

# NEURAL NETWORK AND FUZZY LOGIC

## ASSIGNMENT-2

### PROBLEM 1

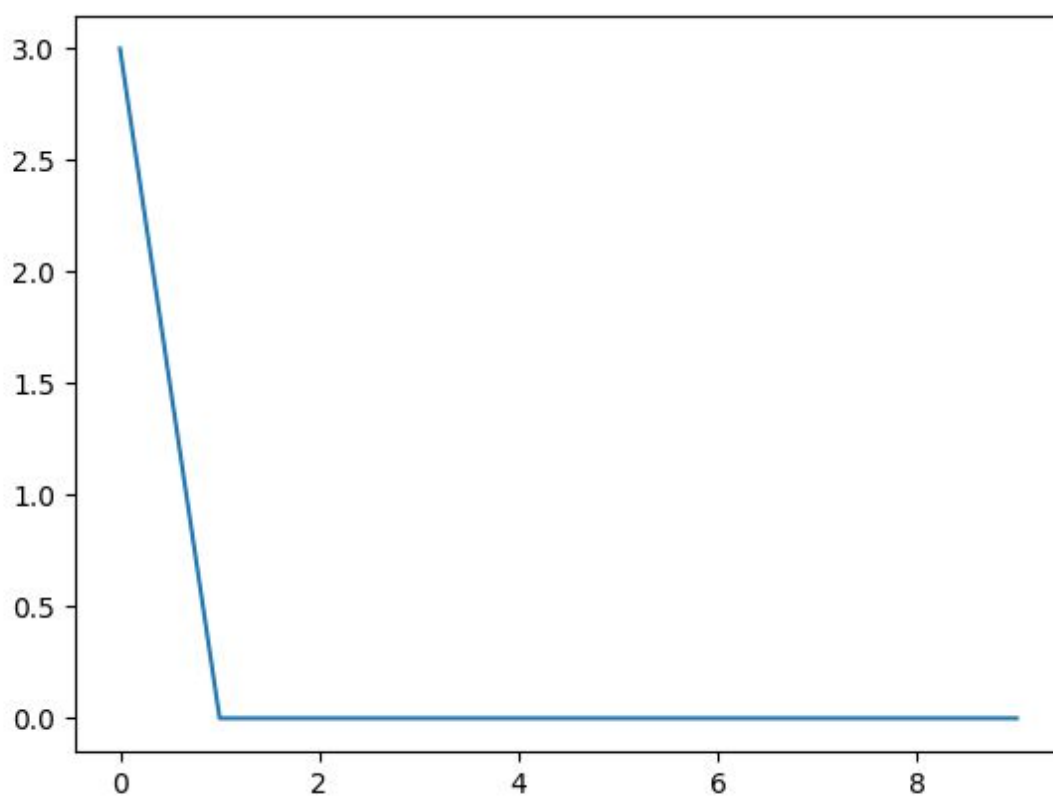
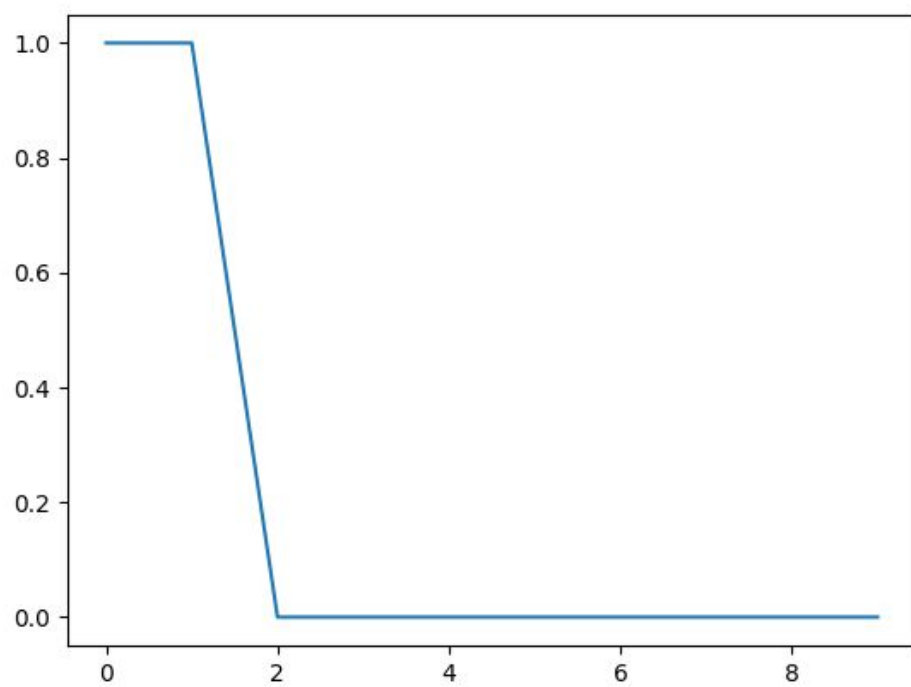
```
1  # hebbian learning
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import pandas as pd
5  import xlrd
6  from pandas import DataFrame
7
8  def hebbLearn1(x, y, nf):
9      if nf==0 :
10         w = [0, 0];
11     else:
12         w = 0;
13     b = 0;
14     xt = np.transpose(x);
15     alpha = 0.1;
16     theta = 0.5;
17     iterations = 10;
18     cost = [];
19     for t in range(iterations):
20         sume = 0.0;
21         # print(w);
22         for m in range(np.size(x[0])):
23             if nf==0 :
24                 am = b + np.matmul(xt[m], w);
25             else:
26                 am = b + xt[m]*w;
27             hm = 1 if am >= theta else 0;
28             if hm != y[m] :
29                 w = w + alpha*y[m]*xt[m];
30                 b = b + alpha*y[m];
31             sume += (y[m]-hm)**2;
32         # print(sume);
33         cost.append(sume);
34     plt.plot(range(iterations), cost);
35     plt.show();
36
37 def main():
38     # AND gate
39     x = [[0, 0, 1, 1],[0, 1, 0, 1]];
40     x = np.array(x);
41     y = [0, 0, 0, 1];
42     y = np.array(y);
43     hebbLearn1(x, y, 0);
```

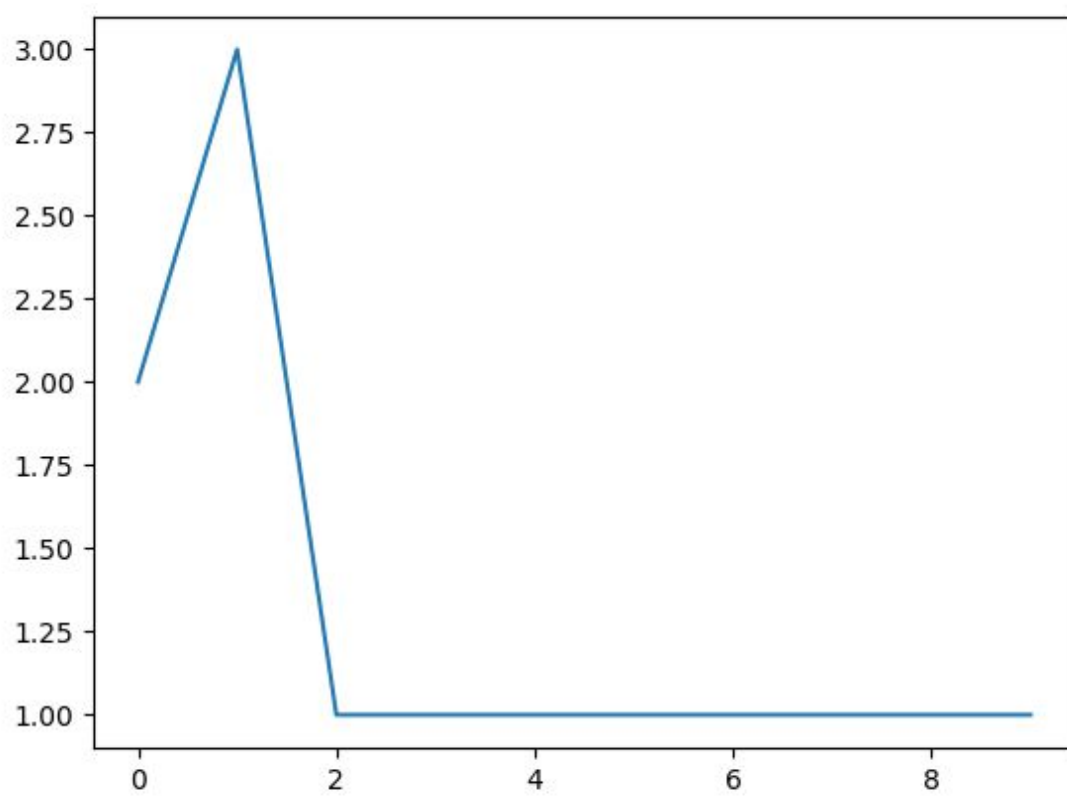
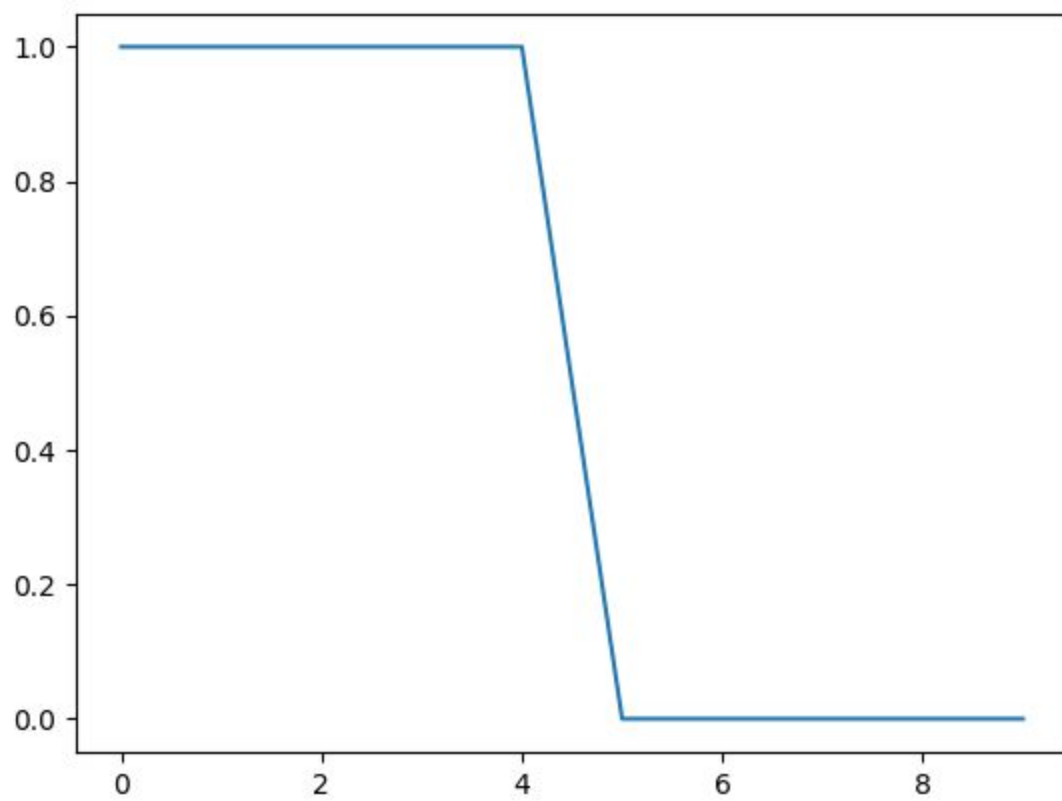
```
37 def main():
38     # AND gate
39     x = [[0, 0, 1, 1], [0, 1, 0, 1]];
40     x = np.array(x);
41     y = [0, 0, 0, 1];
42     y = np.array(y);
43     hebbLearn1(x, y, 0);
44
45     # OR gate
46     x = [[0, 0, 1, 1], [0, 1, 0, 1]];
47     x = np.array(x);
48     y = [0, 1, 1, 1];
49     y = np.array(y);
50     hebbLearn1(x, y, 0);
51
52     # NOT gate
53     x = [0, 1];
54     x = np.array(x);
55     y = [1, 0];
56     y = np.array(y);
57     hebbLearn1(x, y, 1);
58
59     # XOR gate
60     x = [[0, 0, 1, 1], [0, 1, 0, 1]];
61     x = np.array(x);
62     y = [0, 1, 1, 0];
63     y = np.array(y);
64     hebbLearn1(x, y, 0);
65
66     # ANDNOT gate
67     x = [[0, 0, 1, 1], [0, 1, 0, 1]];
68     x = np.array(x);
69     y = [0, 0, 1, 0];
70     y = np.array(y);
71     hebbLearn1(x, y, 0);
72
73     # NAND gate
74     x = [[0, 0, 1, 1], [0, 1, 0, 1]];
75     x = np.array(x);
76     y = [1, 1, 1, 0];
77     y = np.array(y);
78     hebbLearn1(x, y, 0);
79
```

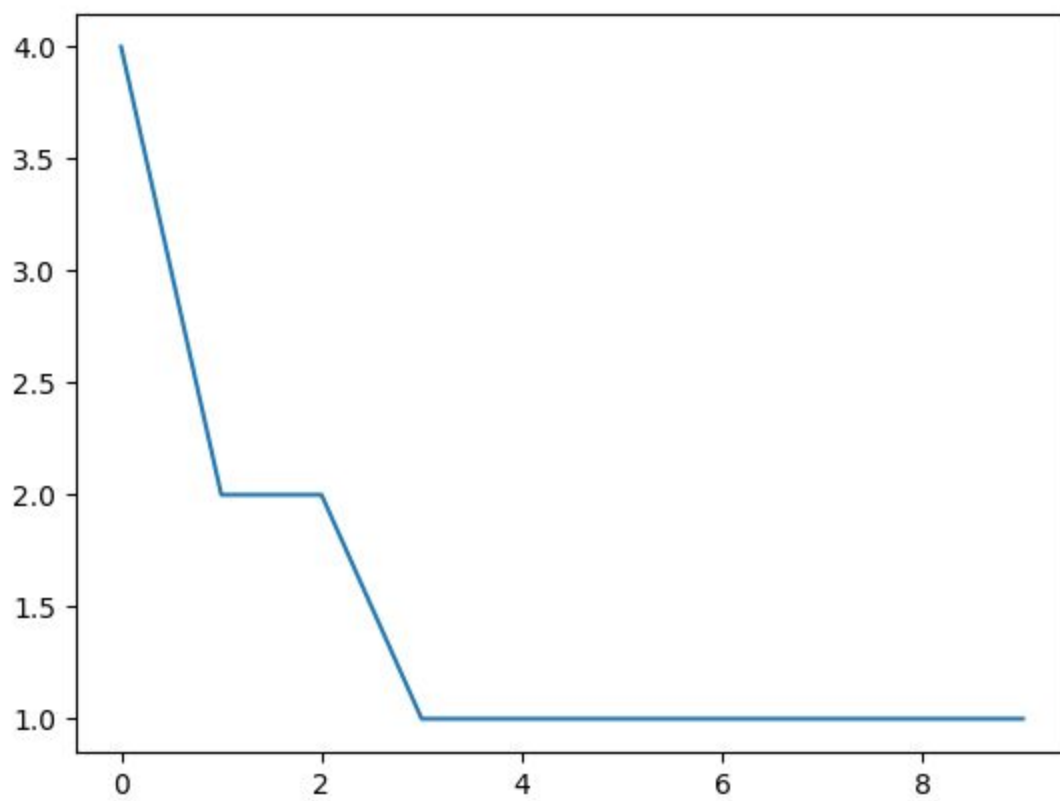
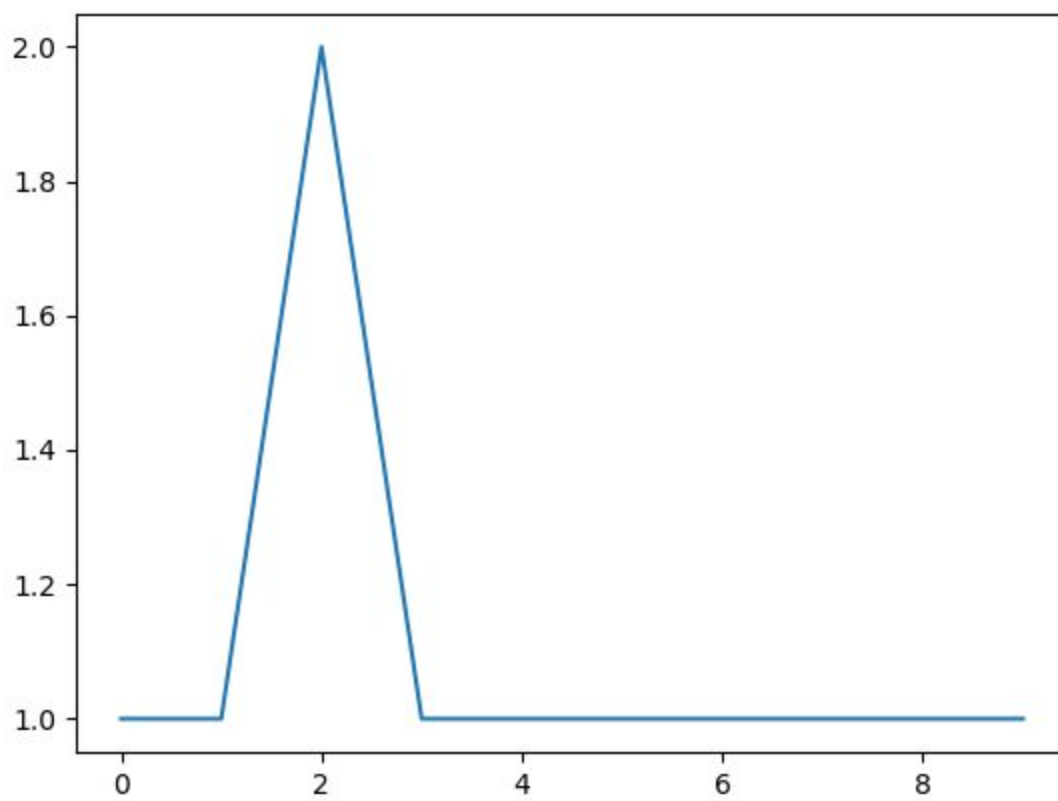
```

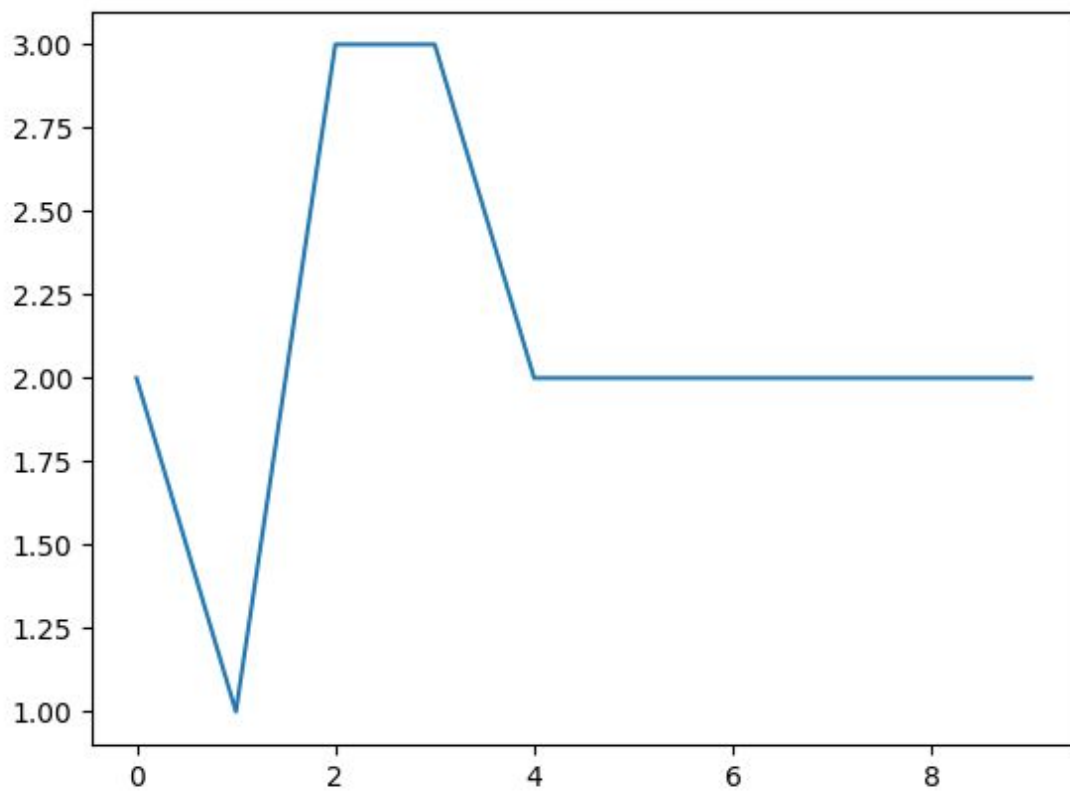
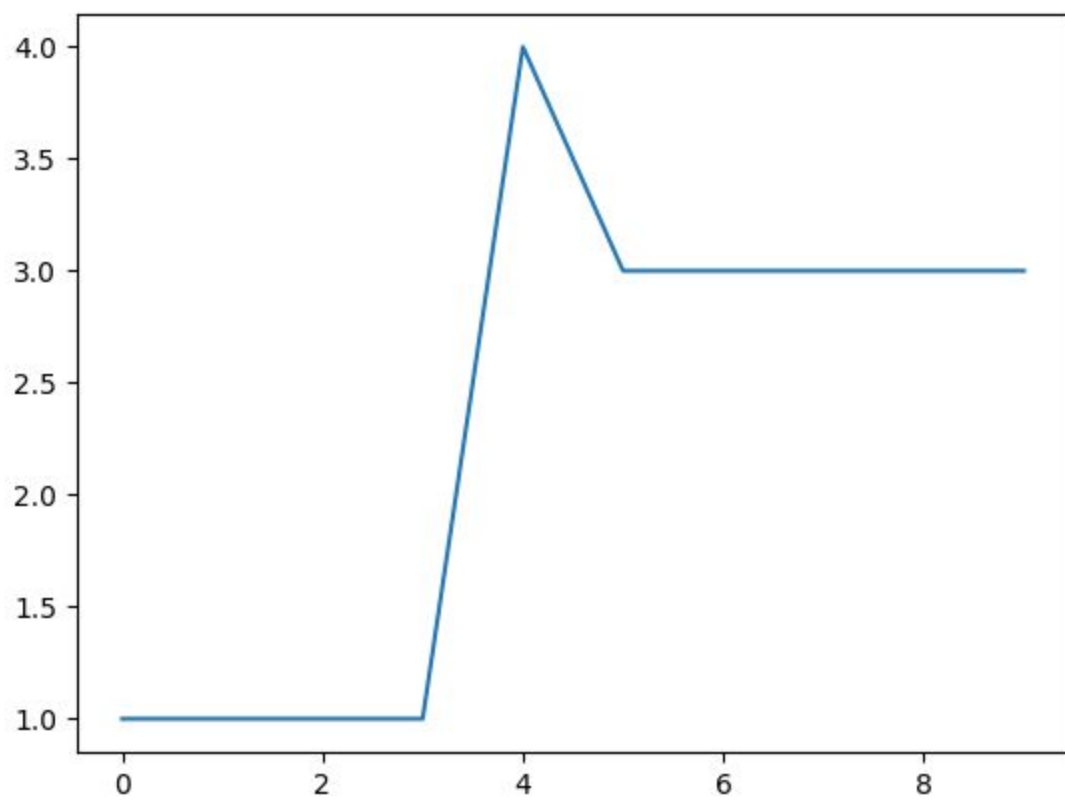
55     y = [1, 0];
56     y = np.array(y);
57     hebbLearn1(x, y, 1);
58
59     # XOR gate
60     x = [[0, 0, 1, 1],[0, 1, 0, 1]];
61     x = np.array(x);
62     y = [0, 1, 1, 0];
63     y = np.array(y);
64     hebbLearn1(x, y, 0);
65
66     # ANDNOT gate
67     x = [[0, 0, 1, 1],[0, 1, 0, 1]];
68     x = np.array(x);
69     y = [0, 0, 1, 0];
70     y = np.array(y);
71     hebbLearn1(x, y, 0);
72
73     # NAND gate
74     x = [[0, 0, 1, 1],[0, 1, 0, 1]];
75     x = np.array(x);
76     y = [1, 1, 1, 0];
77     y = np.array(y);
78     hebbLearn1(x, y, 0);
79
80     # NOR gate
81     x = [[0, 0, 1, 1],[0, 1, 0, 1]];
82     x = np.array(x);
83     y = [1, 0, 0, 0];
84     y = np.array(y);
85     hebbLearn1(x, y, 0);
86
87     # XNOR gate
88     x = [[0, 0, 1, 1],[0, 1, 0, 1]];
89     x = np.array(x);
90     y = [1, 0, 0, 1];
91     y = np.array(y);
92     hebbLearn1(x, y, 0);
93
94 if __name__ == "__main__":
95     main()
96

```









## PROBLEM 2

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 import xlrd
5 from pandas import DataFrame
6 from numpy.linalg import inv
7 import math
8 from sklearn.cross_validation import train_test_split
9 def find_max(x):
10     index = 0
11     maxi = x[index]
12     for i in range(len(x)):
13         if x[i]>maxi:
14             maxi = x[i]
15             index = i
16     return index
17
18 def sigmoid(x):
19     temp = 1.0 + np.exp(-x)
20     return (1.0/temp)
21
22 def main():
23     data = pd.read_excel('dataset.xlsx')
24     var1 = data['row1']
25     var2 = data['row2']
26     var3 = data['row3']
27     var4 = data['row4']
28     var5 = data['row5']
29     var6 = data['row6']
30     var7 = data['row7']
31     y = data['row8']
32     x = []
33     for i in data.index:
34         temp = []
35         # temp.append(1)
36         temp.append(var1[i])
37         temp.append(var2[i])
38         temp.append(var3[i])
39         temp.append(var4[i])
40         temp.append(var5[i])
41         temp.append(var6[i])
42         temp.append(var7[i])
43         x.append(temp)
```



```

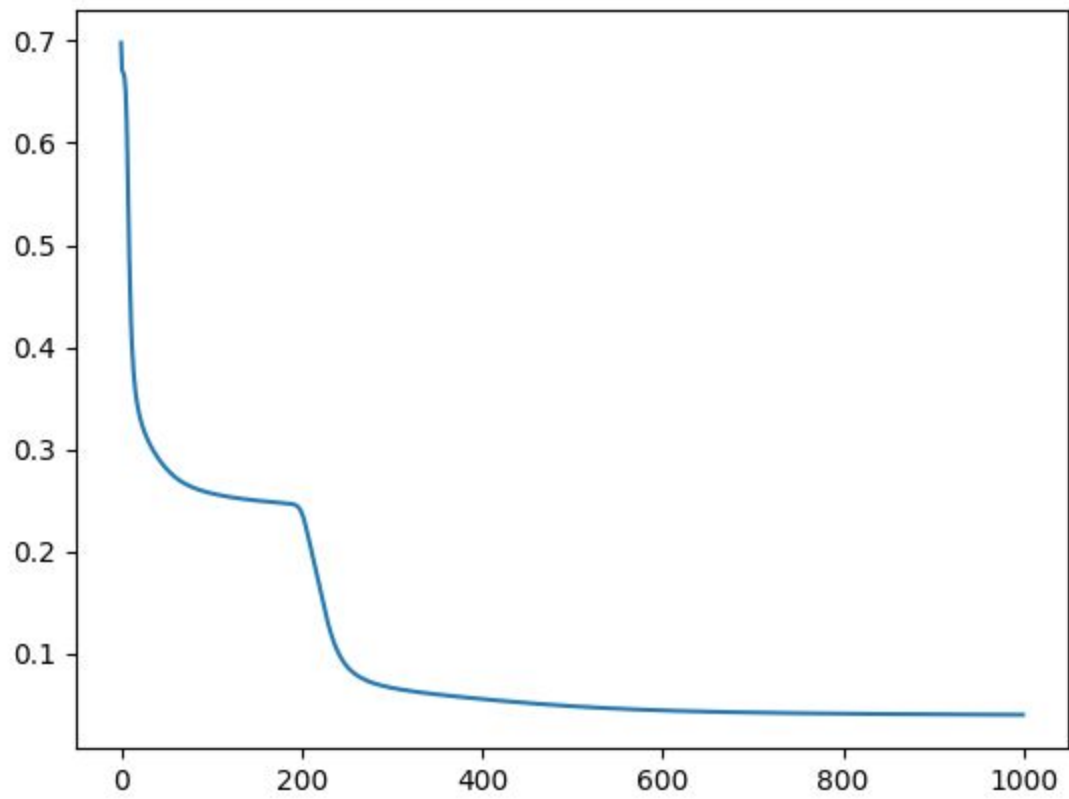
43     x.append(temp)
44
45     x = np.array(x)
46     y = np.array(y)
47     # x = sigmoid(x)
48     # normalizing the data
49     mean = np.sum(x,axis = 0)/len(x)
50     variance = (np.sum((x - mean)**2,axis = 0))/len(x)
51     x = (x - mean)/variance
52
53     xtra, xte, ytr, yte = train_test_split(x, y, train_size=0.7)
54     # print(sigmoid(xtra))
55     nhid = 6
56     nout = 3
57     # print(len(xtra[0]))
58     yt = []
59     for i in range(len(ytr)):
60         if ytr[i] == 1: yt.append([1, 0, 0])
61         elif ytr[i] == 2: yt.append([0, 1, 0])
62         else: yt.append([0, 0, 1])
63
64     w11 = np.zeros((nhid, len(xtra[0])))
65     w12 = np.zeros((nout, nhid))
66     b1 = np.zeros(nhid)
67     b2 = np.zeros(nout)
68     yt = np.array(yt)
69     eta = 0.1
70     iteration = 1000
71     cost = np.zeros(iteration)
72     cost_test = np.zeros(iteration)
73     for ite in range(iteration):
74         for m in range(len(xtra)):
75             out0 = xtra[m]
76             out1 = sigmoid(np.matmul(out0, w11.T)+b1)
77             # print(out1)
78             out2 = sigmoid(np.matmul(out1, w12.T)+b2)
79             delta2 = (yt[m]-out2)*out2*(1-out2)
80             # print(np.matmul(delta2, w12))
81             delta1 = np.matmul(delta2, w12)*out1*(1-out1)
82             # print(delta1)
83             w12 = w12 + eta*np.outer(delta2, out1)
84             b2 = b2 + eta*delta2
85             w11 = w11 + eta*np.outer(delta1, out0)

```

```

73     for ite in range(iteration):
74         for m in range(len(xtra)):
75             out0 = xtra[m]
76             out1 = sigmoid(np.matmul(out0, w11.T)+b1)
77             # print(out1)
78             out2 = sigmoid(np.matmul(out1, w12.T)+b2)
79             delta2 = (yt[m]-out2)*out2*(1-out2)
80             # print(np.matmul(delta2, w12))
81             delta1 = np.matmul(delta2, w12)*out1*(1-out1)
82             # print(delta1)
83             w12 = w12 + eta*np.outer(delta2, out1)
84             b2 = b2 + eta*delta2
85             w11 = w11 + eta*np.outer(delta1, out0)
86             b1 = b1 + eta*delta1
87             cost[ite] += np.sum((yt[m]-out2)**2)
88
89         # print(w11)
90         # print(w12)
91         # testing the data
92         # accuracy is the objective function
93         cnt = 0
94         tot = 0
95         for m in range(len(xte)):
96             out0 = xte[m]
97             out1 = sigmoid(np.matmul(out0, w11.T)+b1)
98             out2 = sigmoid(np.matmul(out1, w12.T)+b2)
99             cost_test[ite] += np.sum((yte[m]-out2)**2)
100            ind = find_max(out2)+1
101            if ind==yte[m] : cnt+=1
102            tot+=1
103
104        cost /= len(xtra)
105        cost_test /= len(xte)
106        plt.plot(range(iteration), cost)
107        # plt.plot(range(iteration), cost_test)
108        plt.show()
109
110        print('Accuracy : ')
111        print(cnt/tot)
112
113 if __name__ == "__main__":
114     main()
115

```

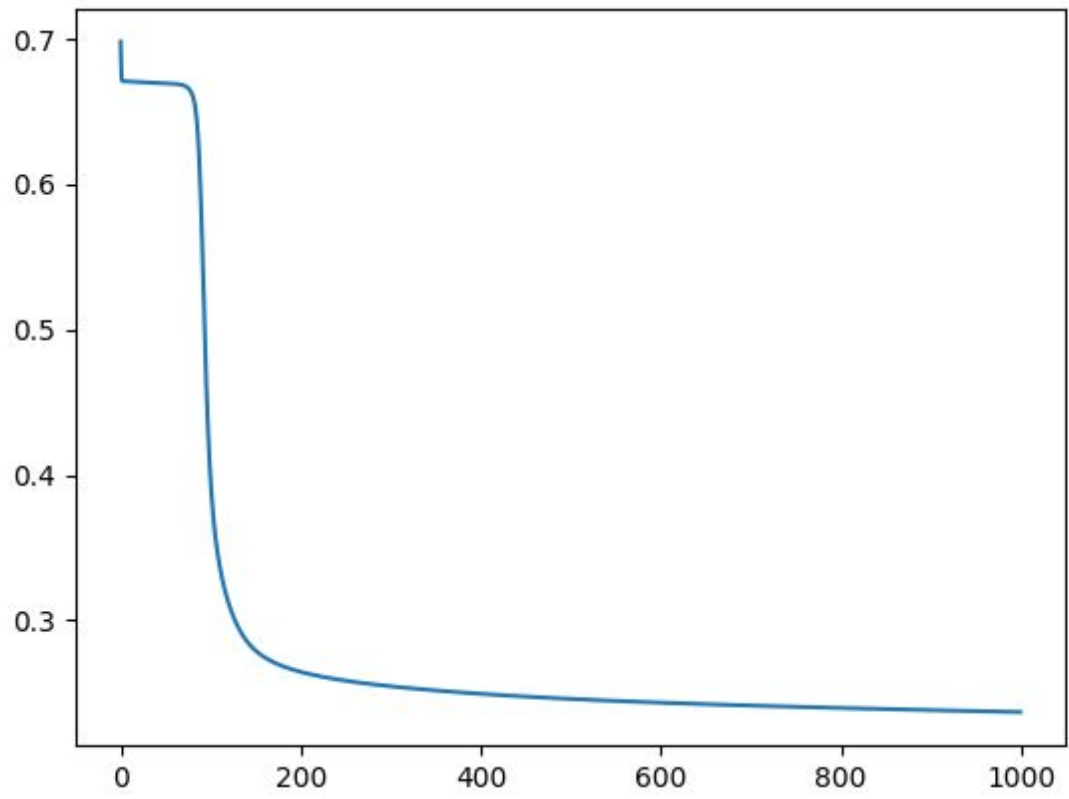


PROBLEM 3

```

67     w13 = np.zeros((nout, nhid2))
68     b1 = np.zeros(nhid1)
69     b2 = np.zeros(nhid2)
70     b3 = np.zeros(nout)
71     yt = np.array(yt)
72     eta = 0.1
73     iteration = 1000
74     cost = np.zeros(iteration)
75     cost_test = np.zeros(iteration)
76     for ite in range(iteration):
77         for m in range(len(xtra)):
78             out0 = xtra[m]
79             out1 = sigmoid(np.matmul(out0, w11.T)+b1)
80             out2 = sigmoid(np.matmul(out1, w12.T)+b2)
81             out3 = sigmoid(np.matmul(out2, w13.T)+b3)
82
83             delta3 = (yt[m]-out3)*out3*(1-out3)
84             delta2 = np.matmul(delta3, w13)*out2*(1-out2)
85             delta1 = np.matmul(delta2, w12)*out1*(1-out1)
86
87             w13 = w13 + eta*np.outer(delta3, out2)
88             b3 = b3 + eta*delta3
89             w12 = w12 + eta*np.outer(delta2, out1)
90             b2 = b2 + eta*delta2
91             w11 = w11 + eta*np.outer(delta1, out0)
92             b1 = b1 + eta*delta1
93             cost[ite] += np.sum((yt[m]-out3)**2)
94
95         # print(w11)
96         # print(w12)
97         # testing the data
98         # accuracy is the objective function
99         cnt = 0
100        tot = 0
101        for m in range(len(xte)):
102            out0 = xte[m]
103            out1 = sigmoid(np.matmul(out0, w11.T)+b1)
104            out2 = sigmoid(np.matmul(out1, w12.T)+b2)
105            out3 = sigmoid(np.matmul(out2, w13.T)+b3)
106            cost_test[ite] += np.sum((yte[m]-out3)**2)
107            ind = find_max(out3)+1
108            if ind==yte[m] : cnt+=1
109            tot+=1

```



PROBLEM 4

```

52
53 x = np.array(x)
54 y = np.array(y)
55 # x = sigmoid(x)
56 # normalizing the data
57 mean = np.sum(x,axis = 0)/len(x)
58 variance = (np.sum((x - mean)**2,axis = 0))/len(x)
59 x = (x - mean)/variance
60
61
62
63 xtra, xte, ytr, yte = train_test_split(x, y, train_size=0.7)
64
65 yt = []
66 for i in range(len(ytr)):
67     if ytr[i] == 1: yt.append([1, 0, 0])
68     elif ytr[i] == 2: yt.append([0, 1, 0])
69     else: yt.append([0, 0, 1])
70
71 nhid = 8
72 kmeans = KMeans(n_clusters=nhid, random_state=0).fit(xtra)
73 mu = kmeans.cluster_centers_
74
75 # training the data
76 h = np.zeros((len(xtra), nhid))
77 for i in range(len(xtra)):
78     for j in range(nhid):
79         h[i][j] = basis_func(xtra[i], mu[j])
80
81 w = np.matmul(pinv(h), yt)
82
83 # testing the data
84 # accuracy is the objective function
85 cnt = 0
86 tot = 0
87 ht = np.zeros((len(xte), nhid))
88 for i in range(len(xte)):
89     for j in range(nhid):
90         ht[i][j] = basis_func(xte[i], mu[j])
91
92 yp = np.matmul(ht, w)
93 for i in range(len(yp)):
94     ind = find_max(yp[i])+1
95     if ind==yte[i] : cnt+=1
96     tot+=1
97
98 print('Accuracy : ')
99 print(cnt/tot)

```

## PROBLEM 5

```
54 y = np.array(y)
55 # x = sigmoid(x)
56 # normalizing the data
57 mean = np.sum(x,axis = 0)/len(x)
58 variance = (np.sum((x - mean)**2,axis = 0))/len(x)
59 x = (x - mean)/variance
60
61 xtra, xte, ytr, yte = train_test_split(x, y, train_size=0.7)
62
63 yt = []
64 for i in range(len(ytr)):
65     if ytr[i] == 1: yt.append([1, 0, 0])
66     elif ytr[i] == 2: yt.append([0, 1, 0])
67     else: yt.append([0, 0, 1])
68
69 # finding the w1 matrix ie input to hidden layer(pretraining)
70 nhid = 8
71 nout = len(xtra[0])
72 # print(len(xtra[0]))
73 yt = []
74 for i in range(len(ytr)):
75     if ytr[i] == 1: yt.append([1, 0, 0])
76     elif ytr[i] == 2: yt.append([0, 1, 0])
77     else: yt.append([0, 0, 1])
78
79 w11 = np.zeros((nhid, len(xtra[0])))
80 w12 = np.zeros((nout, nhid))
81 b1 = np.zeros(nhid)
82 b2 = np.zeros(nout)
83 yt = np.array(yt)
84 eta = 0.001
85 iteration = 100
86 cost = np.zeros(iteration)
87 cost_test = np.zeros(iteration)
88 for ite in range(iteration):
89     for m in range(len(xtra)):
90         out0 = xtra[m]
91         out1 = np.matmul(out0, w11.T)+b1
92         # print(out1)
93         out2 = np.matmul(out1, w12.T)+b2
94         delta2 = (xtra[m]-out2)
95         delta1 = np.matmul(delta2, w12)
96         w12 = w12 + eta*np.outer(delta2, out1)
97         b2 = b2 + eta*delta2
98         w11 = w11 + eta*np.outer(delta1, out0)
```

```

        # print(out1)
        out2 = np.matmul(out1, w12.T)+b2
        delta2 = (xtra[m]-out2)
        delta1 = np.matmul(delta2, w12)
        w12 = w12 + eta*np.outer(delta2, out1)
        b2 = b2 + eta*delta2
        w11 = w11 + eta*np.outer(delta1, out0)
        b1 = b1 + eta*delta1
        cost[ite] += np.sum((xtra[m]-out2)**2)

out1 = np.matmul(xtra, w11.T)+b1

# hidden layer to rbfnn
nhid = 8
kmeans = KMeans(n_clusters=nhid, random_state=0).fit(out1)
mu = kmeans.cluster_centers_

# training the data
h = np.zeros((len(out1), nhid))
for i in range(len(out1)):
    for j in range(nhid):
        h[i][j] = basis_func(out1[i], mu[j])

w = np.matmul(pinv(h), yt)

# testing the data
# accuracy is the objective function
cnt = 0
tot = 0
out1_test = np.matmul(xte, w11.T)+b1
ht = np.zeros((len(out1_test), nhid))
for i in range(len(out1_test)):
    for j in range(nhid):
        ht[i][j] = basis_func(out1_test[i], mu[j])
yp = np.matmul(ht, w)
for i in range(len(yp)):
    ind = find_max(yp[i])+1
    if ind==yte[i] : cnt+=1
    tot+=1

print('Accuracy : ')
print(cnt/tot)

if __name__ == "__main__":
    main()

```