Course Number: EEL 5840

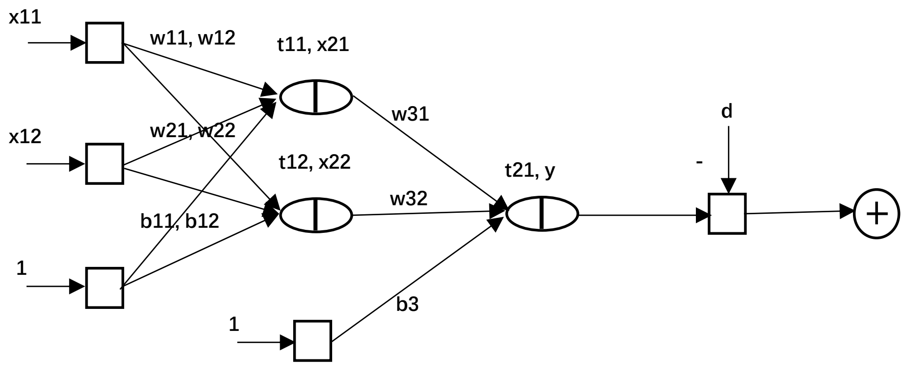
UF ID: 0699 6263
Name: Fang Zhu

Homework 6:

Q1.

Define your network's architecture (processing units in the input, hidden and output layers) and use a sigmoid activation function in the hidden layer. Explain why you used that number of hidden processing elements.

The architecture of the network is as follow:



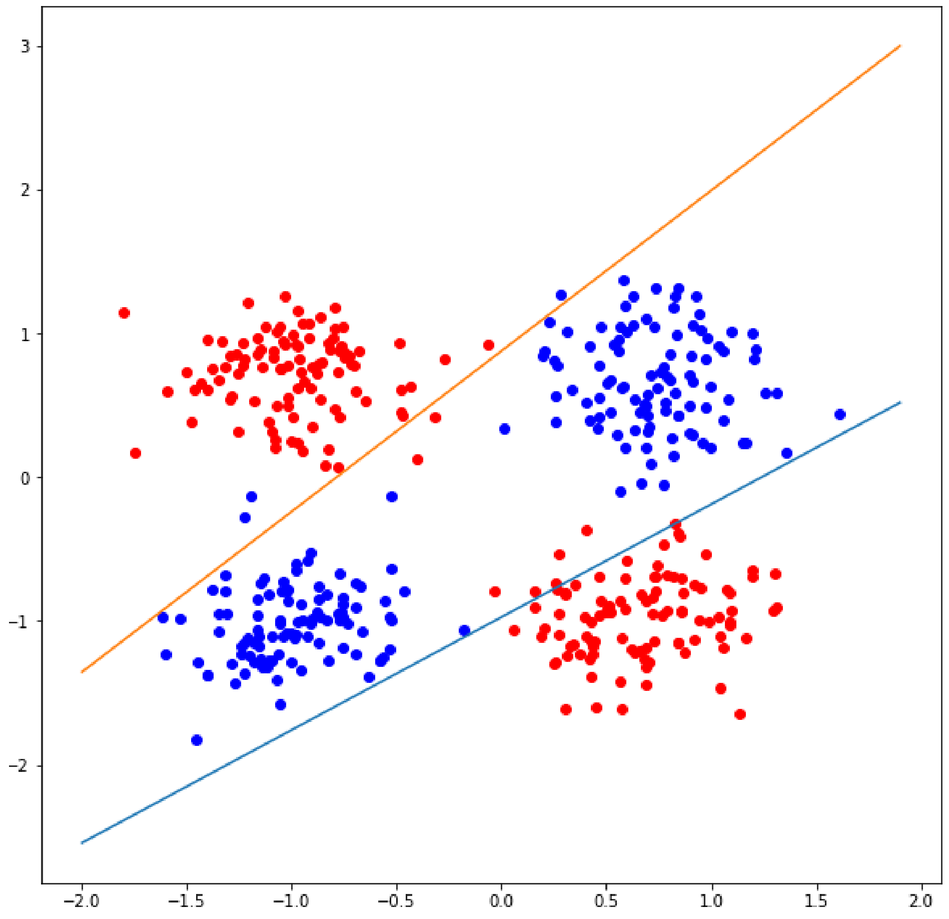
According to the universal Approximation Theorem, the three layers architecture can solve the problem. Because the distribution of the data is can be divided into two parts and one line cannot split the two classes, so we need two processing elements as the hidden layer. The output layer would help to adjust the value of points in the three parts from the hidden layer to be two classes.

Q2.

Plot the corresponding decision boundaries after the networks have finished training. As well, see how well your solution performs on

samples that do not belong to the training set by generating additional samples that follow the same pattern. Keep these generated points within a square that has length 2 on each of its sides.

The boundary of the classification is as follow:



The parameters weights and biases for the lines are: w11 = -4.659, w12 = -4.432, w21 = 5.882, w22 = 4.112, b11 = 4.912, b12 = 3.442. We can get the two lines: y = 0.792x-0.835; y=1.078x-0.837.

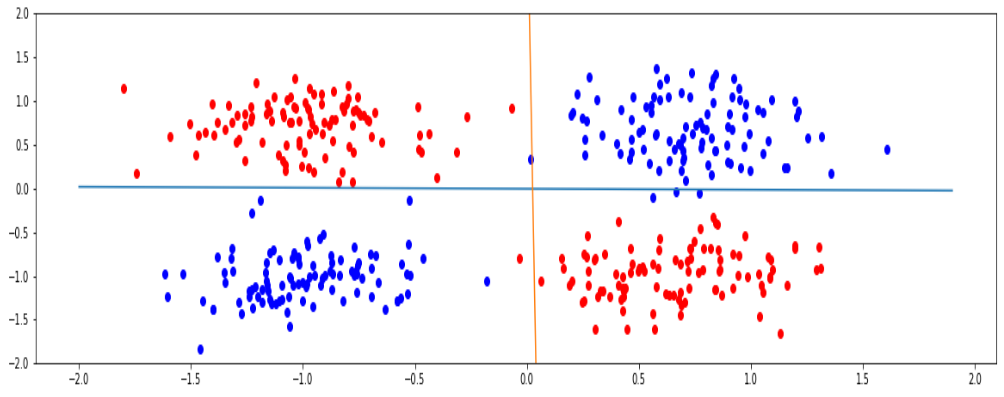
I take 4 test data from four quadrants: (0.32, 0.92), (0.78, -1.82), (-0.76, 1.54), (-0.77, -1.29). The answer for the four points are 0.128287556431, 0.98403482964, 0.982469450258, 0.168603857649. The two classes can be divided.

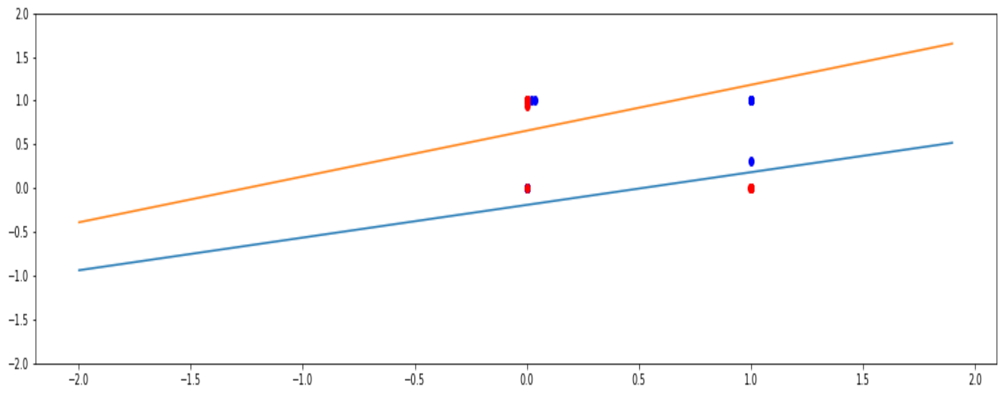
However, in the picture we can find the problem that the two line cannot deal with the point closed to zero. The rate of correct classification is 0.7925.

Q3.

Can multiple architectures solve this problem? If no, why not? If yes, give at least two examples. Which one would you choose and why? Justify your answer.

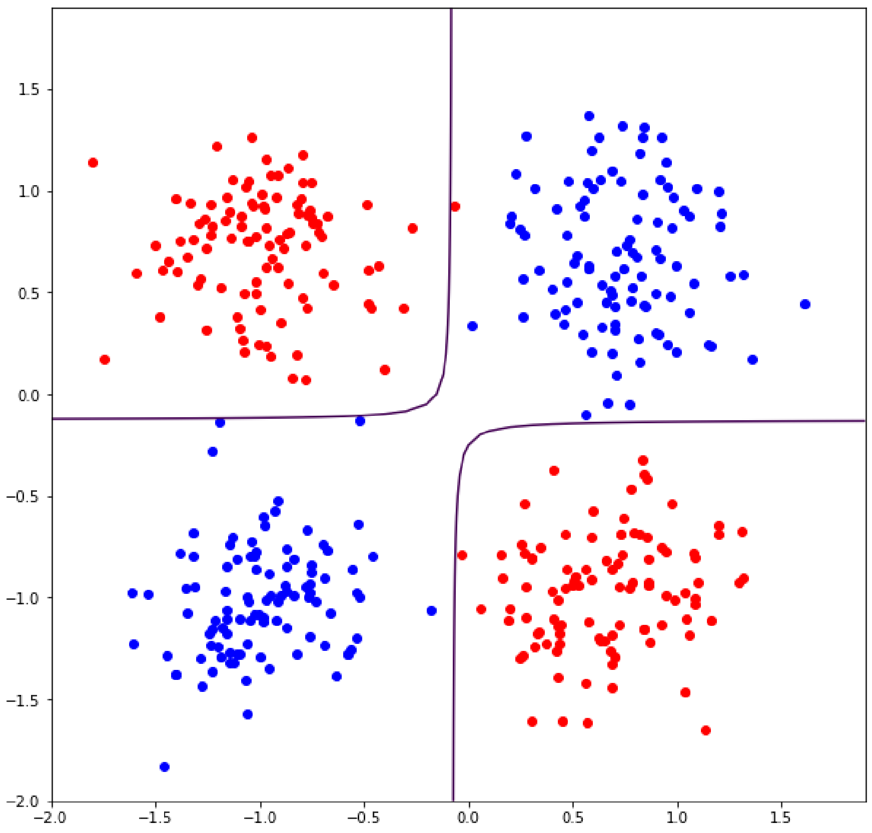
1. Yes, the multiple architectures can solve the problem, however the training data set is not enough to train, and the local optimal problems may happen. So sometimes we cannot get the right classification. I tried may times, the best rate of classification is 0.9975. The plot of the boundaries is as follow:





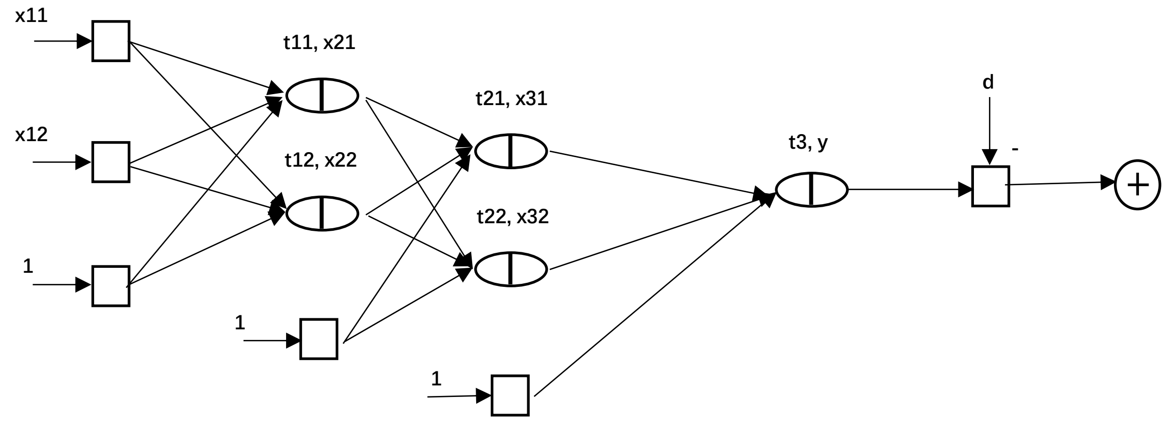
In the first layer we can classified the points into four quadrants, in the second layer I draw two lines to classifies the four "points" into three parts. The output layer help to adjust the three parts to be two classes. Because of the first layer accumulate the sparse points from different the four quadrants, the classification would be much easier.

2. The second solution is to add a PE in the first layer, the product of two input, as we known that the sign of the product of two input correspond to the two inputs, it means that it can classify the data into two class without hidden layer. So, we only need two layers and one PE for output layer. The rate of correction is 0.9525, which is much better. The plot of boundary is as follow:

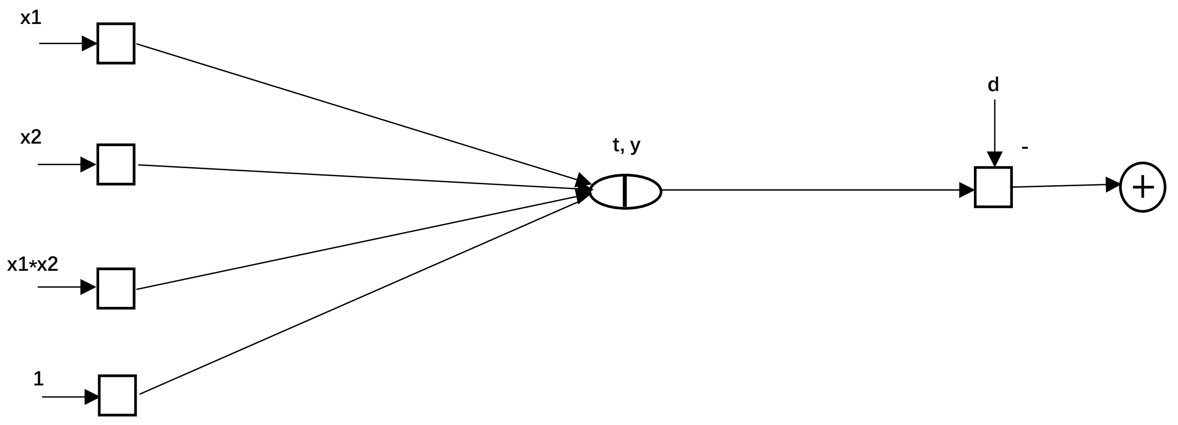


Finally, the architecture of the two examples are as follow:

1.



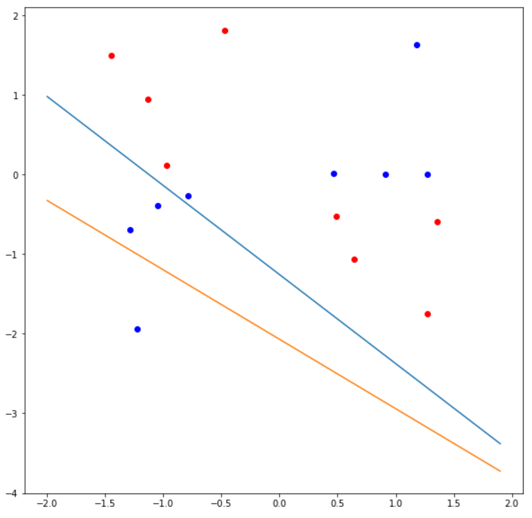
2.

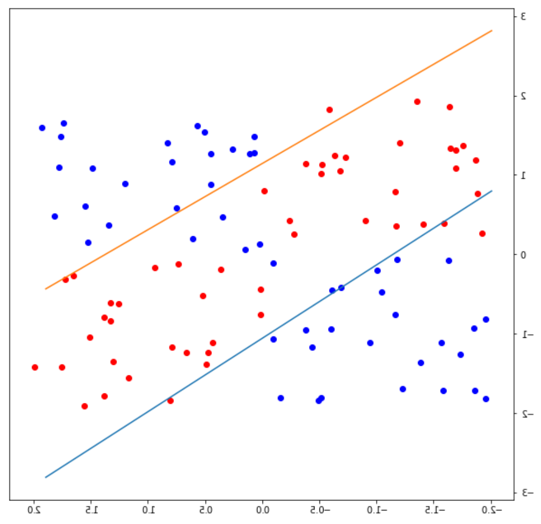


Q4.

Choose 8 points for each class, 4 points in each quadrant, as your training data. Are the decision boundaries optimal? If not, what is the best solution? Comment on the generalization ability of your network with such fewer points.

The boundaries are not optimal, the right rate for the training data (the accuracy) is 0.615. The plot is as below. The solution I take is to add more data. So, I generalize 100 data, the rate becomes much better (0.8325). The two plots are as below.





Q5.

Lastly, explain what solution you would choose if you were going to hand-select the parameters of the network. Does the multi-layer neural network learn the solution that you expected?

I will choose the first architecture in question 3, because this would help to simplify this problem to be a XOR problem. Because of the sigmoid PE, to make the sparse point closed to one and zero, so the parameters in the first input layer to the first hidden layer would be ([0, 100, 0] [100, 0, 0]), ([w11, w21, b11] [w12, w22, b12]). This would make the negative net closed to 0 after the sigmoid function and the positive net closed to 1. Then the parameters for the first hidden layer to the second hidden layer would be ([100, -100, 50] [100, -100, -50]), ([w31, w41, b21] [w32, w42, b22]). This would classify the points into three parts. The parameters for the final hidden layer and the output layer are ([100, -100, 50]), ([w31, w41, b21] [w32, w42, b22]).

Q6.

How would you improve the generalization accuracy of the multi-layer neural network?

To improve the accuracy, I add a new input x1*x2, beforehand. The accuracy is 0.955. The reason is the same as the second architecture in question 3.

Code 1:

```
import numpy as np

import matplotlib.pyplot as plt

import math

import time

import random


def der_sig(x):

    t = (1/(1+np.exp(-x)))*(1-(1/(1+np.exp(-x))))

    return t


w1 = np.random.uniform(size=(2, 3))

w2 = np.random.uniform(size=(1, 3))

x1 = np.zeros([1,3])

x2 = np.zeros([1,3])


step1 = 0.03

step2 = 0.03

test_data = np.zeros(8)

test_data[0] = 0.32

test_data[1] = 0.92

test_data[2] = 0.78
```

```
test_data[3] = -1.82

test_data[4] = -0.76

test_data[5] = 1.54

test_data[6] = -0.77

test_data[7] = -1.29

#sigmoid = 1/(1+np.exp())


data = np.loadtxt("HW6_Data.txt")

validate = data[:,2]

training = data[:,0:2]

length = validate.size

fig = plt.figure(figsize=(10,10))



for j in range(1000):

    for i in range(length):

        x1[0,0] = training[i,0]

        x1[0,1] = training[i,1]

        x1[0,2] = 1

        t1 = w1@x1.T #2x1

        x2[0,0] = 1/(1+np.exp(-t1[0,0]))

        x2[0,1] = 1/(1+np.exp(-t1[1,0]))
        x2[0,2] = 1
        t2 = w2@x2.T
        y = 1/(1+np.exp(-t2))
        e = validate[i]-y
        w2 = w2 + step2*e*np.asscalar(der_sig(t2))*x2
        w1                    =                    w1                    +
((step1*np.asscalar(der_sig(t2))*e)*x1.T@np.multiply(der_sig(t1).T,w2[0,0:
2])).T


t = np.arange(-2,2,0.1)
z1 = (-w1[0,0]/w1[0,1])*t + (-w1[0,2]/w1[0,1])
z2 = (-w1[1,0]/w1[1,1])*t + (-w1[1,2]/w1[1,1])

p1 = fig.add_subplot(*[1,1,1])
p1.scatter(training[0:100,0],training[0:100,1],c="b")
p1.scatter(training[100:200,0],training[100:200,1],c="b")
p1.scatter(training[200:300,0],training[200:300,1],c="r")
p1.scatter(training[300:400,0],training[300:400,1],c="r")
p1.plot(t,z1)
p1.plot(t,z2)
```

```
#test data

count = 0

for i in range(400):

    #input layer

    n11 = np.random.uniform(-2, 2)

    n12 = np.random.uniform(-2, 2)

    #final weight

    k11 = w1[0,0]

    k12 = w1[1,0]

    k21 = w1[0,1]

    k22 = w1[1,1]

    v11 = w1[0,2]

    v12 = w1[1,2]

    k31 = w2[0,0]

    k32 = w2[0,1]

    v3 = w2[0,2]

    #hidden layer 1

    p11 = k11*n11 + k21*n12 + v11

    p12 = k12*n11 + k22*n12 + v12

    n21 = 1/(1+np.exp(-p11))

    n22 = 1/(1+np.exp(-p12))

    p3 = k31*n21 + k32*n22 + v3
    n = 1/(1+np.exp(-p3))
    if (n11*n12 >= 0):
        true = 0
        if n<=0.5:
            count = count + 1
    else:
        true = 1
        if n>=0.5:
            count = count + 1
print(count/400)


print('finish')
```

Code 2:

```
import numpy as np
import matplotlib.pyplot as plt
import math
import time
import random


def der_sig(x):
    t = (1/(1+np.exp(-x))).T@(1-(1/(1+np.exp(-x))))
    return t


w1 = 5*np.random.uniform(size=(2, 3))

w2 = 5*np.random.uniform(size=(2, 3))

w3 = 5*np.random.uniform(size=(1, 3))

x1 = np.zeros([1,3])

x2 = np.zeros([1,3])

x3 = np.zeros([1,3])

x = np.zeros(400)

v = np.zeros(400)

step1 = 0.5

step2 = 0.5

step3 = 0.5

test_data = np.zeros(8)

test_data[0] = 0.32

test_data[1] = 0.92

test_data[2] = 0.78

test_data[3] = -1.82

test_data[4] = -0.76

test_data[5] = 1.54

test_data[6] = -0.77

test_data[7] = -1.29
```

```
#sigmoid = 1/(1+np.exp())


data = np.loadtxt("HW6_Data.txt")
validate = data[:,2]
training = data[:,0:2]
length = validate.size
fig = plt.figure(figsize=(20,10))
for j in range(1000):
    for i in range(length):
        x1[0,0] = training[i,0]
        x1[0,1] = training[i,1]
        x1[0,2] = 1
        t1 = w1@x1.T #2x1
        x2[0,0] = 1/(1+np.exp(-t1[0,0]))
        x2[0,1] = 1/(1+np.exp(-t1[1,0]))
        x2[0,2] = 1
        x[i] = x2[0,0]
        v[i] = x2[0,1]
        t2 = w2@x2.T
        x3[0,0] = 1/(1+np.exp(-t2[0,0]))
        x3[0,1] = 1/(1+np.exp(-t2[1,0]))
        x3[0,2] = 1
        t3 = w3@x3.T
        y = 1/(1+np.exp(-t3))
        e = validate[i]-y
        w3 = w3 + step3*e*np.asscalar(der_sig(t3))*x3
        w2 = w2 +
((step2*np.asscalar(der_sig(t3))*e)*x2.T@np.multiply(der_sig(t2).T,w3[0,0:
2])).T
        w1 = w1 +
((step1*np.asscalar(der_sig(t3))*e)*x1.T@np.multiply(der_sig(t1).T,(w2[0,0
:2]*der_sig(t2[0])*w3[0,0]+w2[1,0:2]*der_sig(t2[1])*w3[0,1]))).T
        #w1 = w1 +
(np.matrix(np.asscalar(step1*e*der_sig(t3))*np.multiply(der_sig(t1),(np.m
atrix(w2[:,0:2]).T@np.multiply(der_sig(t2).T,np.matrix(w3[0,0:2]).T))))@x1)
        if j == 999:
            print(y)


t = np.arange(-2,2,0.1)
z11 = (-w1[0,0]/w1[0,1])*t + (-w1[0,2]/w1[0,1])
z12 = (-w1[1,0]/w1[1,1])*t + (-w1[1,2]/w1[1,1])
z21 = (-w2[0,0]/w2[0,1])*t + (-w2[0,2]/w2[0,1])
z22 = (-w2[1,0]/w2[1,1])*t + (-w2[1,2]/w2[1,1])
```

```
p1 = fig.add_subplot(*[2,1,1])

p1.scatter(training[0:100,0],training[0:100,1],c="b")

p1.scatter(training[100:200,0],training[100:200,1],c="b")

p1.scatter(training[200:300,0],training[200:300,1],c="r")

p1.scatter(training[300:400,0],training[300:400,1],c="r")

p1.plot(t,z11)

p1.plot(t,z12)

ymin, ymax = -2, 2

p1.set_ylim([ymin,ymax])


p2 = fig.add_subplot(*[2,1,2])

p2.scatter(x[0:100],v[0:100],c="b")

p2.scatter(x[100:200],v[100:200],c="b")

p2.scatter(x[200:300],v[200:300],c="r")

p2.scatter(x[300:400],v[300:400],c="r")

p2.plot(t,z21)

p2.plot(t,z22)

ymin, ymax = -2, 2

p2.set_ylim([ymin,ymax])


#test data

for i in range(4):
```

```
#input layer

n11 = test_data[i*2]

n12 = test_data[i*2+1]

#final weight

k11 = w1[0,0]

k12 = w1[1,0]

k21 = w1[0,1]

k22 = w1[1,1]

v11 = w1[0,2]

v12 = w1[1,2]

k31 = w2[0,0]

k32 = w2[1,0]

k41 = w2[0,1]

k42 = w2[1,1]

v21 = w2[0,2]

v22 = w2[1,2]

k51 = w3[0,0]

k52 = w3[0,1]

v3 = w3[0,2]

#hidden layer 1

p11 = k11*n11 + k21*n12 + v11

p12 = k12*n11 + k22*n12 + v12
```

```
    n21 = 1/(1+np.exp(-p11))

    n22 = 1/(1+np.exp(-p12))

    p21 = k31*n21 + k42*n22 + v21

    p22 = k32*n21 + k42*n22 + v22

    n31 = 1/(1+np.exp(-p21))

    n32 = 1/(1+np.exp(-p22))

    h = k51*n31 + k52*n32 + v3

    n = 1/(1+np.exp(-h))

    print(n)


count = 0

for i in range(400):

    #input layer

    n11 = np.random.uniform(-2, 2)

    n12 = np.random.uniform(-2, 2)

    #final weight

    k11 = w1[0,0]

    k12 = w1[1,0]

    k21 = w1[0,1]

    k22 = w1[1,1]

    v11 = w1[0,2]

    v12 = w1[1,2]
```

```
k31 = w2[0,0]

k32 = w2[1,0]

k41 = w2[0,1]

k42 = w2[1,1]

v21 = w2[0,2]

v22 = w2[1,2]

k51 = w3[0,0]

k52 = w3[0,1]

v3 = w3[0,2]

#hidden layer 1

p11 = k11*n11 + k21*n12 + v11

p12 = k12*n11 + k22*n12 + v12

n21 = 1/(1+np.exp(-p11))

n22 = 1/(1+np.exp(-p12))

p21 = k31*n21 + k42*n22 + v21

p22 = k32*n21 + k42*n22 + v22

n31 = 1/(1+np.exp(-p21))

n32 = 1/(1+np.exp(-p22))

h = k51*n31 + k52*n32 + v3

n = 1/(1+np.exp(-h))

if (n11*n12 >= 0):

    true = 0
```

```
            if n<=0.5:
                count = count + 1
        else:
            true = 1
            if n>=0.5:
                count = count + 1
print(count/400)
print('finish')
```

Code 3:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 13 14:13:29 2017

@author: JSZJZ
"""


import numpy as np
import matplotlib.pyplot as plt
import math
import time
import random
```

```
def der_sig(x):

      t = (1/(1+np.exp(-x)))*(1-(1/(1+np.exp(-x))))

      return t


w1 = np.random.uniform(size=(2, 4))

w2 = np.random.uniform(size=(1, 3))

x1 = np.zeros([1,4])

x2 = np.zeros([1,3])


step1 = 0.03

step2 = 0.03

test_data = np.zeros(8)

test_data[0] = 0.32

test_data[1] = 0.92

test_data[2] = 0.78

test_data[3] = -1.82

test_data[4] = -0.76

test_data[5] = 1.54

test_data[6] = -0.77

test_data[7] = -1.29

#sigmoid = 1/(1+np.exp())
```

```
data = np.loadtxt("HW6_Data.txt")

validate = data[:,2]

training = data[:,0:2]

length = validate.size

fig = plt.figure(figsize=(10,10))



for j in range(1000):

    for i in range(length):

        x1[0,0] = training[i,0]

        x1[0,1] = training[i,1]

        x1[0,2] = x1[0,0]*x1[0,1]

        x1[0,3] = 1

        t1 = w1@x1.T #2x1

        x2[0,0] = 1/(1+np.exp(-t1[0,0]))

        x2[0,1] = 1/(1+np.exp(-t1[1,0]))

        x2[0,2] = 1

        t2 = w2@x2.T

        y = 1/(1+np.exp(-t2))

        e = validate[i]-y

        w2 = w2 + step2*e*np.asscalar(der_sig(t2))*x2
    w1 = w1 + ((step1*np.asscalar(der_sig(t2))*e)*x1.T@np.multiply(der_sig(t1).T,w2[0,0:2])).T

x1 = np.arange(-2,2,0.1)
x2 = np.arange(-2,2,0.1)
X,Y = np.meshgrid(x1,x2)
Z1 = w1[0,0]*X + w1[0,1]*Y + w1[0,2]*X*Y + w1[0,3]

p1 = fig.add_subplot(*[1,1,1])
p1.scatter(training[0:100,0],training[0:100,1],c="b")
p1.scatter(training[100:200,0],training[100:200,1],c="b")
p1.scatter(training[200:300,0],training[200:300,1],c="r")
p1.scatter(training[300:400,0],training[300:400,1],c="r")
#p1.contour(X,Y,Z1,0)
p1.contour(X,Y,Z1,0)

#test data
count = 0
for i in range(400):
    #input layer
    n11 = np.random.uniform(-2, 2)
```

```
n12 = np.random.uniform(-2, 2)

n13 = n11*n12

#final weight

k11 = w1[0,0]

k12 = w1[1,0]

k21 = w1[0,1]

k22 = w1[1,1]

k31 = w1[0,2]

k32 = w1[1,2]

v11 = w1[0,3]

v12 = w1[1,3]

k41 = w2[0,0]

k42 = w2[0,1]

v3 = w2[0,2]

#hidden layer 1

p11 = k11*n11 + k21*n12 + k31*n13 + v11

p12 = k12*n11 + k22*n12 + k32*n13 + v12

n21 = 1/(1+np.exp(-p11))

n22 = 1/(1+np.exp(-p12))

p3 = k41*n21 + k42*n22 + v3

n = 1/(1+np.exp(-p3))

if (n11*n12 >= 0):
        true = 0
        if n<=0.5:
            count = count + 1
    else:
        true = 1
        if n>=0.5:
            count = count + 1
print(count/400)


print('finish')
```