EE7207 NN Assignment Report

LI RIXUAN G1801134G

Question 1

Assuming that the RBF neural network has 20 neurons in the hidden layer, find center vectors for the 20 neurons using SOM neural network.

My thinking: These 20 center vectors are exactly the weights in SOM neural network. So I initialize weights by normal distribution with 33 (-1,+1) small numbers. Followed by three essential process: Competition, Cooperation, Weights adaptation. In these successive process, we should focus on continuously changing parameters:

$$\sigma(n) = \sigma_0 \exp(-\frac{n}{\tau_1}) \quad ; \ \eta(n) = \eta_0 \exp(-\frac{n}{\tau_2}) \quad ; \ h_{ji}(n) = \exp(-\frac{{d_{ji}}^2}{2\sigma_{(n)}^2})$$

Width parameter $\sigma(n)$ and learning-rate parameter $\eta(n)$ are affected by iteration times n, in this way, neighborhood function is also changing. So the adaptive process: $w_j(n+1) = w_j(n) + \eta(n) h_{ji}(n)[X - w_j(n)]$ is affected by the above continuously changing parameters.

Getting center vectors' python code is as below:

```
import math
import numpy as np

Learn_YITA = 0.1;  max_iteration = 500

train_sample = np.loadtxt('./data_train.csv', dtype=float, delimiter=',')

class Neuron:  #single neuron
    def __init__(self, row, col):
        self.weight = np.random.normal(0.0, 1.0, 33)  # get 33 random weight
        self.row = row
        self.col = col

    def GETdistance(self, sample):
        return np.sqrt(np.sum(np.square(sample- self.weight)))

    def NEWeight(self, sample, Learn_YITA, h):
        self.weight = self.weight + Learn_YITA*h*(sample - self.weight)

class NeuronNET: #neuron sets
    def __init__(self):
        self.NNET = []
        for row in range(0, 4):
            for col in range(0, 5):
                self.NNET.append(Neuron(row, col)) #VIP

    def mindistance(self,sample):
        minD = 0
        min_posi = -1
        for Neu_posi in range(0, 20):
            D = self.NNET[Neu_posi].GETdistance(sample)
            if min_posi == -1:
                minD = D
```

```
                min_posi = Neu_posi
            elif D < minD:
                minD = D
                min_posi = Neu_posi
        return min_posi

# Create A somNN
NN = NeuronNET()
# iterate to get trainned weights
for iteration in range(0, max_iteration):

    for i in range(0, 330):
        sample = train_sample[i, :]
        min_posi = NN.mindistance(sample)
        WINNER = NN.NNET[min_posi]

   # for every neuron get their distance to winner then mew weight could form
        for ID in range(0, 20):
            PosWIN = np.array([WINNER.row, WINNER.col])
            PosEVERY = np.array([NN.NNET[ID].row, NN.NNET[ID].col])
            d = np.sqrt(np.sum(np.square(PosEVERY - PosWIN)))
            delta = 2 * np.exp(-iteration/max_iteration)
            h = np.exp(-d**2/(2*delta**2))
            NN.NNET[ID].NEWeight(sample, Learn_YITA, h)

CV = np.ndarray([20,33])
for i in range(0,20):
    CV[i,:] = NN.NNET[i].weight
np.savetxt('CenterVec.csv', CV, '%.6f', delimiter='  ')
```

Center vectors are saved as a (.csv) document for check and future usage. We could also add some code to see the cluster result during the iteration process for better understanding:

499iteration-> 20, 13, 24, 21, 43, 13, 31, 19, 16, 19, 11, 3, 6, 3, 16, 11, 15, 11, 16, 19,

It's shown that after 500 times iteration the cluster is distributed quite fine. The final center vector is partly shown below:

-1.946981	-1.172470	0.205082	-0.902107	-0.410691	-1.282705	-0.258519	-0.658543	0.002810	-0.470546	-0.470378	-0.7	
-0.477472	-0.601359	-0.029485	-0.679823	-0.348837	-0.688440	-0.542532	-0.231930	-0.197813	-0.170868	-0.456670	-0	
0.309499	-0.010444	-0.126530	0.080613	-0.312401	0.252495	-0.366548	0.265432	-0.461684	0.248304	-0.422040	0.41524	
0.335875	0.378206	-0.087589	0.469862	-0.291267	0.636261	-0.242671	0.643664	-0.423611	0.608222	-0.303406	0.724779	
0.339379	0.538144	-0.071730	0.606988	-0.306420	0.766245	-0.249837	0.790982	-0.381346	0.770056	-0.255876	0.827265	
-0.653437	-1.002814	-0.070964	-1.186656	0.026060	-1.288165	0.334641	-0.941408	0.303672	-0.789827	-0.067405	-0.90	
0.053825	-0.633965	-0.194359	-0.882550	-0.183929	-0.671023	0.006210	-0.480572	-0.011617	-0.412511	-0.199150	-0.4	
0.331173	-0.242183	-0.156706	-0.081006	-0.215259	0.070986	-0.150193	0.017111	-0.355142	0.078163	-0.333055	0.2580	
0.326285	0.163044	-0.128583	0.371995	-0.202504	0.438507	-0.194482	0.431124	-0.359755	0.427847	-0.316548	0.573197	
0.260357	0.421687	-0.113040	0.568763	-0.186431	0.649121	-0.193723	0.681683	-0.327451	0.670343	-0.280641	0.743995	
0.206174	-0.291301	0.446905	-0.653022	0.645148	-0.840567	0.972473	-1.392493	0.950923	-1.360602	0.492493	-1.32412	
0.307239	-0.114933	0.336015	-0.378619	0.536950	-0.577894	0.815133	-0.913386	0.812786	-1.001616	0.620653	-1.09339	
0.316856	-0.069809	0.207641	-0.046421	0.252045	-0.127109	0.429810	-0.343852	0.339307	-0.357841	0.433764	-0.40343	
0.227159	0.142584	-0.003025	0.322706	0.070421	0.264162	0.166707	0.143754	0.036289	0.168448	0.140704	0.231381	0.
0.008900	0.302827	-0.230592	0.526461	0.128006	0.433333	0.229330	0.475592	-0.036377	0.457454	0.005299	0.519097	-
0.324680	0.298961	0.831877	-0.129591	0.992284	-0.590214	1.344371	-1.288511	1.372091	-1.560116	1.012020	-1.667816	
0.342594	0.364090	0.730214	0.067025	0.979720	-0.393161	1.263715	-0.937592	1.330205	-1.267178	1.228040	-1.450685	
0.329842	0.330388	0.654713	0.263480	0.785299	-0.059689	1.028914	-0.467873	1.082801	-0.714021	1.267205	-0.903938	
0.255205	0.434223	0.307652	0.456233	0.434414	0.329888	0.645986	0.095909	0.698129	0.001786	0.935695	-0.115683	1.
0.159015	0.482316	-0.120441	0.569808	0.322588	0.445182	0.454459	0.477733	0.222954	0.441097	0.505624	0.369269	0.

Question 2

Assuming the RBF neural network has only 1 neuron in the output layer, determine the weights from the hidden layer to output layer using the linear least square estimation algorithm.

My thinking: Choose commonly used Gaussian basis function, initial its width σ=1.5. Then just calculate estimated weights $W = (\Phi^T\Phi)^{-1}\Phi^T d$ step by step. Add a bias term to the weights. No iteration need, so it's much straight forward than Question1.

Getting estimated weights' python code part is as below:

```
import math
import numpy as np

input_dim = 33; bias = 1

CenterVec = np.loadtxt('./CenterVec.csv', delimiter='  ', dtype=float)
data_train = np.loadtxt('./data_train.csv',delimiter=',', dtype=float)
label_train = np.loadtxt('./label_train.csv', delimiter=',', dtype=float)
data_test = np.loadtxt('./data_test.csv', delimiter=',', dtype=float)

# claculate PHI--->distance between CenterVec and sample
PHI = np.ndarray([330,21])
for i in range(0, 330):
    PHI_row = []
    for j in range(0,20):
        CV_row = CenterVec[j,:];  deta = 1.5
        distance = 
sum(np.exp(-np.square(CV_row-data_train[i,:]/(2*deta**2))))
        PHI_row.append(distance)
    PHI_row.append(bias)
    PHI[i,:] = np.array(PHI_row,dtype=float)
# using LSE method to calculate WEIGHTS
WEIGHTS = np.linalg.inv(np.transpose(PHI) @ PHI) @ np.transpose(PHI) @ 
label_train
```

The result of weights are:

[0.42823569, -1.25723447, 1.33417983, -0.48274879,\n
-2.90833306, 0.86768626, 0.49103258, -3.17424631,\n
2.41291994, 2.71123575, 0.52821417, -1.67386369,\n
0.7195139 , 1.92755066, -1.5037657 , -0.18649565,\n
0.80467508, 0.11648976, -0.7530848 , 0.11718663,\n -13.86378143])

Question 3

Calculate the classification accuracy of the training data.

The following accuracy calculation codes are appended to codes in Question2:

```
# using data_train to test accuracy
TestVAL = PHI @ WEIGHTS
right = 0; wrong = 0; TestLABLE = 0
for i in range(0,330):
    if TestVAL[i] > 0:
        TestLABLE = 1
```

```
    else:
        TestLABLE = -1
    if TestLABLE == label_train[i]:
        right+=1
    else:
        wrong+=1
accuracy = right/(right+wrong)

print('accuracy = %.5f'%(accuracy))
print('rightNUM = %d  wrongNUM = %d'%(right,wrong))
```

The accuracy result is shown below:

accuracy = 0.94848

rightNUM = 313 wrongNUM = 17

Question 4

Predict the labels for testing data.

The following labels prediction codes are appended to codes in Question3:

```
# predict labels of test data
PHIT = np.ndarray([21,21],dtype=float)
for i in range(0, 21):
    PHIT_row = []
    for j in range(0,20):
        row = CenterVec[j,:]; delta = 1.5
        distance = 
np.sum(np.exp(-np.square(row-data_test[i,:]/(2*delta**2))))
        PHIT_row.append(distance)
    PHIT_row.append(bias)
    PHIT[i,:] = np.array(PHIT_row,dtype=float)
RBFVAL = PHIT @ WEIGHTS

# get predict labels
predictLAB = np.zeros([21], dtype=float)
for i in range(0,21):
    if RBFVAL[i] > 0:
        predictVAL = 1
    else:
        predictVAL = -1
    predictLAB[i] = predictVAL

print('PredictLabel: '); print(predictLAB)
```

The predict labels result is shown below:

PredictLabel:

[1. 1. 1. -1. 1. -1. 1. -1. 1. -1. 1. -1. 1. 1. 1. -1. 1. -1. 1. -1. 1.]