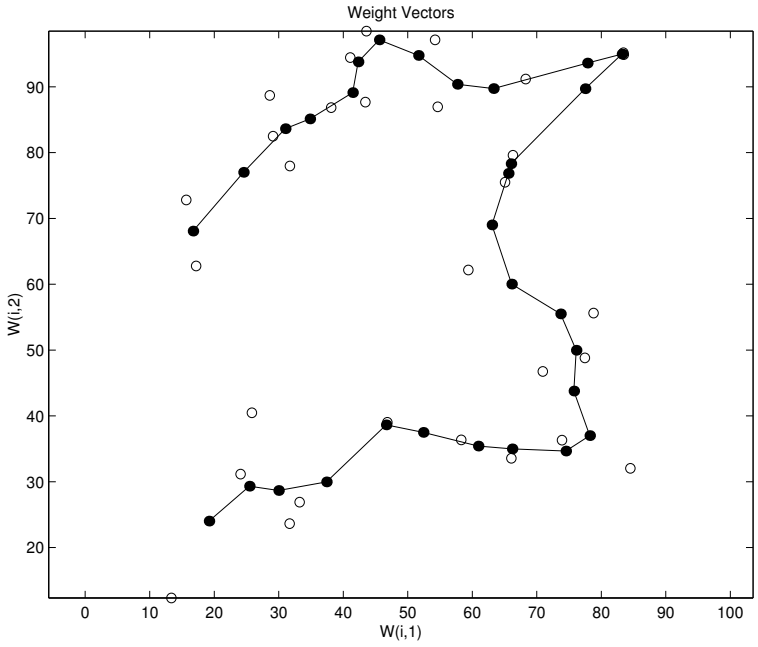
$$\begin{bmatrix} 0.4176 & 0.5634 & 0.9314 & 0.6573 & 0.5634 & 0.4176 \end{bmatrix}$$

which shows points (1,4) and (2,3) (having maximum values) can be linked so as to maximize the total length of the links for the given test points.

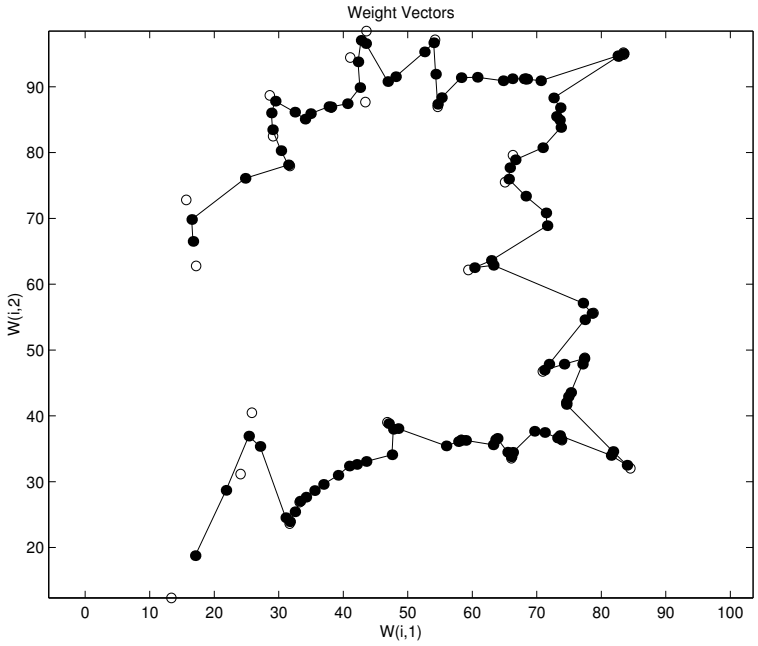
Solution to problem 8:

Traveling salesman problem is solved using SOM and the final output is plotted for the following different cases:

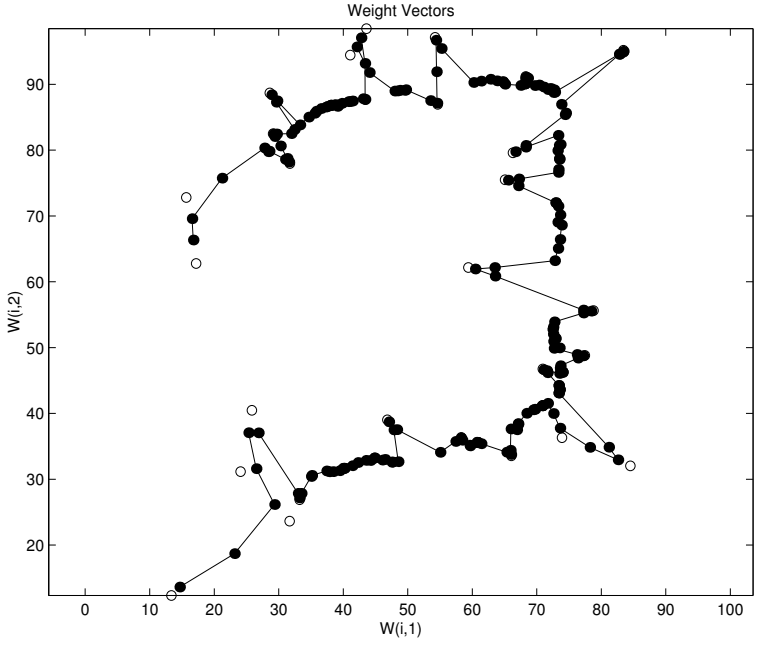
(a) 30 cities, 30 units



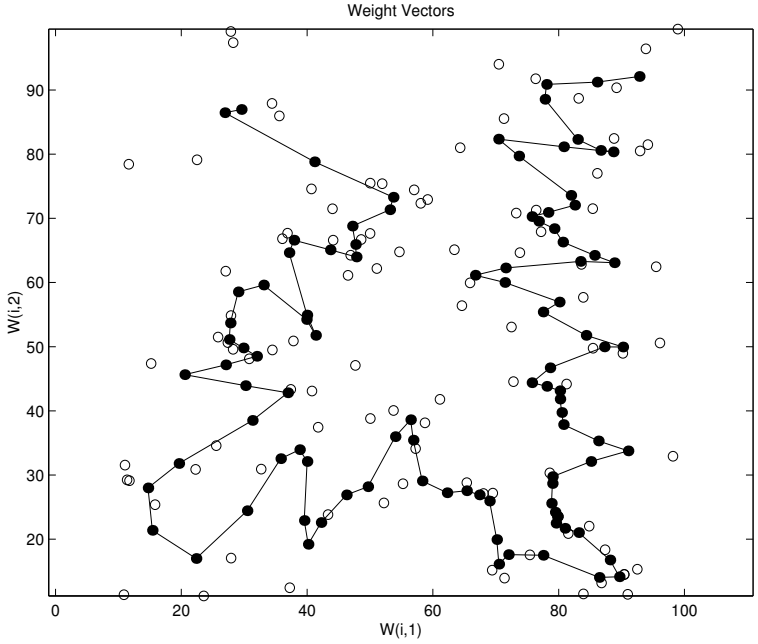
(b) 30 cities, 100 units



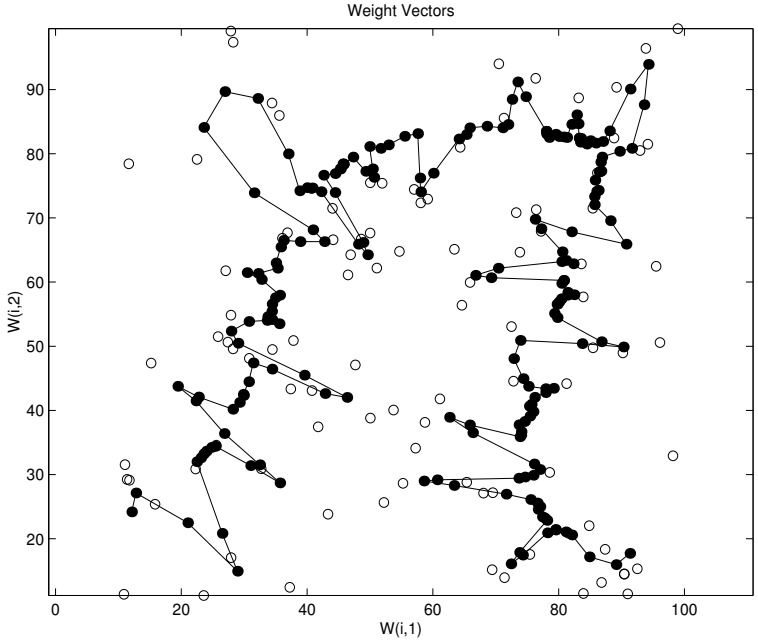
(c) 30 cities, 500 units



(d) 100 cities, 100 units



(e) 100 cities, 500 units



The Matlab program used for to implement this problem is:

```
function[ ]=prob8_8()
%X - coordinates
coords(:,1)=rand(30,1);
%Y - coordinates
coords(:,2)=rand(30,1);
figure
plot(coords(:,1),coords(:,2),'o');
net=newsom([0 1;0 1],[200]);
net.trainParam.epochs=1000;
net=train(net,coords');
hold on
plotsom(net.iw{1,1},net.layers{1}.distances);
hold off
```

Discussion: Traveling salesman problem is solved using SOM. The network is trained by presenting the coordinate value of each city as input. Normally the number of units (shaded circles) in SOM are made much larger than the number of cities(hollow circles). The results of SOM for different number of cities and units are shown in above figures. It shows that an approximate tour can be obtained by increasing the number of units.