

Scilab Manual for
Neural Networks
by Dr G V Maha Lakshmi
Electronics Engineering
Sreenidhi Institute Of Science And
Technology¹

Solutions provided by
Dr G V Maha Lakshmi
Electronics Engineering
Sreenidhi Institute Of Science And Technology

June 3, 2026

¹Funded by a grant from the National Mission on Education through ICT, <http://spoken-tutorial.org/NMEICT-Intro>. This Scilab Manual and Scilab codes written in it can be downloaded from the "Migrated Labs" section at the website <http://scilab.in>

Contents

List of Scilab Solutions	3
1 Generate AND NOT function using McCulloch-Pitts neural net	4
2 McCulloch-Pitts Net for XOR function	7
3 Hebb Net to classify two dimensional input patterns	11
4 Hetro associative neural net	13
5 Discrete Hopfield net	15
6 Kohonen self organizing maps	17
7 Learning Vector Quantisation	19
8 Full Counter Propagation Network for given input pair	22
9 ART1 Neural Net	24
10 MLP	26

List of Experiments

Solution 1.1	1	4
Solution 2.2	2	7
Solution 3.3	3	11
Solution 4.4	4	13
Solution 5.5	5	15
Solution 6.6	6	17
Solution 7.7	7	19
Solution 8.8	8	22
Solution 9.9	9	24
Solution 10.10	10	26

Experiment: 1

Generate AND NOT function using McCulloch-Pitts neural net

Scilab code Solution 1.1 1

```
1 //Generate AND NOT function using McCulloch-Pitts
   neural net
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 //Generate weights and threshold value
8 disp('Enter the weights');
9 w1=input('Weight w1=');
10 w2=input('Weight w2=');
11 disp('Enter Threshold Value');
12 theta=input('theta=');
13 y=[0 0 0 0];
14 x1=[0 0 1 1];
15 x2=[0 1 0 1];
16 z=[0 0 1 0];
```

```

17 con=1;
18 while con
19     zin=x1*w1+x2*w2;
20     for i=1:4
21         if zin(i)>=theta
22             y(i)=1;
23         else
24             y(i)=0;
25         end
26     end
27     disp('Output of Net');
28     disp(y);
29     if y==z
30         con=0;
31     else
32         disp('Net is not learning enter another set of
33             weights and Threshold value');
34         w1=input('weight w1=');
35         w2=input('weight w2=');
36         theta=input('theta=');
37     end
38     disp('Mcculloch-Pitts Net for ANDNOT function');
39     disp('Weights of Neuron');
40     disp(w1);
41     disp(w2);
42     disp('Threshold value');
43     disp(theta);
44
45 //Truth Table
46 //X1 X2 Y
47 //0 0 0
48 //0 1 0
49 //1 0 1
50 //1 1 0
51
52 //Output
53 //Enter the weights

```

```

54 //Weight w1=1
55 //Weight w2=1
56 //
57 // Enter Threshold Value
58 //theta=0.1
59 //
60 // Output of Net
61 //
62 //      0.      1.      1.      1.
63 //
64 // Net is not learning enter another set of weights
    and Threshold v
65 //      alue
66 //weight w1=1
67 //weight w2=-1
68 //theta=1
69 //
70 // Output of Net
71 //
72 //      0.      0.      1.      0.
73 //
74 // Mcculloch-Pitts Net for ANDNOT function
75 //
76 // Weights of Neuron
77 //
78 //      1.
79 //
80 //      - 1.
81 //
82 // Threshold value
83 //
84 //      1.

```

Experiment: 2

McCulloch-Pitts Net for XOR function

Scilab code Solution 2.2 2

```
1 //McCulloch-Pitts for XOR function
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 //Getting weights and threshold value
8 disp('Enter weights');
9 w11=input('Weight w11=');
10 w12=input('weight w12=');
11 w21=input('Weight w21=');
12 w22=input('weight w22=');
13 v1=input('weight v1=');
14 v2=input('weight v2=');
15 disp('Enter Threshold Value');
16 theta=input('theta=');
17 x1=[0 0 1 1];
18 x2=[0 1 0 1];
19 z=[0;1;1;0];
```

```

20 con=1;
21 while con
22     zin1=x1*w11+x2*w21;
23     zin2=x1*w21+x2*w22;
24     for i=1:4
25         if zin1(i)>=theta
26             y1(i)=1;
27         else
28             y1(i)=0;
29         end
30         if zin2(i)>=theta
31             y2(i)=1;
32         else
33             y2(i)=0;
34         end
35     end
36     yin=y1*v1+y2*v2;
37     for i=1:4
38         if yin(i)>=theta;
39             y(i)=1;
40         else
41             y(i)=0;
42         end
43     end
44     disp('Output of Net');
45     disp(y);
46     if y == z
47         con=0;
48     else
49         disp('Net is not learning enter another set of
50             weights and Threshold value');
51         w11=input('Weight w11=');
52         w12=input('weight w12=');
53         w21=input('Weight w21=');
54         w22=input('weight w22=');
55         v1=input('weight v1=');
56         v2=input('weight v2=');
57         theta=input('theta=');

```

```

57     end
58 end
59 disp('McCulloch-Pitts Net for XOR function');
60 disp('Weights of Neuron Z1');
61 disp(w11);
62 disp(w21);
63 disp('weights of Neuron Z2');
64 disp(w12);
65 disp(w22);
66 disp('weights of Neuron Y');
67 disp(v1);
68 disp(v2);
69 disp('Threshold value');
70 disp(theta);
71
72 //Truth Table
73 //X1  X2  Y
74 //0   0   0
75 //0   1   1
76 //1   0   1
77 //1   1   0
78
79 //Output
80
81 // Enter weights
82 //Weight w11=1
83 //weight w12=-1
84 //Weight w21=-1
85 //weight w22=1
86 //weight v1=1
87 //weight v2=1
88 //
89 // Enter Threshold Value
90 //theta=1
91 //
92 // Output of Net
93 //
94 //      0.

```

```
95 //      1.
96 //      1.
97 //      0.
98 //
99 // McCulloch–Pitts Net for XOR function
100 //
101 // Weights of Neuron Z1
102 //
103 //      1.
104 //
105 //     - 1.
106 //
107 // weights of Neuron Z2
108 //
109 //     - 1.
110 //
111 //      1.
112 //
113 // weights of Neuron Y
114 //
115 //      1.
116 //
117 //      1.
118 //
119 // Threshold value
120 //
121 //      1.
```

Experiment: 3

Hebb Net to classify two dimensional input patterns

Scilab code Solution 3.3 3

```
1 //Hebb Net to classify two dimensional input
  patterns
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 //Input Patterns
8 E=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1];
9 F=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 -1 -1
  -1];
10 x(1,1:20)=E;
11 x(2,1:20)=F;
12 w(1:20)=0;
13 w=w'
14 t=[1 -1];
15 b=0;
16 for i=1:2
17     w=w+x(i,1:20)*t(i);
```

```

18     b=b+t(i);
19 end
20 disp('Weight matrix');
21 disp(w);
22 disp('Bias ');
23 disp(b);
24
25 //Output
26 //
27 // Weight matrix
28 //
29 //
30 //      column 1 to 18
31 //
32 //      0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
33 //      0.    0.    0.    0.    0.//      0.    0.    2.
34 //
35 //      column 19 to 20
36 //      2.    2.
37 //
38 // Bias
39 //
40 //      0.

```

Experiment: 4

Hetro associative neural net

Scilab code Solution 4.4 4

```
1 //Hetro associative neural net
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 x=[1 1 0 0;1 0 1 0;1 1 1 0;0 1 1 0];
8 t=[1 0;1 0;0 1;0 1];
9 w=zeros(4,2);
10 for i=1:4
11     w=w+x(i,1:4) '*t(i,1:2);
12 end
13 disp('Weight matrix');
14 disp(w);
15
16
17 //Autotassociative net to store the vector
18
19 x=[1 1 -1 -1];
20 xv=[1;1;-1;-1];
21 w=zeros(4,4);
```

```

22 w=x'*x;
23 yin=x*w;
24 for i=1:4
25     if yin(i)>0
26         y(i)=1;
27     else
28         y(i)=-1;
29     end
30 end
31 disp('Weight matrix');
32 disp(w);
33 if xv==y
34     disp('The vector is a Known Vector');
35 else
36     disp('The vector is an Unknown Vector');
37 end
38
39 //Output
40 // Weight matrix
41 //
42 //     2.     1.
43 //     1.     2.
44 //     1.     2.
45 //     0.     0.
46 //
47 // Weight matrix
48 //
49 //     1.     1.    -1.    -1.
50 //     1.     1.    -1.    -1.
51 //    -1.    -1.     1.     1.
52 //    -1.    -1.     1.     1.
53 //
54 // The vector is a Known Vector

```

Experiment: 5

Discrete Hopfield net

Scilab code Solution 5.5 5

```
1 //Discrete Hopfield net
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 x=[1 1 1 0];
8 tx=[0 0 1 0];
9 w1=(2*x'-1);
10 w2=(2*x-1);
11 w=w1*w2;
12
13 for i=1:4
14     w(i,i)=0;
15 end
16 con=1;
17 y=[0 0 1 0];
18 while con
19     up=[4 2 1 3];
20     for i=1:4
21         yin(up(i))=tx(up(i))+y*w(1:4,up(i));
```

```
22         if yin(up(i))>0
23             y(up(i))=1;
24         end
25     end
26     if y==x
27         disp('Convergence has been obtained');
28         disp('The Converged Ouput');
29         disp(y);
30         con=0;
31     end
32 end
33
34 //Output
35 // Convergence has been obtained
36 //
37 // The Converged Ouput
38 //
39 // 1. 1. 1. 0.
```

Experiment: 6

Kohonen self organizing maps

Scilab code Solution 6.6 6

```
1 //Kohonen self organizing maps
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 x=[1 1 0 0;0 0 0 1;1 0 0 0;0 0 1 1];
8 alpha=0.6;
9
10 //initial weight matrix
11 w=rand(4,2);
12 disp('Initial weight matrix');
13 disp(w);
14 con=1;
15 epoch=0;
16 while con
17     for i=1:4
18         for j=1:2
19             D(j)=0;
20             for k=1:4
21                 D(j)=D(j)+(w(k,j)-x(i,k))^2;
```

```

22         end
23     end
24     for j=1:2
25         if D(j)==min(D)
26             J=j;
27         end
28     end
29     w(:,J)=w(:,J)+alpha*(x(i,:)'-w(:,J));
30 end
31 alpha=0.5*alpha;
32 epoch=epoch+1;
33 if epoch==300
34     con=0;
35 end
36 end
37 disp('Weight Matrix after 300 epoch');
38 disp(w);
39
40 //Output
41 // Initial weight matrix
42 //
43 //     0.2113249     0.6653811
44 //     0.7560439     0.6283918
45 //     0.0002211     0.8497452
46 //     0.3303271     0.685731
47 //
48 // Weight Matrix after 300 epoch
49 //
50 //     0.9671633     0.0277033
51 //     0.4283588     0.0261632
52 //     0.0000092     0.5968633
53 //     0.0137532     0.9869153

```

Experiment: 7

Learning Vector Quantisation

Scilab code Solution 7.7 7

```
1 //Learning Vector Quantization
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 s=[1 1 0 0;0 0 0 1;0 0 1 1;1 0 0 0;0 1 1 0];
8 st=[1 2 2 1 2];
9 alpha=0.6;
10
11 //initial weight matrix first two vectors of input
    patterns
12 w=[s(1,:);s(2,:)]';
13 disp('Initial weight matrix');
14 disp(w);
15
16 //set remaining as input vector
17 x=[s(3,:);s(4,:);s(5,:)];
18 t=[st(3);st(4);st(5)];
19 con=1;
20 epoch=0;
```

```

21 while con
22     for i=1:3
23         for j=1:2
24             D(j)=0;
25             for k=1:4
26                 D(j)=D(j)+(w(k,j)-x(i,k))^2;
27             end
28         end
29         for j=1:2
30             if D(j)==min(D)
31                 J=j;
32             end
33         end
34         if J==t(i)
35             w(:,J)=w(:,J)+alpha*(x(i,:)'-w(:,J));
36         else
37             w(:,J)=w(:,J)-alpha*(x(i,:)'-w(:,J));
38         end
39     end
40     alpha=0.5*alpha;
41     epoch=epoch+1;
42     if epoch==100
43         con=0;
44     end
45 end
46 disp('Weight Matrix after 100 epochs');
47 disp(w);
48
49 //Output
50 // Initial weight matrix
51 //
52 //     1.    0.
53 //     1.    0.
54 //     0.    0.
55 //     0.    1.
56 //
57 // Weight Matrix after 100 epochs
58 //

```

59	//	1.	0.
60	//	0.2040471	0.561484
61	//	0.	0.9583648
62	//	0.	0.438516

Experiment: 8

Full Counter Propagation Network for given input pair

Scilab code Solution 8.8 8

```
1 //Full counter propagation network for given input
   pair
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 //set initial weights
8 v=[0.6 0.2;0.6 0.2;0.2 0.6; 0.2 0.6];
9 w=[0.4 0.3;0.4 0.3];
10 x=[0 1 1 0];
11 y=[1 0];
12 alpha=0.3;
13 for j=1:2
14     D(j)=0;
15     for i=1:4
16         D(j)=D(j)+(x(i)-v(i,j))^2;
17     end
18     for k=1:2
```

```

19         D(j)=D(j)+(y(k)-w(k,j))^2;
20     end
21 end
22 for j=1:2
23     if D(j)==min(D)
24         J=j;
25     end
26 end
27 disp('After one step the weight matrix are');
28 v(:,J)=v(:,J)+alpha*(x'-v(:,J))
29 w(:,J)=w(:,J)+alpha*(y'-w(:,J))
30 disp('v')
31 disp(v)
32 disp('w')
33 disp(w)
34
35 //Output
36 //
37 // After one step the weight matrix are
38 //
39 // v
40 //
41 //     0.42     0.2
42 //     0.72     0.2
43 //     0.44     0.6
44 //     0.14     0.6
45 //
46 // w
47 //
48 //     0.58     0.3
49 //     0.28     0.3

```

Experiment: 9

ART1 Neural Net

Scilab code Solution 9.9 9

```
1 //ART1 Neural Net
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 b=[0.57 0.0 0.3;0.0 0.0 0.3;0.0 0.57 0.3;0.0 0.47
    0.3];
8 t=[1 1 0 0;1 0 0 1;1 1 1 1];
9 vp=0.4;
10 L=2;
11 x=[1 0 1 1];
12 s=x;
13 ns=sum(s);
14 y=x*b;
15 con=1;
16 while con
17     for i=1:3
18         if y(i)==max(y)
19             J=i;
20         end
```

```

21     end
22     x=s.*t(J,:);
23     nx=sum(x);
24     if nx/ns >= vp
25         b(:,J)=L*x(:)/(L-1+nx);
26         t(J,:)=x(1,:);
27         con=0;
28     else
29         y(J)=-1;
30         con=1;
31     end
32     if y+1==0
33         con=0;
34     end
35 end
36 disp('Top Down Weights');
37 disp(t);
38 disp('Bottom up Weights');
39 disp(b);
40
41 //Output
42 // Top Down Weights
43 //
44 //     1.     1.     0.     0.
45 //     1.     0.     0.     1.
46 //     1.     1.     1.     1.
47 //
48 // Bottom up Weights
49 //
50 //     0.57     0.6666667     0.3
51 //     0.         0.             0.3
52 //     0.         0.             0.3
53 //     0.         0.6666667     0.3

```

Experiment: 10

MLP

Scilab code Solution 10.10 10

```
1 //MLP Algorithm and implementation
2 //Windows 10
3 //Scilab 5.4.1
4 clear;
5 clc;
6
7 deff('y=f(x)', 'y=1/(1+exp(-x))')
8 Wih=[0.1, -0.3;0.3,0.4];
9 Who=[0.4;0.5]
10 i=[0.2,0.6];
11 t=0.7;
12 a=10;
13 for k=1:3
14     printf('\n\n\nAfter Iteration %i :\n\n',k)
15     disp(Wih, 'Wih = ')
16     disp(Who, 'Who = ')
17     a1=i*Wih;
18     disp(a1, 'a = ')
19     h=[f(a1(1)),f(a1(2))]
20     disp(h, 'h = ')
21     b1=h*Who
```

```

22 disp(b1, 'b1 =')
23 o=f(b1)
24 disp(o, 'o = ')
25 d=o*(1-o)*(t-o)
26 disp(d, 'd =')
27 for j=1:2
28     e(1,j)=h(j)*(1-h(j))*d*Who(j)
29 end
30 disp(e, 'e =')
31 dWho=a*h'*d;
32 disp(dWho, 'dWho =')
33 Who=Who+dWho;
34 dWih=a*i'*e;
35 disp(dWih, 'dWih =')
36 Wih=Wih+dWih;
37 end
38
39 //Output
40 //After Iteration 1 :
41 //
42 //
43 // Wih =
44 //
45 //     0.1   -0.3
46 //     0.3    0.4
47 //
48 // Who =
49 //
50 //     0.4
51 //     0.5
52 //
53 // a =
54 //
55 //     0.2    0.18
56 //
57 // h =
58 //
59 //     0.549834    0.5448789

```

```

60 //
61 // b1 =
62 //
63 // 0.492373
64 //
65 // o =
66 //
67 // 0.6206653
68 //
69 // d =
70 //
71 // 0.0186786
72 //
73 // e =
74 //
75 // 0.0018493 0.002316
76 //
77 // dWho =
78 //
79 // 0.102701
80 // 0.1017755
81 //
82 // dWih =
83 //
84 // 0.0036986 0.004632
85 // 0.0110958 0.0138961
86 //
87 //
88 //
89 // After Iteration 2 :
90 //
91 //
92 // Wih =
93 //
94 // 0.1036986 -0.295368
95 // 0.3110958 0.4138961
96 //
97 // Who =

```

```
98 //
99 // 0.502701
100 // 0.6017755
101 //
102 // a =
103 //
104 // 0.2073972 0.189264
105 //
106 // h =
107 //
108 // 0.5516642 0.5471753
109 //
110 // b1 =
111 //
112 // 0.6065989
113 //
114 // o =
115 //
116 // 0.6471646
117 //
118 // d =
119 //
120 // 0.0120646
121 //
122 // e =
123 //
124 // 0.0015 0.0017989
125 //
126 // dWho =
127 //
128 // 0.066556
129 // 0.0660144
130 //
131 // dWih =
132 //
133 // 0.0030001 0.0035978
134 // 0.0090002 0.0107933
135 //
```

```
136 //
137 //
138 // After Iteration 3 :
139 //
140 //
141 // Wih =
142 //
143 // 0.1066987 -0.2917702
144 // 0.320096 0.4246894
145 //
146 // Who =
147 //
148 // 0.569257
149 // 0.6677899
150 //
151 // a =
152 //
153 // 0.2133973 0.1964596
154 //
155 // h =
156 //
157 // 0.5531478 0.5489575
158 //
159 // b1 =
160 //
161 // 0.6814715
162 //
163 // o =
164 //
165 // 0.6640671
166 //
167 // d =
168 //
169 // 0.008016
170 //
171 // e =
172 //
173 // 0.0011279 0.0013254
```

```
174 //  
175 // dWho =  
176 //  
177 // 0.0443403  
178 // 0.0440044  
179 //  
180 // dWih =  
181 //  
182 // 0.0022558 0.0026508  
183 // 0.0067674 0.0079525
```
