*National University of Singapore*

*School of Electrical and Computer Engineering*

# Assignment 3

*Student:*
Wanry Lin

*Matriculation Number:*
A0000001X

*EE5904 Neural Network*

May 10, 2023

**Note: my MATLAB code is attached as the appendix in the end of the report.**

# Question 1

(a)

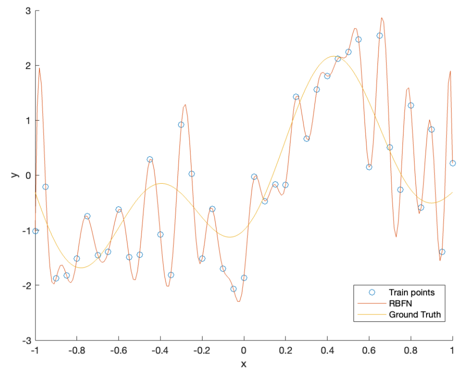



Figure 1: (a) The fitting result of RBFN by using training centers

It is obvious that the model is overfitting, since we can find it even learn the random noise of the training set.

(b)

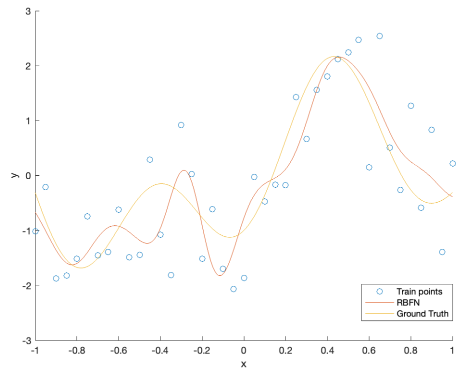


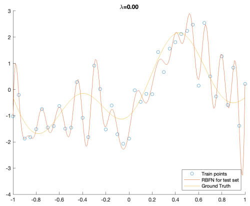

Figure 2: (b) The fitting result of RBFN by using 15 random centers

Randomly selecting centers from the data can reduce the degree of overfitting. The result of (b) is much better than that of (a). But the noise in train data still damage the overall performance.

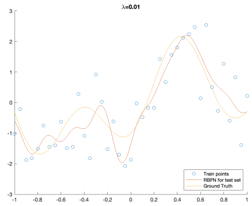
(c) $\lambda$ is set to the values [0, 0.01, 0.1, 1, 10, 100]. The mini square error is used to evaluate the performance of the regression. The result is shown in the following table and figure.

Table 1: Fitting performance with different regularization

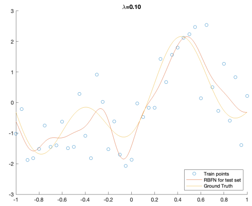
| MSE \ $\lambda$ | 0 | 0.01 | 0.1 | 1 | 10 | 100 |
|---|---|---|---|---|---|---|
| train set | 0.0389 | 0.4442 | 0.4583 | 0.4792 | 0.6491 | 1.3993 |
| test set | 0.6910 | 0.1572 | 0.1410 | 0.1136 | 0.2023 | 0.8647 |



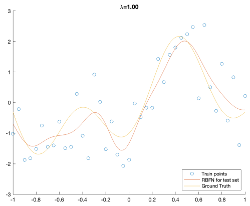
(a) $\lambda = 0$



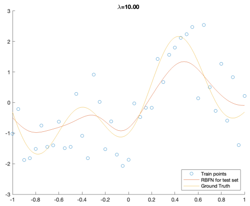
(b) $\lambda = 0.01$



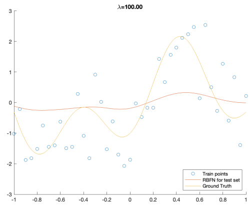
(c) $\lambda = 0.1$



(d) $\lambda = 1$



(e) $\lambda = 10$



(f) $\lambda = 100$

Figure 3: (c) The fitting result of RBFN by using different regularization

As the value of $\lambda$ grows from 0 to 100, the model turns from overfitting to proper-fitting and end with under-fitting. From Fig.3, we can find that when $1 < \lambda < 10$ the model is most likely proper fitting. As I test the MSE of changing the value of $\lambda$ from 1 to 10, the best performance appears when $\lambda = 2$ and the minimum MSE = 0.1084.

# Question 2

(a)

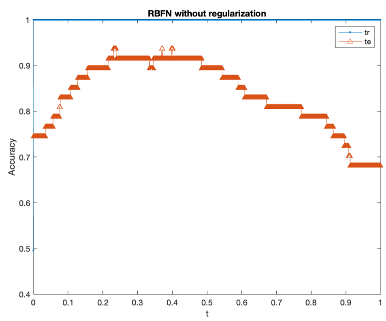
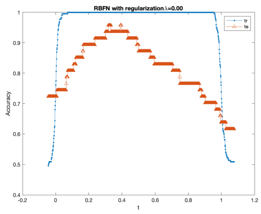
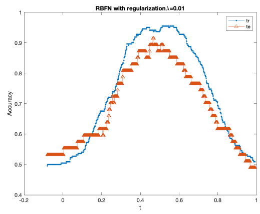


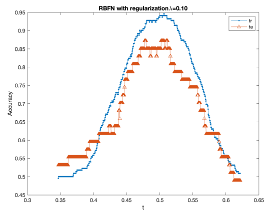

Figure 4: RBFN without regularization



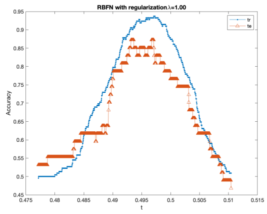
(a) $\lambda = 0$



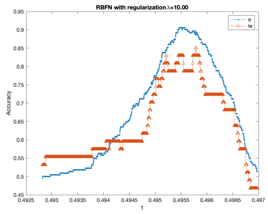
(b) $\lambda = 0.01$



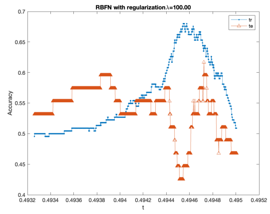
(c) $\lambda = 0.1$



(d) $\lambda = 1$



(e) $\lambda = 10$



(f) $\lambda = 100$

Figure 5: RBFN with different $\lambda$ regularization

For RBFN without regularization, it leads to overfitting and result in poor generalization by comparing Fig.4 and Fig.5. For RBFN with regularization, as the value of $\lambda$ increases, the accuracy of training set decreases but the performance of training set and test set is more and more closer. It illustrates that the smoothness reduces the performance gap between training set and test set. But if the $\lambda$ is too large, the smoothness constraint dominates and less account is taken for training data error, which damages both performance of training and test set.

(b) From Fixed Centers Selected at Random,

$$\sigma_i = \frac{d_{max}}{\sqrt{2M}} = 0.26$$

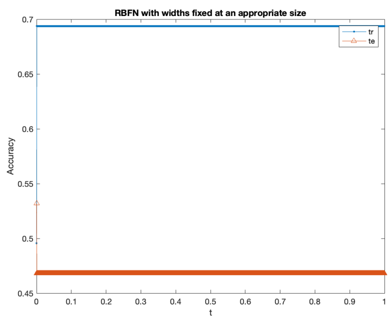
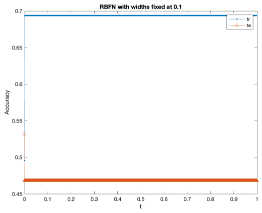
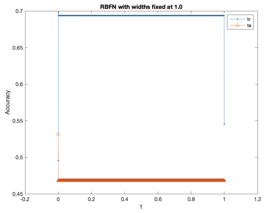



Figure 6: RBFN with widths fixed at an appropriate size

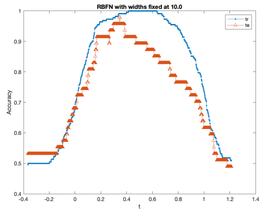
Compared with Fig.6 and Fig.4, the performance of fixed width is worse. Because the $\sigma$ is too small, which means M = 100 is redundant for this classifier. We should reduce the value of M and increase the value of $\sigma$.



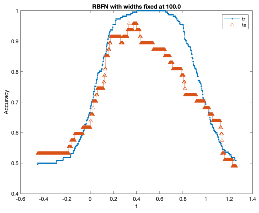

(a) $width = 0.1$



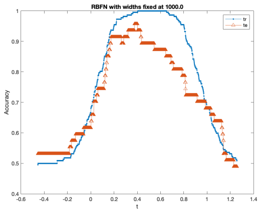

(b) $width = 1$



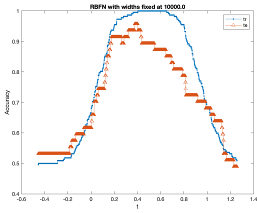

(c) $width = 10$




(d) $width = 100$




(e) $width = 1000$




(f) $width = 10000$

Figure 7: RBFN with different width

Effective width 0.1, 1.0 are too small and 100, 1000, 10000 are too large. The individual RBFs are too peaked or too flat. Therefore, the most appropriate width should be around 10 according to Fig.7.

(c)

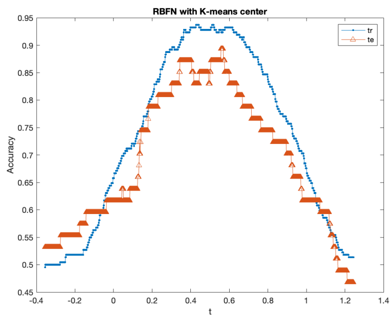



Figure 8: RBFN with 2 centers from K-means and widths = 10

The performance of using K-means is not better than random choosing centers by comparing Fig.7(c) and Fig.8.

(a) K-means center 1    (b) Training set average with label 1

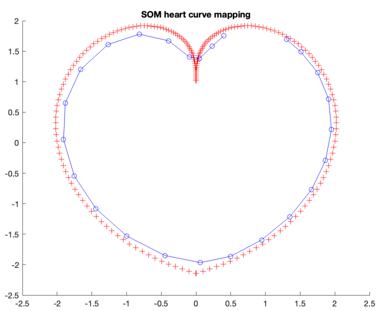
(c) K-means center 2    (d) Training set average with label 0

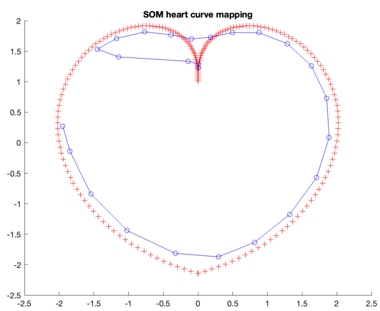
Figure 9: Visualization of centers

From Fig.9, the K-means almost perfectly generate the center of the 2 classes. The centers are almost same to the average of 2 classes. But we can find the image brightness of K-means center is not as large as average one, comparing Fig.9(a) and Fig.9(b). Therefore, when computing distance, there will be a longer distance using average instead of K-means center, which results in the a little bit worse performance when using K-means.

# Question 3

(a)



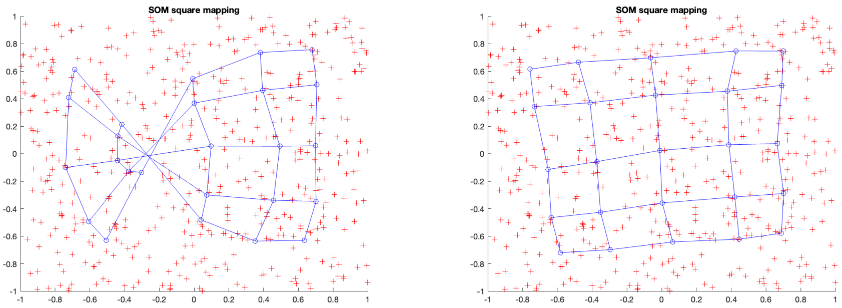




(a) most of results    (b) few of results

Figure 10: SOM for heart curve mapping

The SOM mapping result shows that it can almost mapping the shape of the curve except the northeast of the curve in Fig.10(a). However, sometimes, if the initial weight is not so perfect, the mapping curve will be like Fig.10(b).

(b)




(a) random seed: 5904 (b) random seed: 9045

Figure 11: SOM for square mapping

The SOM mapping result is shown in Fig.11. In this section, the initial weight is important too. After I change the random seed from 5904 to 9045, SOM can map the correct shape.

(c) -1

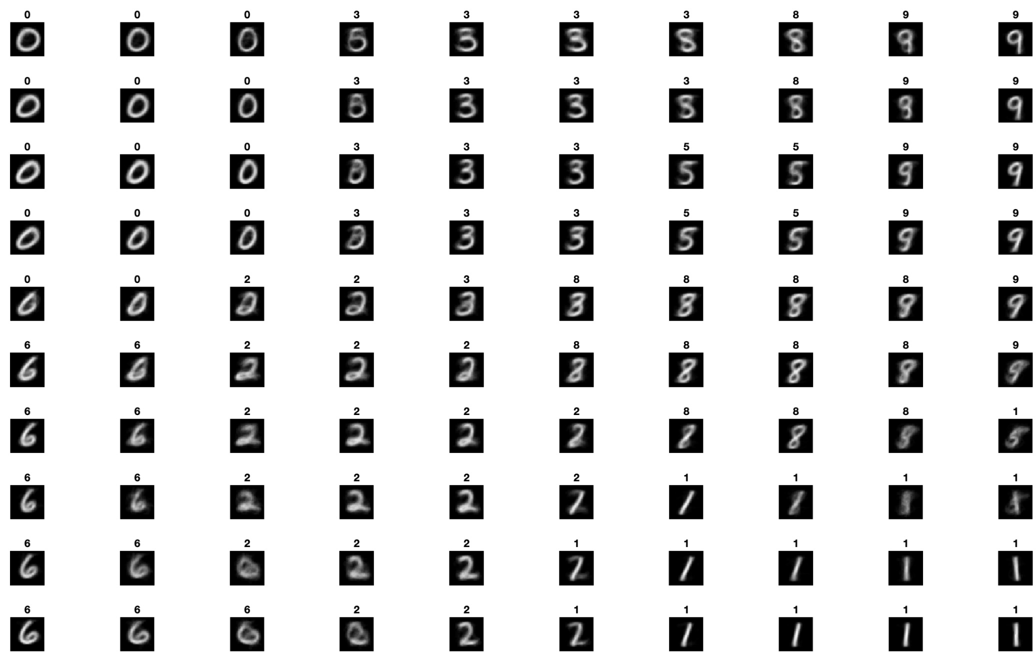
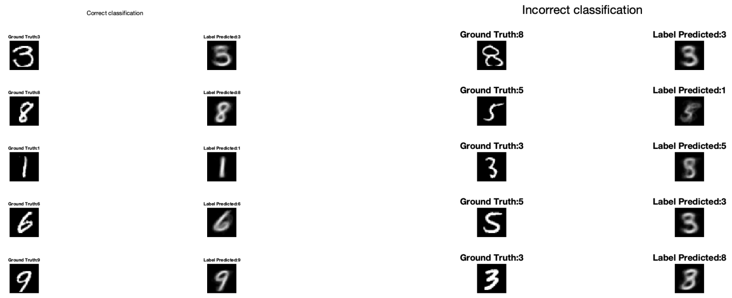


Figure 12: the semantic map of the trained SOM

Actually, it is easily to find the pattern in this semantic map. I use Nearest Neighbor to classify the neurons' label.These patterns are organized or clustered around the same category. For example, the pattern of 0 and 6, 3 and 8 are similar to each other, which locates near too. Moreover, the neurons at the joint are blur and ambiguous, which looks like the mixture of the surrounding neurons.

(d) -2




(a) some correct samples

(b) some incorrect samples

Figure 13: SOM classification visualization

The accuracy is **77.34%**. The classification mostly happens when the test data is blur and ambiguous.

# Appendix

```
% EE5904 Neural Network
% Assignment 3

%% Question 1

% (a)
clc
clear all
close all
% initialization data set
rng(5904); % random seed
train_x = -1:0.05:1;
train_y = ...
    1.2*sin(pi*train_x)-cos(2.4*pi*train_x)+randn(1,length(train_x));
test_x = -1:0.01:1;
test_y = 1.2*sin(pi*test_x)-cos(2.4*pi*test_x);
% RBF matrix
r = train_x' - train_x;
RBF = exp(-r.^2./(2*0.1^2));
w = RBF^-1*train_y';
% predict on test
r = test_x' - train_x;
RBF = exp(-r.^2./(2*0.1^2));
pred_test_y = (RBF*w)';
% plot the result
figure()
```

```
hold on
plot(train_x,train_y,'o')
plot(test_x,pred_test_y)
plot(test_x,test_y)
xlabel('x')
ylabel('y')
legend('Train points','RBFN','Ground Truth')
hold off

%(b)
% Choose centers
rand_center = datasample(train_x,15);
% Training stage
r = train_x' - rand_center;
RBF = exp(-r.^2./(2*0.1^2));
w = pinv(RBF)*train_y';
% Test stage
r = test_x' - rand_center;
RBF = exp(-r.^2./(2*0.1^2));
pred_test_y = (RBF*w)';
% plot result
figure()
hold on
plot(train_x,train_y,'o')
plot(test_x,pred_test_y)
plot(test_x,test_y)
xlabel('x')
ylabel('y')
legend('Train points','RBFN','Ground Truth')
hold off

%(c)
MSE_train = [];
MSE_test = [];
count = 1;
for lambda = [0,0.01,0.1,1,10,100]
    % Training
    r = train_x' - train_x;
    RBF = exp(-r.^2./(2*0.1^2));
    w =pinv(RBF'*RBF+lambda*eye(length(RBF)))*RBF'*train_y';
    % Train set performance
    pred_train_y = (RBF*w)';
    MSE_train = [MSE_train,sum((pred_train_y - ...
        train_y).^2)/length(pred_train_y)];
    % Test set performance
    r = test_x' - train_x;
    RBF = exp(-r.^2./(2*0.1^2));
    pred_test_y = (RBF*w)';
    MSE_test = [MSE_test,sum((pred_test_y - ...
        test_y).^2)/length(pred_test_y)];
    % Plot
    fig = figure();
    hold on
    plot(train_x,train_y,'o')
    %     plot(train_x,pred_train_y)
    plot(test_x,pred_test_y)
    plot(test_x,test_y)
```

```
    legend('Train points','RBFN for test set','Ground ...
        Truth','Location','southeast')
    title(join(['\lambda=',sprintf('%.2f',lambda)]))
    hold off
    saveas(fig,sprintf('q1_c%d.png',count))
    count = count + 1;
end

% best lambda
for lambda = 1:10
    % Training
    r = train_x' - train_x;
    RBF = exp(-r.^2./(2*0.1^2));
    w =pinv(RBF'*RBF+lambda*eye(length(RBF)))*RBF'*train_y';
    % Train set performance
    pred_train_y = (RBF*w)';
    MSE_train = [MSE_train,sum((pred_train_y - ...
        train_y).^2)/length(pred_train_y)];
    % Test set performance
    r = test_x' - train_x;
    RBF = exp(-r.^2./(2*0.1^2));
    pred_test_y = (RBF*w)';
    MSE_test = [MSE_test,sum((pred_test_y - ...
        test_y).^2)/length(pred_test_y)];
    % Plot
    fig = figure();
    hold on
    plot(train_x,train_y,'o')
    %     plot(train_x,pred_train_y)
    plot(test_x,pred_test_y)
    plot(test_x,test_y)
    legend('Train points','RBFN for test set','Ground ...
        Truth','Location','southeast')
    title(join(['\lambda=',sprintf('%d',lambda)]))
    hold off
    saveas(fig,sprintf('q1_c%d.png',count))
    count = count + 1;
end
[min_MSE,min_idx] = min(MSE_test);

%% Question 2
% 7 4 classes

% (a)
clc
clear all
close all
% read the handwritten data
load('MNIST_database.mat')
column_no = 1;
tmp=reshape(train_data(:,column_no),28,28);
imshow(double(tmp));
close all
% find the location of classes 7,4
% train data
trainIdx = find(train_classlabel==7 | train_classlabel==4);
TrLabel = train_classlabel(trainIdx);
TrLabel(TrLabel==7) = 1;
```

```
TrLabel(TrLabel==4) = 0;
Train_Data = train_data(:,trainIdx);
% test data
testIdx = find(test_classlabel==7 | test_classlabel==4);
TeLabel = test_classlabel(testIdx);
TeLabel(TeLabel==7) = 1;
TeLabel(TeLabel==4) = 0;
Test_Data = test_data(:,testIdx);
deviation = 100;

% determine the weights of RBFN without regularization
% Traning
distances = pdist(Train_Data'); % compute pairwise Euclidean ...
    distances
r = squareform(distances); % convert the pairwise distances ...
    into a distance matrix
RBF = exp(-r.^2./(2*deviation^2));
w =inv(RBF)*TrLabel';
TrPred = (RBF*w)';
% Prediction
r = dist(Test_Data',Train_Data);
RBF = exp(-r.^2./(2*deviation^2));
TePred = (RBF*w)';
% Plot
figure();
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
plot(thr,TrAcc,'.- ',thr,TeAcc,'^-');legend('tr','te');
title('RBFN without regularization')
xlabel('t');
ylabel('Accuracy')

count = 2;
for lambda = [0,0.01,0.1,1,10,100]
    % determine the weights of RBFN without regularization
    % Traning
    distances = pdist(Train_Data');
    r = squareform(distances);
    RBF = exp(-r.^2./(2*deviation^2));
    w =pinv(RBF'*RBF+lambda*eye(size(RBF,2)))*RBF'*TrLabel';
    TrPred = (RBF*w)';
    % Prediction
    r = dist(Test_Data',Train_Data);
    RBF = exp(-r.^2./(2*deviation^2));
    TePred = (RBF*w)';
    % Plot
    fig = figure();
```

```
    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
    plot(thr,TrAcc,'.- ',thr,TeAcc,'^-');legend('tr','te');
    title(join(['RBFN with ...
        regularization',sprintf('\\lambda=%.2f',lambda)]))
    xlabel('t');
    ylabel('Accuracy')
    %     saveas(fig,sprintf('q2_a%d.png',count))
    count = count + 1;
end

% (b)
clc
close all
% Choose centers
rng(5904);
rand_center = datasample(Train_Data,100,2);
% Traning
r = dist(Train_Data',rand_center);
deviation = sqrt(max(r,[],'all'))/sqrt(2*size(rand_center,2));
RBF = exp(-r.^2./(2*deviation^2));
w =pinv(RBF)*TrLabel';
TrPred = (RBF*w)';
% Prediction
r = dist(Test_Data',rand_center);
RBF = exp(-r.^2./(2*deviation^2));
TePred = (RBF*w)';
% Plot
figure();
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
plot(thr,TrAcc,'.- ',thr,TeAcc,'^-');legend('tr','te');
title('RBFN with widths fixed at an appropriate size')
xlabel('t');
ylabel('Accuracy')

```

```
count = 2;
for deviation = [0.1,1,10,100,1000,10000]
    % Traning
    r = dist(Train_Data',rand_center);
    RBF = exp(-r.^2./(2*deviation^2));
    w =pinv(RBF)*TrLabel';
    TrPred = (RBF*w)';
    % Prediction
    r = dist(Test_Data',rand_center);
    RBF = exp(-r.^2./(2*deviation^2));
    TePred = (RBF*w)';
    % Plot
    fig = figure();
    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
    plot(thr,TrAcc,'.- ',thr,TeAcc,'^-');legend('tr','te');
    title(join(['RBFN with widths fixed at ...
       ',sprintf('%.1f',deviation)]));
    xlabel('t');
    ylabel('Accuracy')
%     saveas(fig,sprintf('q2_b%d.png',count))
    count = count + 1;
end


% (c)
clc
close all
% Choose centers using K means
k = 2;
[¬, kcenters] = kmeans(Train_Data', k);
kcenters = kcenters';
% Traning
r = dist(Train_Data',kcenters);
deviation = 10; % from q2_b
RBF = exp(-r.^2./(2*deviation^2));
w =pinv(RBF)*TrLabel';
TrPred = (RBF*w)';
% Prediction
r = dist(Test_Data',kcenters);
RBF = exp(-r.^2./(2*deviation^2));
TePred = (RBF*w)';
% Plot
figure();
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
plot(thr,TrAcc,'.- ',thr,TeAcc,'^-');legend('tr','te');
title('RBFN with K-means center')
xlabel('t');
ylabel('Accuracy')
% visualize center
figure();
title('K-means Centers 1')
imshow(reshape(kcenters(:,1),[28,28]));
figure();
title('K-means Centers 2')
imshow(reshape(kcenters(:,2),[28,28]));
% visualize training set average
figure();
title('Training set average 1')
imshow(reshape(mean(Train_Data(:,TrLabel==1),2),[28,28]));
figure();
title('Training set average 2')
imshow(reshape(mean(Train_Data(:,TrLabel==0),2),[28,28]));

%% Question 3

% (a)
clc
clear all
close all
rng(5904);
% Train data
t = linspace(-pi,pi,200);
trainX = [t.*sin(pi*sin(t)./t); 1-abs(t).*cos(pi*sin(t)./t)]; ...
    % 2x200 matrix, column-wise points
train_fig = figure();
plot(trainX(1,:),trainX(2,:),'+r');
close(train_fig);

% SOM
% initialization network
M = 1;
N = 25;
neurons = rand(2,N);
sigma0 = sqrt(M^2+N^2)/2;
iteration = 500;
eta0 = 0.1;
tau = iteration/log(sigma0); % I guess log in HW3 is ln
d0 = 1:N;
for epoch = 1:iteration
    etan = eta0*exp(-epoch/iteration);
    sigma = sigma0*exp(-epoch/tau);
    for i = 1:size(trainX,2)
```

```
%          distance = sum(dist(trainX(:,i),neurons),1);
        distance = sum((trainX(:,i) - neurons).^2,1);
        [¬,winner] = min(distance,[],2);
        d = abs(d0-winner);
        h = exp(-d.^2/(2*sigma^2));
        % Update
        neurons = neurons + etan*h.*(trainX(:,i) - neurons);
    end
end

% plot the SOM result
figure();
hold on
plot(trainX(1,:),trainX(2,:),'+r');
plot(neurons(1,:),neurons(2,:),'o-b');
title("SOM heart curve mapping")
hold off

%(b)
clc
close all
trainX = rands(2,500); % 2x500 matrix, column-wise points
% SOM
% initialization network
rng(9045);
M = 5;
N = 5;
neurons = rand(2,M,N);
sigma0 = sqrt(M^2+N^2)/2;
iteration = 500;
eta0 = 0.1;
tau = iteration/log(sigma0); % I guess log in HW3 is ln
d0 = 1:N;
for epoch = 1:iteration
    etan = eta0*exp(-epoch/iteration);
    sigma = sigma0*exp(-epoch/tau);
    for i = 1:size(trainX,2)
        distance = squeeze(sum((trainX(:,i) - neurons).^2,1))';
        [¬,winner] = min(distance,[],'all','linear');
        k = ceil(winner/5);
        n = winner - (k-1)*5;
        d_j = (d0 - n).^2;
        d_i = (d0 - k).^2;
        d_square = d_j' + d_i;
        h = exp(-d_square.^2/(2*sigma^2));
        h = permute(repmat(h,[1,1,2]),[3 2 1]);
        % Update
        neurons = neurons + etan*h.*(trainX(:,i) - neurons);
    end
end

% plot the SOM result
figure();
hold on
plot(trainX(1,:),trainX(2,:),'+r');
for i = 1:5
    for j = 1:5
        % left and right neighbors
```

```
        if i+1 ≤ 5
            plot([neurons(1,i,j),neurons(1,i+1,j)],[neurons(2,i,j),neurons(2,i+1,j
        end
        % top and bottom neighbors
        if j+1 ≤ 5
            plot([neurons(1,i,j),neurons(1,i,j+1)],[neurons(2,i,j),neurons(2,i,j+1
        end
    end
end
title("SOM square mapping")
hold off

%% Question 3 c
%  7 4 classes

%(c)-1
clc
clear all
close all
rng(5904);
% read the handwritten data
load('MNIST_database.mat')
% find the location of classes 7,4
% train data
trainIdx = find(train_classlabel≠7 & train_classlabel≠4);
TrLabel = train_classlabel(trainIdx);
Train_Data = train_data(:,trainIdx);
% test data
testIdx = find(test_classlabel≠7 & test_classlabel≠4);
TeLabel = test_classlabel(testIdx);
Test_Data = test_data(:,testIdx);
% input data
Data = cat(2,Train_Data,Test_Data);
Label = cat(2,TrLabel,TeLabel);

% SOM
% initialization network
M = 10;
N = 10;
neurons = rand(size(Data,1),M,N);
sigma0 = sqrt(M^2+N^2)/2;
iteration = 1000;
eta0 = 0.1;
tau = iteration/log(sigma0); % I guess log in HW3 is ln
d0 = 1:N;
for epoch = 1:iteration
    etan = eta0*exp(-epoch/iteration);
    sigma = sigma0*exp(-epoch/tau);
    for i = 1:size(Data,2)
        distance = squeeze(sum((Data(:,i) - neurons).^2,1))';
        [¬,winner] = min(distance,[],'all','linear');
        k = ceil(winner/N);
        n = winner - (k-1)*N;
        d_j = (d0 - n).^2;
        d_i = (d0 - k).^2;
        d_square = d_j' + d_i;
        h = exp(-d_square.^2/(2*sigma^2));
        h = permute(repmat(h,[1,1,size(Data,1)]),[3 2 1]);
```

```
        % Update
        neurons = neurons + etan*h.*(Data(:,i) - neurons);
    end
end

% map label using nearest neighbour
reshaped_neurons = reshape(neurons,[size(Data,1),1,100]);
true_label = [0,1,2,3,5,6,8,9];
for i = 1:8
    label = true_label(i);
    truth(i,:) = mean(Train_Data(:,TrLabel==label),2);
end
for i = 1:100
    [idx, ¬] = knnsearch(truth,reshaped_neurons(:,1,i)', 'K', 1);
    neuron_label(i) = true_label(idx);
end

% plot SOM
fig = figure;
fig.WindowState = 'maximized' ;
title('semantic map')
for A = 1:100
    subplot(10,10,A)
    graph = reshape(reshaped_neurons(:,1,A),[28,28]);
    imshow(graph);
    title(sprintf('%d',neuron_label(A)))
end

%%
%(c)-1
clc
clear all
close all
rng(5904);
% read the handwritten data
load('MNIST_database.mat')
% find the location of classes 7,4
% train data
trainIdx = find(train_classlabel≠7 & train_classlabel≠4);
TrLabel = train_classlabel(trainIdx);
Train_Data = train_data(:,trainIdx);
% test data
testIdx = find(test_classlabel≠7 & test_classlabel≠4);
TeLabel = test_classlabel(testIdx);
Test_Data = test_data(:,testIdx);

% SOM
% initialization network
M = 10;
N = 10;
neurons = rand(size(Train_Data,1),M,N);
sigma0 = sqrt(M^2+N^2)/2;
iteration = 1000;
eta0 = 0.1;
tau = iteration/log(sigma0); % I guess log in HW3 is ln
d0 = 1:N;
for epoch = 1:iteration
    etan = eta0*exp(-epoch/iteration);
```

```
    sigma = sigma0*exp(-epoch/tau);
    for i = 1:size(Train_Data,2)
        distance = squeeze(sum((Train_Data(:,i) - ...
            neurons).^2,1))';
        [¬,winner] = min(distance,[],'all','linear');
        k = ceil(winner/N);
        n = winner - (k-1)*N;
        d_j = (d0 - n).^2;
        d_i = (d0 - k).^2;
        d_square = d_j' + d_i;
        h = exp(-d_square.^2/(2*sigma^2));
        h = permute(repmat(h,[1,1,size(Train_Data,1)]),[3 2 1]);
        % Update
        neurons = neurons + etan*h.*(Train_Data(:,i) - neurons);
    end
end

% map label using nearest neighbour
reshaped_neurons = reshape(neurons,[size(Train_Data,1),1,100]);
true_label = [0,1,2,3,5,6,8,9];
for i = 1:8
    label = true_label(i);
    truth(i,:) = mean(Train_Data(:,TrLabel==label),2);
end
for i = 1:100
    [idx, ¬] = knnsearch(truth,reshaped_neurons(:,1,i)', 'K', 1);
    neuron_label(i) = true_label(idx);
end

% Test
% TePred
TePred = zeros(size(TeLabel));
counter_1 = 1;
counter_2 = 1;
for i = 1:size(Test_Data,2)
    distance = squeeze(sum((Test_Data(:,i) - neurons).^2,1));
    [¬,winner] = min(distance,[],'all','linear');
    TePred(i) = neuron_label(winner);
    % plot some correct samples
    if TePred(i)==TeLabel(i) && counter_1 ≤ 5
        figure(1)
        sgtitle('Correct classification')
        subplot(5,2,(counter_1-1)*2+1)
        imshow(reshape(Test_Data(:,i),28,28))
        title(sprintf('Ground Truth:%d',TeLabel(1,i)))
        subplot(5,2,(counter_1-1)*2+2)
        imshow(reshape(reshaped_neurons(:,winner),28,28))
        title(sprintf('Label Predicted:%d',TePred(1,i)))
        counter_1 = counter_1 + 1;
    % plot some incorrect samples
    elseif TePred(1,i)≠TeLabel(1,i) && counter_2 ≤ 5
        figure(2)
        sgtitle('Incorrect classification')
        subplot(5,2,(counter_2-1)*2+1)
        imshow(reshape(Test_Data(:,i),28,28))
        title(sprintf('Ground Truth:%d',TeLabel(1,i)))
        subplot(5,2,(counter_2-1)*2+2)
        imshow(reshape(reshaped_neurons(:,winner),28,28))
```

```
        title(sprintf('Label Predicted:%d',TePred(1,i)))
        counter_2 = counter_2 + 1;
    end
end
TeAccr = sum(TePred == TeLabel)/size(Test_Data,2);
```