EE5904 NEURAL NETWORKS

HOMEWORK THREE

Name: Yu Shixin

Matriculation Number.: A0195017E

Date:2019/3/15

Q1. Function Approximation with RBFN

(a) According to Fig.1, we can see that it, the line by using test set to evaluate the RBFN performance, is not very fitting the ideal output with some fluctuation. The MSE of train set is $\mathbf{6.4824 \times 10^{-28}}$ and the MSE of test set is **0.0721**. Thus, the result suggests that the sample train data is **over-fitting**, due to the RBFN model exactly fitting every sample in train set.

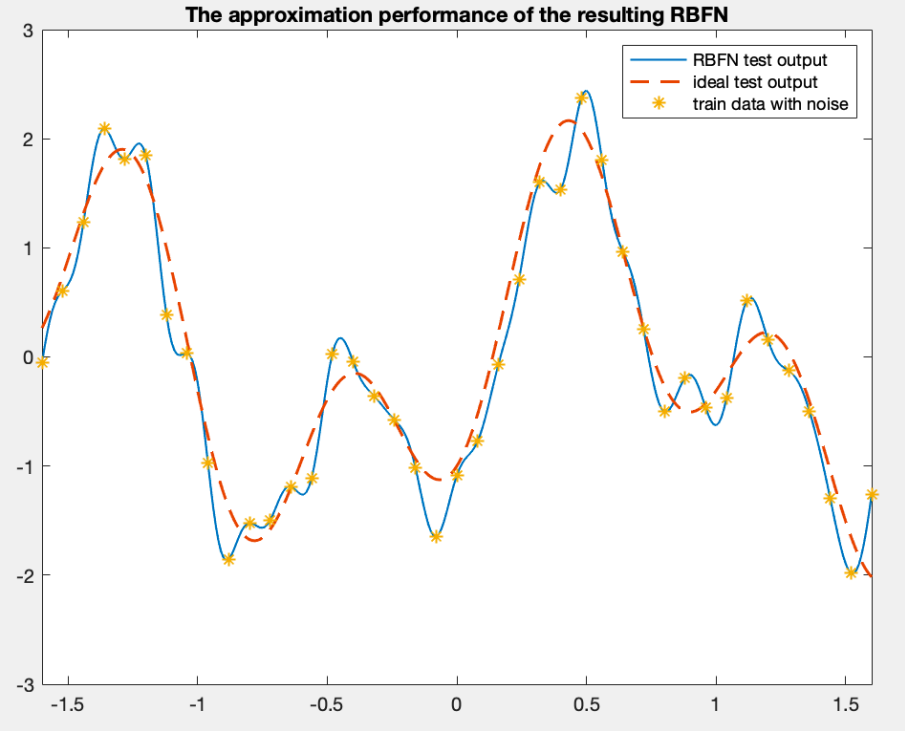


Fig.1 The approximation performance of the resulting RBFN

Code.1 RBFN

```
%parameter
x_train=-1.6:0.08:1.6;%uniform step 0.08
x_test=-1.6:0.01:1.6;%uniform step 0.01
N=length(x_train);
x=randn(1,N);%random Gaussian noise for xtain not for xtest
d=1.2*sin(pi*x_train)-cos(2.4*pi*x_train)+0.3*x;%x with noise
%calculate phi
phi=zeros(N,N);%initialize phi
for i=1:N
    for j=1:N
        r=x_train(i)-x_train(j);
        phi(i,j)=exp(r^2/(-0.02));
    end
end
w=pinv(phi)*d';%get the unique solution w
%test data
phi_test=zeros(length(x_test),N);%initialize phi_test
for i=1:length(x_test)
```

```
    for j=1:N
        r=x_test(i)-x_train(j);
        phi_test(i,j)=exp(r^2/(-0.02));
    end
end
d_test=phi_test*w;
ideal_test=1.2*sin(pi*x_test)-cos(2.4*pi*x_test);
error_train=sum((d-(phi*w)').^2)/N;%mse
error_test=sum((ideal_test-d_test').^2)/length(x_test);
```

(b) In this case, I use the strategy of "Fixed Centers Selected at Random". So I choose 20 points randomly in train set as the centers, and then calculate the coefficient $(-\frac{M}{d_{max}^2}), d_{max} = \max(M) - \min(M)$. Comparing the result with part a, we can find that approximation output of test set by using fixed centers is closer to ideal output than that of test set without fixed centers, which is not only smooth, but also less fluctuant around the ideal output. Thus, the fixed centers can make the approximation performance better. The MSE of train set is **0.0439** and the MSE of test set is **0.0383**. The difference with part a is that the MSE of train set is bigger than that in part a, which means the decrease of overfitting degree, not all the train sample being fitted. The approximation performance and the MATLAB code show as following.

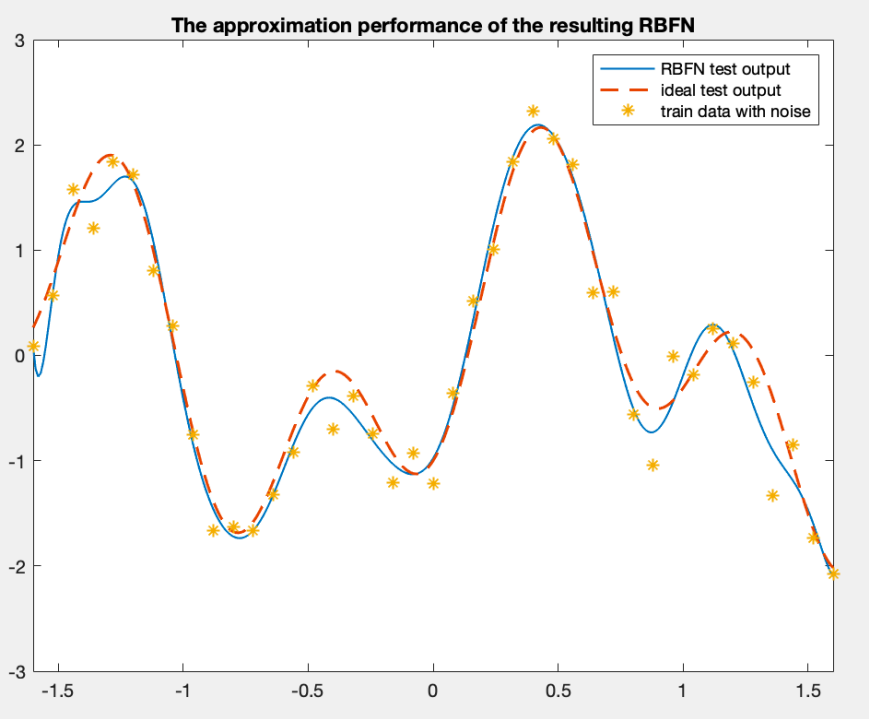


Fig.2 The approximation performance of the resulting RBFN using FXSR

Code.2 RBFN with Fixed Centers Selected at Random

```
%parameter
x_train=-1.6:0.08:1.6;%uniform step 0.08
x_test=-1.6:0.01:1.6;%uniform step 0.01
N=length(x_train);
x=randn(1,N);%random Gaussian noise for xtain not for xtest
d=1.2*sin(pi*x_train)-cos(2.4*pi*x_train)+0.3*x;%x with noise
%calculate phi
rand_index=randperm(N,20);% randomly choose centres 20
M=x_train(rand_index);
coef=length(M)/(-(max(M)-min(M))^2);
phi=zeros(N,length(M));%initialize phi
for i=1:N
    for j=1:length(M)
        r=x_train(i)-M(j);
        phi(i,j)=exp(coef*r^2);
    end
end
phi=[ones(N,1),phi];% bias
w=pinv(phi)*d';%get the unique solution w
%test data
phi_test=zeros(length(x_test),length(M));%initialize phi_test
for i=1:length(x_test)
    for j=1:length(M)
        r=x_test(i)-M(j);
        phi_test(i,j)=exp(coef*r^2);
    end
end
phi_test=[ones(length(x_test),1),phi_test];
d_test=phi_test*w;
ideal_test=1.2*sin(pi*x_test)-cos(2.4*pi*x_test);
error_train=sum((d-(phi*w)').^2)/N;%mse
error_test=sum((ideal_test-d_test').^2)/length(x_test);
```

(c) Apply regulation method in part a, we can find that

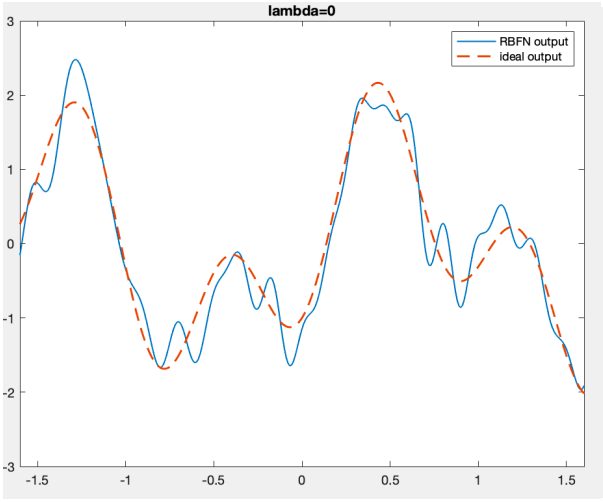
1. It is fluctuant, when lambda=0, which means that it do not exist the regulation. But only a little λ(0.0001), we can find the curve is smoother than that when $\lambda = 0$.

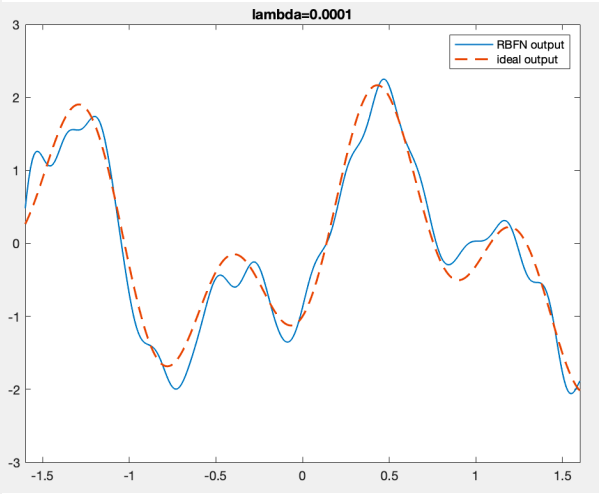
2. With the increase of lambda, we find the curve become more and more smooth. When $\lambda = 1$, the curve is quite smooth than before.

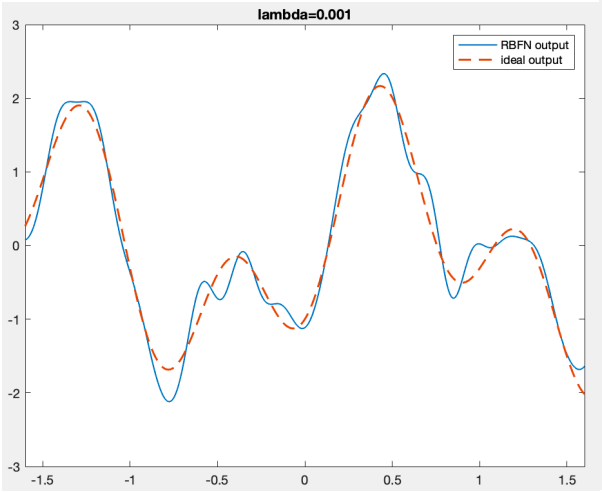
3. However, when the λ big enough (10,50,100), we find that the smoothness constraint dominates and less account is taken for training and test data error.

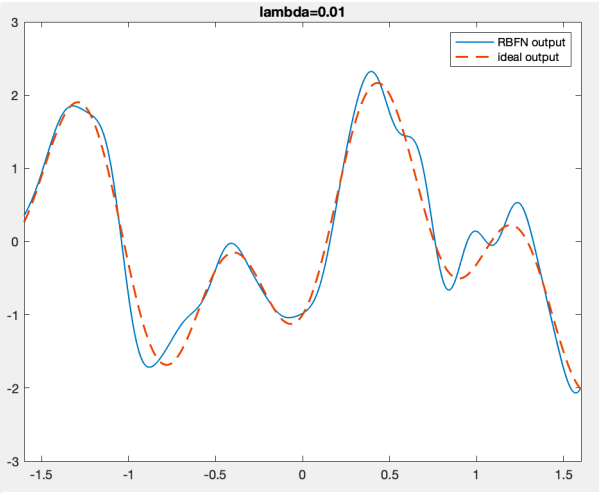
4. The more λ is, the larger MSE of the test is. In this case, when $\lambda = 1000$, the MSE of train data and the MSE of test data increase up to **1.3463** and

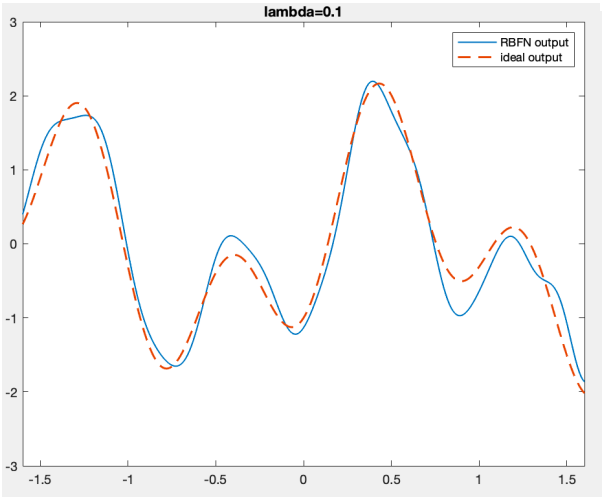
1.2289. So the increase of λ causes the **under-fitting** in RBFN output using test data.

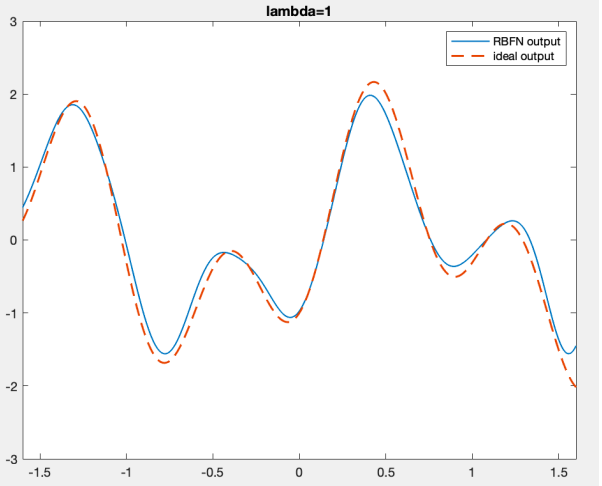


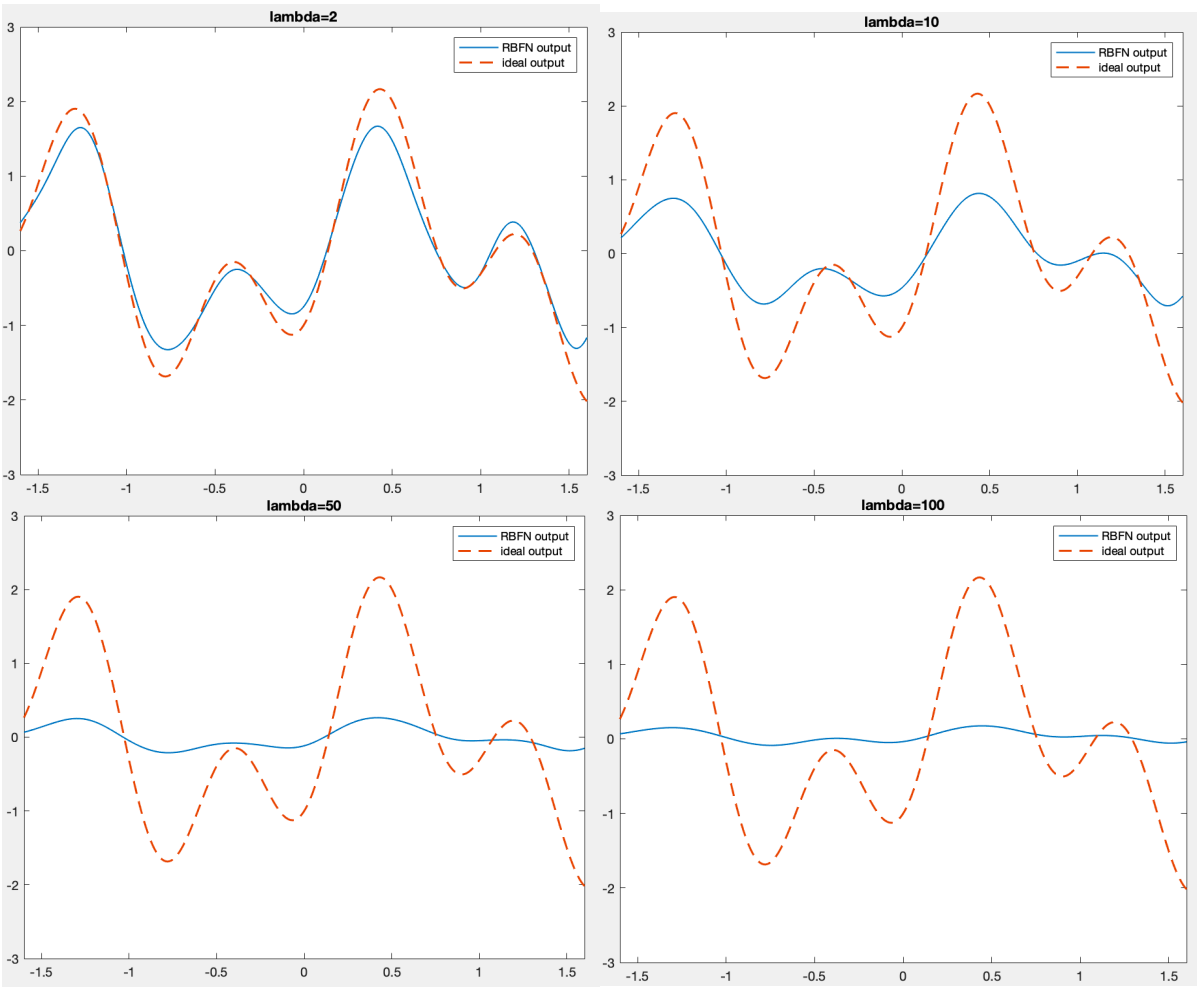












Code.3 RBFN with regulation

```
lambda=1000;%regulation factor
%calculate phi
phi=zeros(N,N);%initialize phi
for i=1:N
    for j=1:N
        r=x_train(i)-x_train(j);
        phi(i,j)=exp(r^2/(-0.02));
    end
end
phi=[ones(N,1),phi];
w=pinv(phi'*phi+lambda*eye(N+1))*phi'*d';%get the unique solution w
%test data
phi_test=zeros(length(x_test),N);%initialize phi_test
for i=1:length(x_test)
    for j=1:N
        r=x_test(i)-x_train(j);
        phi_test(i,j)=exp(r^2/(-0.02));
    end
end
phi_test=[ones(length(x_test),1),phi_test];
d_test=phi_test*w;
ideal_test=1.2*sin(pi*x_test)-cos(2.4*pi*x_test);
error_train=sum((d-(phi*w)').^2)/N;%mse
error_test=sum((ideal_test-d_test').^2)/length(x_test);
```

Q2. Handwritten Digits Classification using RBFN

My matriculation number is **A0195017E**, so I choose classes **1** and **7** to be assigned the label "1", and the remaining classes to be assigned the label "0".

(a) In this part, I use Exact Interpolation Method and apply regulation, given the Gaussian function of RBF, with standard deviation of 100.

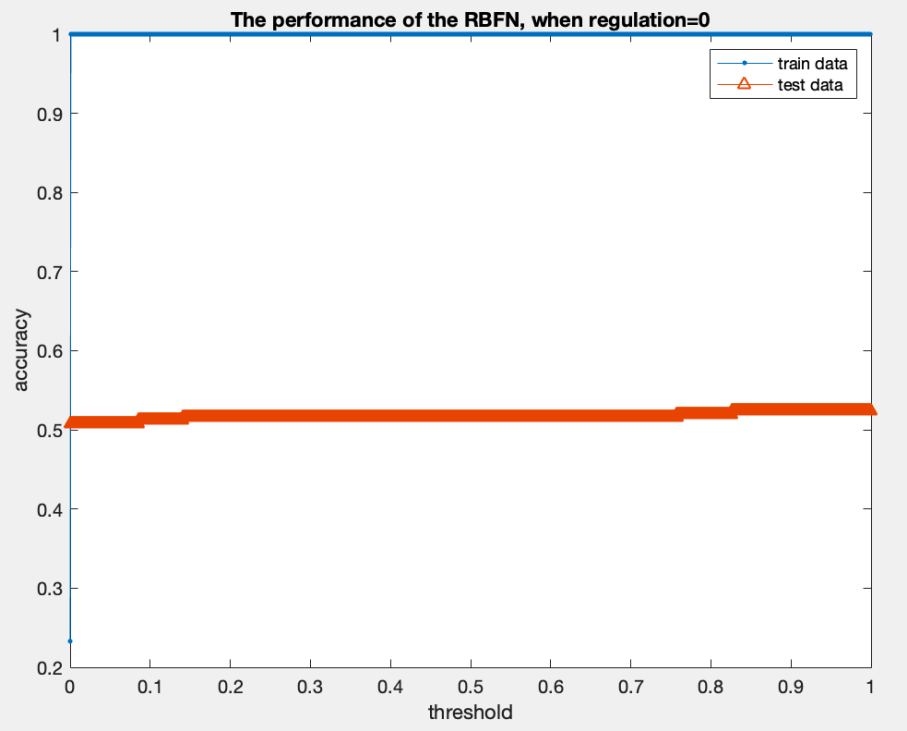
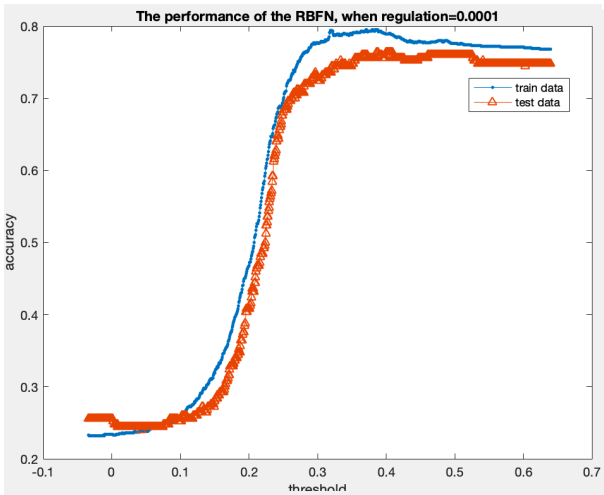
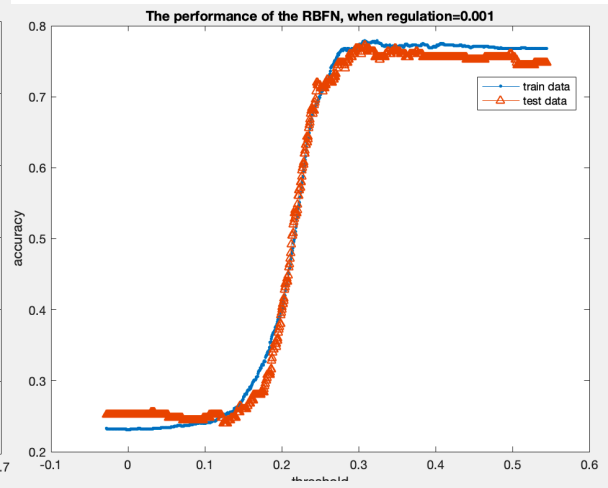
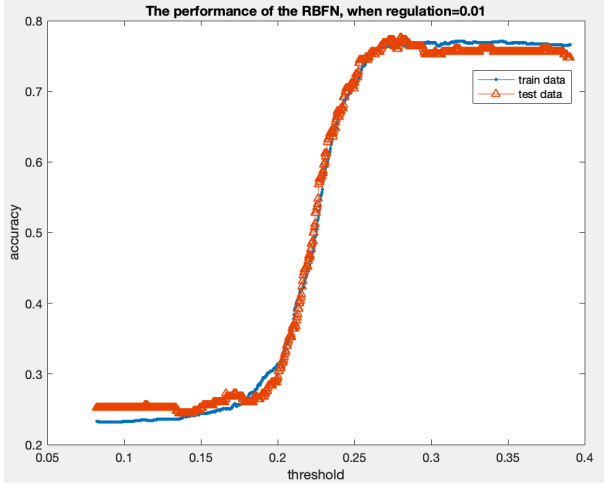


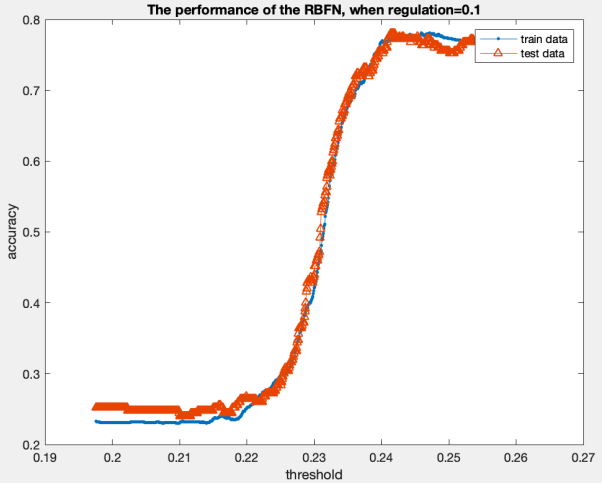
Fig.3 The performance of the RBFN without regulation

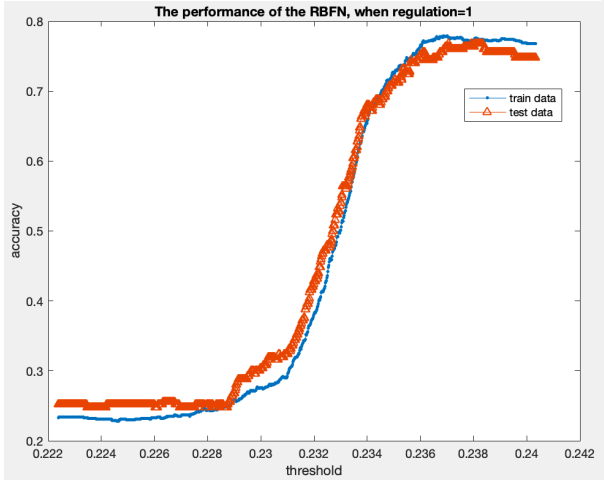
The Fig.3 is the accuracy both train set and test set using RBFN without regulation. The results suggest that the accuracy of train set (23.3%) is lower than that of test set (50.8%), but jump to 100% accuracy once threshold not 0, and keeps until the end. However, the accuracy of test set does not change more, only form 50.8% to 52.4%. Thus, the performance of the RBFN in test set is very bad.

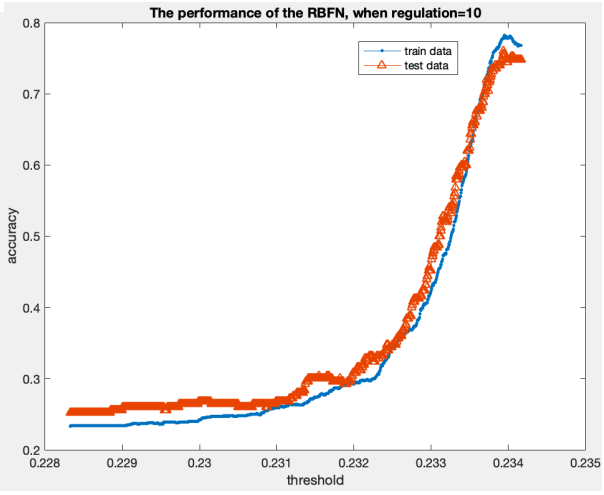

The performance of the RBFN, when regulation=0.0001
train data
test data
accuracy
threshold

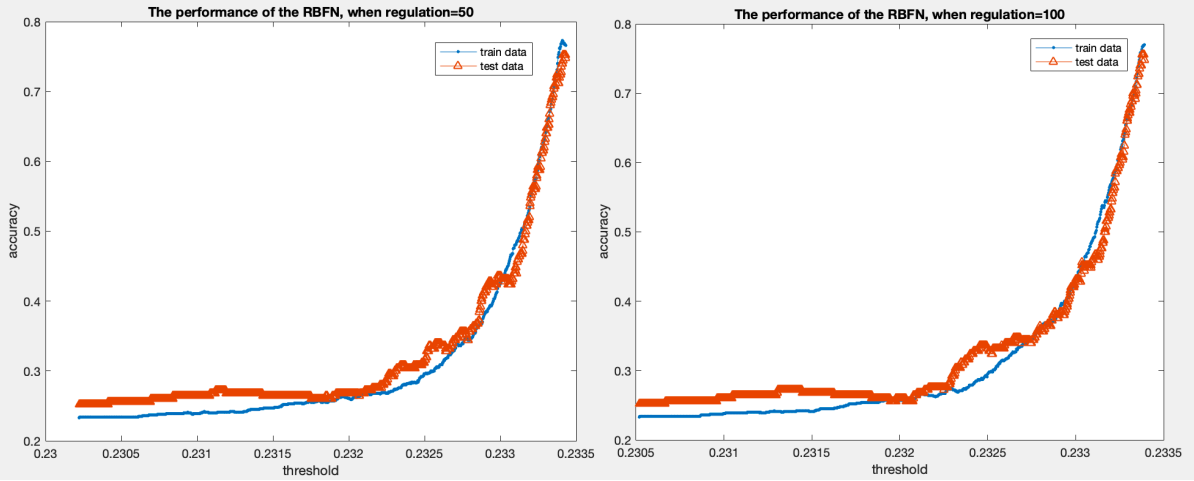

The performance of the RBFN, when regulation=0.001
train data
test data
accuracy

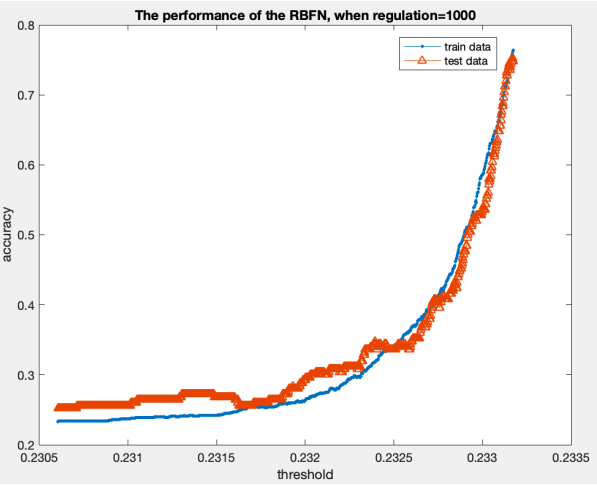

The performance of the RBFN, when regulation=0.01
train data
test data
accuracy
threshold


The performance of the RBFN, when regulation=0.1
train data
test data
accuracy
threshold


The performance of the RBFN, when regulation=1
train data
test data
accuracy
threshold


The performance of the RBFN, when regulation=10
train data
test data
accuracy
threshold





The figures above imply that with the increase of the value of regulation, the accuracy of test set looks like closer to that of train set, and the range of the threshold becomes **narrower**. However, if we print the accuracy of both (table.1), we find that the accuracy of both **do not change too much**, actually! The accuracy of train set fluctuates between **0.795** and **0.764**, and that of test set fluctuates between **0.78** and **0.752**. But we can find that the **increase of the regulation** reduces the overfitting problem, due to the train set accuracy curve is more similar to that of test set, which is higher than accuracy of test data when regulation is small.

Table.1 The accuracy of train set and test set in different regulation factor

	0	0.0001	0.001	0.01	0.1	1	10	50	100	1000
train	0.233	0.795	0.779	0.771	0.781	0.779	0.782	0.773	0.77	0.764
test	0.524	0.764	0.772	0.776	0.780	0.768	0.760	0.752	0.756	0.752

Code.4 2.a

```
%load files
load('mnist_m.mat');

N=length(train_classlabel);
N_test=length(test_classlabel);
trainIdx=find(train_classlabel==1|train_classlabel==7); %find the location
of classes 1,7
testIdx=find(test_classlabel==1|test_classlabel==7);
TrLabel=zeros(1,N);
TeLabel=zeros(1,N_test);
TrLabel(trainIdx)=1;%the selected 2 classes should be assigned the label 1
TeLabel(testIdx)=1;
lambda=1000;%regulation factor
phi=zeros(N,N);
for i=1:N
    for j=1:N
        r=dot(train_data(:,i)-train_data(:,j),train_data(:,i)-
train_data(:,j));
        phi(i,j)=exp(r/(-20000));
    end
end
phi=[ones(N,1),phi];
w=pinv(phi'*phi+lambda*eye(N+1))*phi'*TrLabel';%get the unique solution w
% w=pinv(phi)*TrLabel';
TrPred=phi*w;
phi_test=zeros(N_test,N);%initialize phi_test
for i=1:N_test
    for j=1:N
        r=dot(test_data(:,i)-train_data(:,j),test_data(:,i)-
train_data(:,j));
        phi_test(i,j)=exp(r/(-20000));
    end
end
phi_test=[ones(N_test,1),phi_test];
TePred=phi_test*w;
```

(b) Because of comparing with the result of a, I use the same width size with standard deviation of 100 and regulation factor with 0.

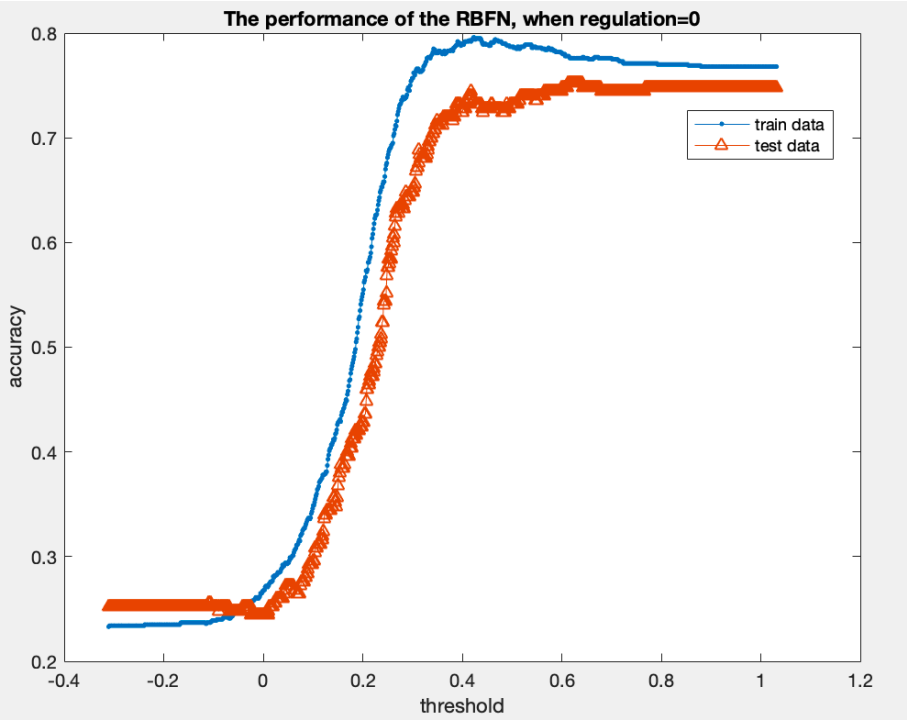


Fig.4 The performance of the RBFN without regulation

Compared with Fig.3 in Q2.a, the accuracy of train set is lower than that in part a, but the accuracy of test set is higher than that in part a. Then I vary the value of width form 0.1 to 10000. The results imply that the accuracy of train set starts at 79.6%, and jumps to 87.8% when width = 7.1. And the accuracy of test set shows a similar trend that starts at 74.8% and reaches the highest point 84% at the same width. With the increase of width, both of them show a downward trend, so a proper width can improve the performance of the RBFN.

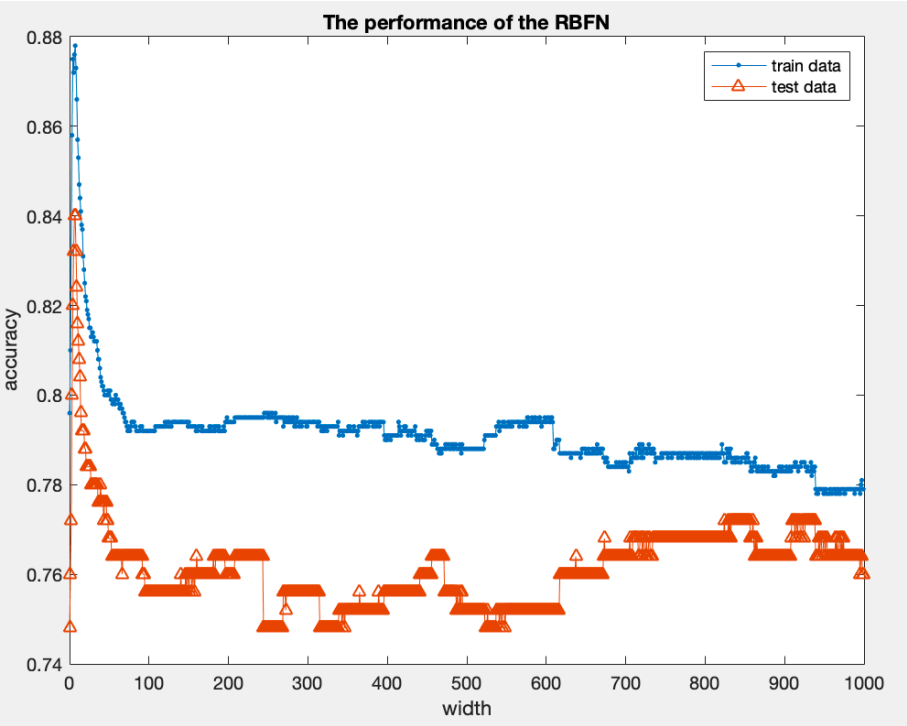


Fig.5 The performance of the RBFN (width 0.1 to 1000)

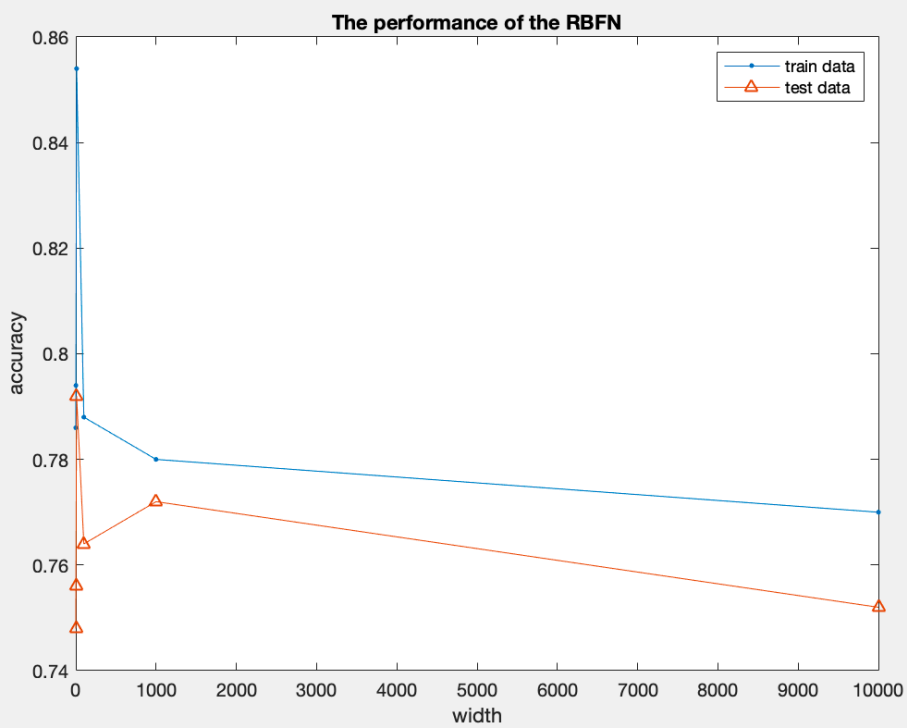


Fig.6 The performance of the RBFN (width 0.1 to 10000)

Code.5 2.b

```
rand_index=randperm(N,100);% randomly choose centres
M=train_data(:,rand_index);
countM=size(M,2);

wid=[0.1,1,10,100,1000,10000];
atr=zeros(1,length(wid));
```

```
ate=zeros(1,length(wid));
for count=1:length(wid)
    width=wid(count);
phi=zeros(N,countM);
for i=1:N
    for j=1:countM
        r=dot(train_data(:,i)-M(:,j),train_data(:,i)-M(:,j));
        phi(i,j)=exp(r/(-2*width.^2));
    end
end
phi=[ones(N,1),phi];
w=pinv(phi'*phi+lambda*eye(countM+1))*phi'*TrLabel';%get the unique
solution w
TrPred=phi*w;
%test data
phi_test=zeros(N_test,countM);%initialize phi_test
for i=1:N_test
    for j=1:countM
        r=dot(test_data(:,i)-M(:,j),test_data(:,i)-M(:,j));
        phi_test(i,j)=exp(r/(-2*width.^2));
    end
end
phi_test=[ones(N_test,1),phi_test];
TePred=phi_test*w;
```

(c) Apply “K-Mean Clustering” with 2 centers, we get 2 centers visualized in Fig.7. Although my classes are 1&7, I get the final image which is similar to “8”. Then I visualize the mean of training images of each label, and get mean 1 and mean 0. We can find that mean 0 is more like center 1&2, and mean 1 is similar to a combination of 1&7. This is because that the label 1 mixes 1&7, and the label 0 is the remainder. Thus, for mean 1, it is very reasonable to be similar with 1&7. But if we assign images 1 as label 1 and images 7 as label 0, and select randomly center 1 in label 1 and center 2 in label 0, we can get more specific images in center 1&2 and mean 1&0.

According to Fig.8, the result imply that the “K-Mean Clustering” shows a good performance, which reaches a high accuracy by using only 2 centers (100 centers in part b), with the accuracy of train set (76.6%) and the accuracy of test set (74.8%). But if we assign label in the way I said above, the accuracy of both may improve in theory.

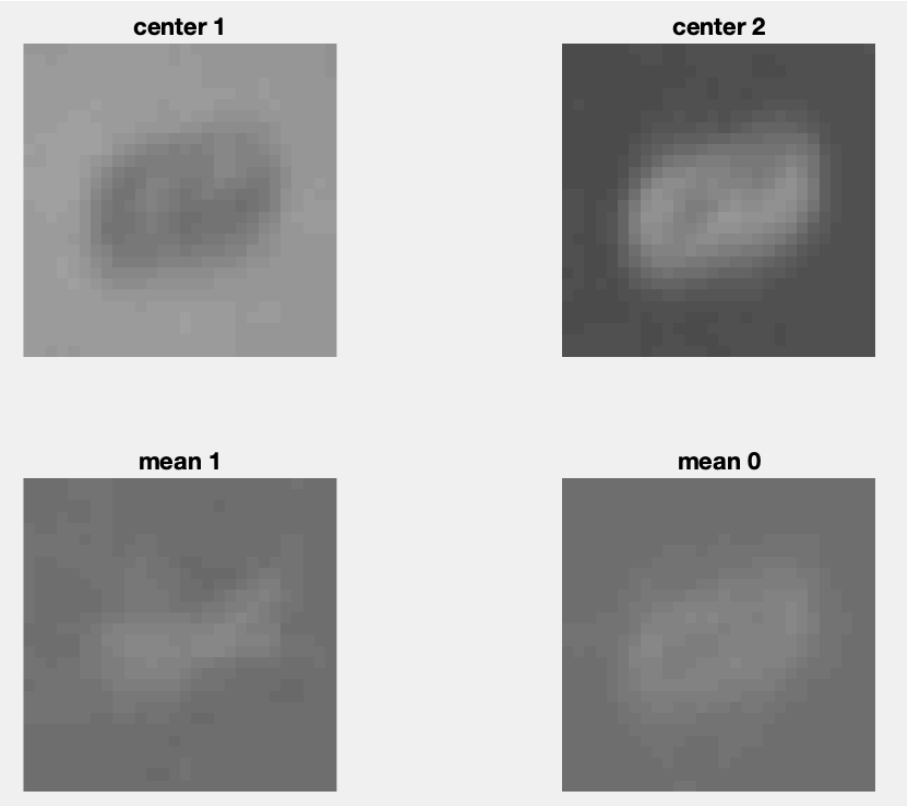


Fig.7 “K-Mean Clustering” center1&2 mean center 1&0

Fig.8 The performance of the RBFN

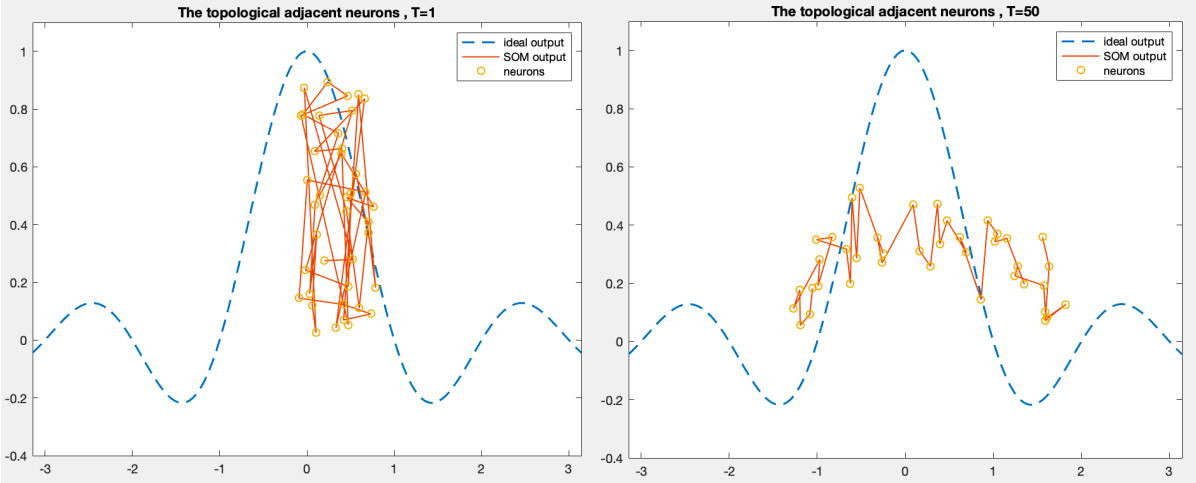
Code.6 “K-Mean Clustering”

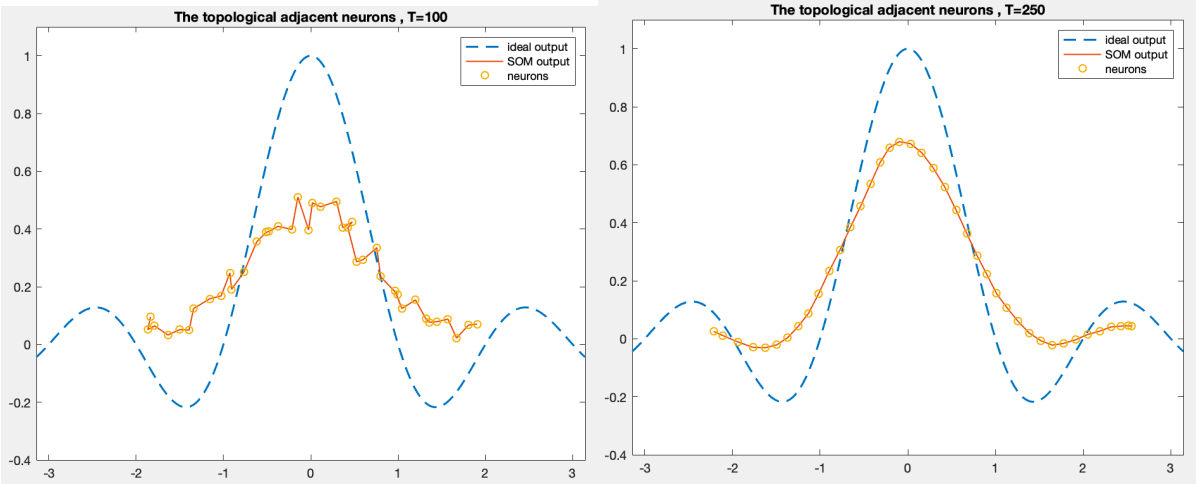
```
%train set
rand_index=randperm(N,2);% randomly choose 2 centres
center=double(train_data(:,rand_index));
trainIdx_0=find(train_classlabel==0|train_classlabel==2|train_classlabel==3
```

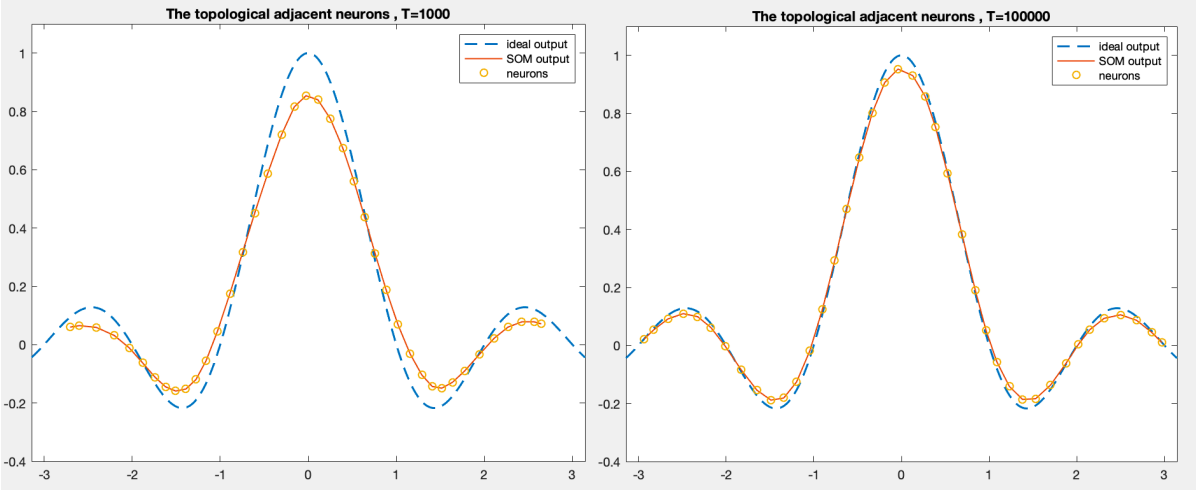
```
|train_classlabel==4|train_classlabel==5|train_classlabel==6|train_classlab
el==8|train_classlabel==9);
for i=1:100 %loop untill convergence
    distance1=dist(train_data',center(:,1));%calculate train data distance
to 1,2
    distance2=dist(train_data',center(:,2));
    center1ind=find(distance1>distance2);
    center2ind=find(distance1<distance2);
    cluster1=train_data(:,center1ind);%divide into 2 part
    cluster2=train_data(:,center2ind);

    error_dist_1=sum(dist(cluster1',center(:,1)))/length(cluster1);
    error_dist_2=sum(dist(cluster2',center(:,2)))/length(cluster2);

    center(:,1)=mean(cluster1,2);
    center(:,2)=mean(cluster2,2);
    cluster1=[];
    cluster2=[];
end
figure(1)
tmp1=reshape(center(:,1),28,28);
subplot(2,2,1);
imshow(tmp1);
title('center 1');
tmp2=reshape(center(:,2),28,28);
subplot(2,2,2);
imshow(tmp2);
title('center 2');

train_1=double(train_data(:,trainIdx));
mean_1=mean(train_1,2);
subplot(2,2,3);
imshow(reshape(mean_1,28,28));
title('mean 1');
train_0=double(train_data(:,trainIdx_0));
mean_0=mean(train_0,2);
subplot(2,2,4);
imshow(reshape(mean_0,28,28));
title('mean 0');
```

Q3. Self-Organizing Map (SOM)

(a) Implement a SOM that maps a 1-dimensional output layer of 40 neurons. I experiment it in different T.







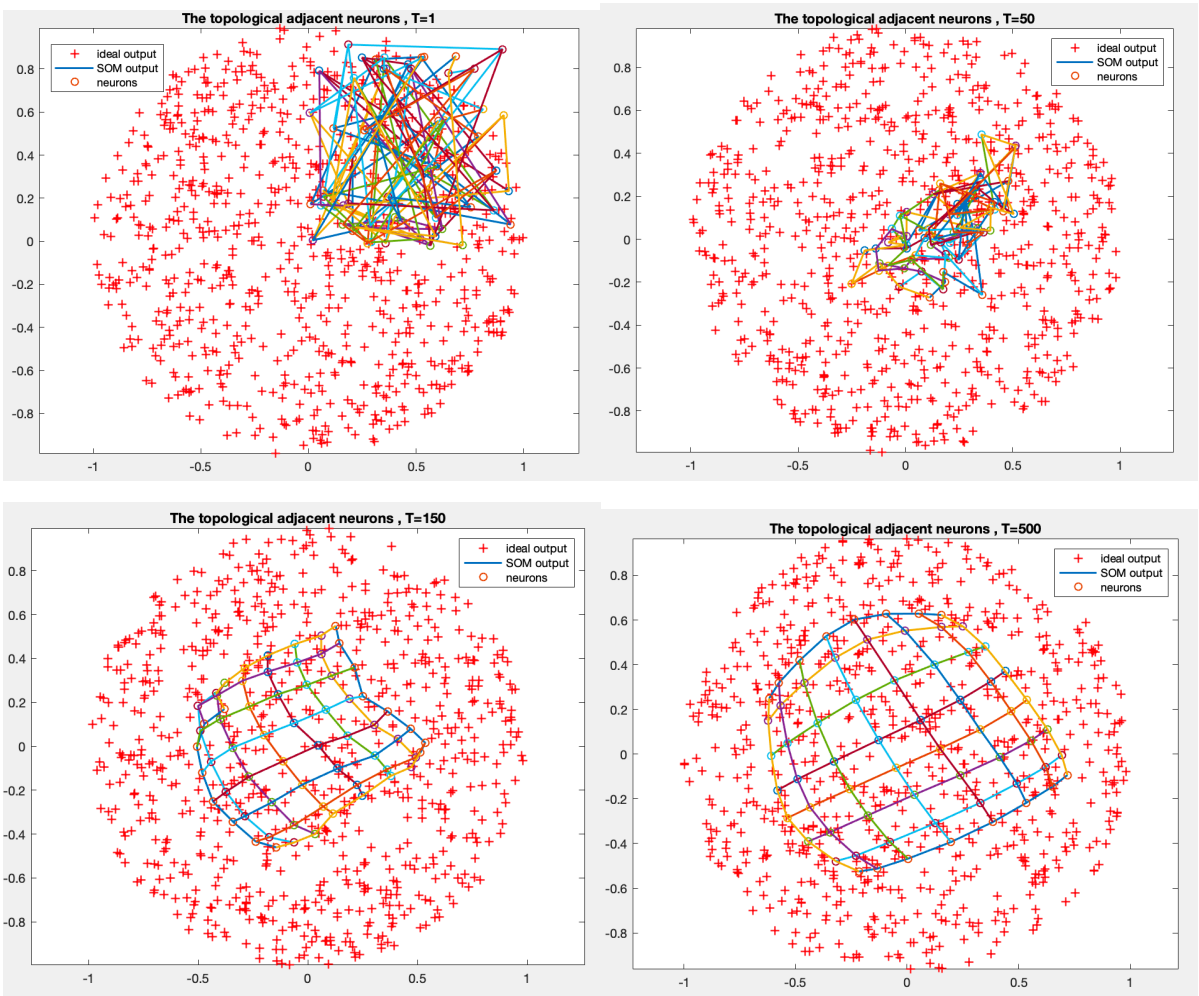
The more epoch is, the better fitting.

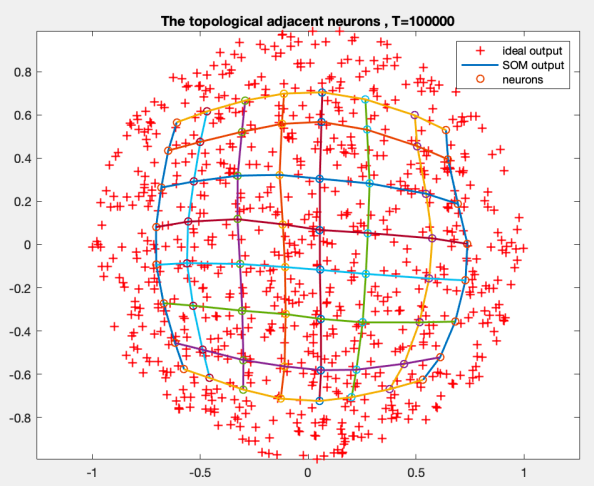
Code.7 SOM

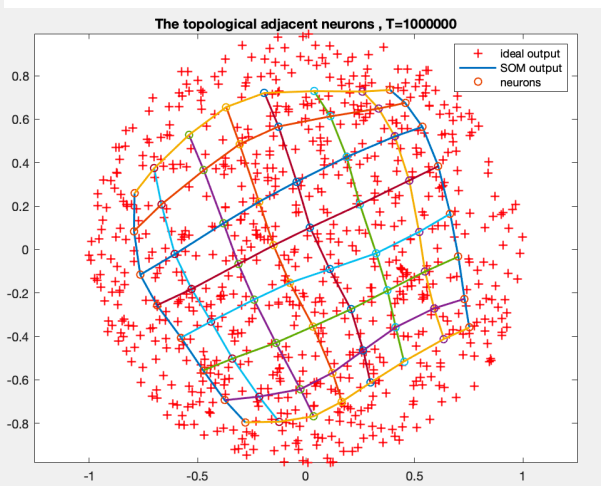
```
%parameter
w=rand(40,2); %randomly init weigth 40 neurons in output layer
sigma0=sqrt(1^2+40^2)/2;%M=1,N=40
```

```
eta0=0.1;
T=100000;%iterations
tau1=T/log(sigma0);
tau2=T;
eta=eta0;
sigma=sigma0;

%algorithm
for n=1:T
    i=randperm(size(trainX,2),1);%randomly select vector x
    %competitive process
    [min_dist,Idx]=min(dist(trainX(:,i)',w'));% 1*2 * 2*40 =1*40
    %adaptation process
    for j=1:40
       h=exp((j-Idx).^2/-(2*sigma.^2));
       w(j,:)=w(j,:)+eta*h*(trainX(:,i)'-w(j,:));
    end
    %update eta&sigma
    eta=eta0*exp(-n/tau2);
    sigma=sigma0*exp(-n/tau1);
end
```

(b) Implement a SOM that maps a 2-dimensional output layer of 64 neurons. I experiment it in different T.







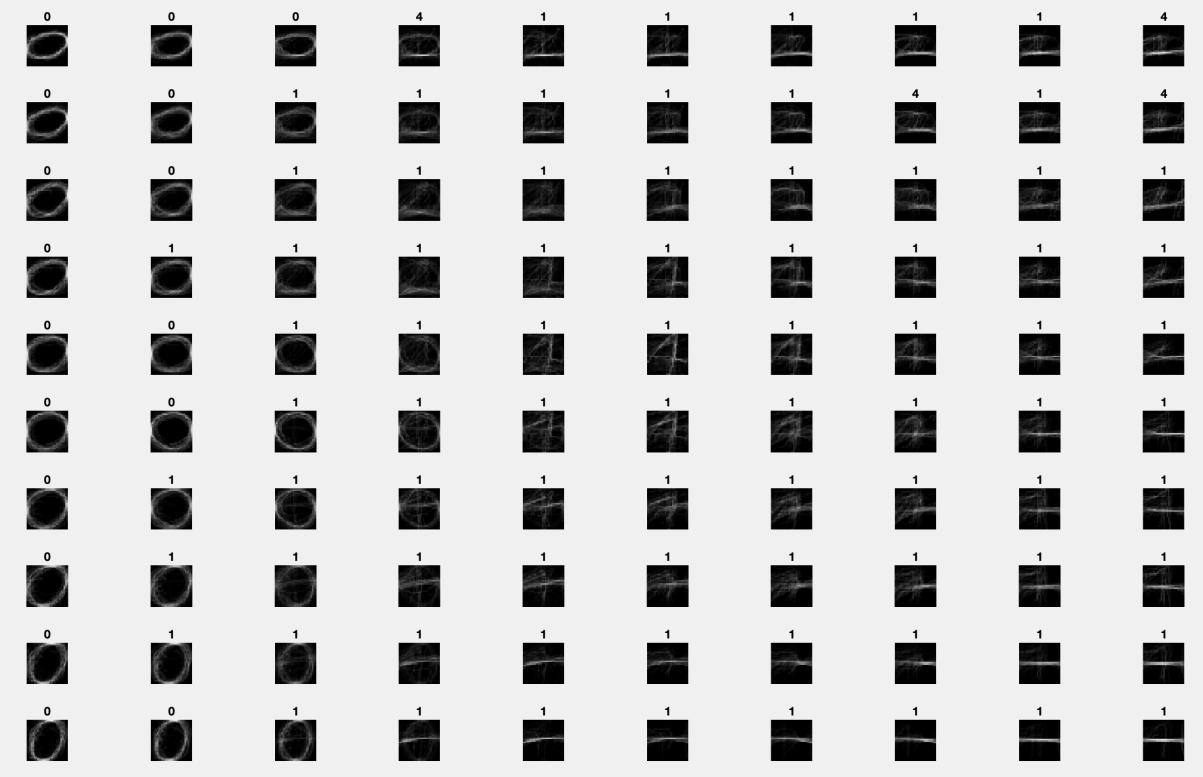
We can find that with the increase of the T, the neurons' distribution is more like a circle, and get better performance.

Code.8 SOM

```
%parameter
w=rand(2,8,8);%randomly init weigth 64 neurons in output layer
sigma0=sqrt(8^2+8^2)/2;%M=8 N=8
eta0=0.1;
T=1000000;%iterations
tau1=T/log(sigma0);
tau2=T;
eta=eta0;
sigma=sigma0;

%algorithm
for n=1:T
    i=randperm(size(trainX,2),1);%randomly select vector x
    %competitive process
    distance=zeros(8,8);
    for row=1:8
        for col=1:8
            distance(row,col)=sqrt((trainX(1,i)-
w(1,row,col)).^2+(trainX(2,i)-w(2,row,col)).^2);
        end
    end
    [min_row,min_col]=find(distance==min(min(distance)));
    %adaptation process
    for row=1:8
        for col=1:8
            h=exp(((row-min_row).^2+(col-min_col).^2)/-(2*sigma.^2));
            w(:,row,col)=w(:,row,col)+eta*h*(trainX(:,i)-w(:,row,col));
        end
    end
    %update eta&sigma
    eta=eta0*exp(-n/tau2);
    sigma=sigma0*exp(-n/tau1);
end

%plot
figure(1)
plot(trainX(1,:),trainX(2,:),'+r');hold on;
```

```
axis equal;
for i=1:8
    plot(w(1,:,i),w(2,:,i),'LineWidth',1.5);
    scatter(w(1,:,i),w(2,:,i),'o');
    hold on;
end
for j=1:8
    w_1 = reshape(w(1,j,:),1,8);
    w_2 = reshape(w(2,j,:),1,8);
    plot(w_1,w_2,'LineWidth',1.5);
    hold on;
end
title(['The topological adjacent neurons , T=',num2str(T)]);
legend('ideal output','SOM output','neurons');
```

(c) My matriculation number is **A0195017E**, so I omit classes **2** and **3**, and use 0,1,4 classes to experiment. The corresponding conceptual map of the trained SOM and visualization of trained weights of each output neuron on a 10*10 map are displayed as below. (I only display the best performance.)

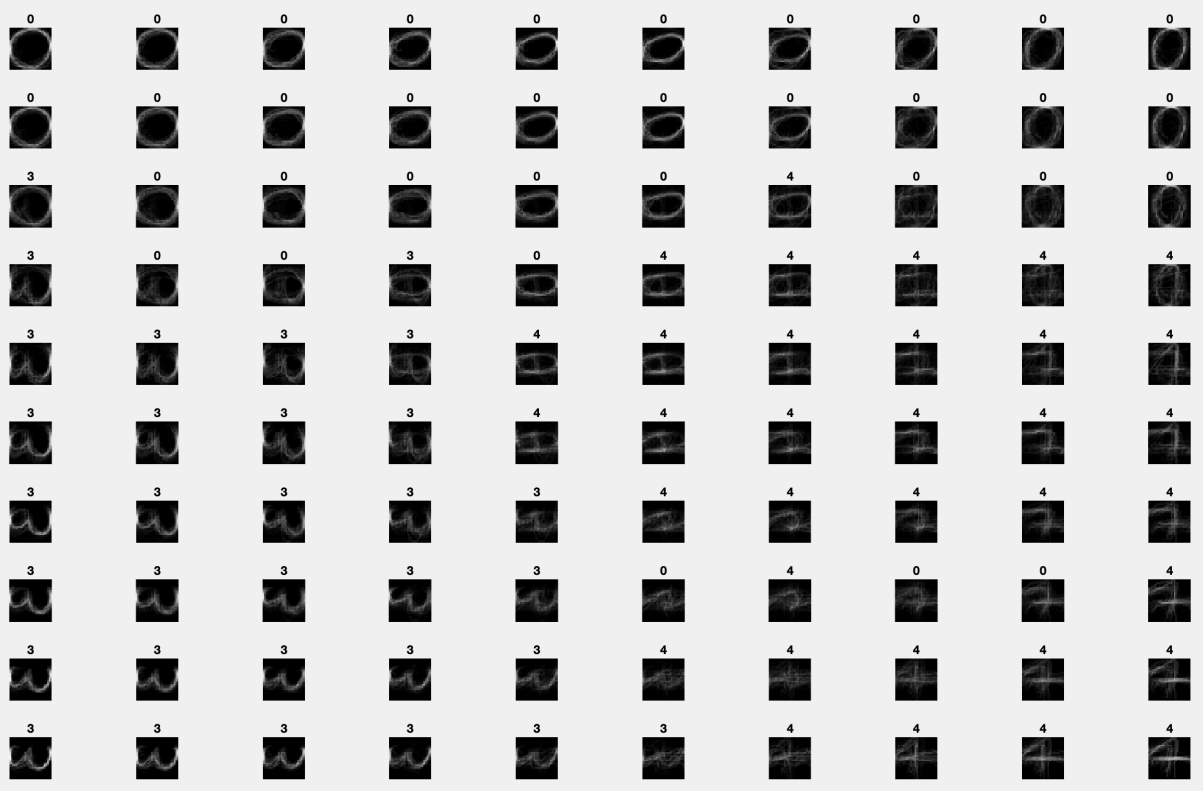


T=100000. Accuracy of test=58.33%

The results imply that the more train epochs, the higher accuracy is. But when the epoch is big enough (T=1000000), the accuracy starts to fall, which means under-fitting.

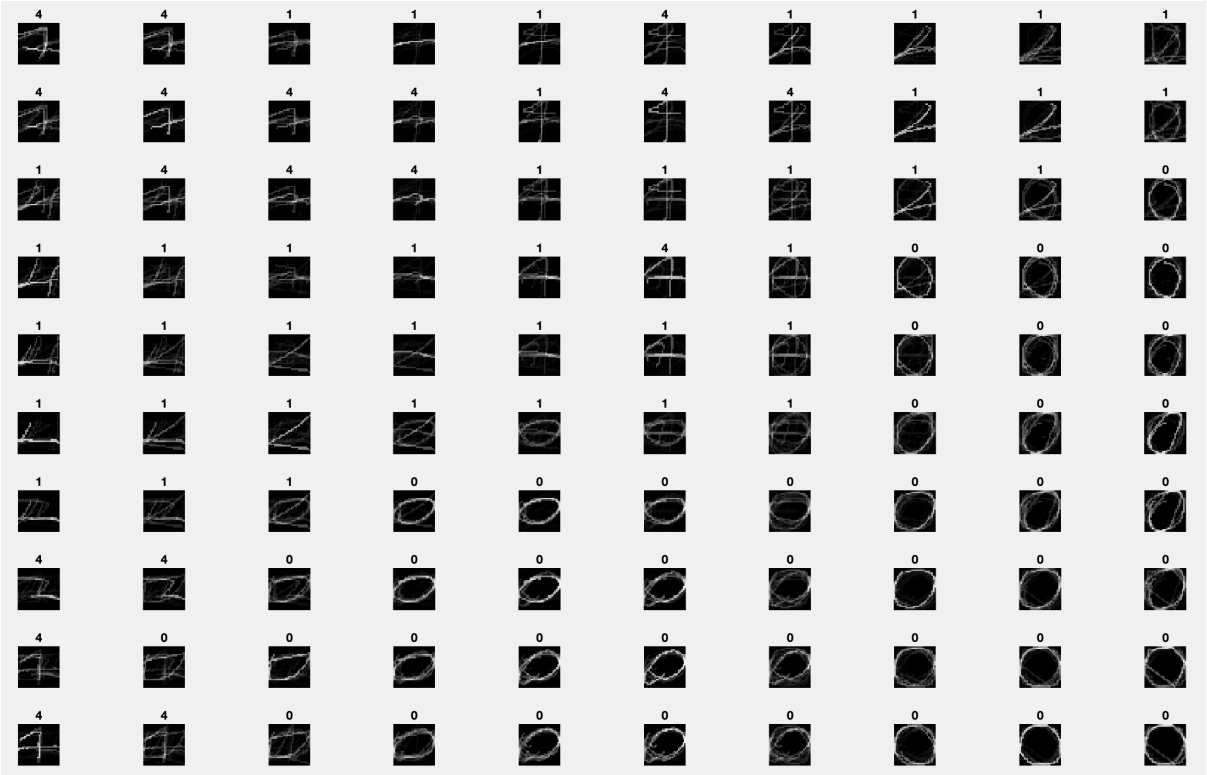
But the accuracy is really low, so I think 2 points.

1. Experiment in **other group (class 0,3,4)**. Although I only run this program in T=10000, I can get a very high accuracy. Thus, I find that the algorithm is related to the feature of data. Actually, 1&4 are always wrong-classified, which is similar to each other. The more different between each class, the better performance will be.



T=10000. Accuracy=96.67%

2.In this experiment, I assign learning rate form slide with 0.1, so I try to increase the learning rate then I find that the accuracy increase a lot!



T=10000. accuracy of train=67.5%, accuracy of test=68.3%

In conclusion, in order to improve the performance, you can

1. Training as much as possible, but not too much;

2. Choosing an appropriate learning rate.

Code.9 3.c

```
trainIdx=find(train_classlabel==0|train_classlabel==1|train_classlabel==4);
%find the location of classes 0,1,4
testIdx=find(test_classlabel==0|test_classlabel==1|test_classlabel==4);
trainX=train_data(:,trainIdx);%the new data
testX=test_data(:,testIdx);
TrLabel=train_classlabel(trainIdx);%the new label
TeLabel=test_classlabel(testIdx);

w=rand(784,10,10);
sigma0=sqrt(10^2+10^2)/2;%M=10 N=10
eta0=0.1;
T=10000;%iterations
tau1=T/log(sigma0);
tau2=T;
eta=eta0;
sigma=sigma0;

for n=1:T
    i=randperm(size(trainX,2),1);
    %competitive process
    distance=zeros(10,10);
```

```
    for row=1:10
        for col=1:10
            distance(row,col)=dot(trainX(:,i)-w(:,row,col),trainX(:,i)-
w(:,row,col));
        end
    end
    [min_row,min_col]=find(distance==min(min(distance)),1);
    %adaptation process
    for row=1:10
        for col=1:10
            h=exp(((row-min_row).^2+(col-min_col).^2)/(-(2*sigma.^2)));
            w(:,row,col)=w(:,row,col)+eta*h*(trainX(:,i)-w(:,row,col));
        end
    end
    %update eta&sigma
    eta=eta0*exp(-n/tau2);
    sigma=sigma0*exp(-n/tau1);
    n
end
%determine label for w
label=zeros(10,10);
dist=zeros(length(TrLabel),1);
for row=1:10
    for col=1:10
        for i=1:length(TrLabel)
            dist(i)=dot(trainX(:,i)-w(:,row,col),trainX(:,i)-w(:,row,col));
        end
        [min_dist,Idx]=min(dist);
        label(row,col)=TrLabel(Idx);
    end
end
count=1;
for row=1:10
    for col=1:10
        image=reshape(w(:,row,col),28,28);
        subplot(10,10,count);
        imshow(image);
        title(num2str(label(row,col)));
        count=count+1;
    end
end

%test set
test_dist=zeros(10,10);
test_label=zeros(1,length(TeLabel));
for i=1:length(TeLabel)
    for row=1:10
        for col=1:10
            test_dist(row,col)=dot(testX(:,i)-w(:,row,col),testX(:,i)-
w(:,row,col));
        end
    end
    [row_test,col_test]=find(test_dist==min(min(test_dist)),1);
    test_label(i)=label(row_test,col_test);
end
accuracy_test=sum(test_label==TeLabel)/length(TeLabel);

train_dist=zeros(10,10);
train_label=zeros(1,length(TrLabel));
for i=1:length(TrLabel)
    for row=1:10
        for col=1:10
```

```
            train_dist(row,col)=dot(trainX(:,i)-w(:,row,col),trainX(:,i)-
w(:,row,col));
        end
    end
    [row_train,col_train]=find(train_dist==min(min(train_dist)),1);
    train_label(i)=label(row_train,col_train);
end
accuracy_train=sum(train_label==TrLabel)/length(TrLabel);
```