

**Name : LUO ZIJIAN**

**Matric.No： A0224725H**

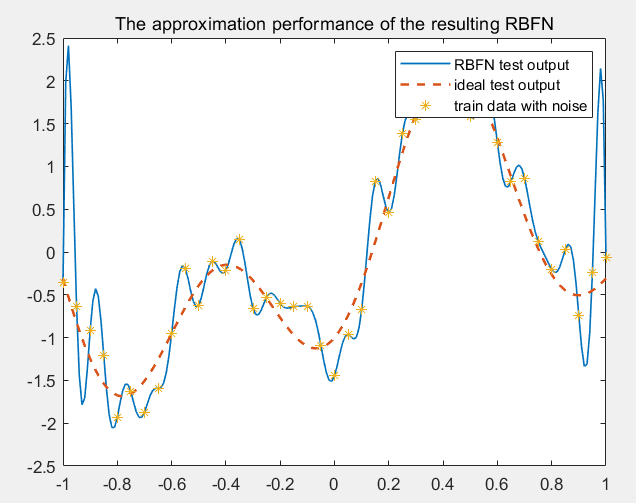
**MUSNET： E0572844**

**Subject： NEURAL NETWORKS**

**Assignment: HOMEWORK THREE**

# Solution 1

1. Follow the instructions in the sides of RBFN, we can get this picture like that



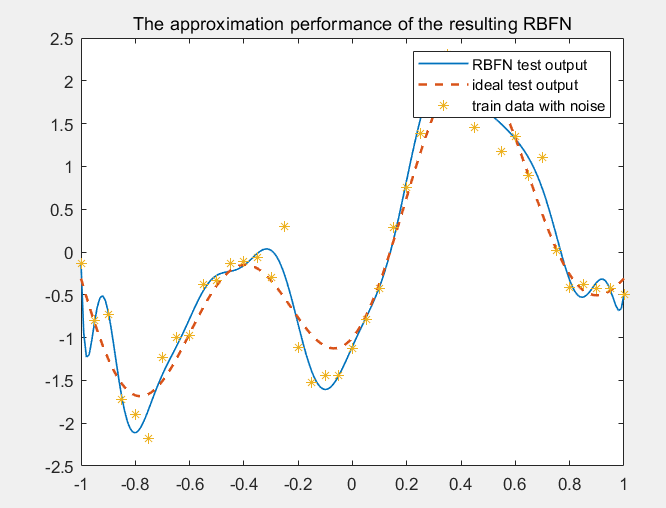
According to the result, it is clear that the output of RBFN test set is not close to the ideal output. The MSE of train set is and the MSE of the test set is . In a word, this simulation is overfitting.

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'); |

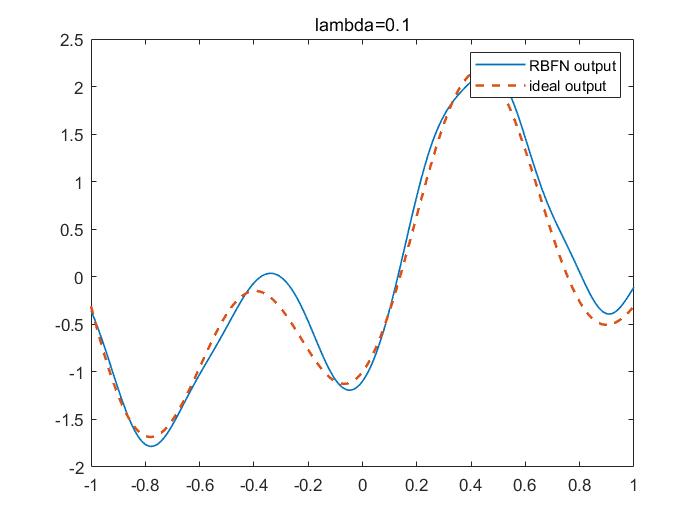
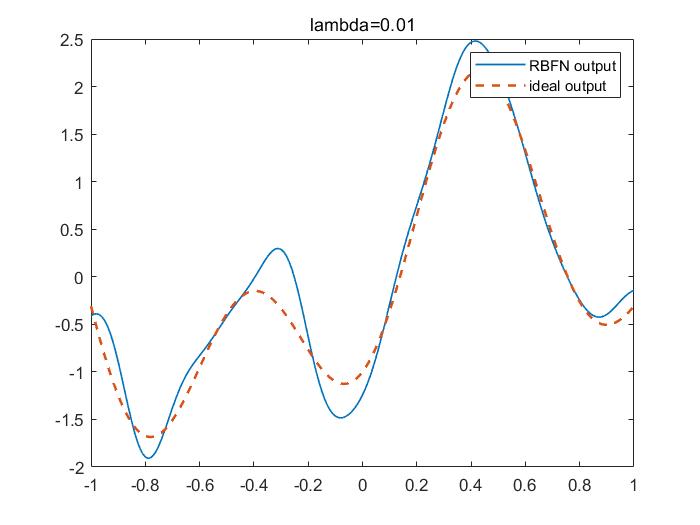
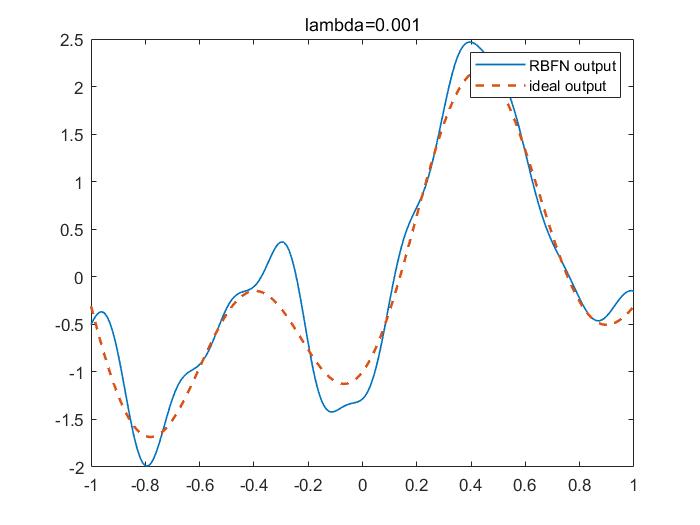
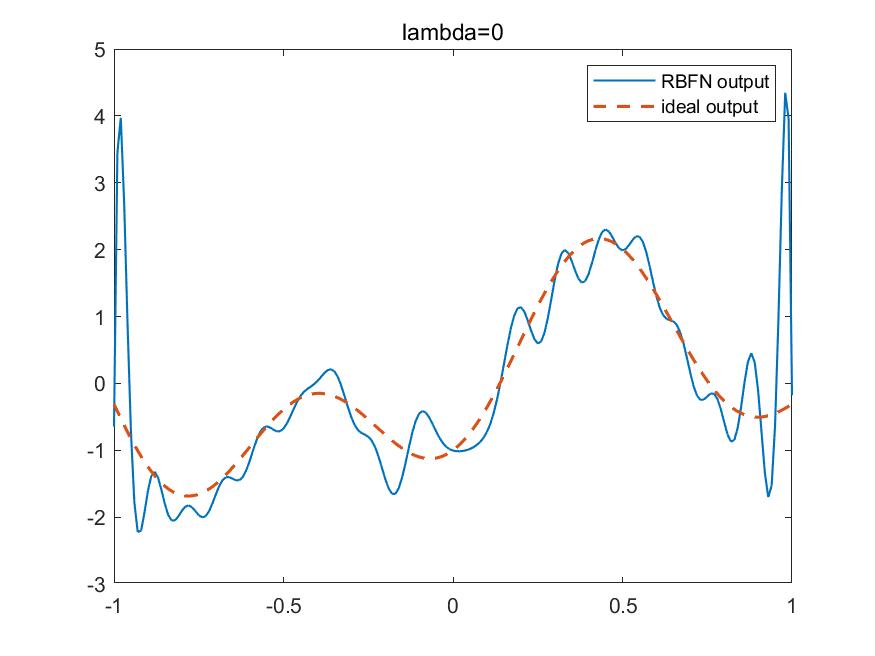
1. For this part, I randomly select 20 centers among the sampling points with the strategy of “Fixed centers selected at random”. Compared to the result of part a, it is clear that the output of test set with the strategy of fixed centers is more close to ideal output than that of test set without fixed 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).

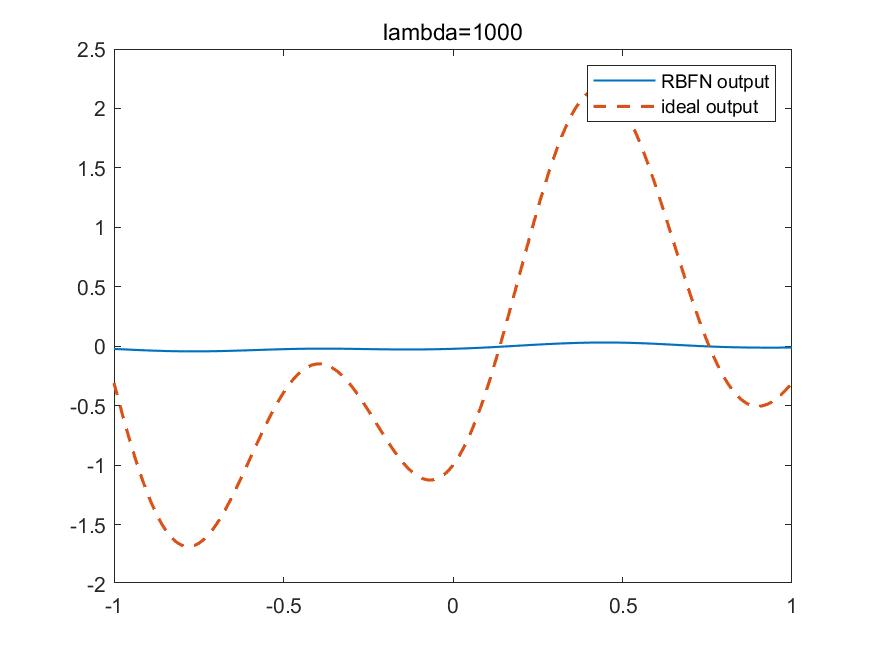
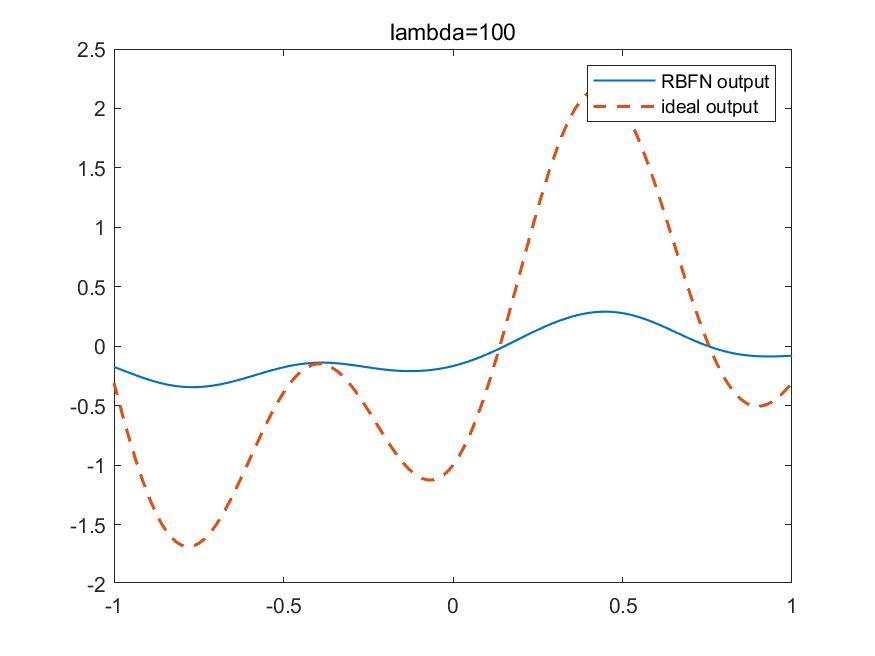
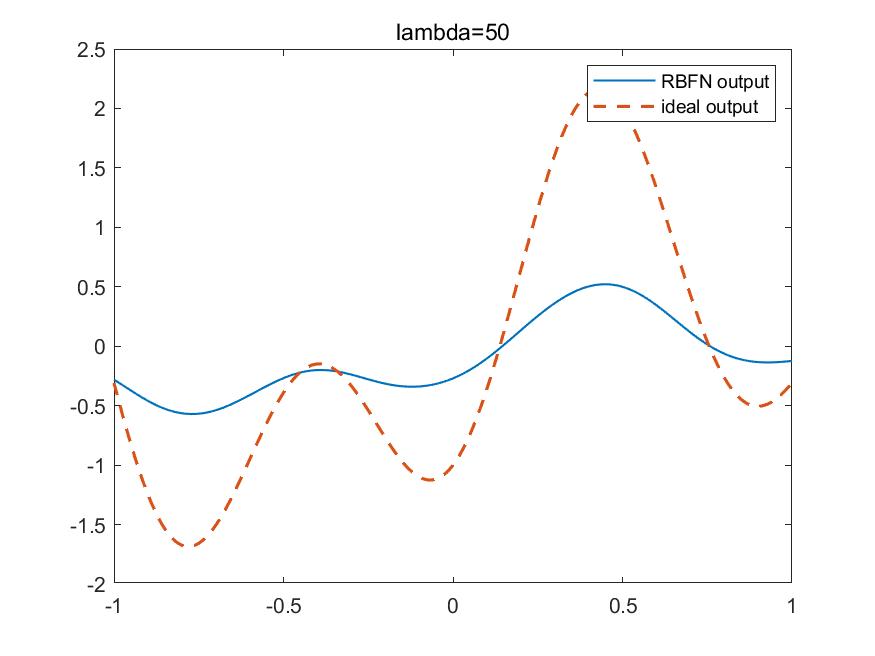
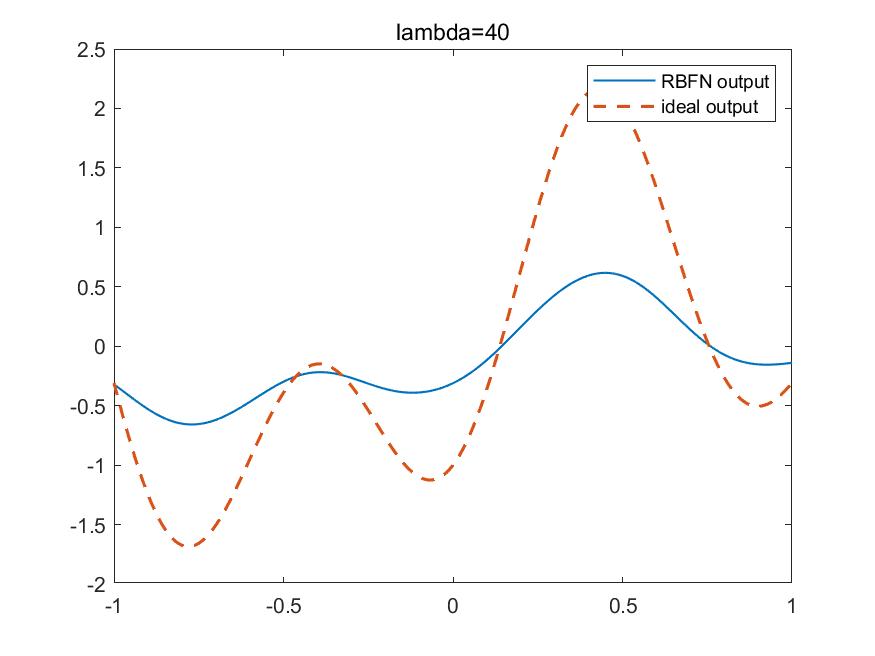
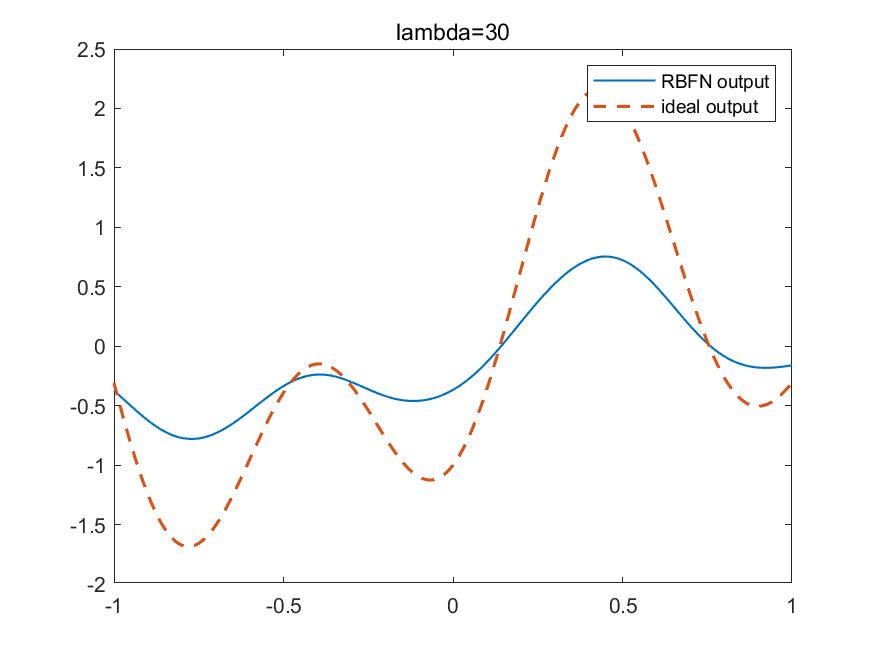
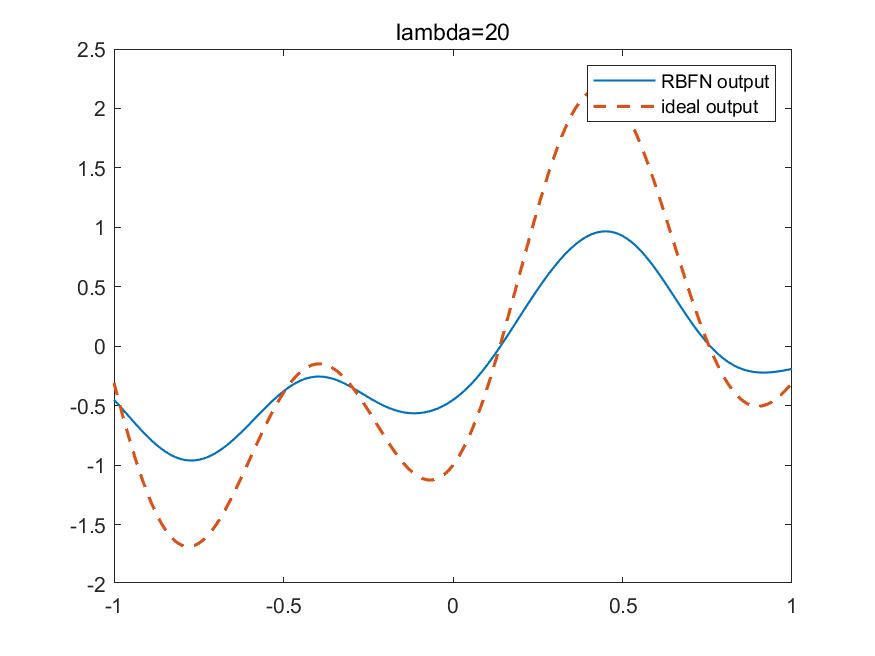
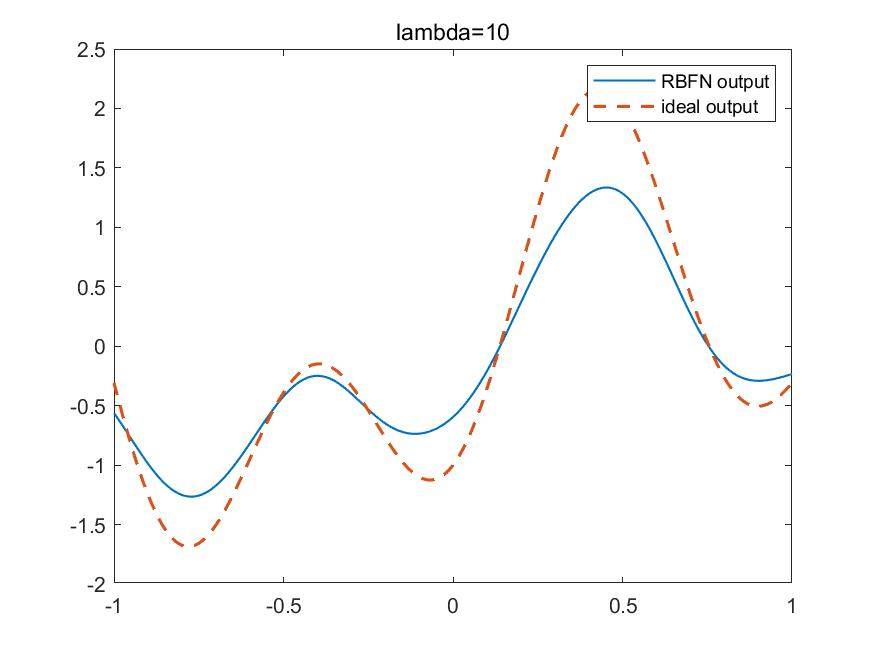
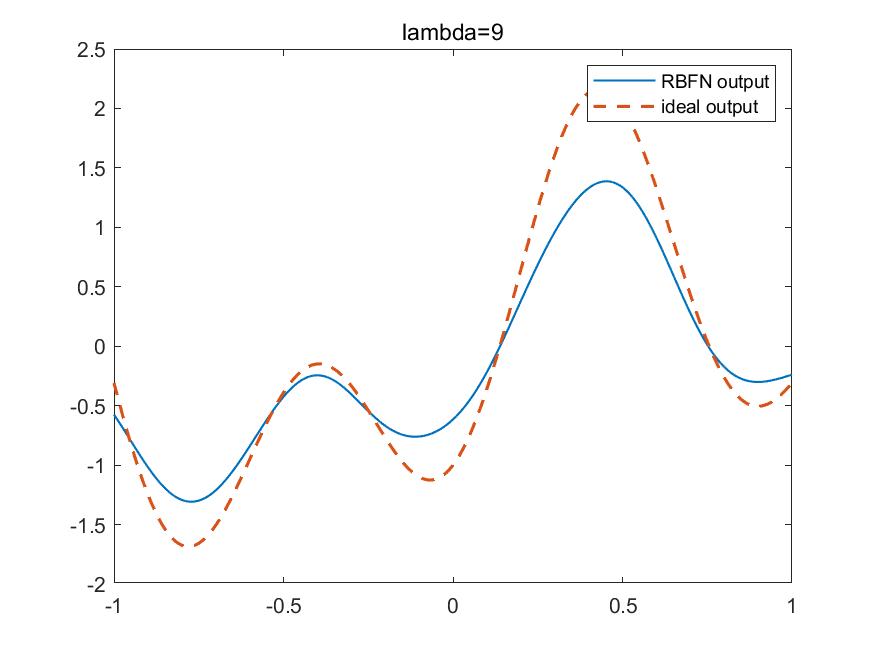
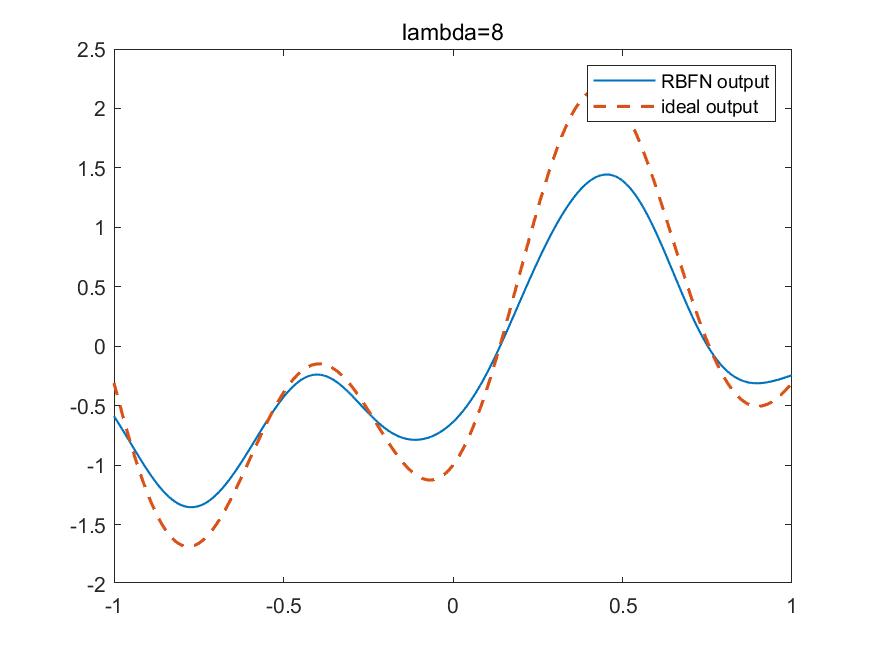
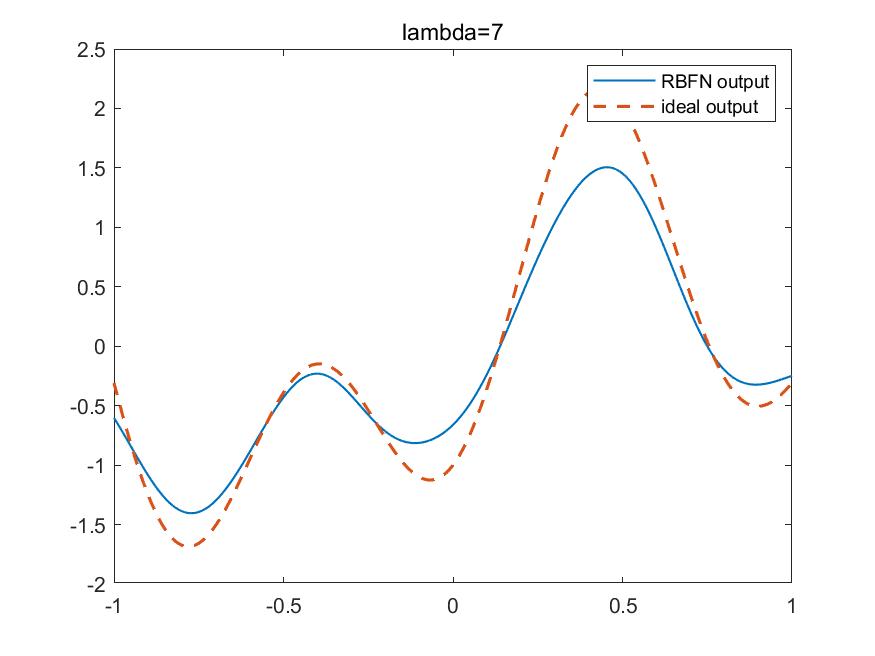
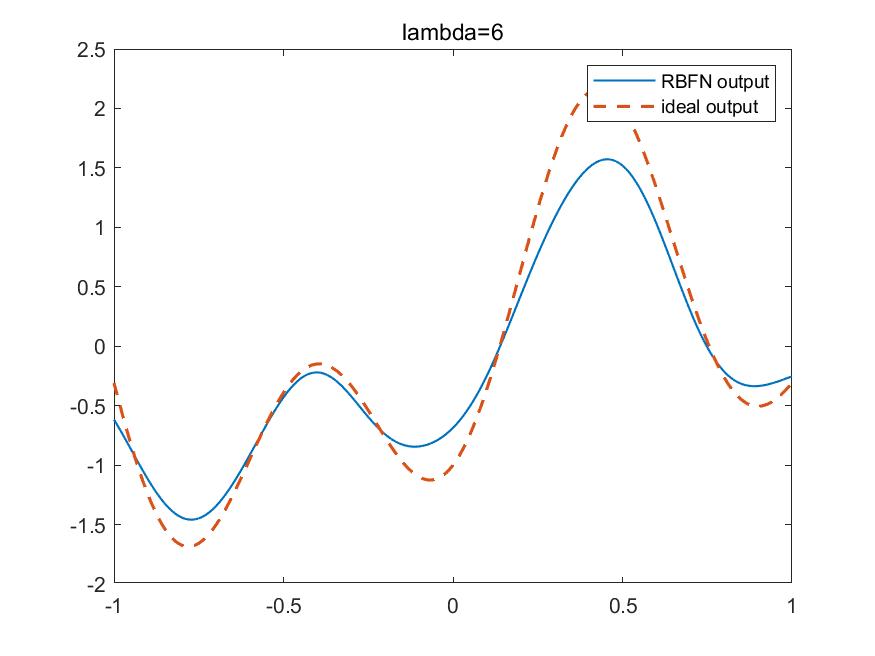
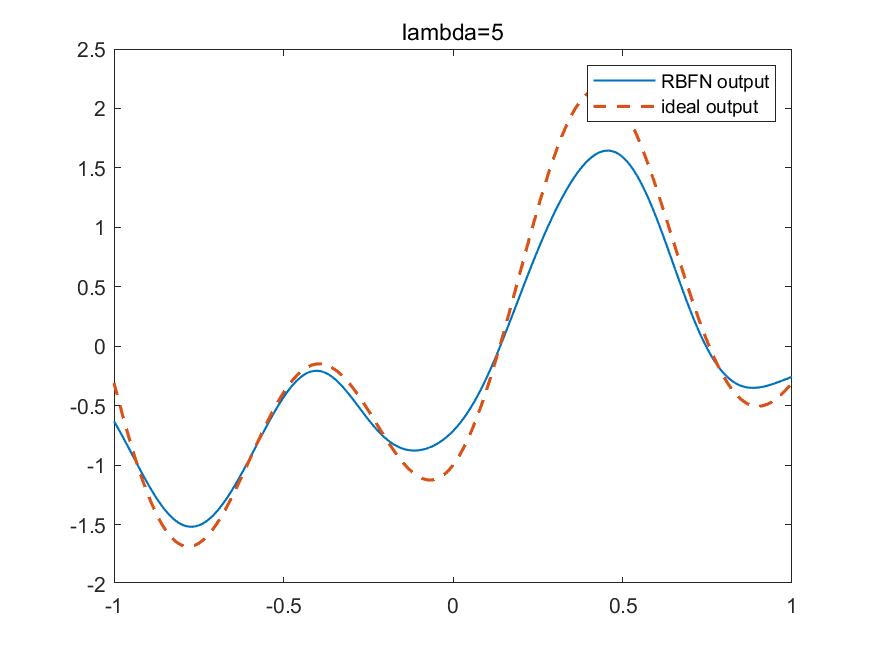
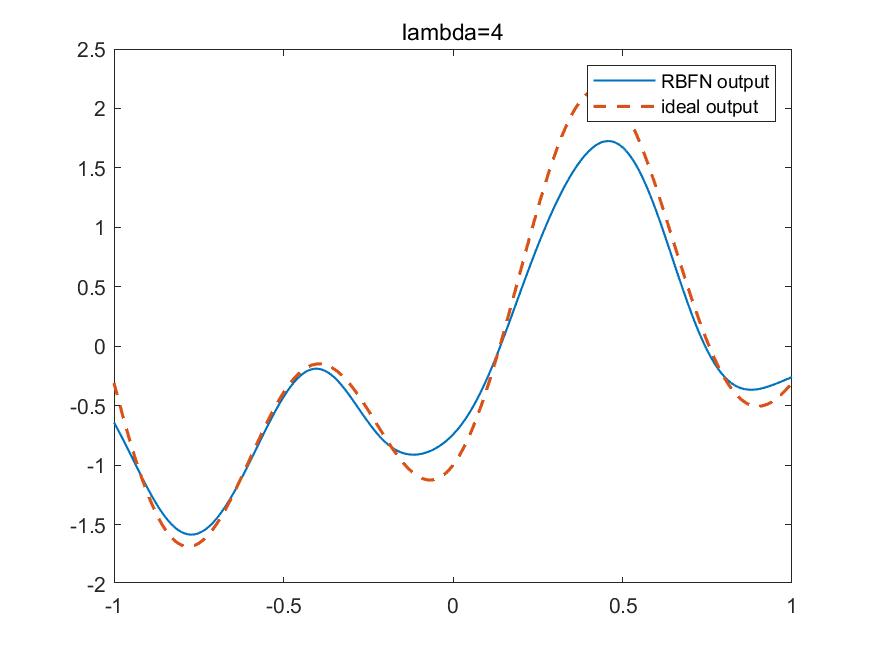
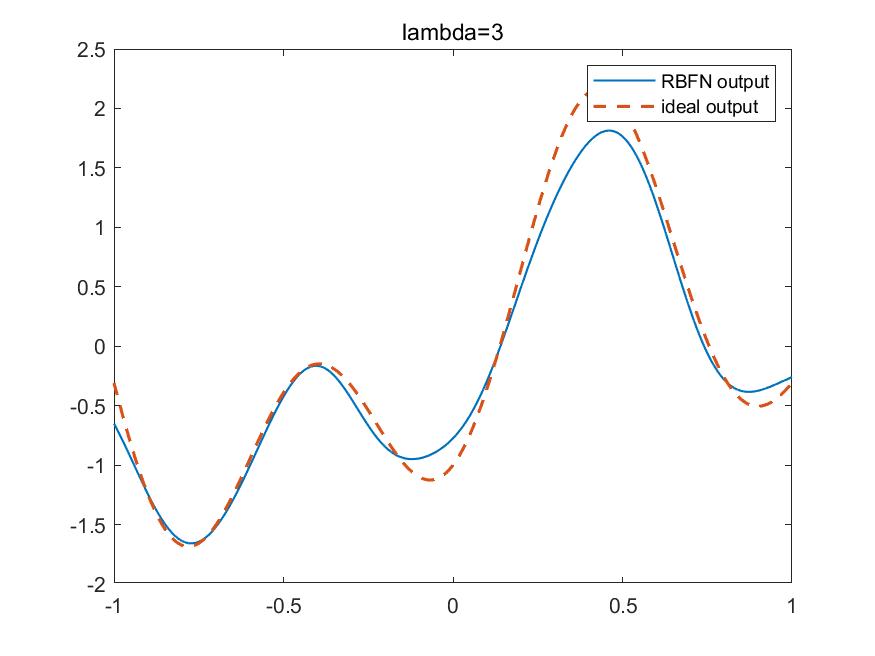
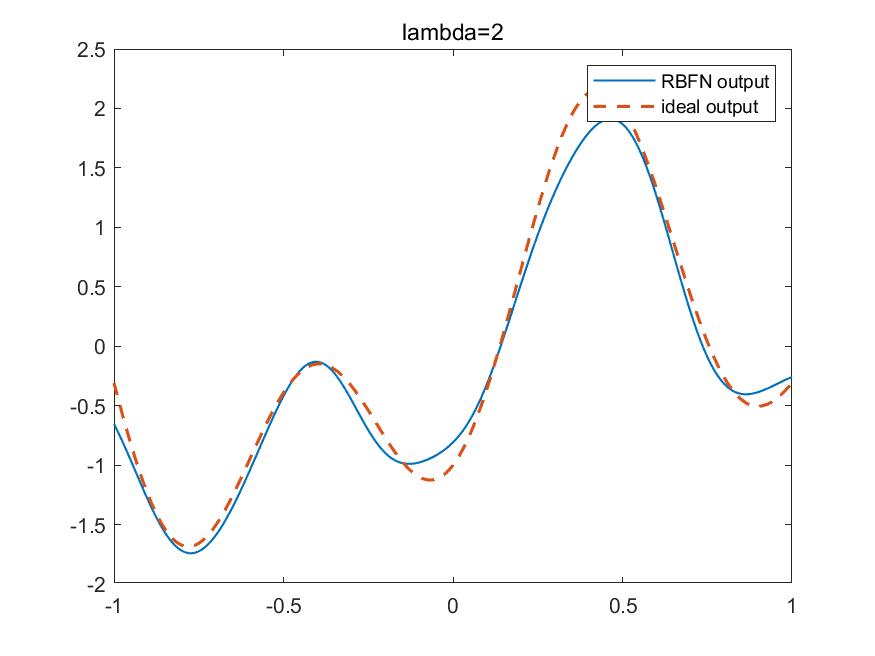
Especially for the difference of MSE of train set between part a and part b, the train error of this part is larger than part a. As a result of overfitting, not all the train samples being fitted.



|  |
| --- |
| %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'); |

1. For this part, I apply regulation method in part a, I get these conclusions.
2. 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.
3. 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.
4. 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.
5. 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.



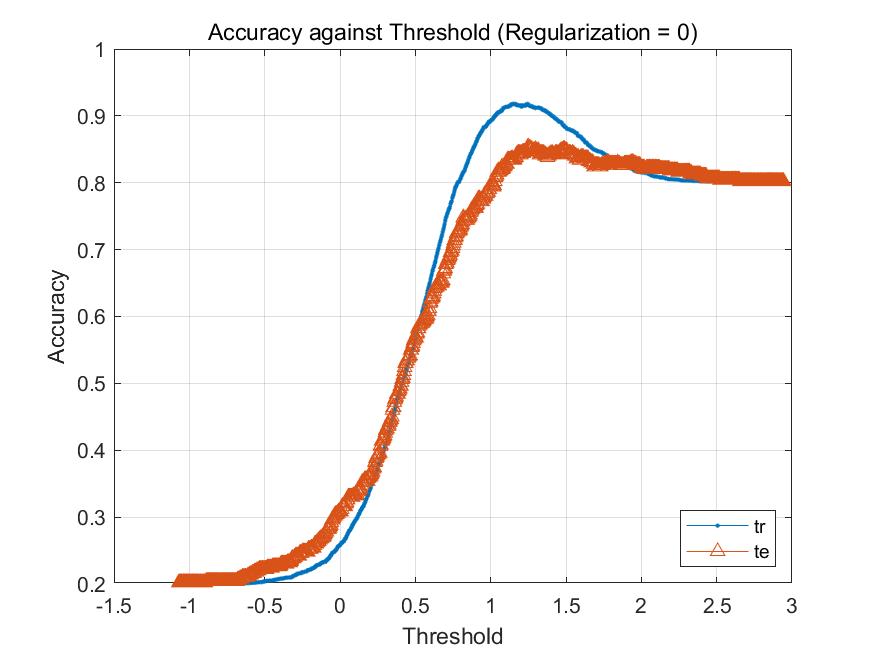


|  |
| --- |
| %init  close all;clear;clc;    %parameter  x\_train=-1:0.05:1;%uniform step 0.08  x\_test=-1:0.01:1;%uniform step 0.01  N=length(x\_train);  x=randn(1,N);%random Gaussian noise for xtain not for xtest  d=1.2\*sin(pi\*x\_train)-cos(2.4\*pi\*x\_train)+0.3\*x;%x with noise    for lambda=[0,0.001,0.01,0.1,1:10,20,30,40,50,100,1000]%regulation 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);  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 |

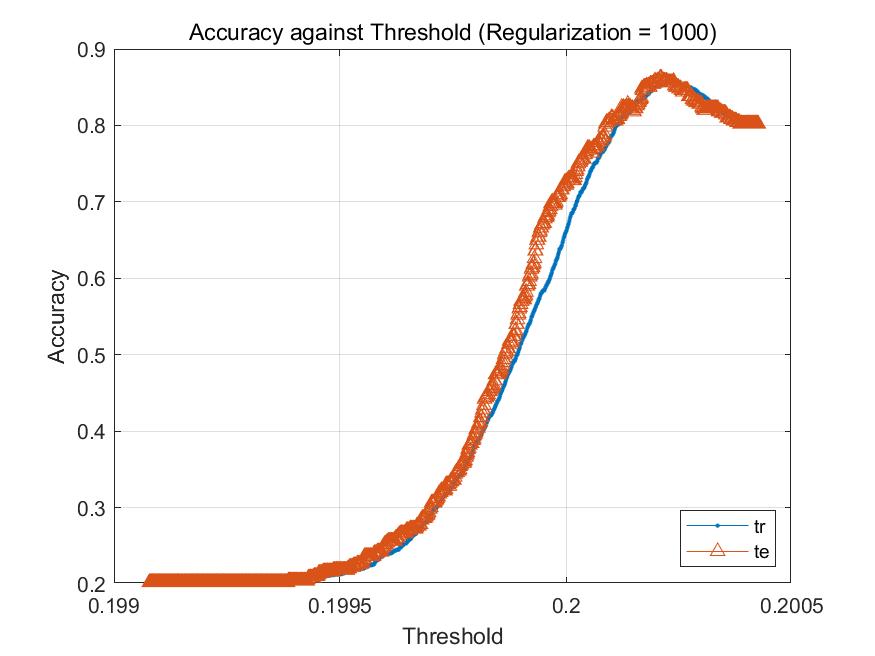
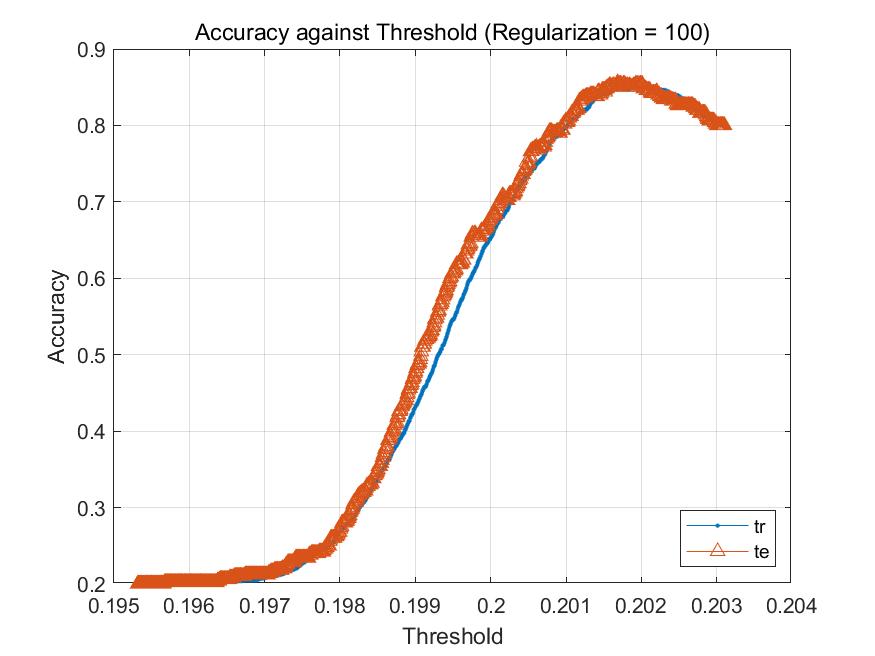
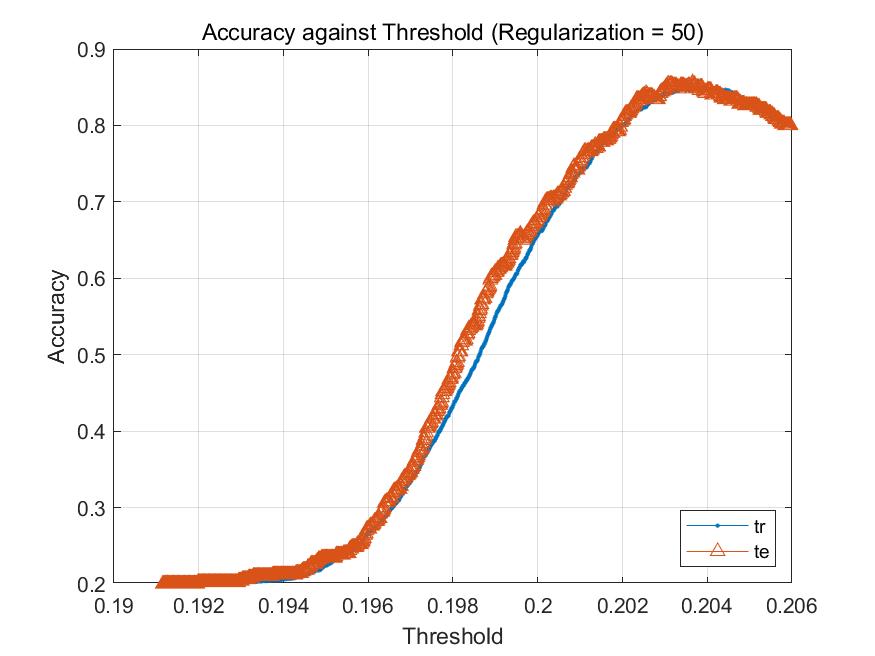
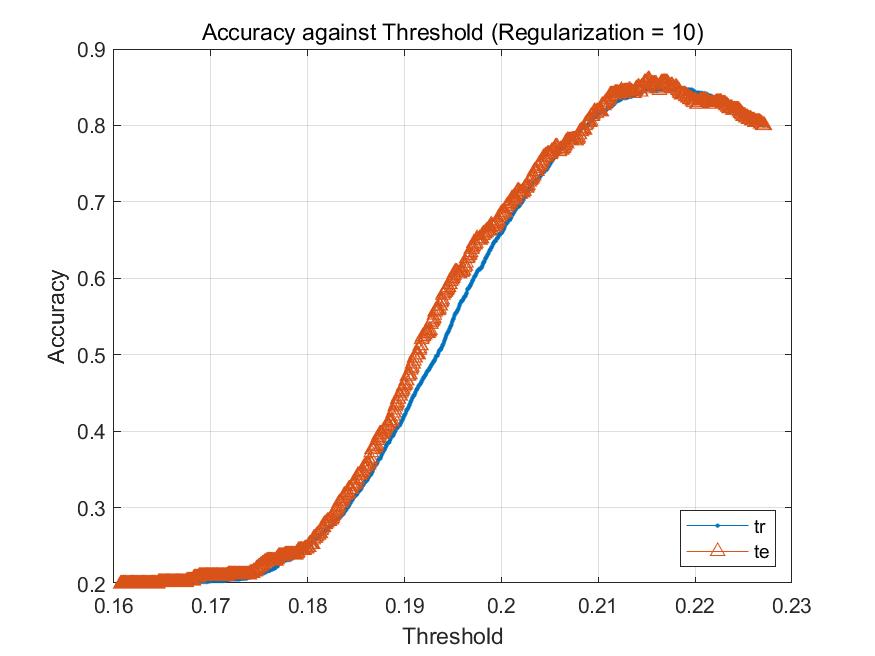
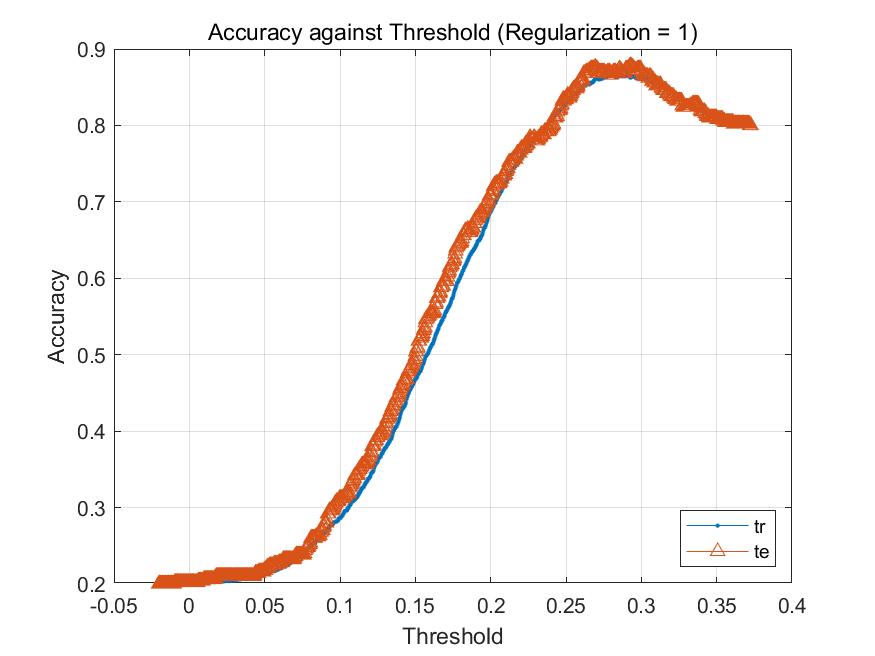
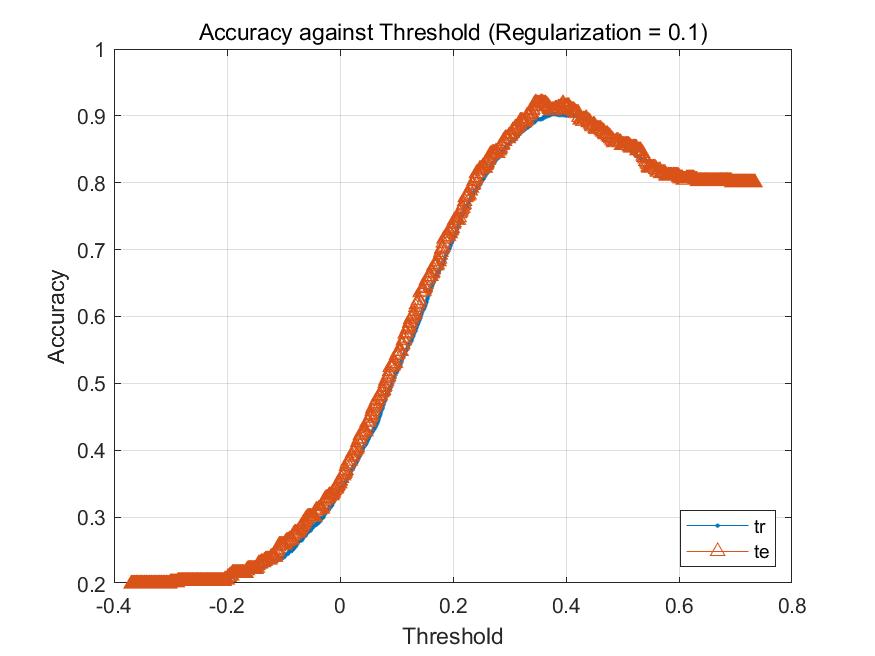
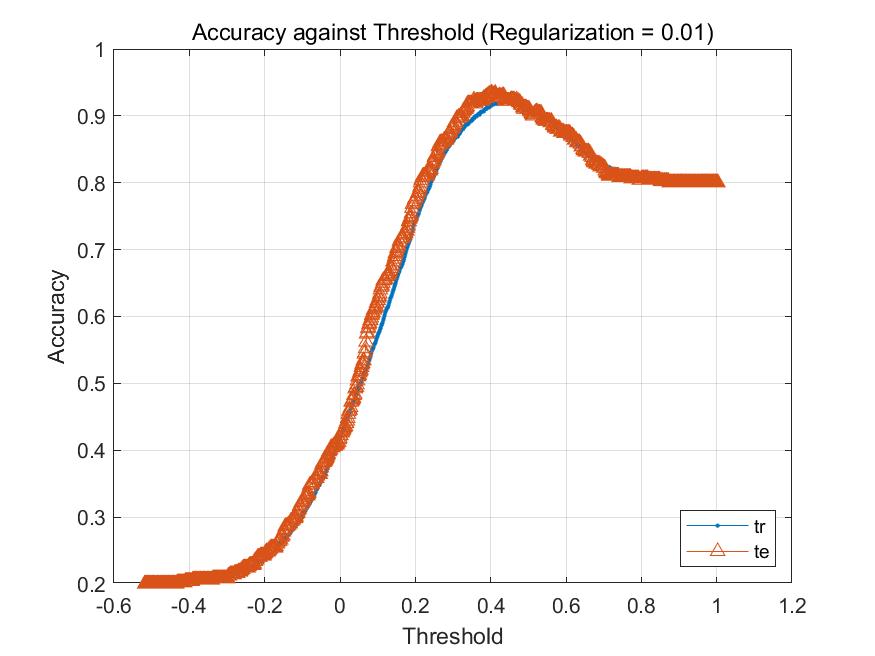
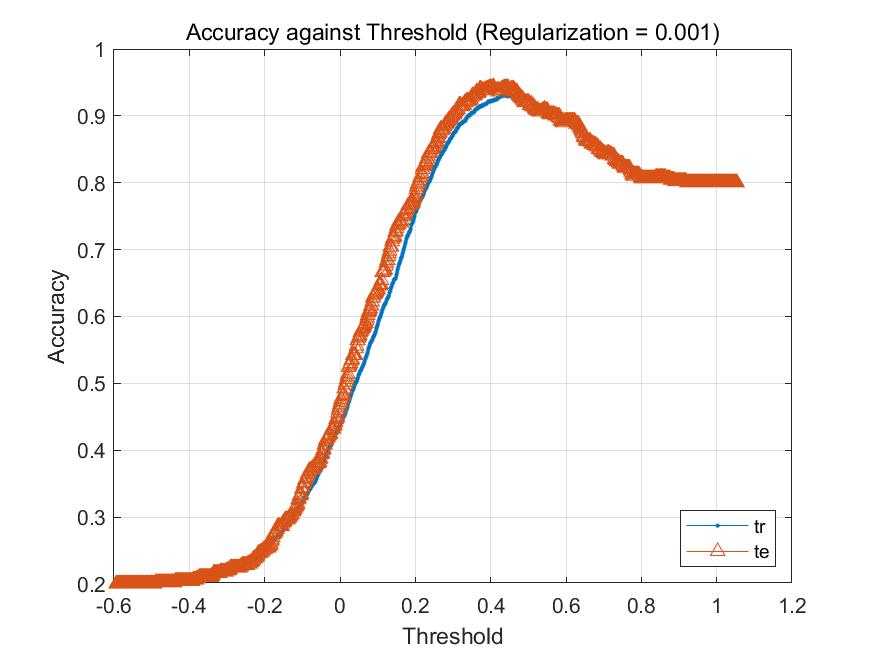
# Solution 2

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”.

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



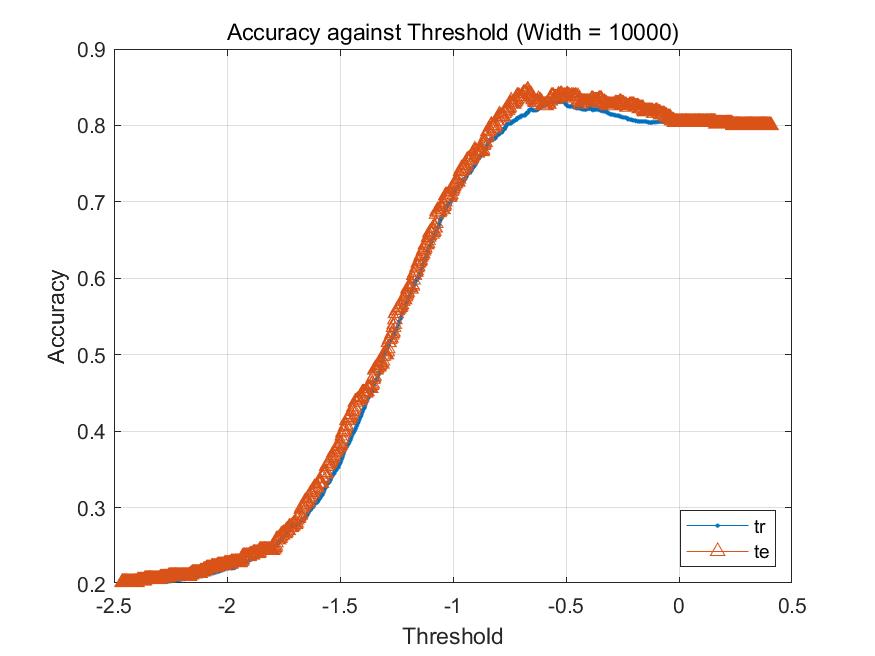
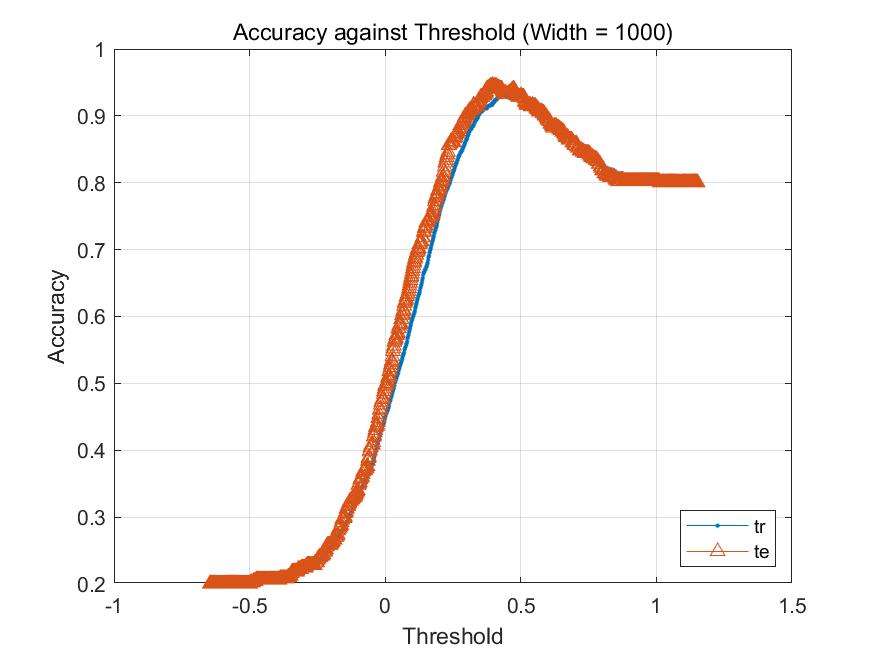
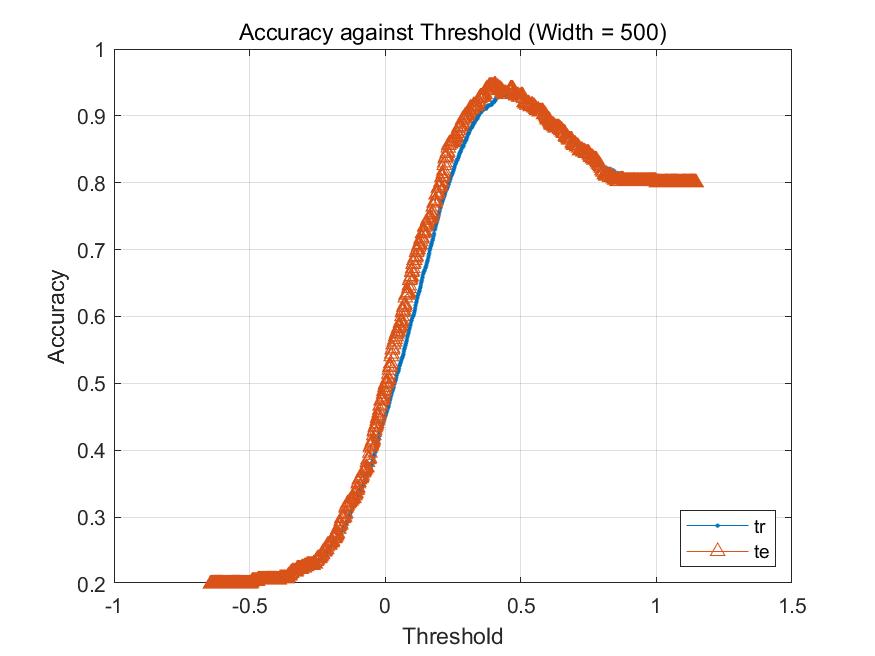
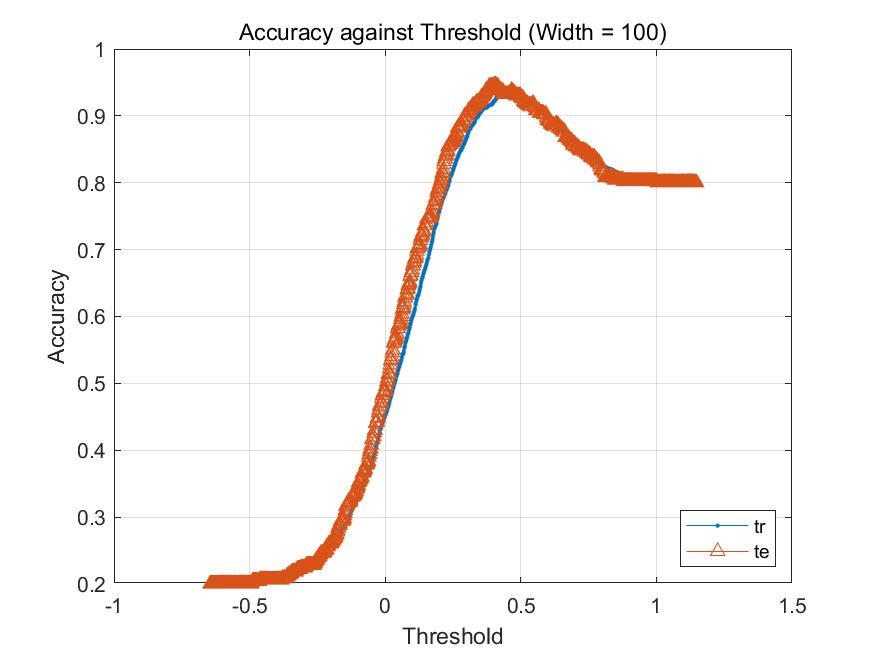
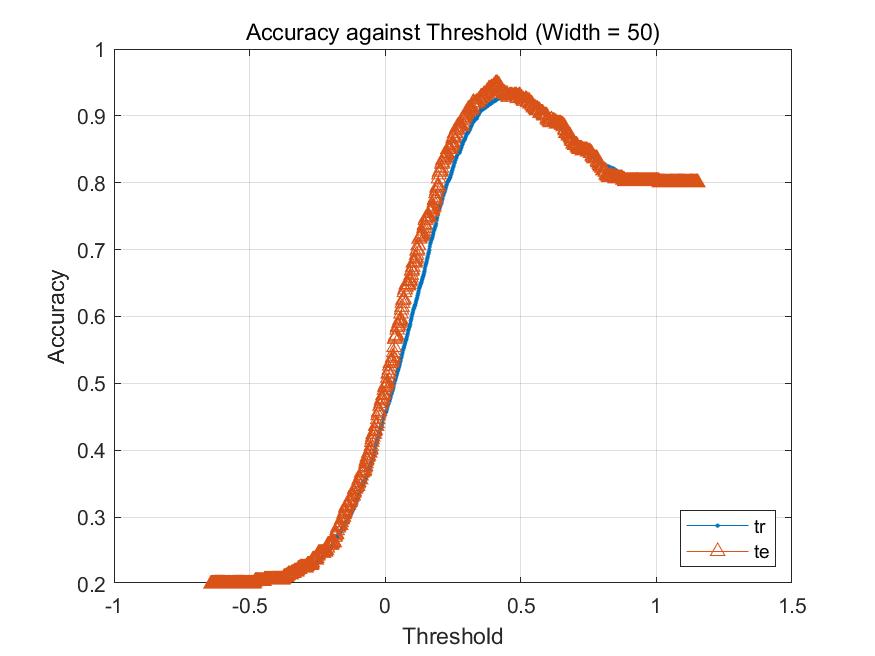
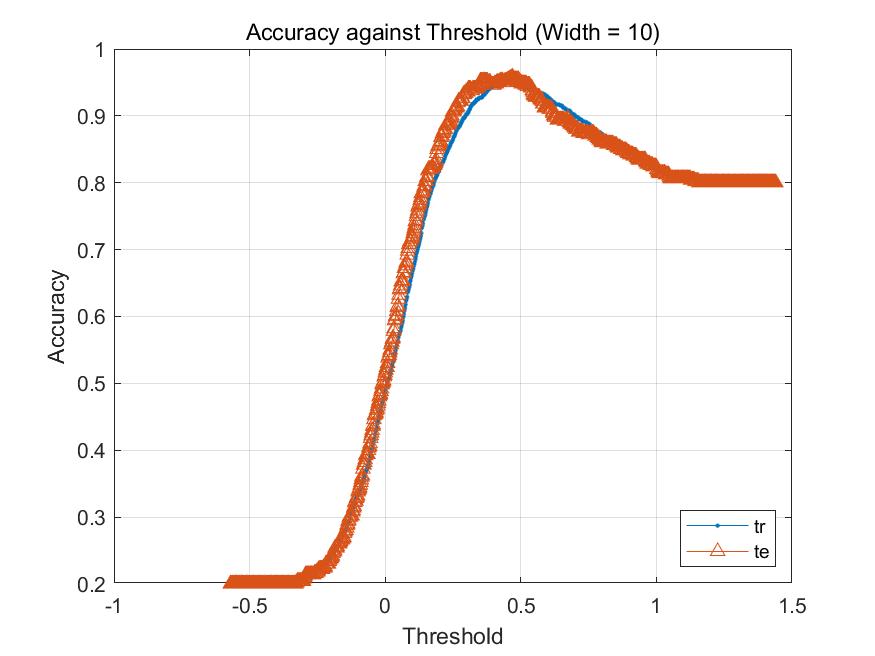
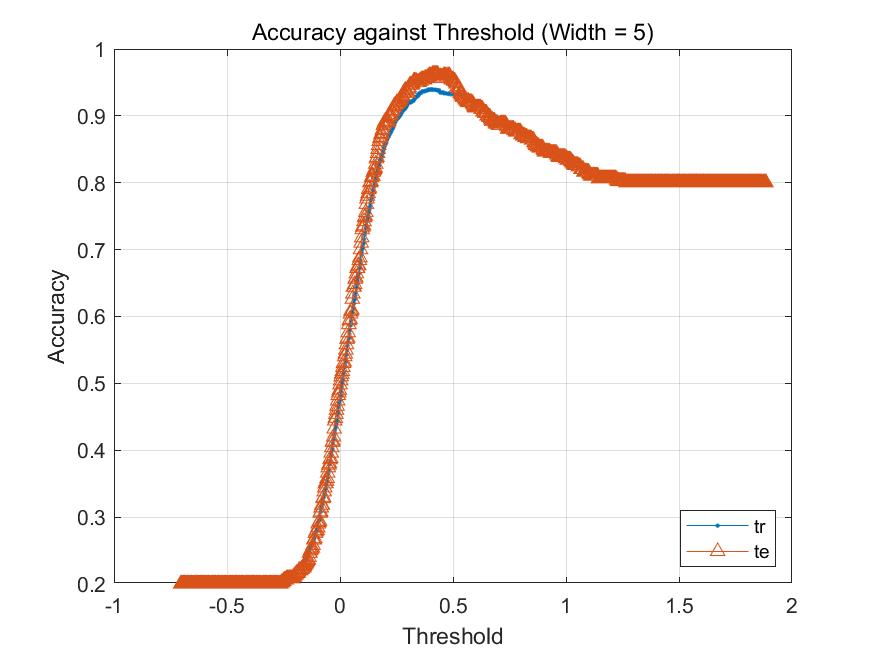
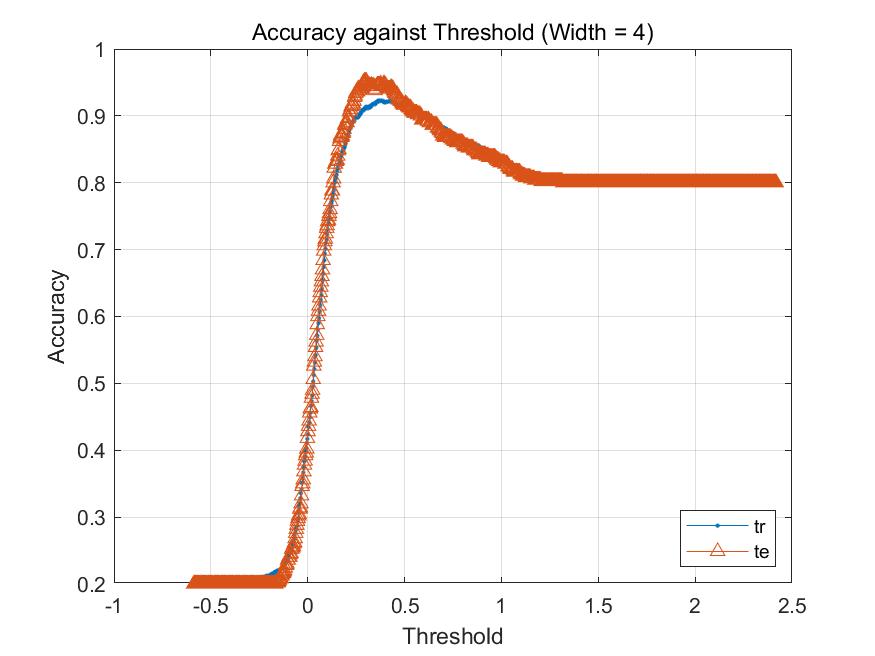
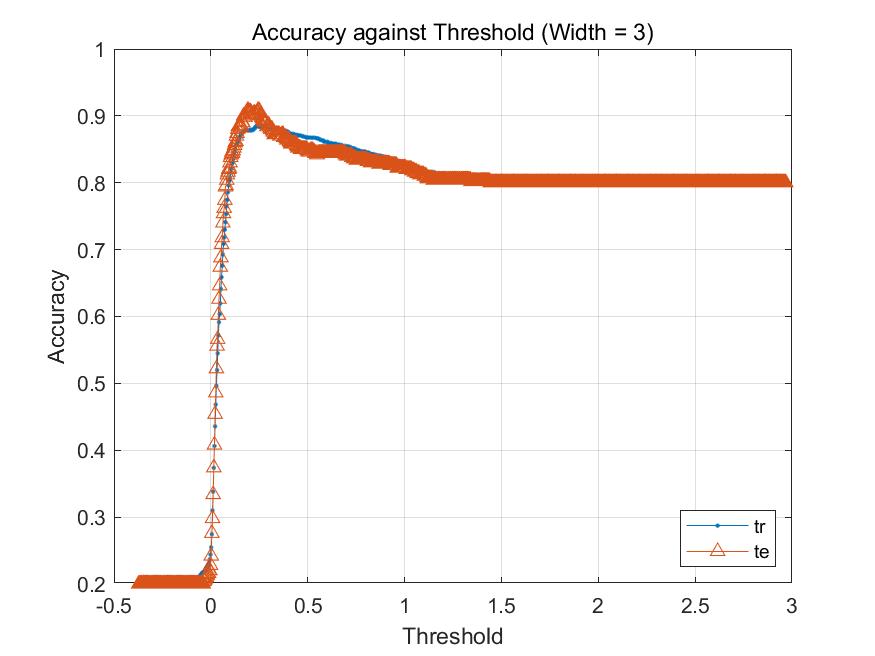
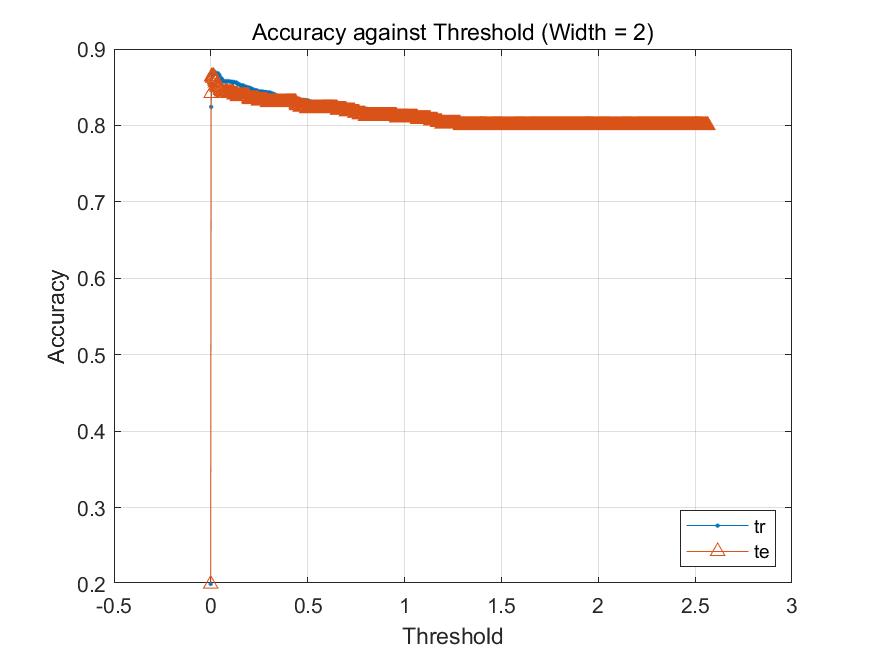
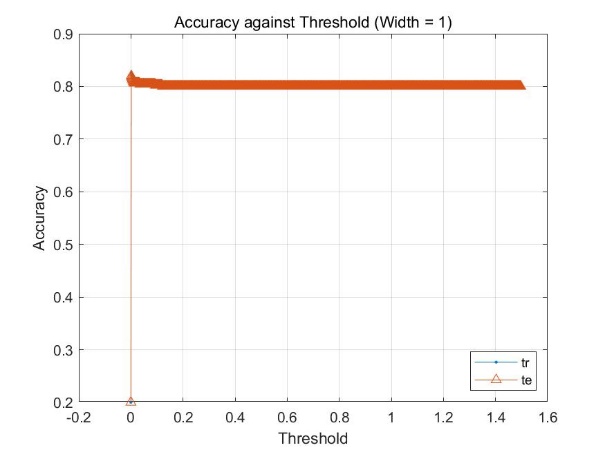
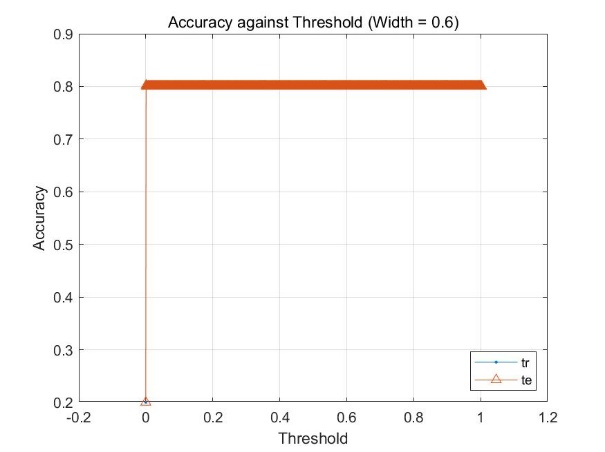
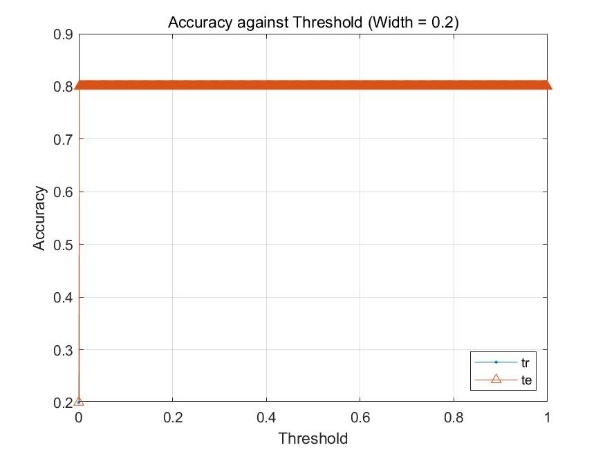
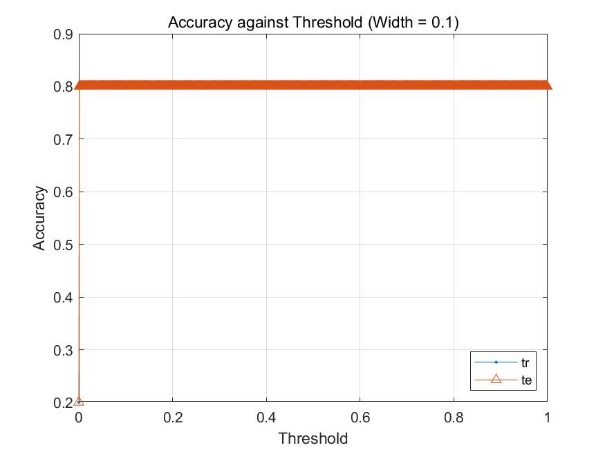
The figure is the accuracy both train set and test set using RBFN without regulation. The result seems that the accuracy of train set(88.5%) is lower than that of test set(78.3%) when threshold is 1.



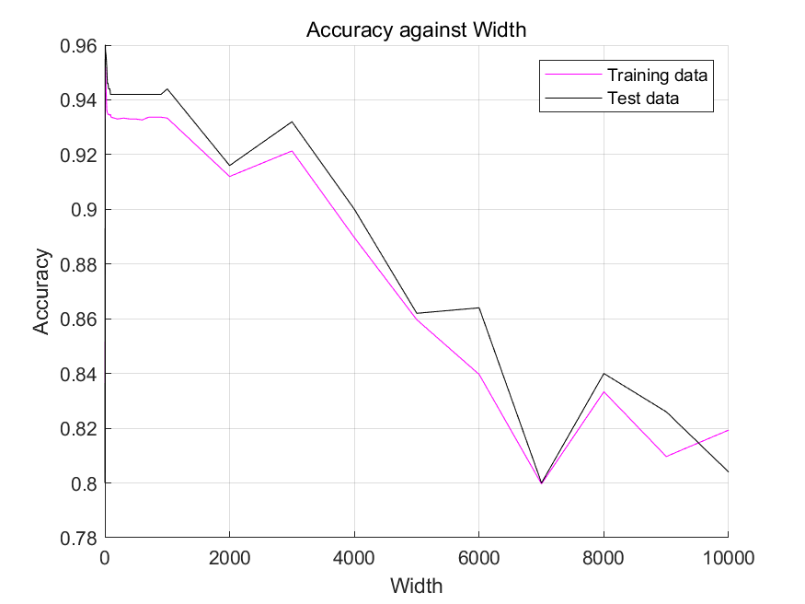
These figures above imply that with the increase of the value of regulation, the accuracy of test set looks like closer to that of train set, and the range of the threshold becomes narrower. However, if we print the accuracy of both, we find that the accuracy of both do not change too much, actually!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

1. 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).

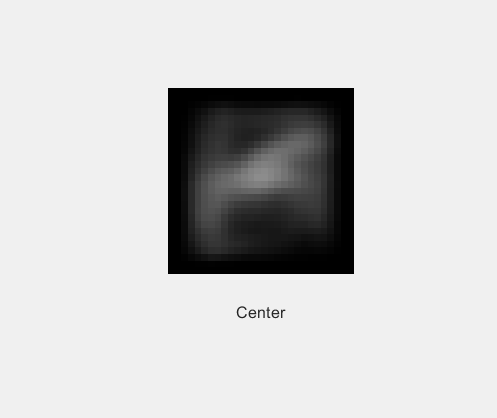
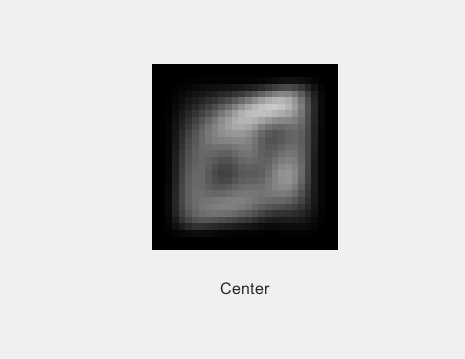


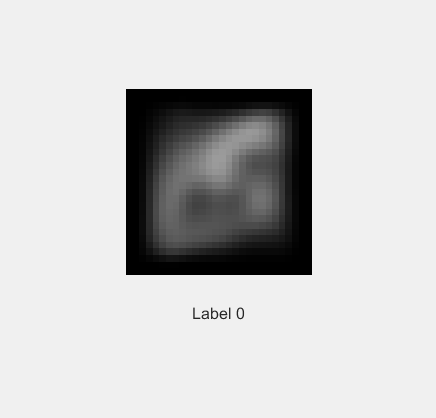
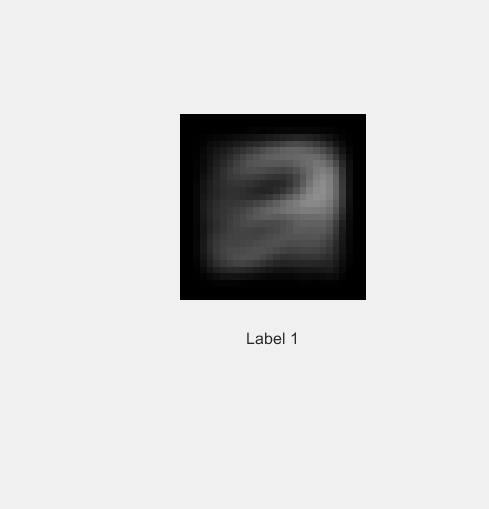
Compared with the result in part a, the accuracy of train set is lower than that in part a, but the accuracy of test set is higher than that in part a. Then I vary the value of width from 0.1 to 10000. The results imply that the accuracy of train set starts at 79.6%, and jumps to 87.8% when width =0.25.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

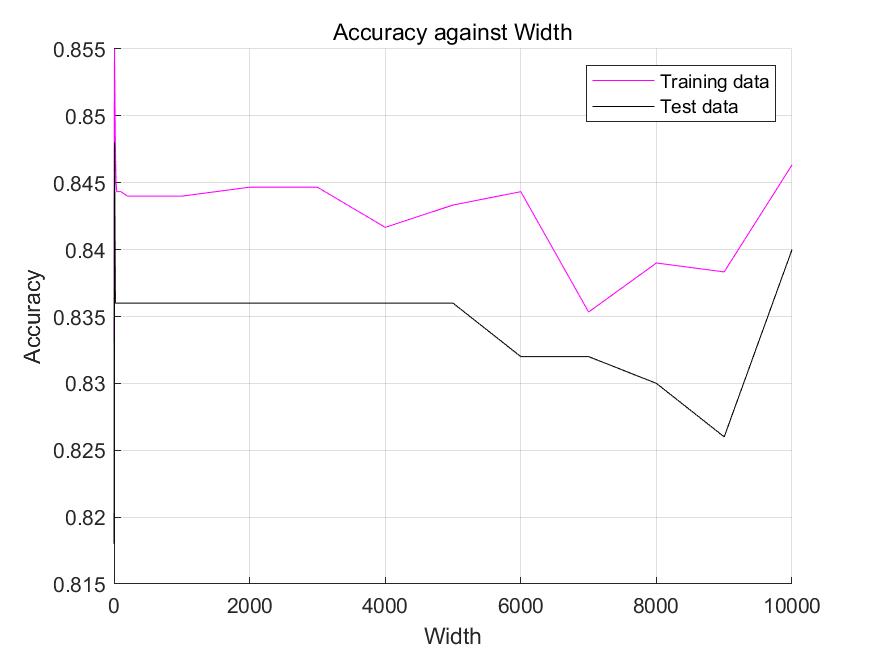
Apply “K-Mean Clustering” with 2 centers, we get 2 centers visualized like this.





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.

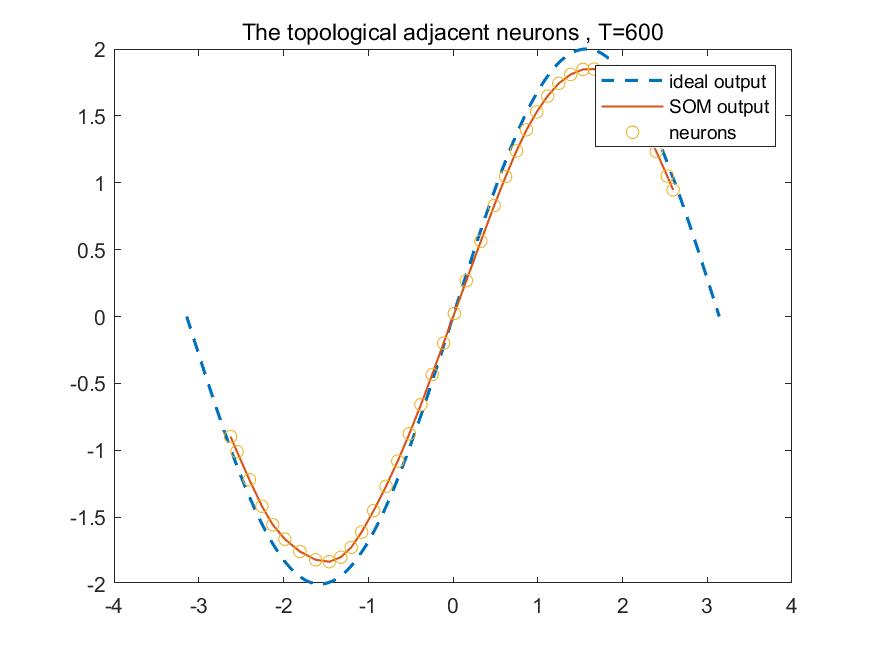
According to below figure, the result imply that the “K-Mean Clustering” shows a good performance, which reaches a high accuracy by using only 2 centers (100 centers in part b), with the accuracy of train set (84.5%) and the accuracy of test set (83.6%).



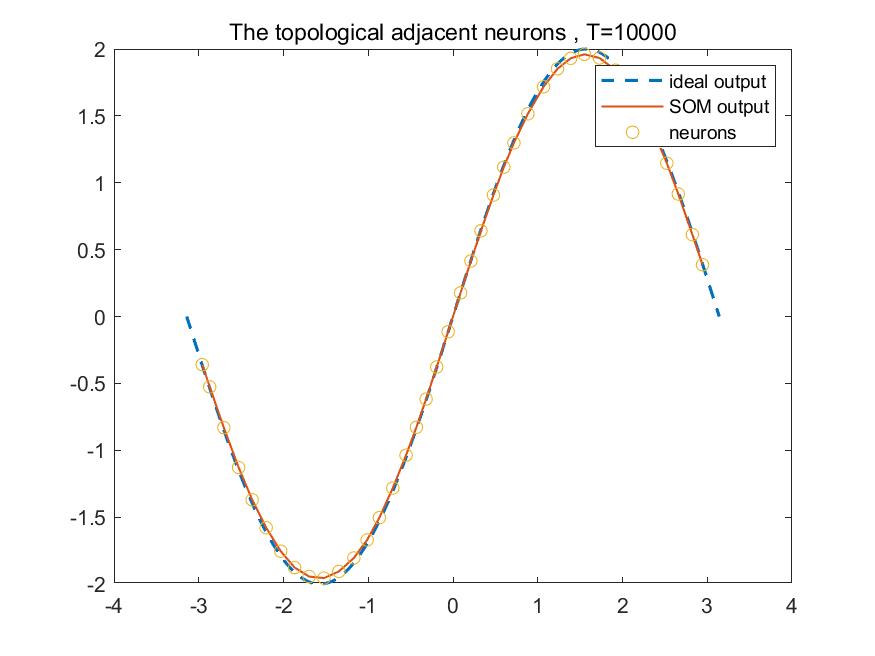
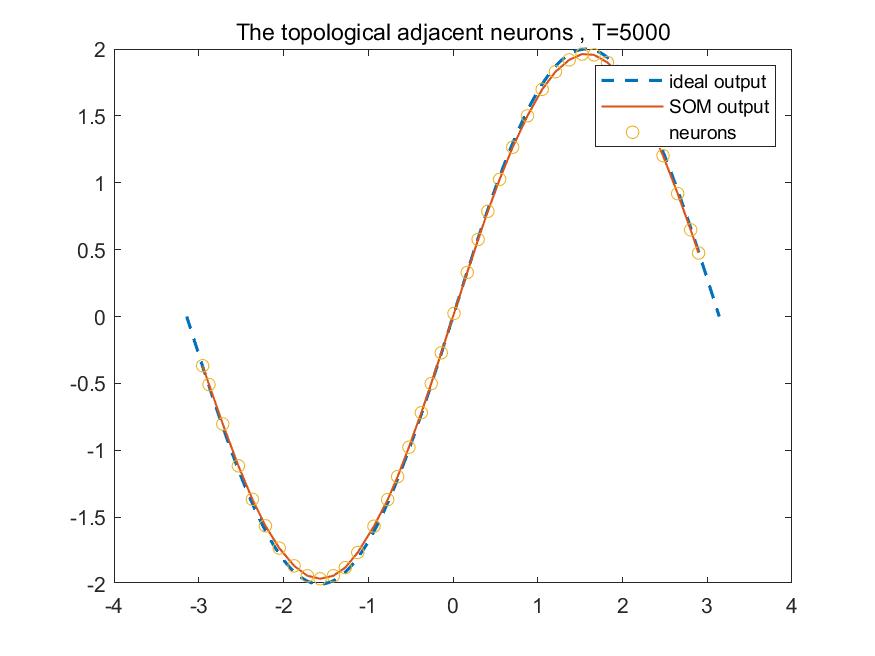
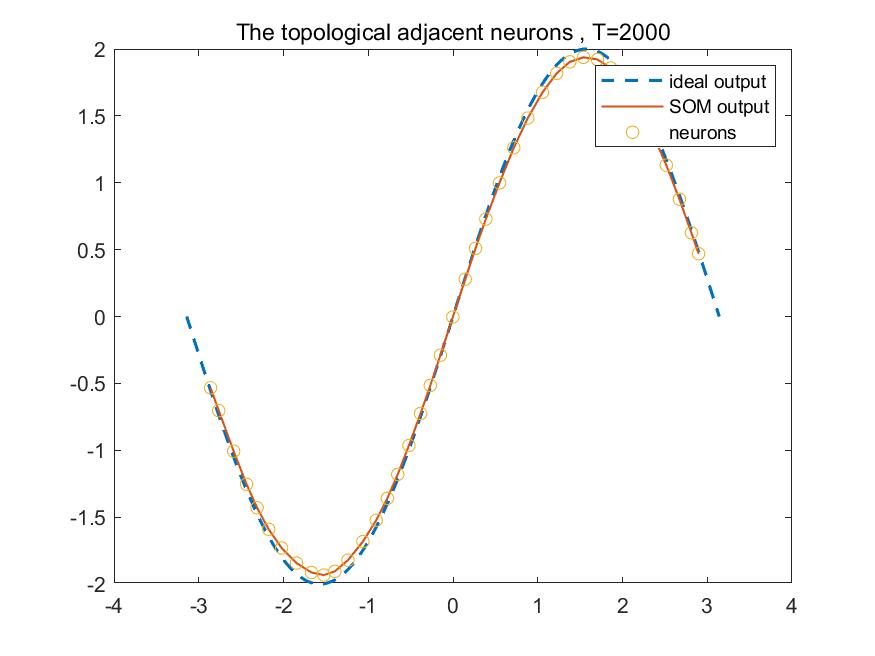
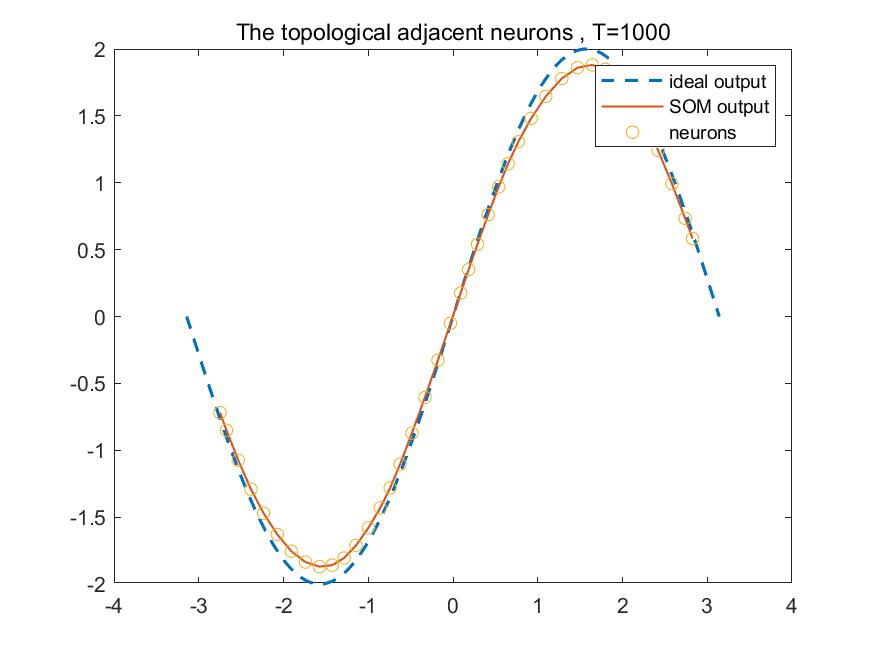
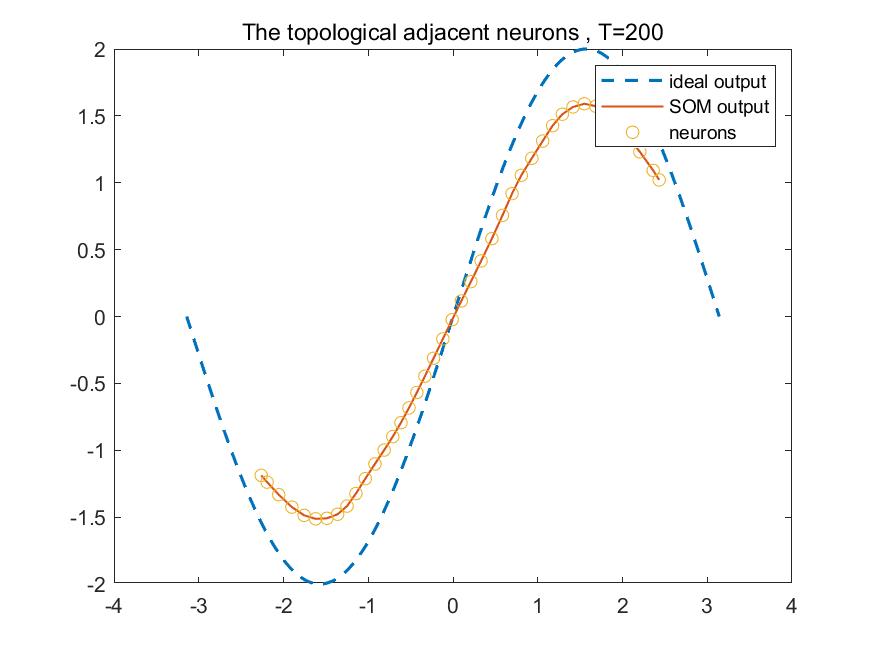
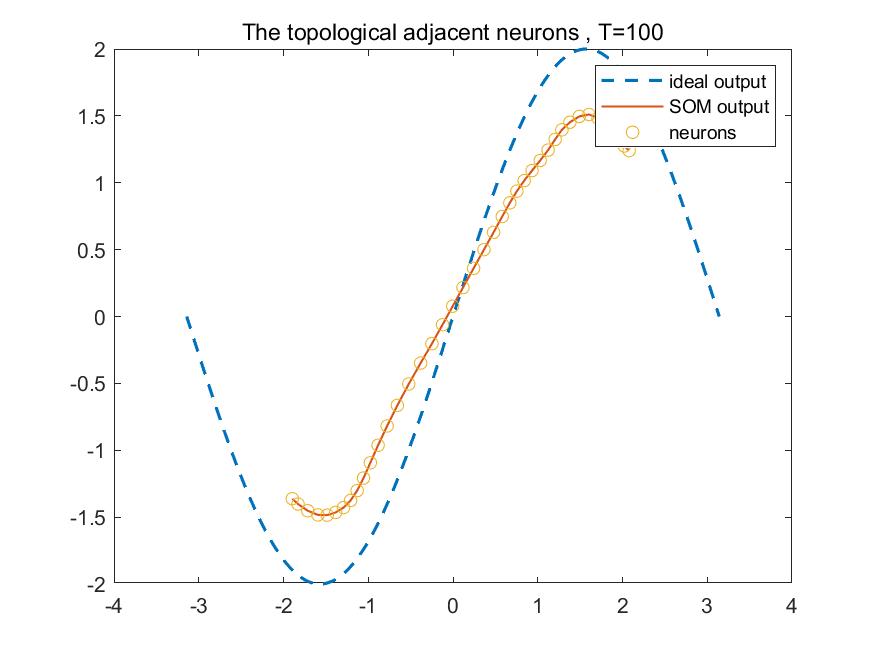
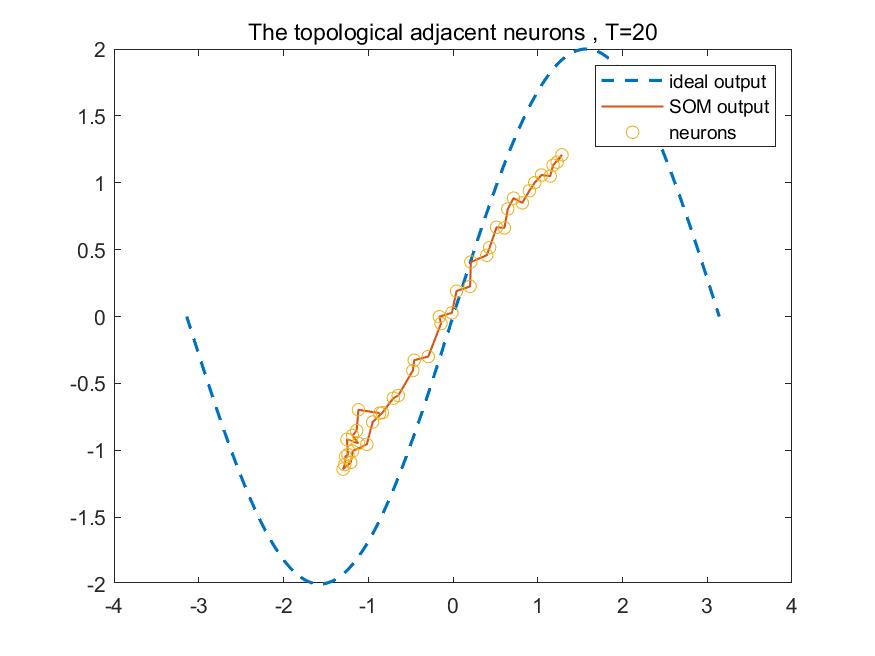
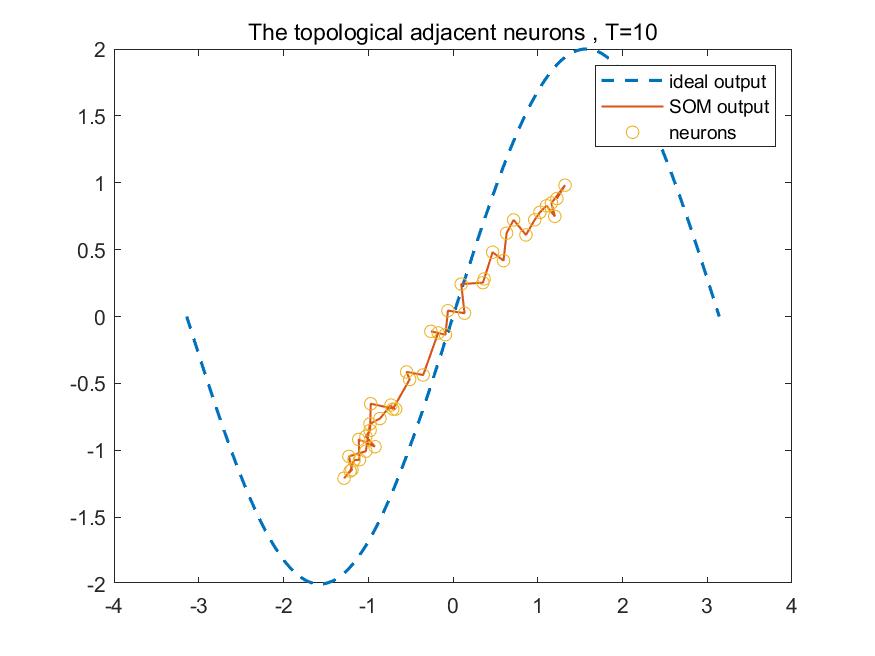
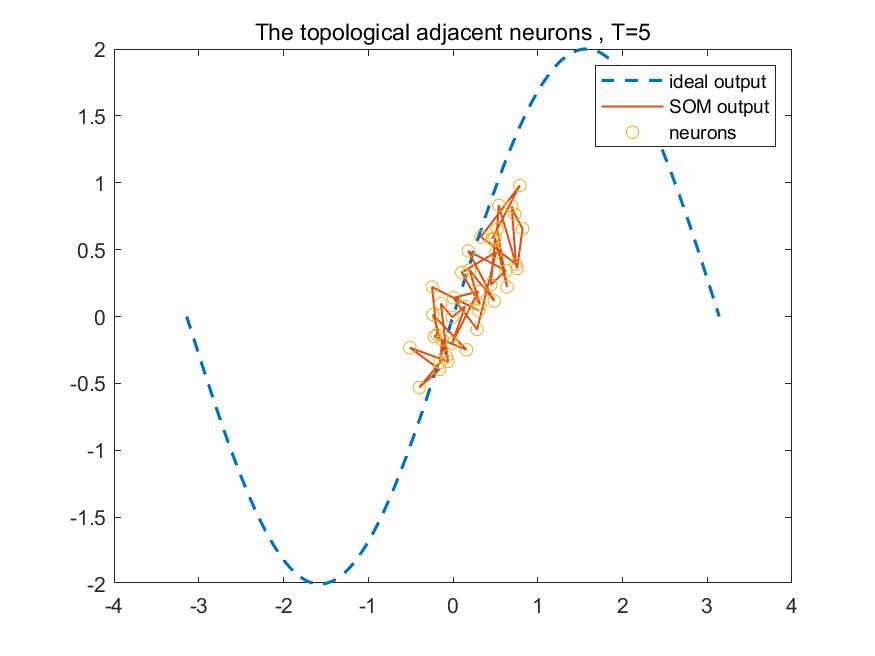
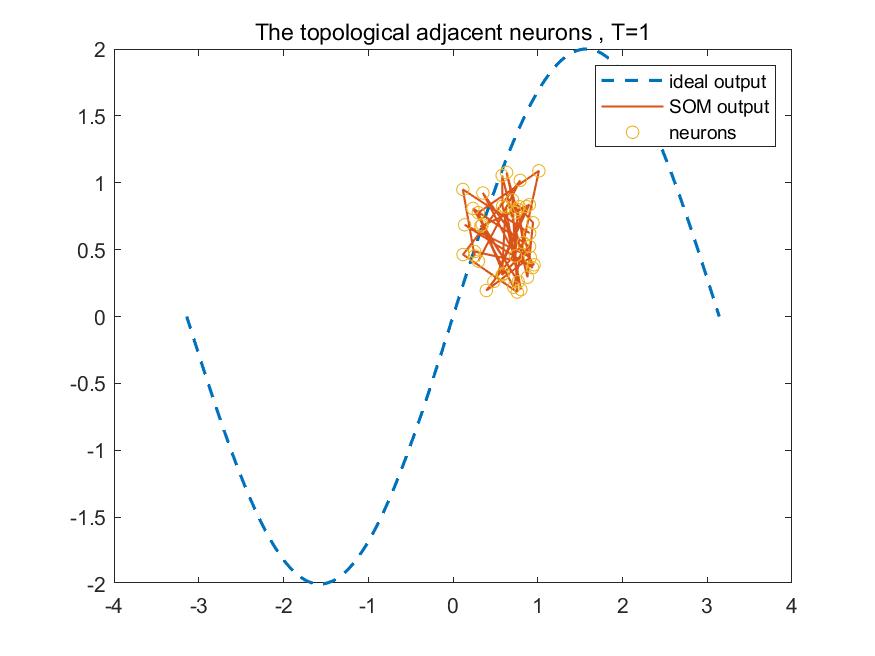
|  |
| --- |
| %% Clear all variables and close all  close all  clear  clc  num\_cen = 2;  mkdir q2\_c\_image  tic    %% Initialise equations and values  load('characters10.mat');  train\_data=im2single(train\_data);  test\_data=im2single(test\_data);  test\_data=test\_data';  train\_data=train\_data';    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

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



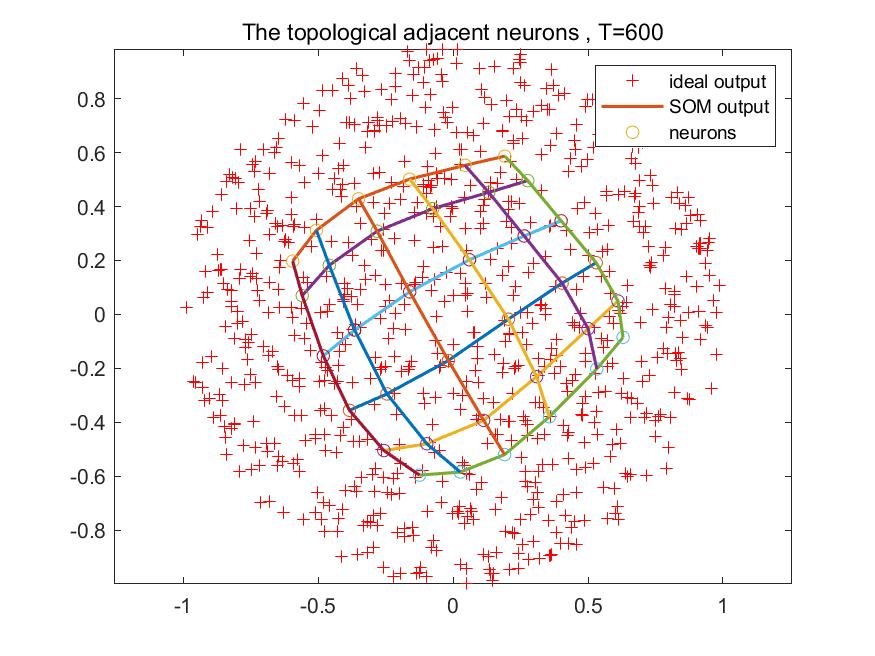
In order to compare the result of different T, I make experiments on T in this value (1,10000)



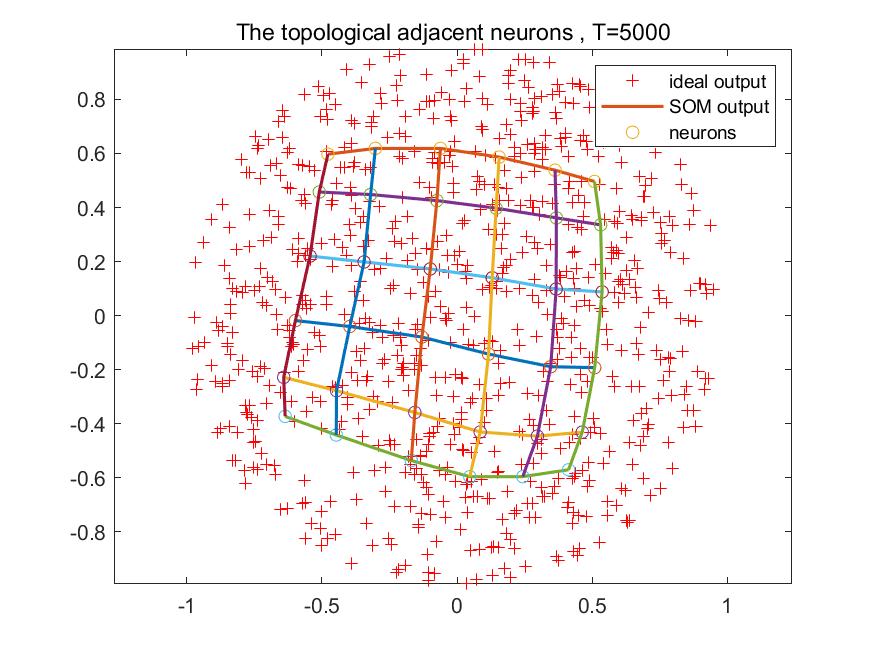
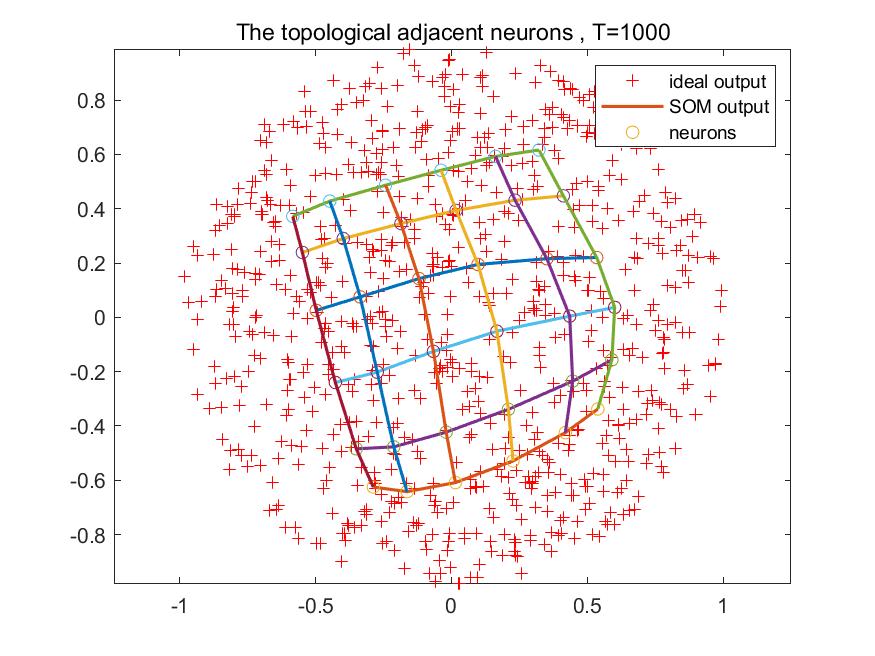
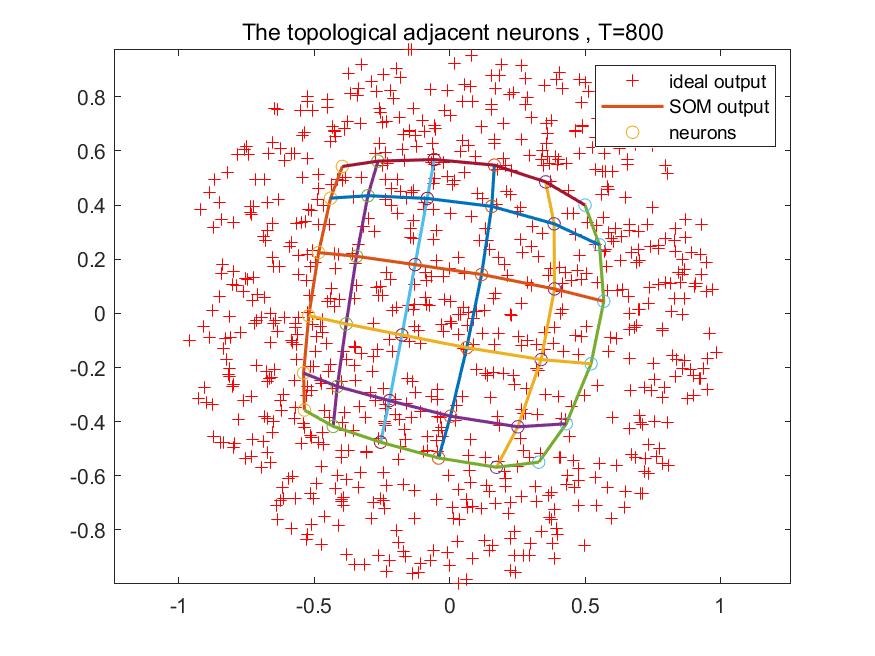
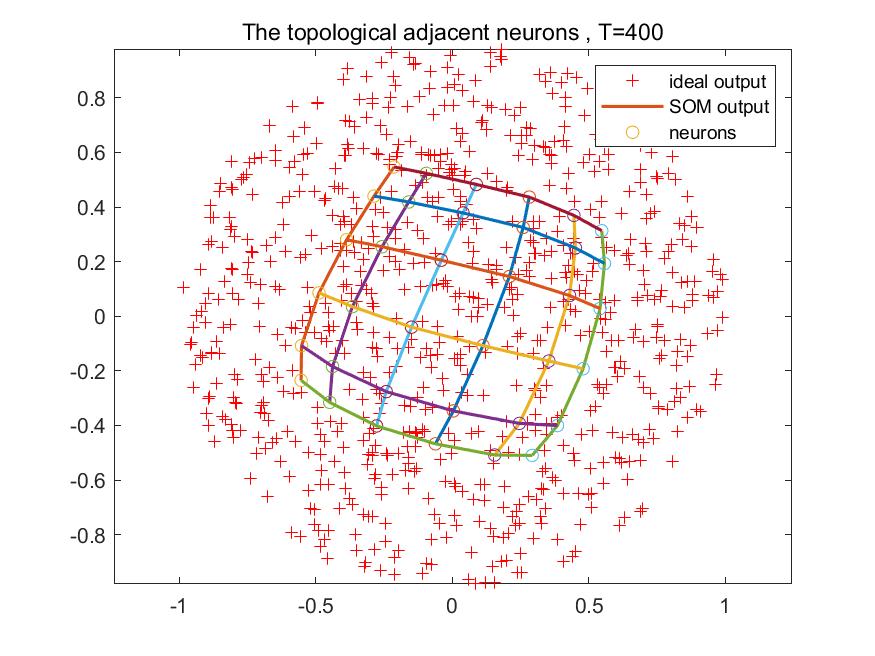
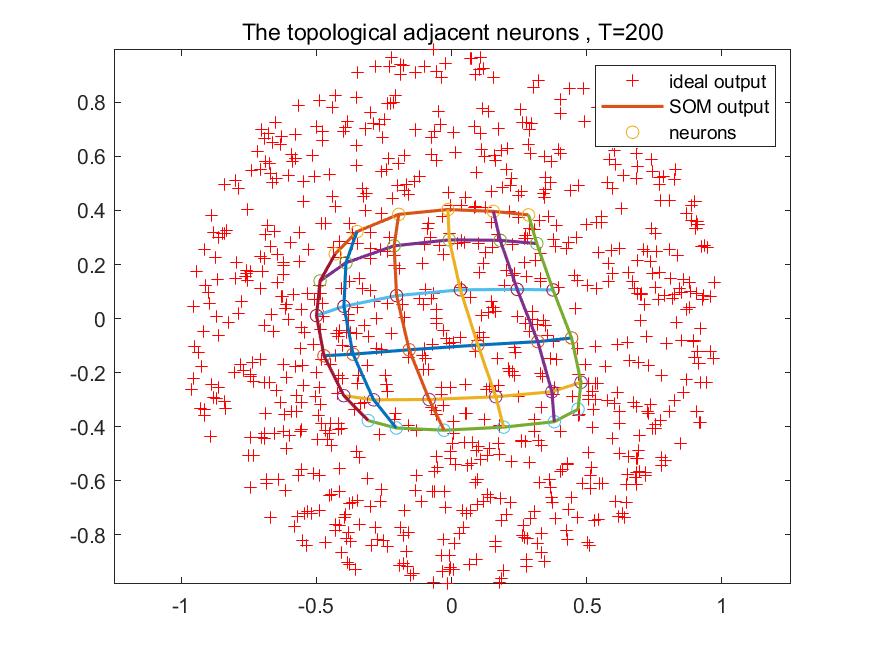
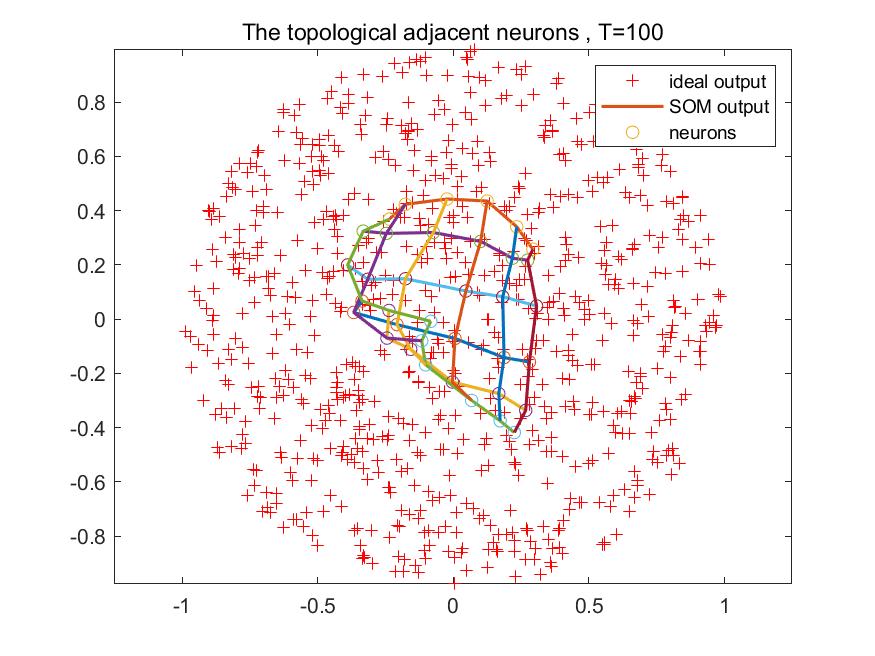
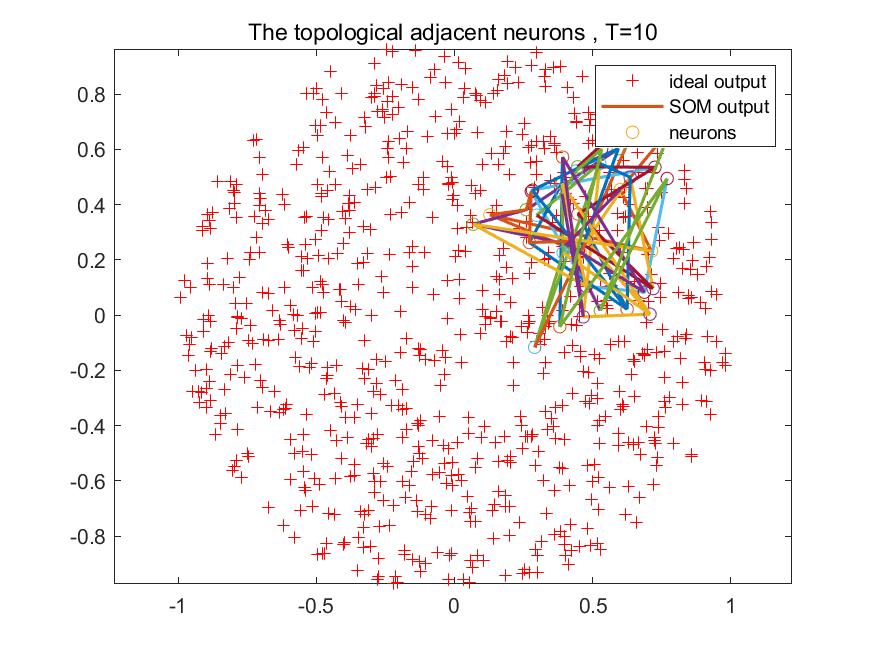
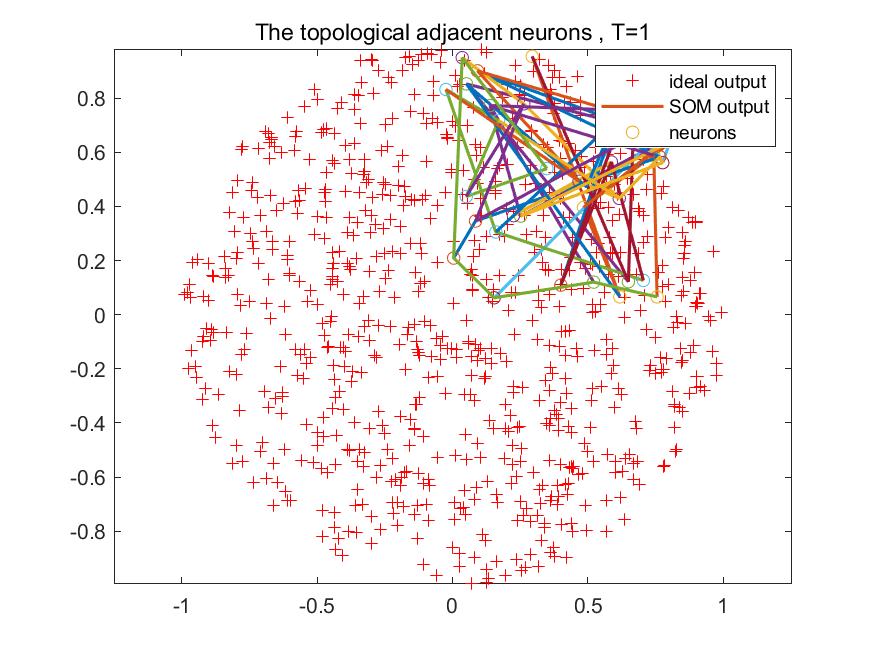
Based on this result, we can conclude that, more epoch is, the better fitting.

|  |
| --- |
| %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),".jpg"));  end |

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



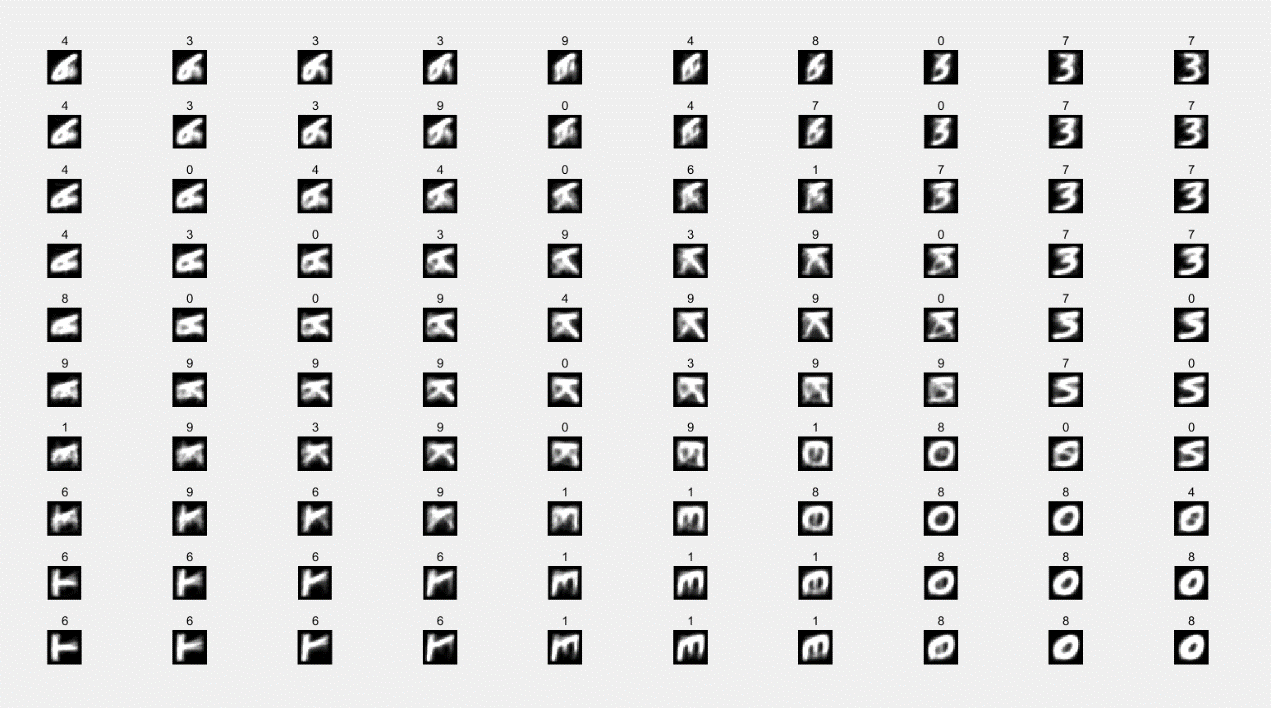
In order to compare the result of different T, I make experiments on T in this value (1,10000)



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.

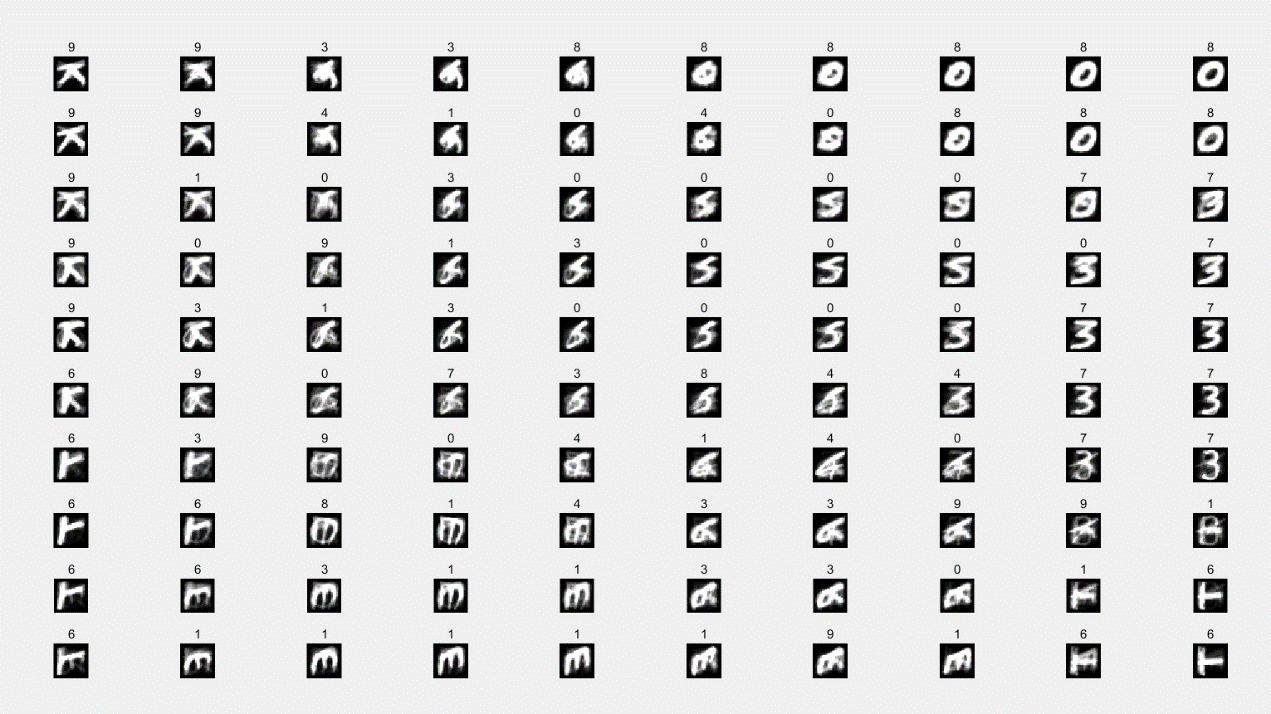
|  |
| --- |
| %init  close all;clear;clc;  mkdir q3\_b\_image    X=randn(800,2);  s2=sum(X.^2,2);  trainX=(X.\*repmat(1\*(gammainc(s2/2,1).^(1/2))./sqrt(s2),1,2))'; %2x800 matrix    %para  w=rand(2,6,6);%randomly init weigth 36 neurons in output layer  sigma0=sqrt(6^2+6^2)/2;%M=6 N=6  eta0=0.1;  T=600;%iterations  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")); |

1. 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,width)  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 |