EE5904 Neural Networks: HW #3

A0206597U Xin Zhengfang

Q1. Function Approximation with RBFN

Q1. (a)

```
clc
clear
% construct data
rng(520) % reproduce seed
train_x = -1:0.05:1;
train_y = 1.2*sin(pi*train_x)-cos(2.4*pi*train_x)+0.3*randn(1,size(train_x,2));
test_x = -1:0.01:1;
test_y = 1.2*sin(pi*test_x)-cos(2.4*pi*test_x);
% RBF matrix
r = abs(train_x' - train_x);
RBF = exp(-r.^2./2/0.1^2);
w = RBF^-1*train_y';
% predict on test
r = abs(test_x' - train_x);
RBF = exp(-r.^2./2/0.1^2);
pred_y = (RBF*w)';
% mean square error on test
MSE_test = sum((pred_y - test_y).^2)/size(pred_y,2);
% predict on train
r = abs(train_x' - train_x);
RBF = exp(-r.^2./2/0.1^2);
pred_y = (RBF*w)';
% mean square error on train
MSE_train = sum((pred_y - train_y).^2)/size(pred_y,2);
```

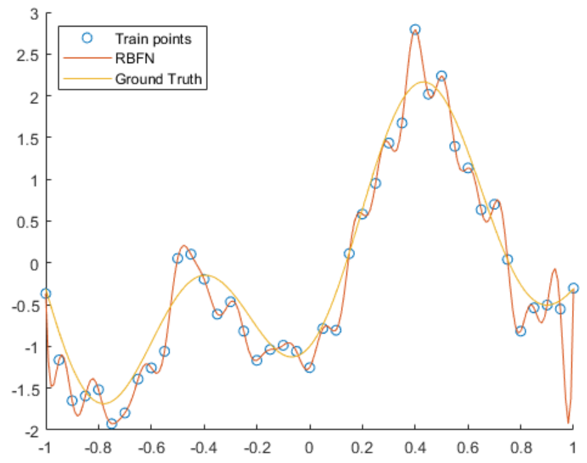

Figure Q1. (a) The fitting result of RBFN by using training centers

The mean square error on the train set is **5.52e-16**, and the test set is **0.119**. Therefore, our model is **overfitting**. **Noise** in the train set is another reason for poor fitting results on the test set.

Q1. (b).

```
clc
clear
% construct data
rng(520) % reproduce seed
train_x = -1:0.05:1;
train_y = 1.2*sin(pi*train_x)-cos(2.4*pi*train_x)+0.3*randn(1,size(train_x,2));
test_x = -1:0.01:1;
test_y = 1.2*sin(pi*test_x)-cos(2.4*pi*test_x);
% Choose centers
rand_cens = datasample(train_x,15,2);
% Training
r = abs(train_x' - rand_cens);
RBF = exp(-r.^2./2/0.1^2);
w = pinv(RBF)*train_y';
% Train accuracy
r = abs(train_x' - rand_cens);
RBF = exp(-r.^2./2/0.1^2);
pred_y = (RBF*w)';
MSE_train = sum((pred_y - train_y).^2)/size(pred_y,2);
% Test accuracy
r = abs(test_x' - rand_cens);
RBF = exp(-r.^2./2/0.1^2);
pred_y = (RBF*w)';
MSE_test = sum((pred_y - test_y).^2)/size(pred_y,2);
```

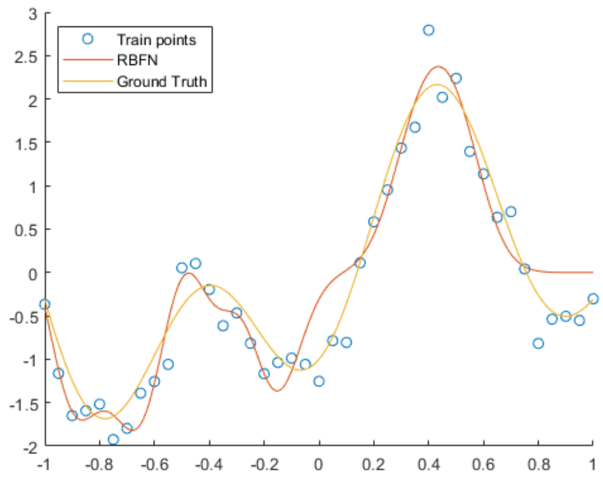

Figure Q1. (b) The fitting result of RBFN by using random centers

The mean square error was **0.1305** on the train and **0.0954** on the test. Therefore, randomly selecting centers from the data can **reduce degree of overfitting**. This result is a little **better** than the method in Q1. (a). But the **noise** of train data still degrades our performance.

Q1. (c)

MATLAB code:

```
clc
clear
% construct data
rng(520) % reproduce seed
train_x = -1:0.05:1;
train_y = 1.2*sin(pi*train_x)-cos(2.4*pi*train_x)+0.3*randn(1,size(train_x,2));
test_x = -1:0.01:1;
test_y = 1.2*sin(pi*test_x)-cos(2.4*pi*test_x);
MSE_train = [];
MSE_test = [];
for i = [0,0.1,1,10,100]
    % Training
    lambda = i;
    r = abs(train_x' - train_x);
    RBF = exp(-r.^2./2/0.1^2);
    w =pinv(RBF'*RBF+lambda*eye(size(RBF,2)))*RBF'*train_y';
    % Train
    r = abs(train_x' - train_x);
    RBF = exp(-r.^2./2/0.1^2);
    pred_y = (RBF*w)';
    MSE_train = [MSE_train;sum((pred_y - train_y).^2)/size(pred_y,2)];
    % Test
    r = abs(test_x' - train_x);
    RBF = exp(-r.^2./2/0.1^2);
    pred_y = (RBF*w)';
    MSE_test = [MSE_test;sum((pred_y - test_y).^2)/size(pred_y,2)];
    % Plot
    fig = figure();
    hold on
    plot(train_x,train_y,'o')
    plot(test_x,pred_y)
    plot(test_x,test_y)
    legend('Train points','RBFN','Ground Truth','Location','northwest')
    title(join(['\lambda=',sprintf('%.1f',i)]))
    hold off
    saveas(fig,sprintf('lambda_%.1f.png',i))
end
```

Table. Q1. (c) Fitting performance with different regularization

	$\lambda = 0.0$	$\lambda = 0.1$	$\lambda = 1.0$	$\lambda = 10.0$	$\lambda = 100.0$
MSE on the train set	0.018	0.0383	0.0512	0.2240	1.0144
MSE on the test set	0.0879	0.0289	0.0165	0.1299	0.8512

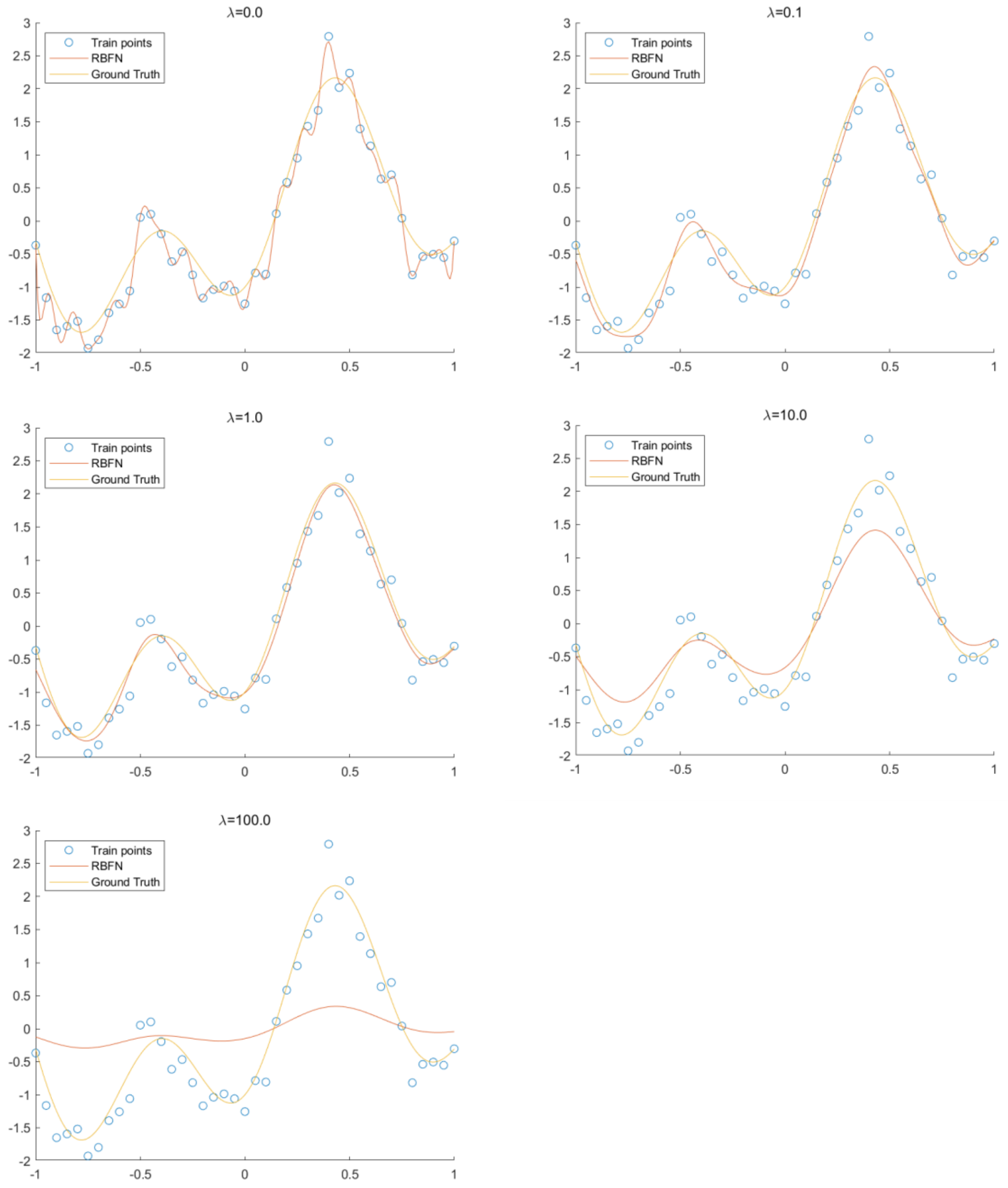

Figure Q1. (c). The fitting result of RBFN by using different regularization

From Figure Q1. (c) As the degree of regularization increases, **the slope becomes smaller** and **overfitting disappears as $\boldsymbol{\lambda}$ increases**. However, if the regularization is **too strong, the model will become under-fitting**, which will seriously affect performance in both training and testing. The Table Q1. (c) and Figure Q1. (c) show **the $\boldsymbol{\lambda = 1.0}$ is best regularization coefficient among $\boldsymbol{0.0, 0.1, 1.0, 10}$ $\boldsymbol{and}$ $\boldsymbol{100}$.**

Q2. Handwritten Digits Classification using RBFN

Q2. (a).

MATLAB code:

```
clc
clear
% load MINST
load('../MNIST_database')
%   A0206597U->9=0 7=1
%   Train data
trainIdx = find(train_classlabel==9 | train_classlabel==7); % find the location of
classes
TrLabel = train_classlabel(trainIdx);
TrLabel(TrLabel==9) = 1;
TrLabel(TrLabel==7) = 0;
Train_Data = train_data(:,trainIdx);
%   Test data
testIdx = find( test_classlabel==9 | test_classlabel==7); % find the location of classes
TeLabel = test_classlabel(testIdx);
TeLabel(TeLabel==9) = 1;
TeLabel(TeLabel==7) = 0;
Test_Data = test_data(:,testIdx);
for lam = [0,0.1,1,10,100]
    % Training
    lambda = lam; % regularization factor;
    cen = Train_Data;
    input = Train_Data;
    r = sum(input.^2,1)' + sum(cen.^2,1) - 2*input'*cen;
    RBF = exp(-r/2/100^2);
    w =inv(RBF'*RBF+lambda*eye(size(RBF,2)))*RBF'*TrLabel';
    % Predict
    %   TrPred
    input = Train_Data;
    r = sum(input.^2,1)' + sum(cen.^2,1) - 2*input'*cen;
    RBF = exp(-r/2/100^2);
    TrPred = (RBF*w)';
    %   TePred
    input = Test_Data;
    r = sum(input.^2,1)' + sum(cen.^2,1) - 2*input'*cen;
    RBF = exp(-r/2/100^2);
    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(sprintf('\\lambda=%.1f',lam))
    saveas(fig,sprintf('q2a/lambda_%.1f.png',lam))
end
```

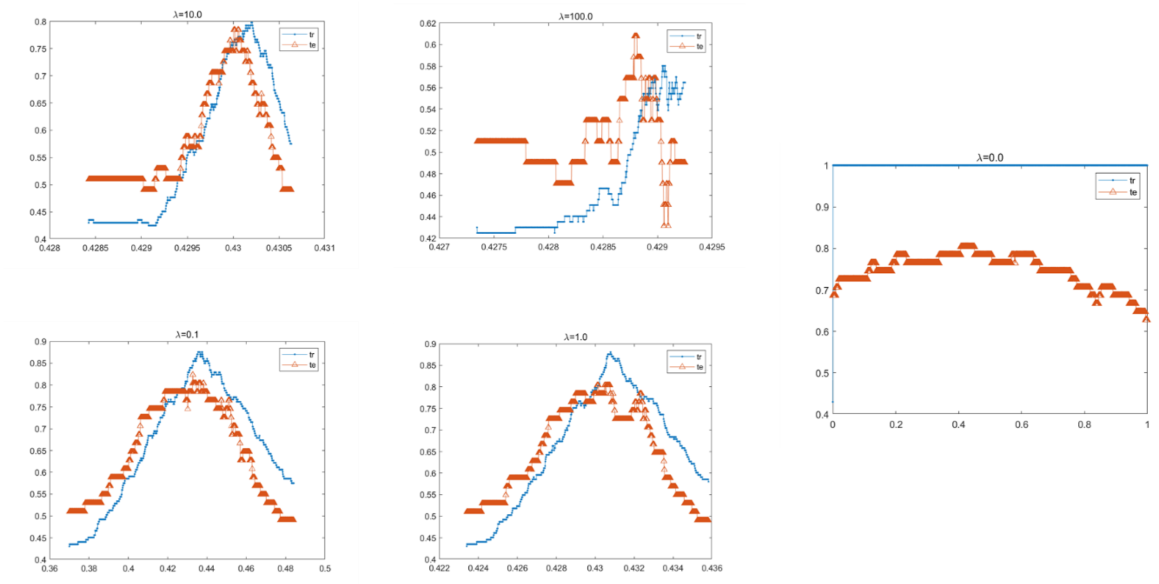

Figure Q2. (a) The performance of RBFN by using different regularization factor λ

- RBFN leads to over-fitting and result in poor generalization without regularization.
- When the regularization factor λ increases, the performance become worse and worse on the training set.
- Regularization can solve the problem because it decreases the performance gap between training set and test set.
- We should carefully choose the regularization factor. If λ is too big, the constraint dominates and less account is taken for training data error, which cause bad performance both on training set and test set. (See $\lambda = 100$)

Q2. (b).

MATLAB code:

```
clc
clear
% load MINST
load('../MNIST_database')
%   A0206597U->9=0 7=1
%   Train data
trainIdx = find(train_classlabel==9 | train_classlabel==7); % find the location of
classes
TrLabel = train_classlabel(trainIdx);
TrLabel(TrLabel==9) = 1;
TrLabel(TrLabel==7) = 0;
```

```
Train_Data = train_data(:,trainIdx);
%   Test data
testIdx = find( test_classlabel==9 | test_classlabel==7); % find the location of classes
TeLabel = test_classlabel(testIdx);
TeLabel(TeLabel==9) = 1;
TeLabel(TeLabel==7) = 0;
Test_Data = test_data(:,testIdx);
for sigma = [-1,0.1,1,10,100,1000,10000]
    % Training
    rng(520)
    rand_cen = datasample(Train_Data,100,2);
    input = Train_Data;
    r = sum(input.^2,1)' + sum(rand_cen.^2,1) - 2*input'*rand_cen;
    if sigma == -1
        sigma = sqrt(max(r,[],'all'))/sqrt(2*size(rand_cen,2));
    end
    RBF = exp(-r/2/sigma^2);
    w = pinv(RBF)*TrLabel';
    % Predict
    %   TrPred
    input = Train_Data;
    r = sum(input.^2,1)' + sum(rand_cen.^2,1) - 2*input'*rand_cen;
    RBF = exp(-r/2/sigma^2);
    TrPred = (RBF*w)';
    %   TePred
    input = Test_Data;
    r = sum(input.^2,1)' + sum(rand_cen.^2,1) - 2*input'*rand_cen;
    RBF = exp(-r/2/sigma^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(sprintf('Width=%.2f',sigma))
    saveas(fig,sprintf('q2b/width_%.2f.png',sigma))
end
```

Q2. (b). (i)

The simplest sigma of fixed centers is

$$\sigma_i = \frac{d_{max}}{\sqrt{2M}} = 0.9506$$

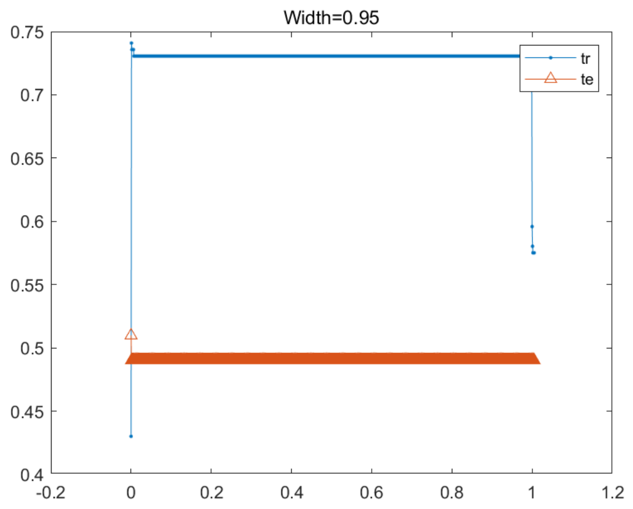

Figure Q2. (b). (i) Width fixed at appropriate size

Individual RBFs are too peaked to obtain a good performance, which means the center points are redundant. The actual M should be smaller.

Q2. (b). (ii)

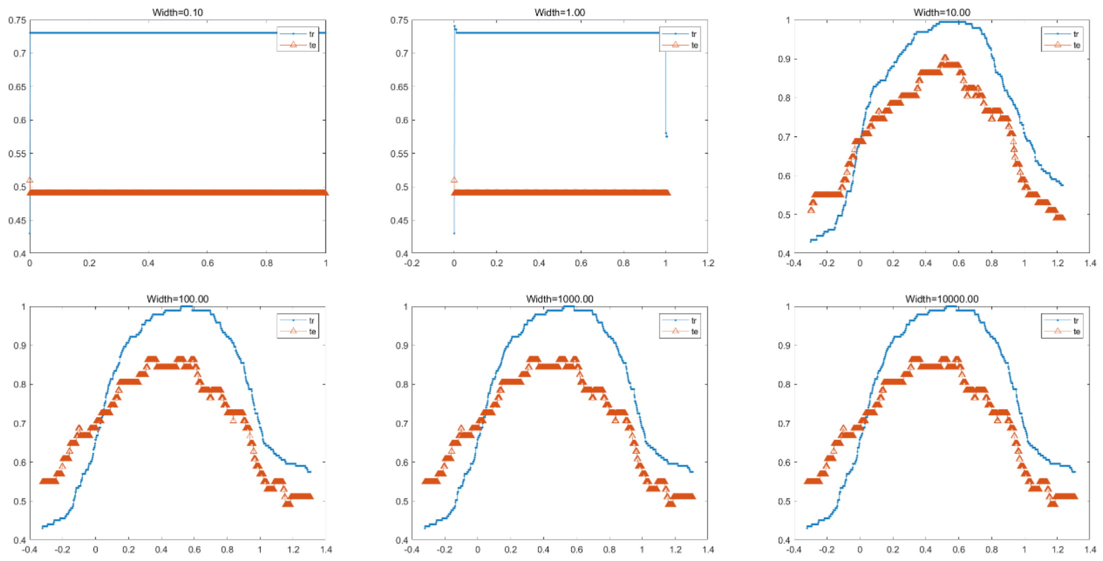

Figure Q2. (b). (ii) The performance of RBFN with different width

- Effective width 0.1, 1.0 are too small and 100, 1000, 10000 are too big. The individual RBFs are too peaked or too flat do not distinguish the difference. Therefore, respective performances are same in respective group.
- Effective width 10.0 is best setting among 0.1, 1.0, 10.0, 100, 1000 and 10000 in my case.

Q2. (c)

MATLAB code:

```
clc
clear
% load MINST Data
load('../MNIST_database')
%   A0206597U->9=0 7=1
%   Train data
trainIdx = find(train_classlabel==9 | train_classlabel==7); % find the location of
classes
TrLabel = train_classlabel(trainIdx);
TrLabel(TrLabel==9) = 1;
TrLabel(TrLabel==7) = 0;
Train_Data = train_data(:,trainIdx);
%   Test data
testIdx = find( test_classlabel==9 | test_classlabel==7); % find the location of classes
TeLabel = test_classlabel(testIdx);
TeLabel(TeLabel==9) = 1;
TeLabel(TeLabel==7) = 0;
Test_Data = test_data(:,testIdx);
% K-means
%   Initialization. centers
rng(520)
cur_cen = datasample(Train_Data,2,2);
prev_cen = zeros(size(cur_cen));
while sum(cur_cen~=prev_cen,'all') ~= 0
    prev_cen = cur_cen;
    %   Assignment
    dis = sum(prev_cen.^2,1)' + sum(Train_Data.^2,1) - 2*prev_cen'*Train_Data;
    [~,label]=min(dis,[],1);
    %   Updating
    cur_cen(:,1) = mean(Train_Data(:,label==1),2);
    cur_cen(:,2) = mean(Train_Data(:,label==2),2);
end
% Training
cen = cur_cen;
input = Train_Data;
r = sum(input.^2,1)' + sum(cen.^2,1) - 2*input'*cen;
RBF = exp(-r/2/100^2);
w = pinv(RBF)*TrLabel';
% Predict
%   TrPred
input = Train_Data;
r = sum(input.^2,1)' + sum(cen.^2,1) - 2*input'*cen;
RBF = exp(-r/2/100^2);
TrPred = (RBF*w)';
%   TePred
input = Test_Data;
r = sum(input.^2,1)' + sum(cen.^2,1) - 2*input'*cen;
RBF = exp(-r/2/100^2);
TePred = (RBF*w)';
% Plot
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
```

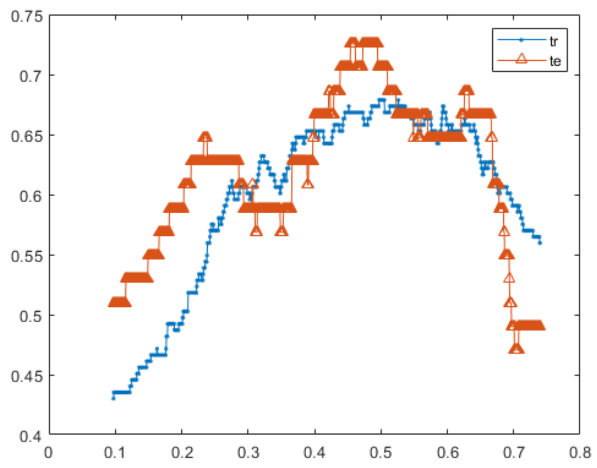

Figure Q2. (c) 1. The performance of RBFN-2

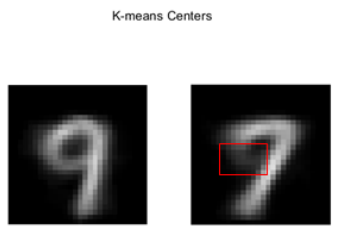

Figure Q2. (c) 2. Visualization of K-means centers

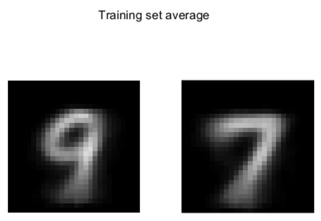

Figure Q2. (c) 3. Visualization of Training set average

The **K-means centers are like the respective training set average**. However, in the training set, there is no **shadow linked line** on the image 7 comparing with the K-means center 7. 7 and 9 are have close similar type, which **results inaccurate performance** of RBFN-2 and K-means centers.

Q3. Self-Organizing Map

Q3. (a).

MATLAB code:

```
clc
clear
% 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
% SOM
%   initialization
neurons = zeros(2,25);
sigma0 = sqrt(25^2+1)/2;
for epoch = 1:500
    lr = 0.1*exp(-epoch/500);
    sigma = sigma0*exp(-epoch/(500/log(sigma0)));
    for i = 1:size(trainX,2)
        dis = sum((trainX(:,i) - neurons).^2,1);
        [~,winner] = min(dis,[],2);
        d = abs([1:25]-winner);
        h = exp(-d.^2/(2*sigma^2));
        % Update
        neurons = neurons + lr*h.*(trainX(:,i) - neurons);
    end
end
hold on
plot(trainX(1,:),trainX(2,:),'+r');
plot(neurons(1,:),neurons(2,:),'o-');
hold off
```

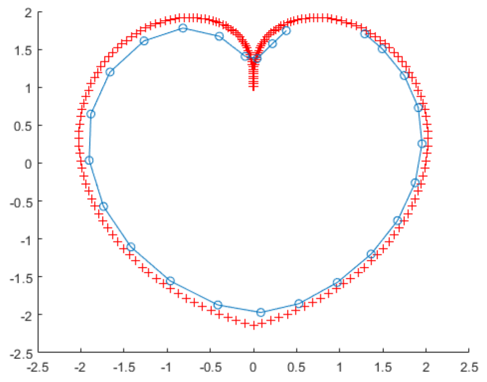

Figure Q3. (a). Visualization of SOM neurons trained by heart curve data

Q3. (b)

MATLAB code:

```
clc
clear
% Train data
rng(720) % reproducibility
trainX = rands(2,500); % 2x500 matrix, column-wise points
% SOM
%   initialization
neurons = randn(2,5,5);
sigma0 = sqrt(5^2+5^2)/2;
for epoch = 1:500
    lr = 0.1*exp(-epoch/500);
    sigma = sigma0*exp(-epoch/(500/log(sigma0)));
    for i = 1:size(trainX,2)
        dis = squeeze(sum((trainX(:,i) - neurons).^2,1))';
        [~,winner] = min(dis,[],'all','linear');
        k = ceil(winner/5);
        n = winner - (k-1)*5;
        d_j = ([1:5] - n).^2;
        d_i = ([1:5] - k).^2;
        d_square = d_j' + d_i;
        h = exp(-d_square./(2*sigma^2));
        % Update
        h = permute(repmat(h,[1,1,2]),[3 2 1]);
        neurons = neurons + lr*h.*(trainX(:,i) - neurons);
    end
end
% plot
plot(trainX(1,:),trainX(2,:),'+r');
hold on
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)],'bo-
')
        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)],'bo-
')
        end
    end
end
hold off
```

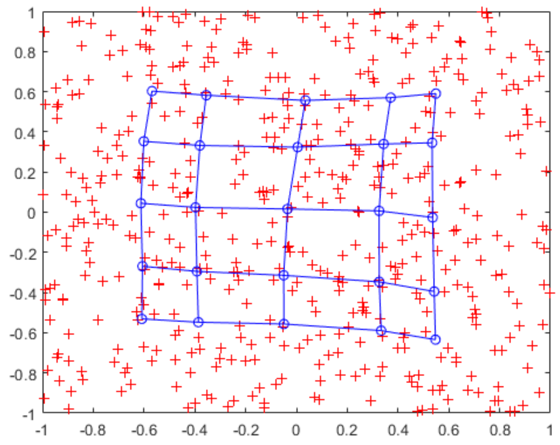

Figure Q3. (b). Visualization of SOM neurons trained by “square” data

Q3. (c)

Q3. (c-1).

```
% load MINST
load('../MNIST_database')
%   A0206597U-> NO:9,7 ->0,1,2,3,4,5,6,8
%   Train data
trainIdx = find(train_classlabel~=9 & train_classlabel~=7); % find the location of
classes
TrLabel = train_classlabel(trainIdx);
Train_Data = train_data(:,trainIdx);
% SOM
%   initialization
neurons = zeros(784,10,10);
sigma0 = sqrt(10^2+10^2)/2;
for epoch = 1:1000
    lr = 0.1*exp(-epoch/1000);
    sigma = sigma0*exp(-epoch/(1000/log(sigma0)));
    for i = 1:size(Train_Data,2)
        dis = squeeze(sum((Train_Data(:,i) - neurons).^2,1))';
        [~,winner] = min(dis,[],'all','linear');
        k = ceil(winner/10);
        n = winner - (k-1)*10;
        d_j = ([1:10] - n).^2;
        d_i = ([1:10] - k).^2;
        d_square = d_j' + d_i;
        h = exp(-d_square./(2*sigma^2));
        % Update
        h = permute(repmat(h,[1,1,784]),[3 2 1]);
        neurons = neurons + lr*h.*(Train_Data(:,i) - neurons);
    end
end
```

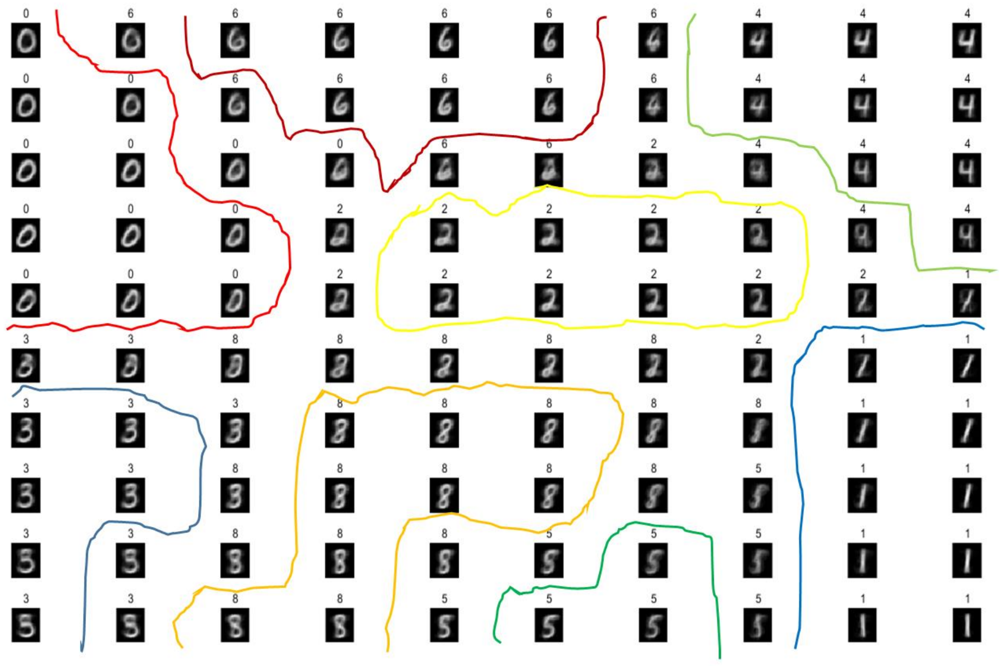
Figure Q3. (c-1) Corresponding semantic map and the trained SOM

The semantic map after trained can shows the different patterns. These patterns are organized or clustered around the same category. For example, the 0 liked patterns like 6, 3 and 8 are located together. And patterns liking 1 are located at the opposite side. The map shows the **structure or semantic related feature**.

Moreover, the **neurons at the border are shows the ambiguity**, which are like all the contacted patterns.

Q3. (c-2).

```
   clc
clear
% load MINST
load('../MNIST_database')
%   A0206597U-> NO:9,7 ->0,1,2,3,4,5,6,8
% load model
load('neurons')
% Predict
%   Test data
testIdx = find(test_classlabel~=9 & test_classlabel~=7); % find the location of classes
TeLabel = test_classlabel(testIdx);
Test_Data = test_data(:,testIdx);
% TePred
TePred = zeros(size(TeLabel));
counter_1 = 1;
counter_2 = 1;
```

```
for i = 1:size(Test_Data,2)
    dis = squeeze(sum((Test_Data(:,i) - neurons).^2,1));
    [~,winner] = min(dis,[],'all','linear');
    k = ceil(winner/10);
    n = winner - (k-1)*10;
    TePred(1,i) = maplabel(n,k);
    % plot some correct samples
    if TePred(1,i)==TeLabel(1,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(neurons(:,n,k),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(neurons(:,n,k),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)
```

```
TeAccr =  0.7789
```

The classification accuracy on the whole test set is **77.89%.** I visualize some samples during classification in Figure Q3. (c-2).

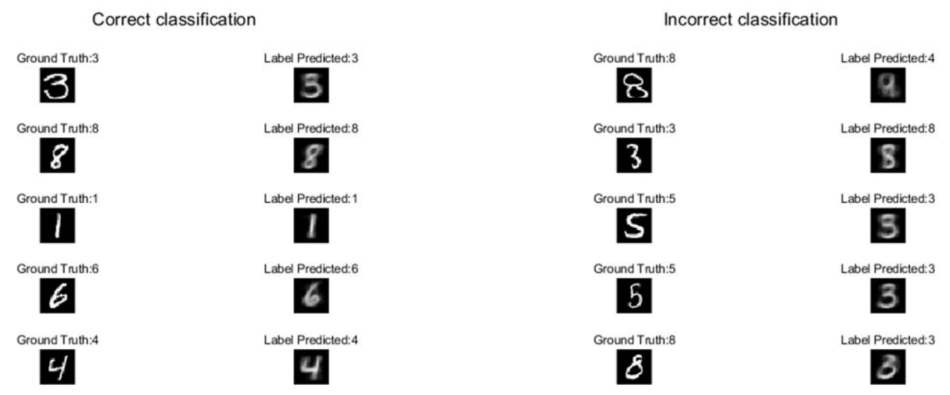

Figure Q3. (c-2). Classification samples visualization

Most of wrong classification happened at the border. For example, 8 was classified as 4 (The corresponding neuron (3,10) is located at the border.). One most confusion always happened is between 3 and 8, which **located close to each other at semantic map.**