**Continious Assignment Report**

Liu Xiangyu G1802061L, the LiuXiangyuG1802061L\_testLabels\_NNCA.mat file in attachment is my result labels for test set. And the zip file is the source code.

1. **Program Steps**
   1. **Initializaiton**
2. Load datasets.
3. Normalize datasets.
4. Specify the number of neurons and the structure of hidden layer.I use a two-dimensional 4x5 structure.
   1. **SOM Networks to find center vectors**
      1. Specify a maximum number of iterations, and initial center vectors, sigma and eta.
      2. Choose a neighborhood function for SOM.
      3. Calculate two time constants for sigma and eta.
      4. Find the winner neuron.
      5. Shrink the cooperating neighborhood and update the weights.
      6. Repeat 3) 4) and 5) at each iteration.
      7. Use the final weight vectors as center vector of RBF when all iterations end.
   2. **RBF Networks to estimate labels**
      1. Choose a basis function of RBF.
      2. Calculate the output matrix based on the input, the center vectos, and the basis function.
      3. Calculate the weight matrix of RBF
      4. Iterate with diffenrent threshold value of the output value and find the optimal threshold value giving the best accuracy.
5. **Appendix: Key Matlab Scripts**

**2.1 RBF.m**

**This script is the entrance of program.**

close all

clear

clc

load('data\_test.mat')

load('data\_train.mat')

load('label\_train.mat')

numNrn=20;

nSomRows=4;

nSomCols=5;

[numSmp, numDim]=size(data\_train);

dataNorm=zeros(numSmp,numDim);

for j=1:numDim

dataNorm(:,j)=(data\_train(:,j)-min(data\_train(:,j)))/range(data\_train(:,j));

end

for j=1:size(data\_test,2)

dataTestNorm(:,j)=(data\_test(:,j)-min(data\_test(:,j)))/range(data\_test(:,j));

end

[cVec, clusterId]=SOM(numNrn, nSomRows, nSomCols, dataNorm, 1000);

gMethod='Gaussian';

gWidth=1;

trainRange=1:330;

testRange=1:330;

rbfW=RBFTrainWeight(numNrn, nSomRows, nSomCols, cVec, dataNorm(trainRange,:), label\_train(trainRange,:), gMethod, gWidth);

[output,rbfValues]=RBFTest(cVec, rbfW, dataNorm(testRange,:), gMethod, gWidth);

results=[0,-1];

for threshold=min(rbfValues):0.001:max(rbfValues)

labels=Threshold(rbfValues, threshold);

correct=0;

correctLabels=label\_train(testRange,:);

for i=1:size(labels, 1)

if labels(i)==correctLabels(i)

correct=correct+1;

end

end

results=[results; [threshold, double(correct)/size(labels,1)]];

end

[maxAccuracy, id]=max(results(:,2));

threshold=results(id,1);

labels=Threshold(rbfValues, threshold);

correct=0;

correctLabels=label\_train(testRange,:);

for i=1:size(labels, 1)

if labels(i)==correctLabels(i)

correct=correct+1;

end

end

double(correct)/size(labels,1)

[output, testValues]=RBFTest(cVec, rbfW, dataTestNorm, gMethod, gWidth);

testLabels=Threshold(testValues, threshold);

**2.2 SOM.m**

**This function starts SOM training process.**

function [cVec, clusterId] = SOM(numNrn, nSomRows, nSomCols, dataNorm, itMax)

[numSmp, numDim]=size(dataNorm);

cVec=rand(numNrn,numDim);

[o(:,1),o(:,2)]=ind2sub([nSomRows,nSomCols],1:numNrn);

sigma0=1;

eta0=0.1;

for it=1:itMax

for n=1:numSmp

dataCur=dataNorm(n,:);

SOMTrain(cVec, o, dataCur, eta0, sigma0, it, itMax);

end

end

clusterId=zeros(numSmp,1);

for i=1:numSmp

clusterId(i)=SOMCompete(dataNorm(i,:), cVec);

end

**2.3 SOMTrain.m**

**Training process at each iteration.**

function w = SOMTrain(w, o, x, eta0, sigma0, it, itMax)

winId=SOMCompete(x, w);

t1=itMax;

t2=itMax/log10(sigma0);

etaN=eta0\*exp(-it/t1);

sigmaN=sigma0\*exp(-it/t2);

for i=1:size(w,1)

h=exp(-sum((o(i,:)-o(winId,:)).^2)/(2\*sigmaN^2));

dw=etaN\*h\*(x-w(i,:));

w(i,:)=w(i,:)+dw;

end

**2.4 RBFTrainWeights.m**

**This function calculates the weights of RBF.**

function rbfW = RBFTrainWeight(numNrn, nSomRows, nSomCols, cVec, dataNorm, lable\_train, gMethod, gWidth)

d=zeros(size(dataNorm, 1), numNrn);

for i=1:size(dataNorm, 1)

xs=repmat(dataNorm(i,:), numNrn, 1);

d(i,:)=EuclidDist(xs, cVec);

end

Phi=zeros(size(dataNorm, 1),1);

if gMethod=='Gaussian'

Phi=exp(-d.^2./(2\*gWidth^2));

end

d=lable\_train;

rbfW=inv(transpose(Phi)\*Phi)\*transpose(Phi)\*d;