National University of Singapore

Name : LUO ZIJIAN

Matric.No： A0224725H

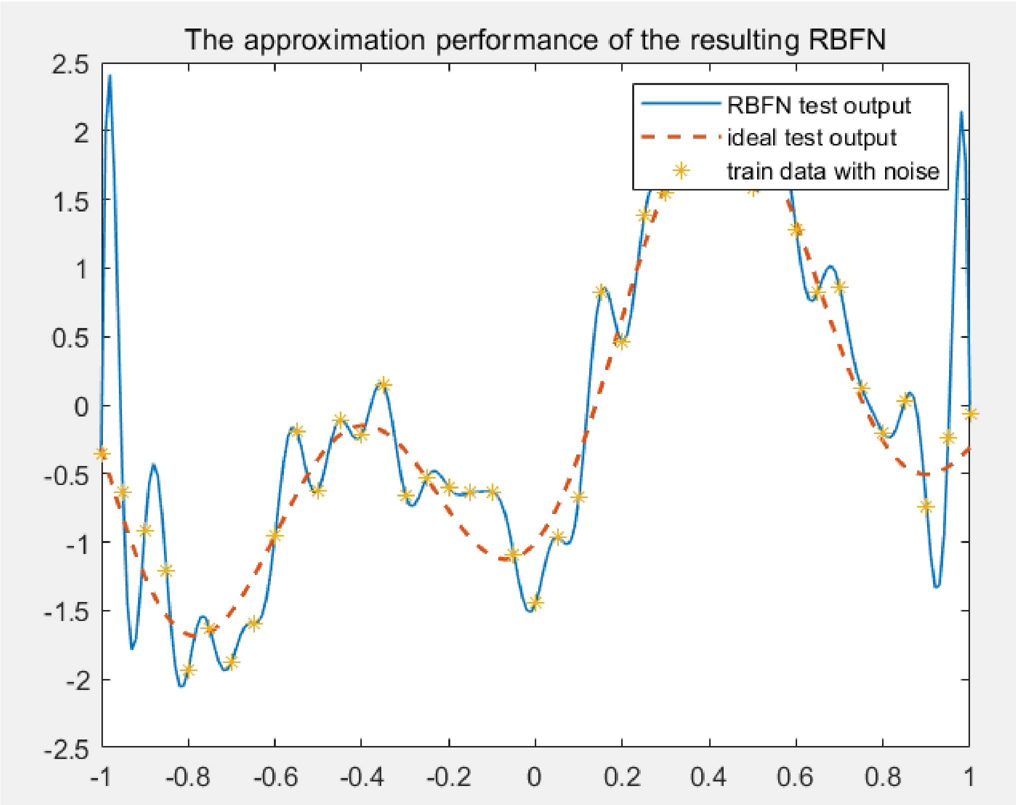
MUSNET： E0572844

Subject： NEURAL NETWORKS

Assignment： HOMEWORK THREE

Solution 1

(a) Follow the instructions in the sides of RBFN, we can get this picture like that



According to the result, it is clear that the output of RBFN test set is not close to the ideal output. The MSE of train set is 3.3524×10^{-18} and the MSE of the test set is 0.2934. In a word, this simulation is overfitting.

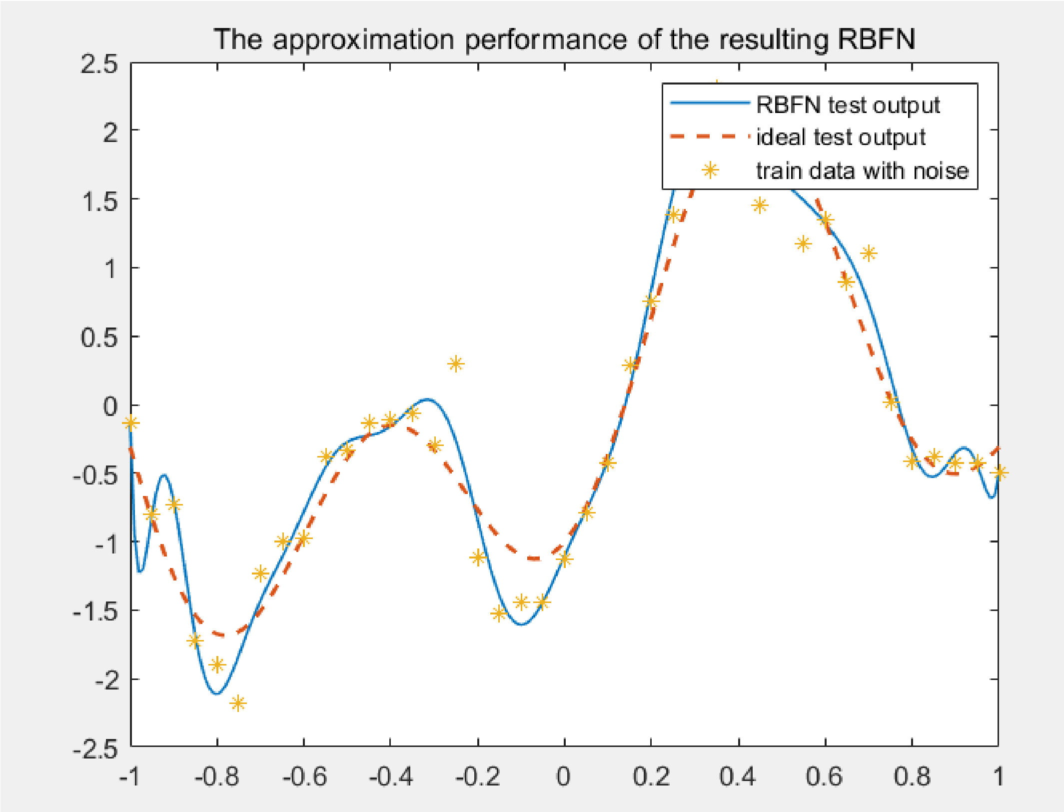
Here is the code

```
%init
close all;clear;clc;

%parameter
x_train=-1:0.05:1;%uniform step 0.08
x_test=-1:0.01:1;%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);
figure(1)
plot(x_test,d_test,'LineWidth',1);
hold on;
plot(x_test,ideal_test,'--','LineWidth',1.5);
hold on;
plot(x_train,d,'*');
hold on;
legend('RBFN test output','ideal test output','train
data with noise');
title('The approximation performance of the resulting
RBFN');
```

(b) For this part, I randomly select 20 centers among the sampling points with the strategy of "Fixed centers selected at random". Compared to the result of part a, it is clear that the output of test set with the strategy of fixed centers is more close to ideal output than that of test set without fixed centers. We can conclude that fixed centers can make the performance better by the result(The MSE of train set is 0.460 and MSE of test set is 0.674).

Especially for the difference of MSE of train set between part a and part b, the train error of this part is larger than part a. As a result of overfitting, not all the train samples being fitted.

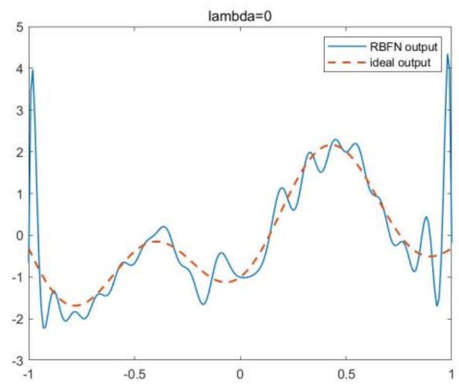
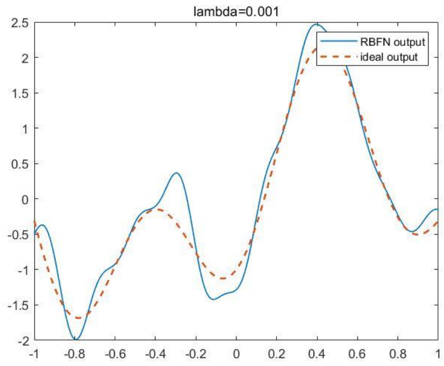
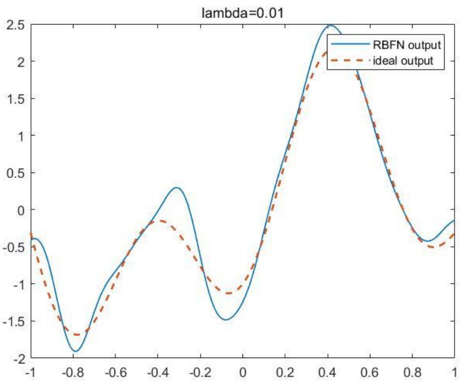
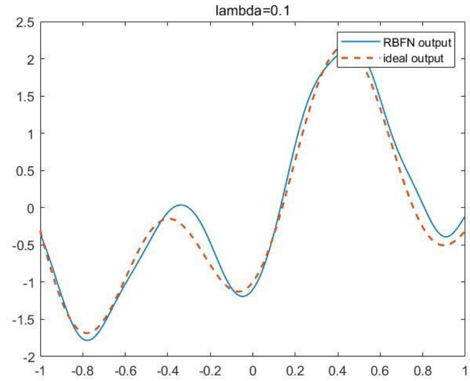
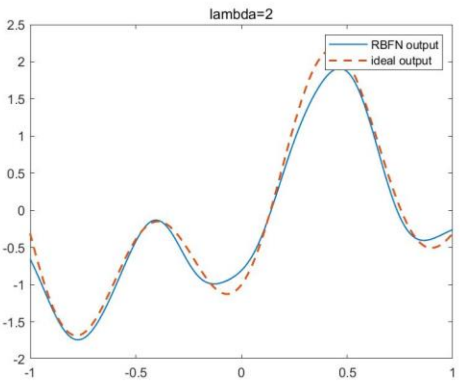
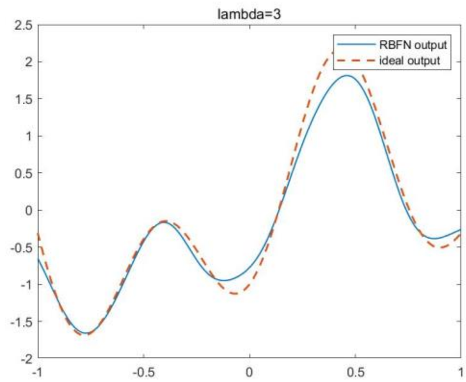
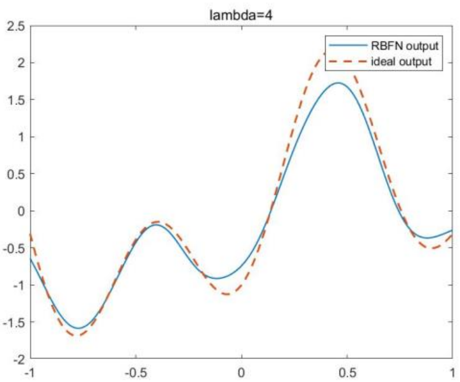
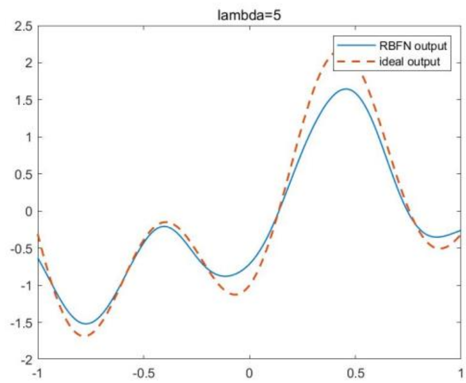


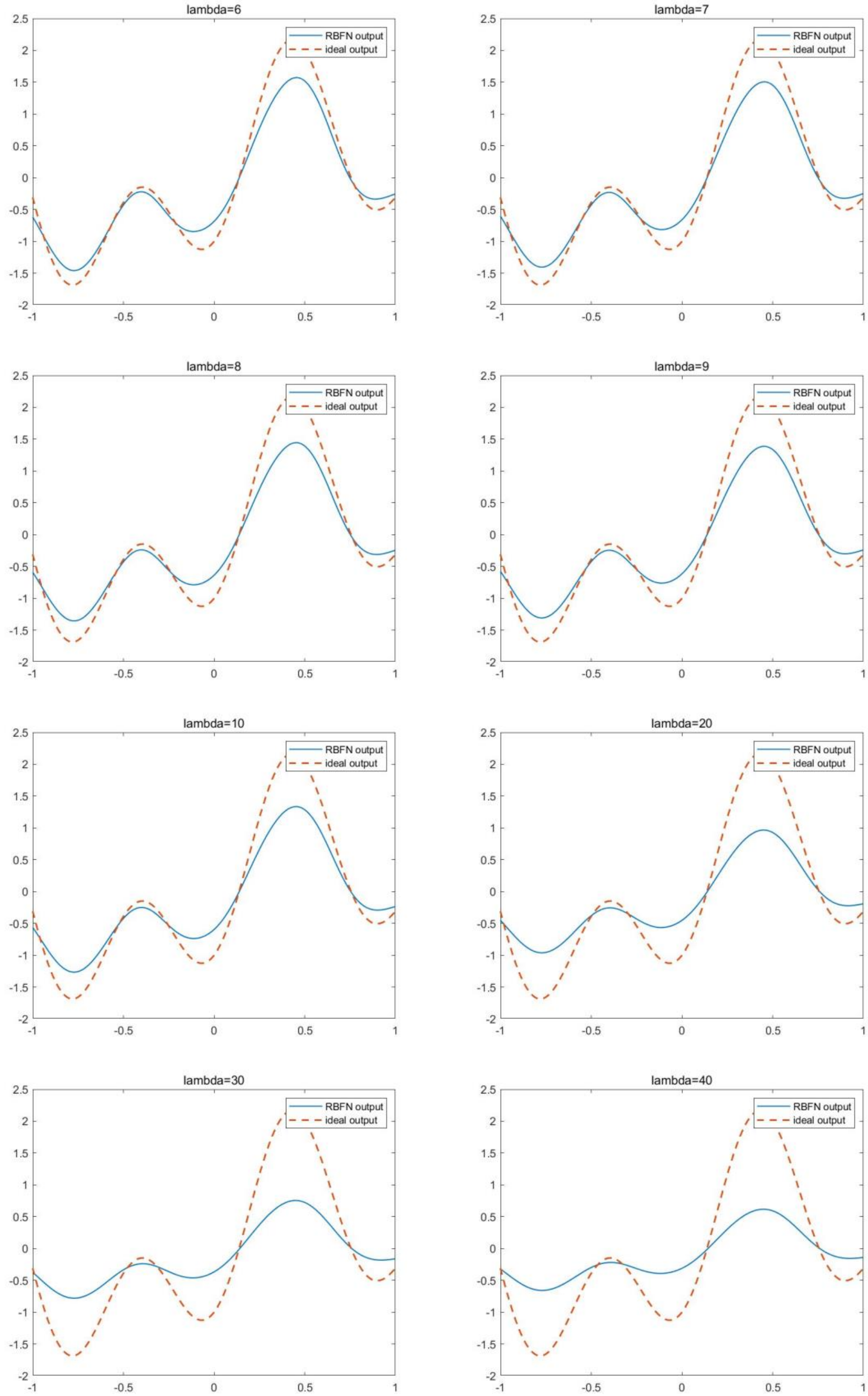

```
%init
close all;clear;clc;

%parameter
x_train=-1:0.05:1;%uniform step 0.08
x_test=-1:0.01:1;%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
```

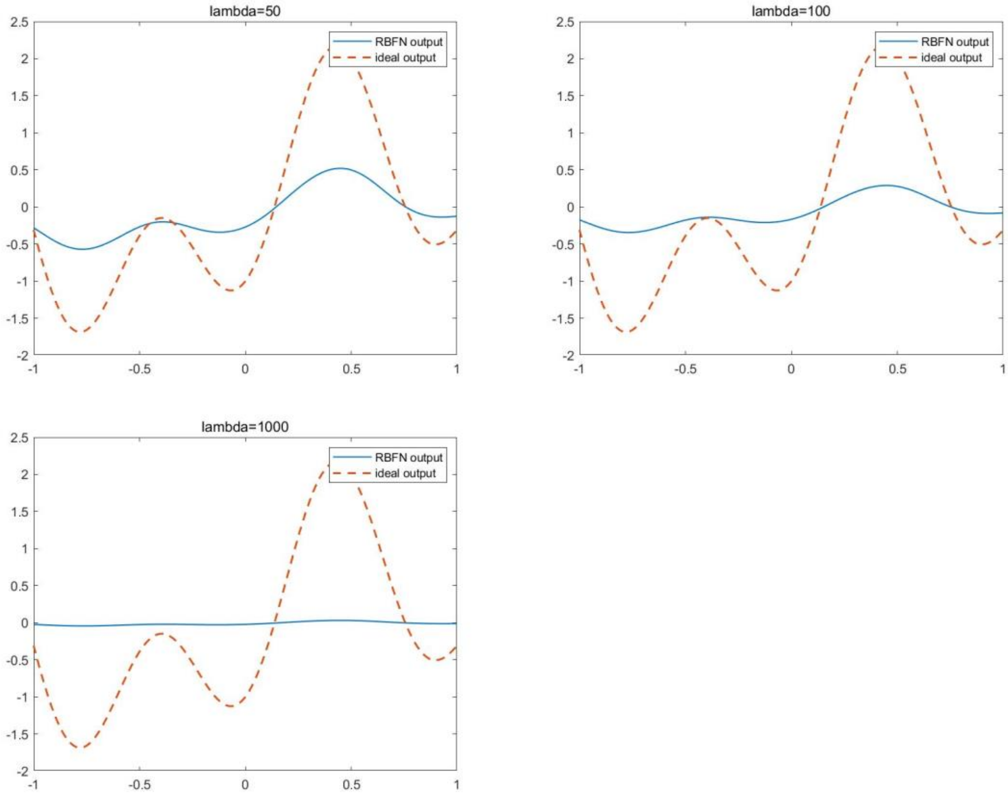
```
w=pinv(phi)*d';%get the unique solution w
%test data
phi_test=zeros(length(x_test),length(M));%initialize
phi_test
for i=1:length(x_test)
    for j=1:length(M)
        r=x_test(i)-M(j);
        phi_test(i,j)=exp(coef*r^2);
    end
end
phi_test=[ones(length(x_test),1),phi_test];
d_test=phi_test*w;
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);
figure(1)
plot(x_test,d_test,'LineWidth',1);
hold on;
plot(x_test,ideal_test,'--','LineWidth',1.5);
hold on;
plot(x_train,d,'*');
hold on;
legend('RBFN test output','ideal test output','train
data with noise');
title('The approximation performance of the resulting
RBFN');
```

(c) For this part, I apply regulation method in part a, I get these conclusions.

1. When lambda equals 0, it means that there is no existence of regulation. However, only a little(0.001), we can find the curve is smoother than that of zero.
2. It is clear that the curve is becoming more and more smooth, with the increase of lambda. Especially for when lambda reach 1, the curve is so smooth than before.
3. However, when the lambda is big enough(50-100), we can conclude that the smoothness constraint dominates and less account is taken for training and test data error.
4. The more lambda is, the larger MSE of the test is. In this case, when lambda reach 1000, the MSE of train set and the MSE of test set increase up to 1.3806 and 1.1842. Therefore, the increase of lambda causes the under-fitting in RBFN output using test data.


lambda=0
RBFN output
ideal output

lambda=0.001
RBFN output
ideal output

lambda=0.01
RBFN output
ideal output

lambda=0.1
RBFN output
ideal output

lambda=2
RBFN output
ideal output

lambda=3
RBFN output
ideal output

lambda=4
RBFN output
ideal output

lambda=5
RBFN output
ideal output


lambda=6
RBFN output
ideal output
lambda=7
RBFN output
ideal output
lambda=8
RBFN output
ideal output
lambda=9
RBFN output
ideal output
lambda=10
RBFN output
ideal output
lambda=20
RBFN output
ideal output
lambda=30
RBFN output
ideal output
lambda=40
RBFN output
ideal output



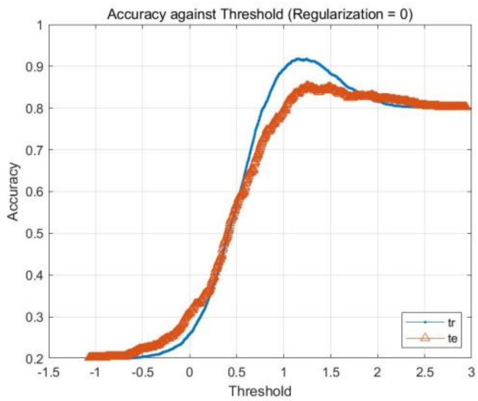

```
%init
close all;clear;clc;

%parameter
x_train=-1:0.05:1;%uniform step 0.08
x_test=-1:0.01:1;%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

for
lambda=[0,0.001,0.01,0.1,1:10,20,30,40,50,100,1000]%regu
lation 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);
    figure
    plot(x_test,d_test,'LineWidth',1);
    hold on;
    plot(x_test,ideal_test,'--','LineWidth',1.5);
    hold on;
    legend('RBFN output','ideal output');
    title(['lambda=',num2str(lambda)]);
    name=num2str(lambda);
    saveas(gcf,name,'jpg');
end
```

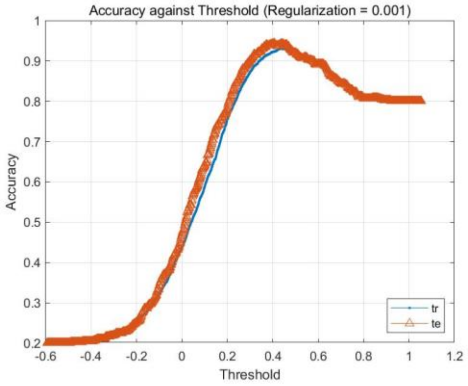
Solution 2

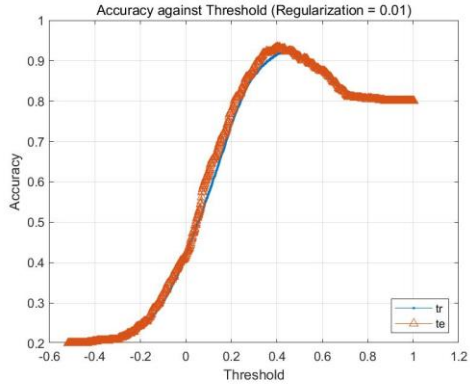
My matriculation number is "A0224725H", so I choose class 2 and 5 to be assigned the label "1", and the remaining classes to be assigned the label "0".

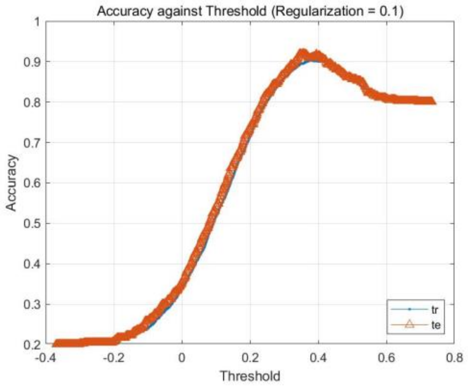
(a) In this part, I use Exact Interpolation Method and apply regulation, given the Gaussian function of RBFN, with standard deviation of 100.

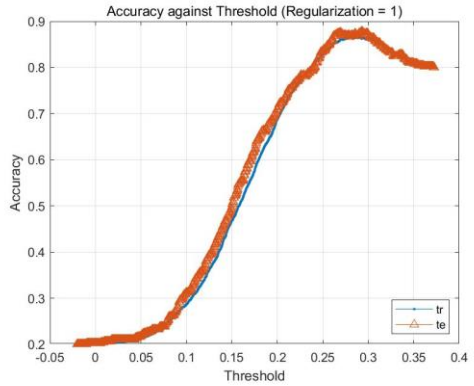


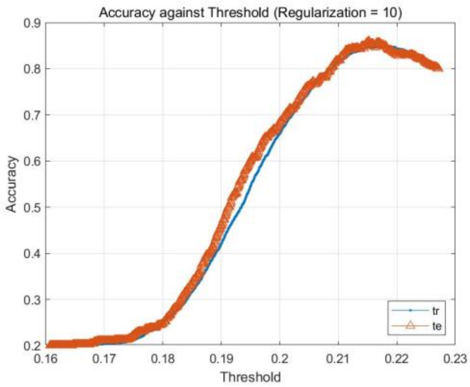
The figure is the accuracy both train set and test set using RBFN without regulation. The result seems that the accuracy of train set(88.5%) is lower than that of test set(78.3%) when threshold is 1.

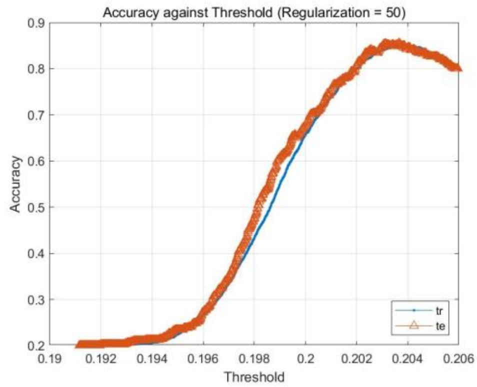


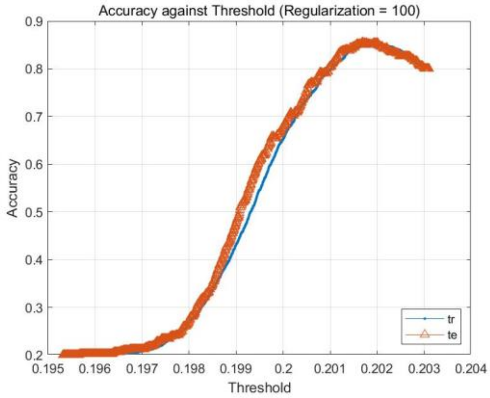


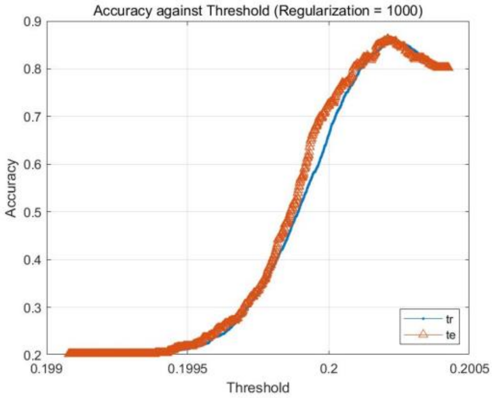












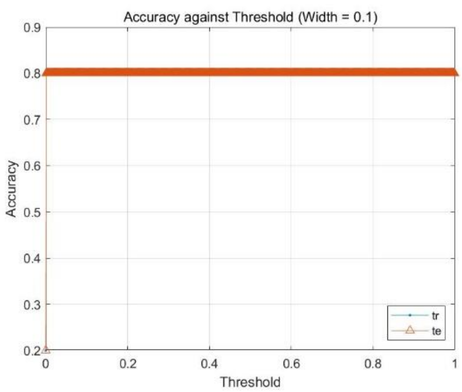
These figures above imply that with the increase of the value of regulation, the accuracy of test set looks like closer to that of train set, and the range of the threshold becomes narrower. However, if we print the accuracy of both, we find that the accuracy of both do not change too much, actually! The accuracy of train set fluctuates between 0.895 and 0.864, and that of test set fluctuates between 0.88 and 0.852.

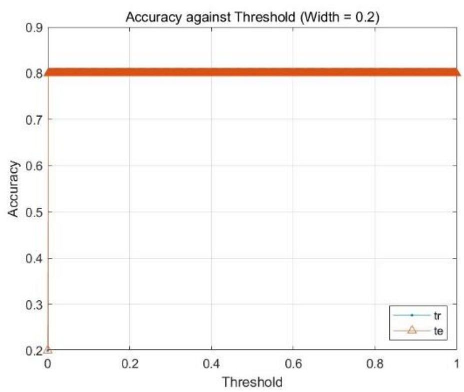
```
%% Clear all variables and close all
close all
clear
clc
sigma = 100;
mkdir q2_a_image
tic

%% Initialise equations and values
load('characters10.mat');
train_data=im2single(train_data);
test_data=im2single(test_data);
test_data=test_data';
train_data=train_data';

trainidx = find(train_label == 2 | train_label == 5);
train_classlabel_logic = logical(train_label(:,:) == 2 |
train_label(:,:) == 5);
train_classlabel_logic =train_classlabel_logic';

testidx = find(test_label == 2 | test_label == 5);
test_classlabel_logic = logical(test_label(:,:) == 2 |
test_label(:,:) == 5);
test_classlabel_logic =test_classlabel_logic';

%% Calculate interpolation matrix and weights
i_mat = cal_i_mat(train_data, sigma,train_data);
i_mat_test = cal_i_mat(test_data, sigma,train_data);

%% Calculate performance and plot graphs
close all
counter = 1;
for reg = [0,0.001, 0.01, 0.1:0.1:1,
10:10:100,200:200:1000]
    disp(reg)
    %if reg == 0
        %w = inv(i_mat)* double(train_classlabel_logic)';
    %else
        w = inv(i_mat'*i_mat + eye(1000) * reg) * i_mat'
```

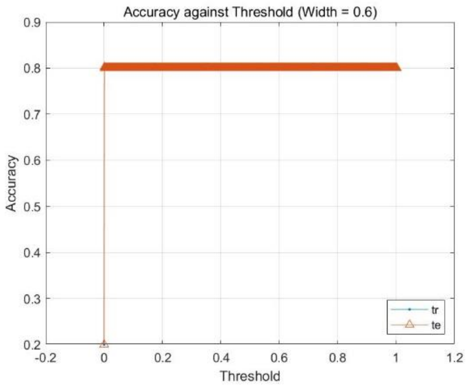
```
* double(train_classlabel_logic)';
    %end

    TrPred = i_mat * w;
    TePred = i_mat_test * w;

    TrLabel = double(train_classlabel_logic);
    TeLabel = double(test_classlabel_logic);

    TrAcc = zeros(1,1000);
    TeAcc = zeros(1,1000);
    thr = zeros(1,1000);
    TrN = length(TrLabel);
    TeN = length(TeLabel);

    for i = 1:1000
        t = (max(TrPred)-min(TrPred)) * (i-1)/1000 +
min(TrPred);
        thr(i) = t;
        TrAcc(i) = (sum(TrLabel(TrPred<t)==0) +
sum(TrLabel(TrPred>=t)==1)) / TrN;
        TeAcc(i) = (sum(TeLabel(TePred<t)==0) +
sum(TeLabel(TePred>=t)==1)) / TeN;
    end

    acc_th(1,counter) = reg;                  % reg
value

    [acc_th(2,counter),thres] = max(TrAcc);         % max
training accuracy
    acc_th(3,counter) = thr(1,thres);

    [acc_th(4,counter),thres] = max(TeAcc);         % max
testing accuracy
    acc_th(5,counter) = thr(1,thres);

    counter = counter + 1;
    %figure;
    plot(thr,TrAcc,'.- ',thr,TeAcc,'^-
');legend('tr','te','Location','southeast');
    grid
    title(strcat('Accuracy against Threshold
(Regularization = ', " ", num2str(reg), ")"))
    ylabel("Accuracy"); xlabel("Threshold");
```

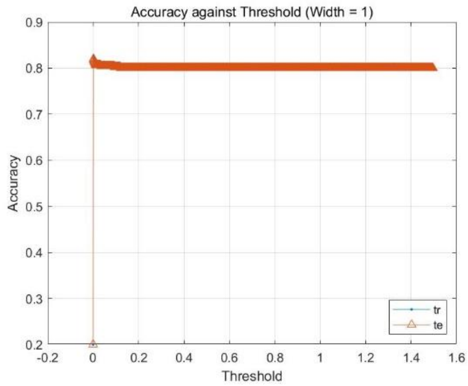
```
saveas(gcf,strcat("q2_a_image/a_",num2str(reg),".bmp"))
end

figure;
hold on
plot(acc_th(1,:),acc_th(2,:),'-m');
plot(acc_th(1,:),acc_th(4,:),'-k');
legend('Training data','Test
data','Location','northeast');
grid
title('Accuracy against Regularization');
ylabel("Accuracy"); xlabel("Regularization");
saveas(gcf,strcat("q2_a_image/a_","acc against
reg",".jpg"))
sort(acc_th,2)
toc

function matrix = cal_i_mat(data, sigma, train_data)
num_data = size(data,2);
num_cen = 1000;
matrix = zeros(num_data,num_cen);
for i = 1:num_data
    for j = 1:num_cen
        disp(['Calculating (' num2str(i) ','
num2str(j),')'])
        matrix(i,j) = exp (  (norm(data(:,i) -
train_data(:,j)))^2  /  (-2*(sigma^2))   )  ;
    end
end
end
```

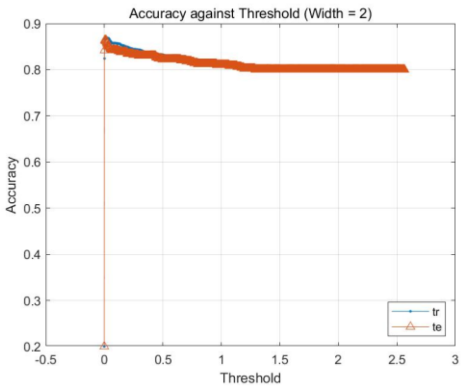
(b) Because of comparing with the result of a, I use the same width with standard deviation of 100 and regulation factor in this range(0,10000).

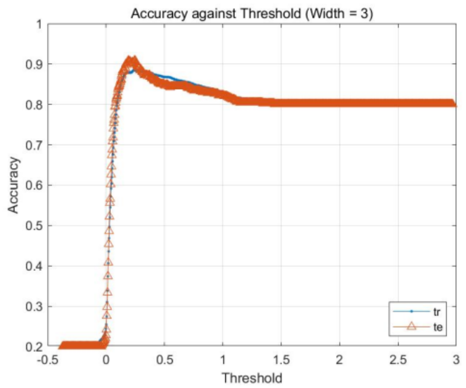


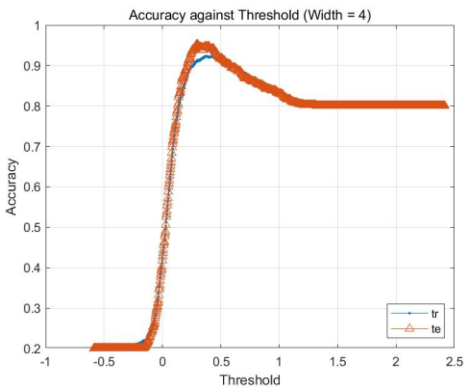


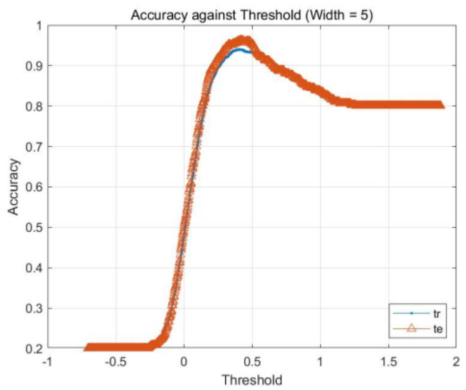

Accuracy against Threshold (Width = 0.6)
Accuracy
Threshold
tr
te

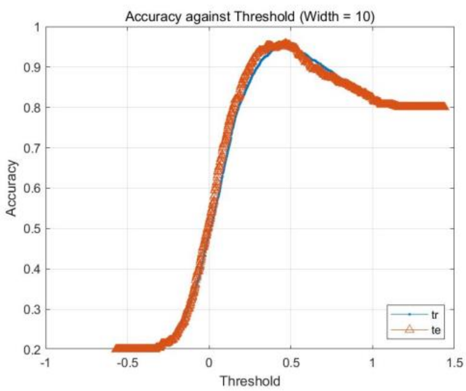

Accuracy against Threshold (Width = 1)
Accuracy
Threshold
tr
te

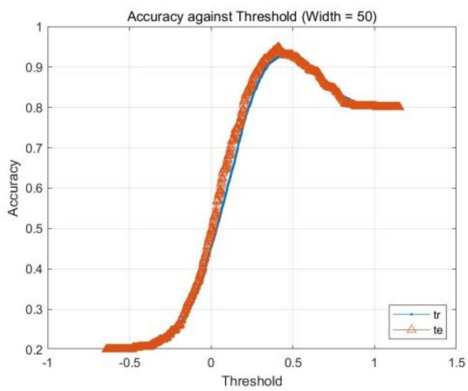

Accuracy against Threshold (Width = 2)
Accuracy
Threshold
tr
te

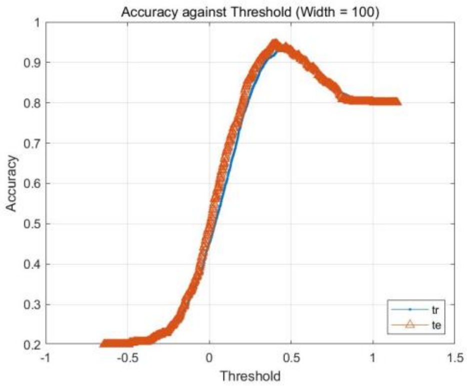

Accuracy against Threshold (Width = 3)
Accuracy
Threshold
tr
te

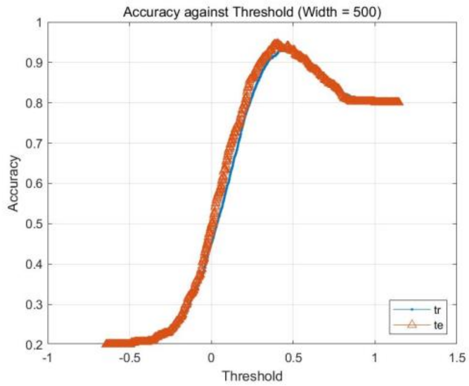

Accuracy against Threshold (Width = 4)
Accuracy
Threshold
tr
te

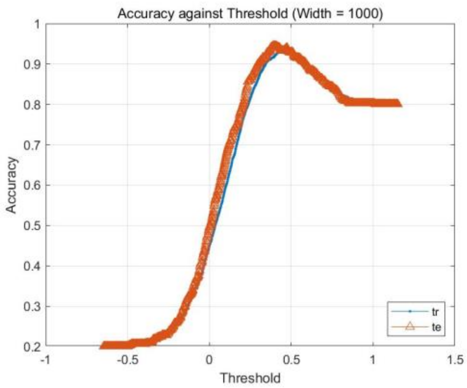

Accuracy against Threshold (Width = 5)
Accuracy
Threshold
tr
te

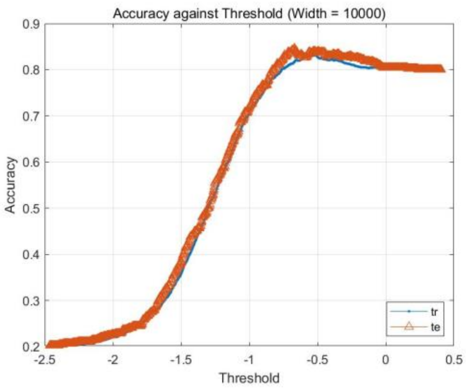

Accuracy against Threshold (Width = 10)
Accuracy
Threshold
tr
te


Accuracy against Threshold (Width = 50)
Accuracy
Threshold
tr
te

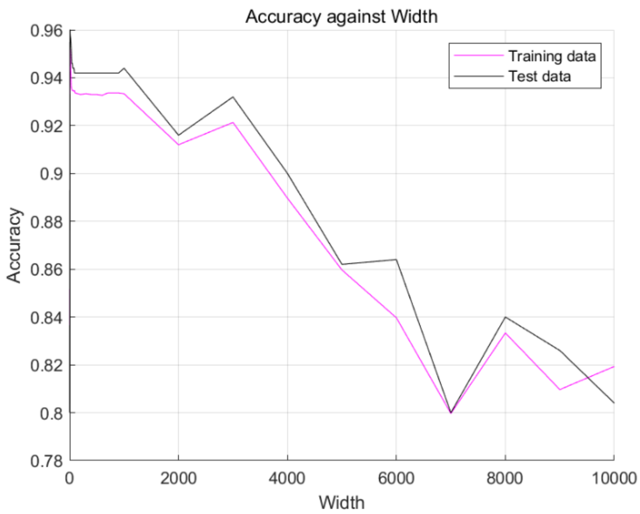








Compared with the result in part a, the accuracy of train set is lower than that in part a, but the accuracy of test set is higher than that in part a. Then I vary the value of width from 0.1 to 10000. The results imply that the accuracy of train set starts at 79.6%, and jumps to 87.8% when width =0.25. 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.




```
% Clear all variables and close all
close all
clear
clc
```

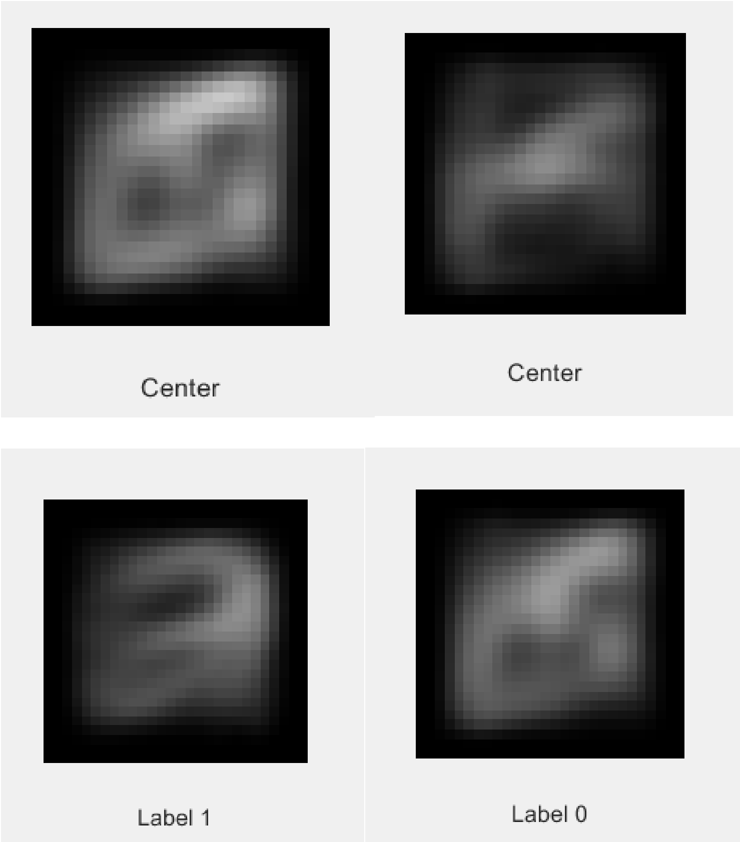
```
num_cen = 100;
mkdir q2_b_image
tic

% Initialise equations and values
load('characters10.mat');
train_data=im2single(train_data);
test_data=im2single(test_data);
test_data=test_data';
train_data=train_data';

trainidx = find(train_label == 2 | train_label ==
5);
train_classlabel_logic = logical(train_label(:,:)
== 2 | train_label(:,:) == 5);
train_classlabel_logic =train_classlabel_logic';

testidx = find(test_label == 2 | test_label == 5);
test_classlabel_logic = logical(test_label(:,:) ==
2 | test_label(:,:) == 5);
test_classlabel_logic =test_classlabel_logic';

% Calculate interpolation matrix and weights
idx = randperm(size(train_data,2));
idx = idx(1,1:num_cen);

cen_data = train_data(:,idx);
cen_label = train_classlabel_logic(:,idx);

for i = 1:num_cen
    dist(1,i) = norm(cen_data(:,i));
end
sigma_o = (max(dist) - min(dist)) /
(sqrt(2*num_cen));

% Calculate performance and plot graphs
close all
counter = 1;
for sigma = [sigma_o, 0.1:0.1:1, 2:1:10, 20:10:100,
200:100:1000, 2000:1000:10000]
%for sigma = [10000]
    disp(sigma)
    i_mat = cal_i_mat(train_data, sigma,cen_data);
    i_mat_test = cal_i_mat(test_data,
```

```
sigma,cen_data);

    w = inv(i_mat'*i_mat) * i_mat' *
double(train_classlabel_logic)';

    TrPred = i_mat * w;
    TePred = i_mat_test * w;

    TrLabel = double(train_classlabel_logic);
    TeLabel = double(test_classlabel_logic);

    TrAcc = zeros(1,1000);
    TeAcc = zeros(1,1000);
    thr = zeros(1,1000);
    TrN = length(TrLabel);
    TeN = length(TeLabel);

    for i = 1:1000
        t = (max(TrPred)-min(TrPred)) * (i-1)/1000 +
min(TrPred);
        thr(i) = t;
        TrAcc(i) = (sum(TrLabel(TrPred<t)==0) +
sum(TrLabel(TrPred>=t)==1)) / TrN;
        TeAcc(i) = (sum(TeLabel(TePred<t)==0) +
sum(TeLabel(TePred>=t)==1)) / TeN;
    end

    acc_th(1,counter) = sigma;                    %
sigma value

    [acc_th(2,counter),thres] = max(TrAcc);        %
max training accuracy
    acc_th(3,counter) = thr(1,thres);

    [acc_th(4,counter),thres] = max(TeAcc);        %
max testing accuracy
    acc_th(5,counter) = thr(1,thres);

    counter = counter + 1;
    %figure;
    plot(thr,TrAcc,'.- ',thr,TeAcc,'^-
');legend('tr','te','Location','southeast');
    grid
    title(strcat('Accuracy against Threshold (Width
```

```
= ', " ", num2str(sigma), ")"))
    ylabel("Accuracy"); xlabel("Threshold");

saveas(gcf,strcat("q2_b_image/b_",num2str(sigma),".
jpg"))
end

figure;
hold on
plot(acc_th(1,:),acc_th(2,:),'-m');
plot(acc_th(1,:),acc_th(4,:),'-k');
legend('Training data','Test
data','Location','northeast');
grid
title('Accuracy against Width');
ylabel("Accuracy"); xlabel("Width");
saveas(gcf,strcat("q2_b_image/b_","acc against
thres",".jpg"))

toc

function matrix = cal_i_mat(data, sigma,
train_data)
num_data = size(data,2);
num_cen = size(train_data,2);
matrix = zeros(num_data,num_cen);
for i = 1:num_data
    for j = 1:num_cen
        disp(['For width = ' num2str(sigma) ',
calculating (' num2str(i) ',' num2str(j),')'])
        matrix(i,j) = exp (  (norm(data(:,i) -
train_data(:,j)))^2  /  (-2*(sigma^2))   )  ;
    end
end
end
```

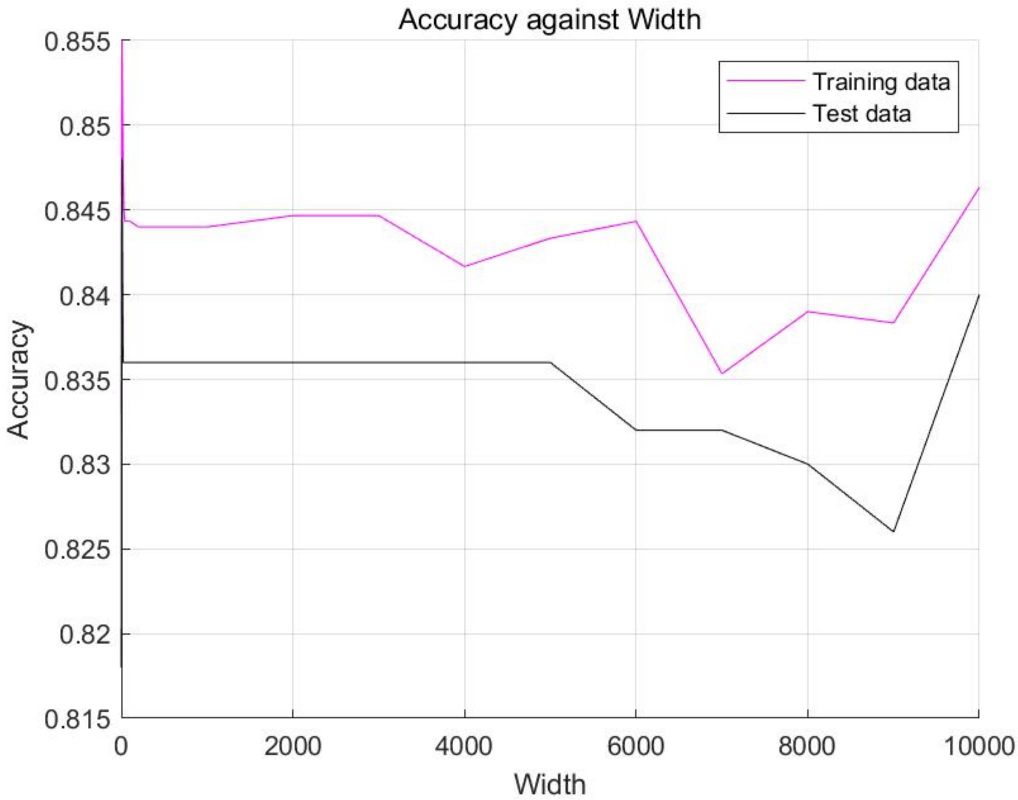
(c)

Apply “K-Mean Clustering” with 2 centers, we get 2 centers visualized like this.



Although my classes are 2&5, 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 2&5, and the label 0 is the remainder. Thus, for mean 1, it is very reasonable to be similar with 2&5. But if we assign images 2 as label 1 and images 5 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 2&5 and mean 1&0.

According to below figure, 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 (84.5%) and the accuracy of test set (83.6%).




```
%% Clear all variables and close all
close all
clear
clc
num_cen = 2;
mkdir q2_c_image
tic

%% Initialise equations and values
load('characters10.mat');
train_data=im2single(train_data);
test_data=im2single(test_data);
test_data=test_data';
train_data=train_data';

trainidx = find(train_label == 2 | train_label ==
5);
train_classlabel_logic = logical(train_label(:,:)
== 2 | train_label(:,:) == 5);
train_classlabel_logic =train_classlabel_logic';

testidx = find(test_label == 2 | test_label == 5);
test_classlabel_logic = logical(test_label(:,:) ==
```

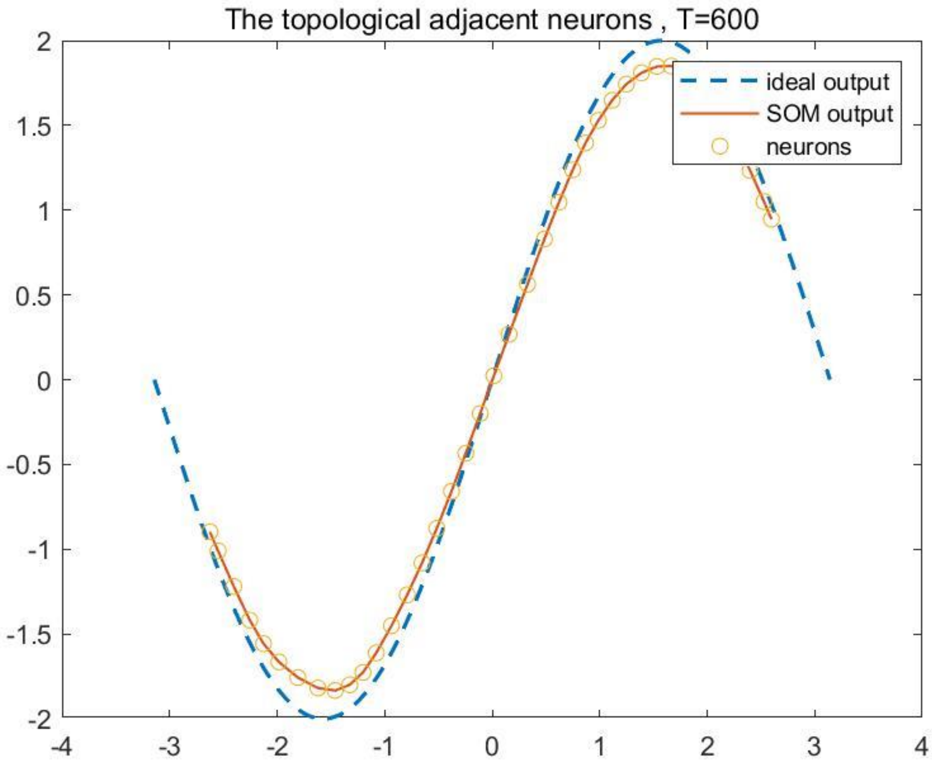
```
2 | test_label(:,:) == 5);
test_classlabel_logic =test_classlabel_logic';

%% Kmeans clustering and calculate width
[idx, center] = kmeans(train_data',num_cen);
idx = idx';
cen_data = center';

%% Calculate interpolation matrix and weights
close all
counter = 1;
for sigma = [1:1:10, 20:10:100, 200:100:1000,
2000:1000:10000]
%for sigma = [1]
    disp(sigma)
    i_mat = cal_i_mat(train_data, sigma,cen_data);
    i_mat_test = cal_i_mat(test_data,
sigma,cen_data);

    w = inv(i_mat'*i_mat) * i_mat' *
double(train_classlabel_logic)';

    TrPred = i_mat * w;
    TePred = i_mat_test * w;

    TrLabel = double(train_classlabel_logic);
    TeLabel = double(test_classlabel_logic);

    TrAcc = zeros(1,1000);
    TeAcc = zeros(1,1000);
    thr = zeros(1,1000);
    TrN = length(TrLabel);
    TeN = length(TeLabel);

    for i = 1:1000
        t = (max(TrPred)-min(TrPred)) * (i-1)/1000 +
min(TrPred);
        thr(i) = t;
        TrAcc(i) = (sum(TrLabel(TrPred<t)==0) +
sum(TrLabel(TrPred>=t)==1)) / TrN;
        TeAcc(i) = (sum(TeLabel(TePred<t)==0) +
sum(TeLabel(TePred>=t)==1)) / TeN;
    end
```

```
    acc_th(1,counter) = sigma;                          %
sigma value

    [acc_th(2,counter),thres] = max(TrAcc);             %
max training accuracy
    acc_th(3,counter) = thr(1,thres);

    [acc_th(4,counter),thres] = max(TeAcc);             %
max testing accuracy
    acc_th(5,counter) = thr(1,thres);

    counter = counter + 1;
    %figure;
    plot(thr,TrAcc,'.- ',thr,TeAcc,'^-
');legend('tr','te','Location','southeast');
    grid
    title(strcat('Accuracy against Threshold (Width
= ', " ", num2str(sigma), ")"))
    ylabel("Accuracy"); xlabel("Threshold");

saveas(gcf,strcat("q2_c_image/c_",num2str(sigma),".
jpg"))
end

figure;
hold on
plot(acc_th(1,:),acc_th(2,:),'-m');
plot(acc_th(1,:),acc_th(4,:),'-k');
legend('Training data','Test
data','Location','northeast');
grid
title('Accuracy against Width');
ylabel("Accuracy"); xlabel("Width");
saveas(gcf,strcat("q2_c_image/c_","acc against
thres",".jpg"))

%% Plot centers and mean
label0idx = find(~train_classlabel_logic == 1);
label1 = train_data(:,trainidx);
label1_mean = mean(label1,2);
label0 = train_data(:,label0idx);
label0_mean = mean(label0,2);

plotimages(cen_data,'Center');          % visualise
```

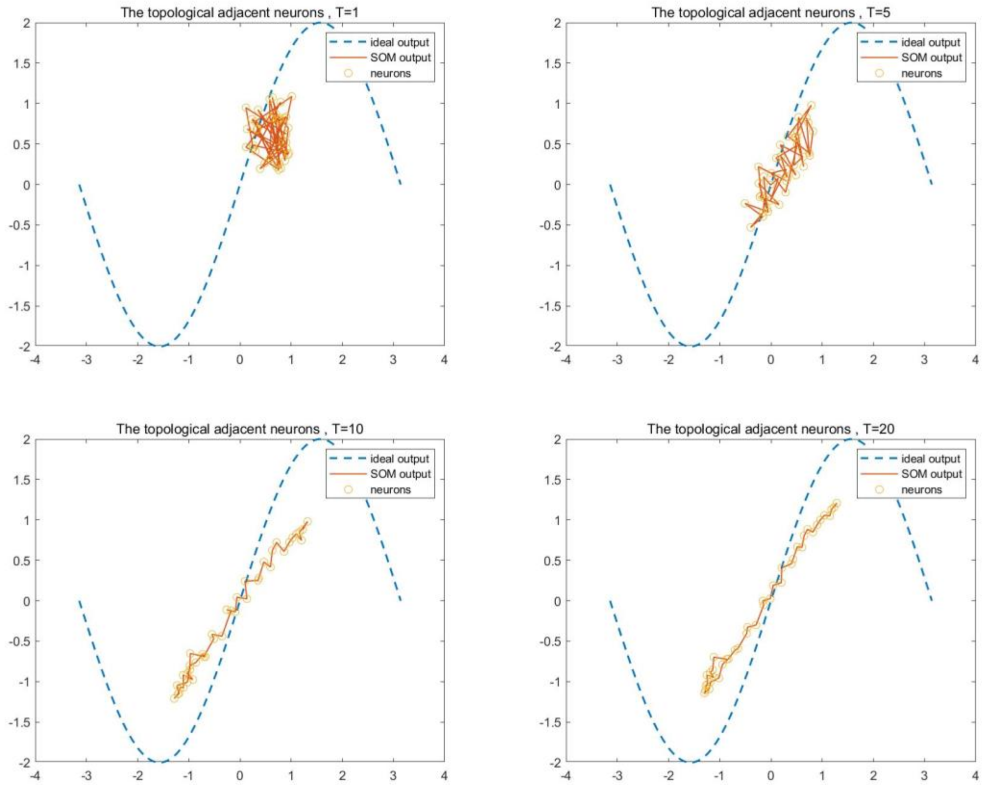
```
centers from kmeans
plotimages(label1_mean,'Label 1');      % visualise
label 1 mean
plotimages(label0_mean,'Label 0');      % visualise
label 0 mean

toc

%% Functions
function plotimages(data,txt)
num_data = size(data, 2);
for i = 1:num_data
    img = reshape(data(:,i),[28 28]);
    figure;
    imshow(img');
    xlabel(txt)
end
end

function matrix = cal_i_mat(data, sigma,
train_data)
num_data = size(data,2);
num_cen = size(train_data,2);
matrix = zeros(num_data,num_cen);
for i = 1:num_data
    for j = 1:num_cen
        disp(['For width = ' num2str(sigma) ',
calculating (' num2str(i) ',' num2str(j),')'])
        matrix(i,j) = exp (  (norm(data(:,i) -
train_data(:,j)))^2  /  (-2*(sigma^2))   )  ;
    end
end
end
```

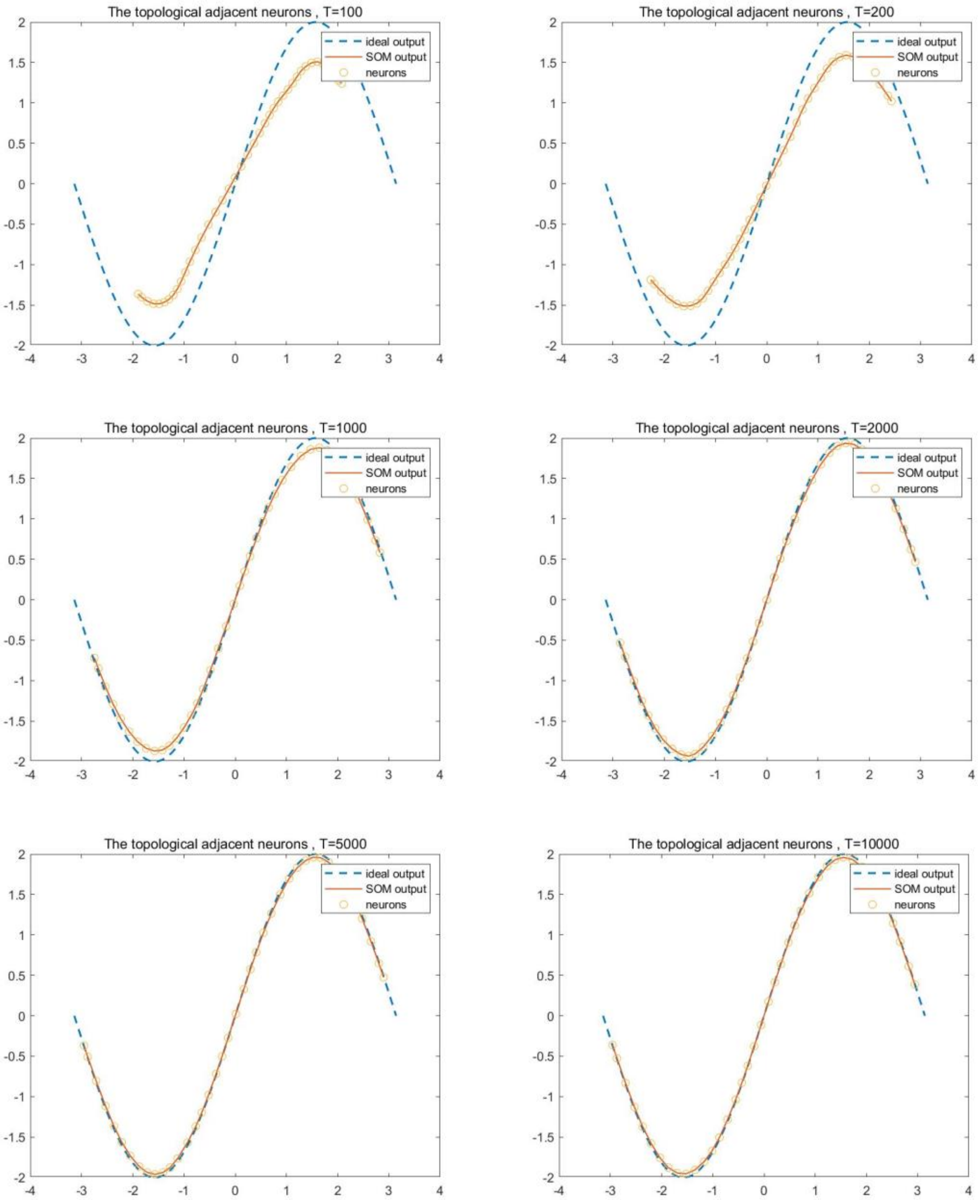
Solution 3

(a) In this part, I design a SOM that maps a layer of 36 neurons. So I set T=600



In order to compare the result of different T, I make experiments on T in this value (1,10000)



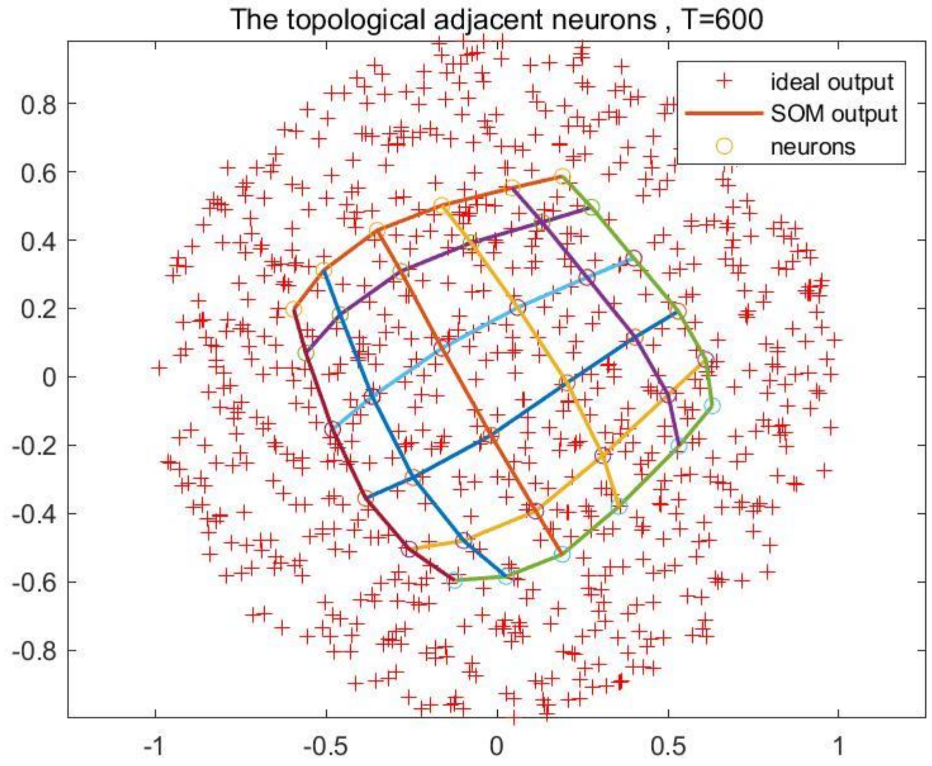


Based on this result, we can conclude that, more epoch is, the better fitting.

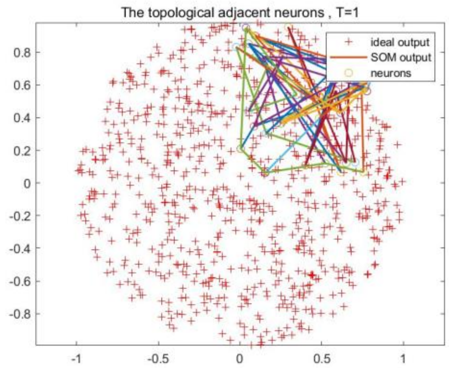
```
%init
close all;clear;clc;
mkdir q3_a_image

x=linspace(-pi,pi,400);
trainX=[x;2*sin(x)];%2x400 matrix

%parameter
w=rand(36,2); %randomly init weigth 36 neurons in
output layer
```

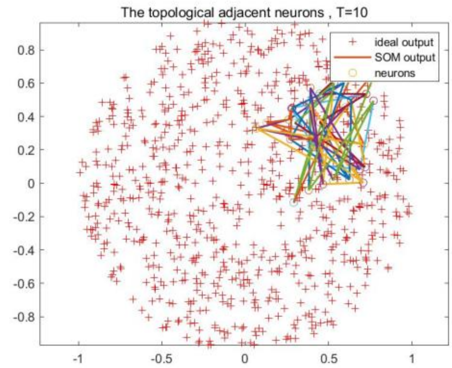
```
sigma0=sqrt(1^2+36^2)/2;%M=1,N=36
eta0=0.1;

for
T=[1:10,20:20:100,200:200:1000,2000:1000:10000]
%T=100;%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*36 =1*36
        %adaptation process
        for j=1:36
            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
    figure(1)
    plot(trainX(1,:),trainX(2,:),'--
','LineWidth',1.5);hold on;
    plot(w(:,1),w(:,2),'LineWidth',1); hold on;
    scatter(w(:,1),w(:,2),'o');hold on;
    title(['The topological adjacent neurons ,
T=',num2str(T)]);
    legend('ideal output','SOM output','neurons');

saveas(gcf,strcat("q3_a_image/a_",num2str(T),".jp
g"));
end
```

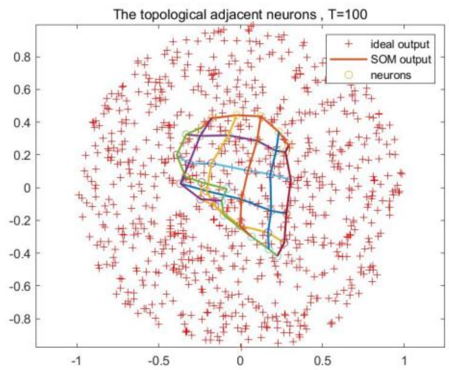
(b) In this part, I design a SOM that maps a 2-dimensional output layer of 36 neurons. Through the requirement, I set T=600.

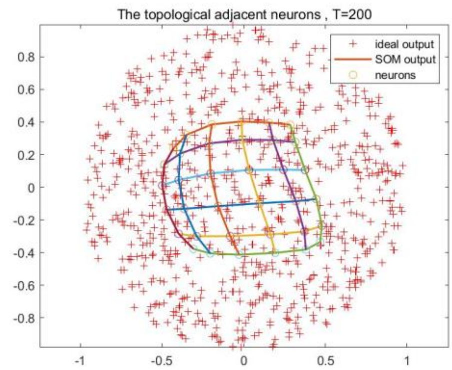


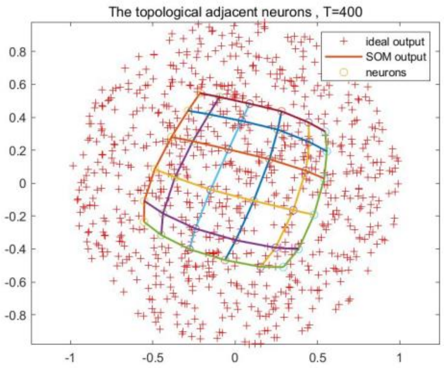
In order to compare the result of different T, I make experiments on T in this value (1,10000)

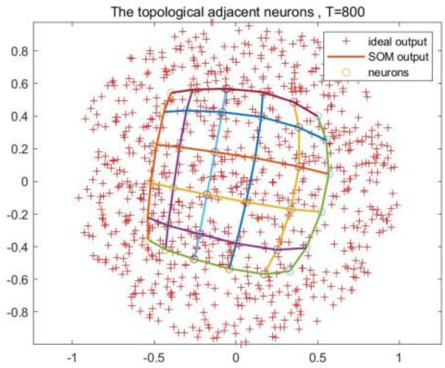


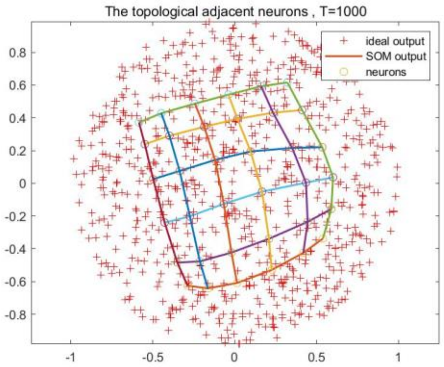


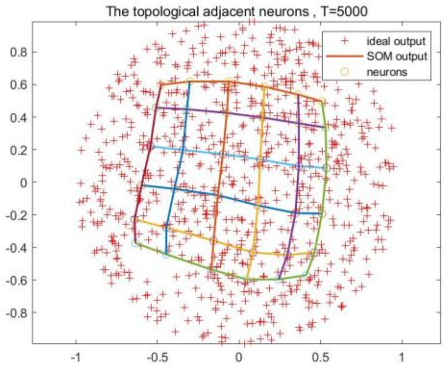












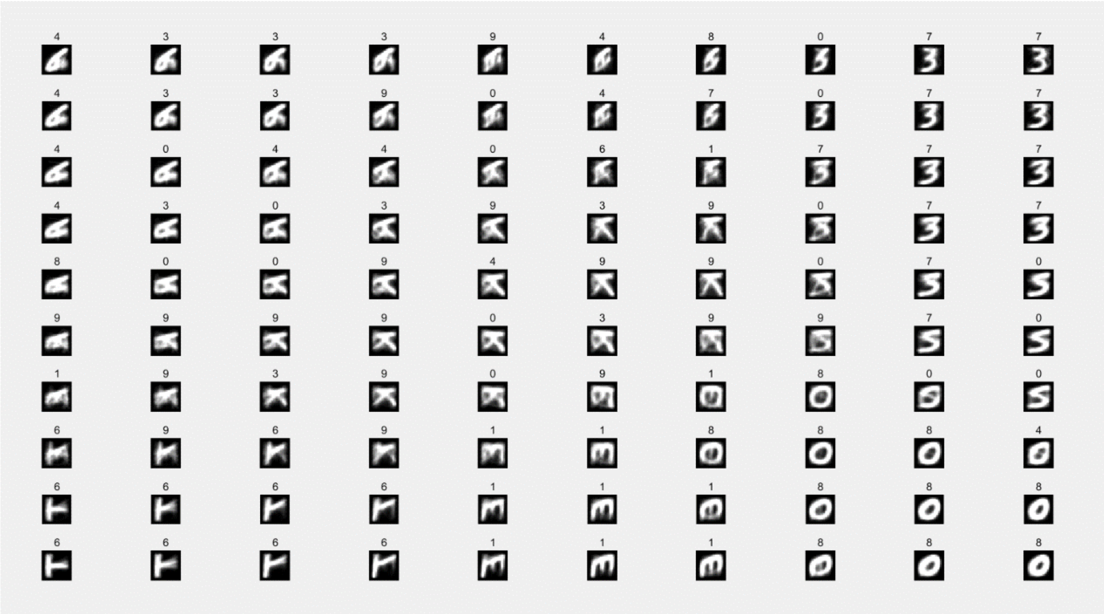
Based on the result, we can make a conclusion that with the increase of T, the neuron's distribution is more like a circle, and get better result.

```
%init
close all;clear;clc;
mkdir q3_b_image

X=randn(800,2);
s2=sum(X.^2,2);
trainX=(X.*repmat(1*(gammainc(s2/2,1).^(1/2))./sqrt
(s2),1,2))'; %2x800 matrix

%para
w=rand(2,6,6);%randomly init weigth 36 neurons in
output layer
sigma0=sqrt(6^2+6^2)/2;%M=6 N=6
eta0=0.1;
T=600;%iterations
tau1=T/log(sigma0);
tau2=T;
eta=eta0;
sigma=sigma0;

%algorithm
```

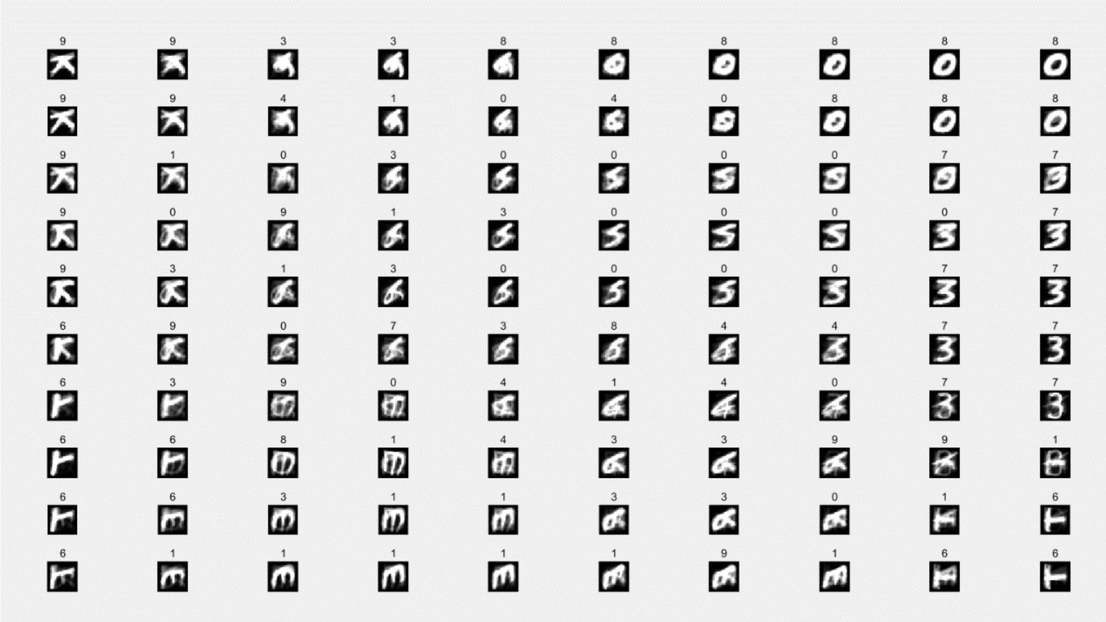
```
for n=1:T
    i=randperm(size(trainX,2),1);%randomly select
vector x
    %competitive process
    distance=zeros(6,6);
    for row=1:6
        for col=1:6
            distance(row,col)=sqrt((trainX(1,i)-
w(1,row,col)).^2+(trainX(2,i)-w(2,row,col)).^2);
        end
    end

[min_row,min_col]=find(distance==min(min(distance))
);
    %adaptation process
    for row=1:6
        for col=1:6
            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=1:6
    plot(w(1,:,i),w(2,:,i),'LineWidth',1.5);
    scatter(w(1,:,i),w(2,:,i),'o');
    hold on;
end
for j=1:6
    w_1 = reshape(w(1,j,:),1,6);
    w_2 = reshape(w(2,j,:),1,6);
    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');
saveas(gcf,strcat("q3_b_image/b_",num2str(T),".jpg"
));
```

(c) My matriculation number is A0224725H, so I omit classes 2 and 5, and use 0,1,3,4,6,7,8,9 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.



The results imply that the more train epochs, the higher accuracy is. But when the epoch is big enough (T=10000), the accuracy starts to fall, which means under-fitting.
But the accuracy reach the best result(63.5%), so I think the initial learning rate is 0.1, so I try to increase the learning rate then I find that the accuracy increases a lot! When the learning rate reaches 1, we can get the accuracy of test set is 63.8%



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.

```
%% Clear all
close all
clear
clc
tic

%% Load labels and data

mkdir q3_c_image
%load
load('characters10.mat');

train_data=im2single(train_data);
test_data=im2single(test_data);
test_data=test_data';
train_data=train_data';

trainIdx = find(train_label ==0 | train_label==
1|train_label == 3 | train_label ==4|train_label == 6 |
train_label == 7|train_label == 8 | train_label == 9);
testIdx = find(test_label == 0 | test_label==
1|test_label == 3 | test_label ==4|test_label == 6 |
test_label == 7|test_label == 8 | test_label == 9);

train_label=train_label';
```

```
test_label=test_label';

train_data = train_data(:,trainIdx);
train_label = train_label(:,trainIdx);

test_data = test_data(:,testIdx);
test_label = test_label(:,testIdx);

trainX = train_data;
%% Initialise
N = 784;                                      % dimnension of
input vector
vert_M = 10;                                   % vertical
neurons
hor_M = 10;                                    % horizontal
neurons
M = vert_M * hor_M;                             % number of
output neurons
iter = 10000;                                   % number of
iterations
som_width = 10;                                % size of SOM
(width)
som_height = 10;                               % size of SOM
(height)
init_rate = 1;                               % initial rate
init_width = sqrt( (10^2 + 10^2)) / 2;      % initial
width
init_w = rand(N,M);                             % initial weight
w = init_w;                                    % initial weight
grid = getgrid(vert_M,hor_M);                    % initialise
grid
label(1:vert_M,1:hor_M) = inf;                   % initialise
label matrix

%% Algorithm start
for n = 0:iter
    disp(strcat('Iteration: ', int2str(n)))
    [rate, width] = getparam(init_rate,
init_width,n,iter);
    for idx = 1:600
        sample = trainX(:,idx);     % get a sample vector
        [winner,grid_col,grid_row,dis] =
getwinner(w,sample,M,vert_M,hor_M);    % get winning
neuron
```

```
        h =
getneighbourhood(vert_M,hor_M,grid_row,grid_col,width);
   % find influence neighbourhood
        label(grid_row,grid_col) = train_label(:,idx);
        reshape_h = reshape(h',[1,100]);
        for i = 1:M
            w(:,i) = w(:,i) + rate * reshape_h(1,i) *
(sample - w(:,i));       % calculate new weight
        end
    end
end


%% Plot SOM
reshaped_label = reshape(label',[1 100]);
for A = 1:100
    subplot(10,10,A)
    graph = reshape(2*(w(:,A)),[28 28]);
    imshow(graph');
    title(sprintf('%0d',reshaped_label(1,A)));
end

%% Calculate training and test accuracy
test_acc =
getacc(test_data,test_label,w,reshaped_label);
train_acc =
getacc(train_data,train_label,w,reshaped_label);

toc

%% Functions
function accuracy = getacc(data,data_label,w,som_labels)
num_inputs = size(data,2);
num_weights = size(w,2);
for i = 1:num_inputs
    min = inf;
    for j = 1:num_weights
    diff = norm(data(:,i) - w(:,j));
    if diff < min
        min = diff;
        min_idx = j;
    end
    end
    test_grid(1,i) = som_labels(1,min_idx);
```

```
end
counter = 0;
for i = 1:num_inputs
   if test_grid(1,i) == double(data_label(1,i))
      counter = counter + 1;
   end
end
accuracy = counter/num_inputs;
end

function grid = getgrid(x, y)
num = 1;
for i=1:x
    for j=1:y
        grid(i,j) = num;
        num = num + 1;
    end
end
end

function [rate, width] = getparam(init_rate,
init_width,n,iter)
    rate = init_rate * exp(-n/iter);
    T1 = iter/(log(init_width));
    width = init_width * exp(-n /T1);
end

function [winner,grid_col,grid_row,dis] =
getwinner(w,sample,M,vert_M,hor_M)
for i = 1:M
    dis(1,i) = getnorm(w(:,i),sample);
end
winner = find(dis==min(dis));
winner = winner(1,1);
grid_col = mod(winner,hor_M);
if grid_col == 0
    grid_col = 10;
end
grid_row = ceil(winner/vert_M);
end

function h =
getneighbourhood(vert_M,hor_M,grid_row,grid_col,wid
th)
```

```
for i = 1:vert_M
    for j = 1:hor_M
        d(i,j) = -1 * (getnorm( [i j] , [grid_row
grid_col] ) )^2;
        h(i,j) = exp(d(i,j) / (2*width^2));
    end
end
end

function dist = getnorm(a,b)
dist = norm(a-b);
end
```