NUS
National University of Singapore

Faculty of Engineering

Title	Homework 3
Module	EE5904
Name	Zavier Ong Jin Jie
Matriculation No.	A0138993L

Q1. Function Approximation with RBFN

1a. Exact Interpolation method

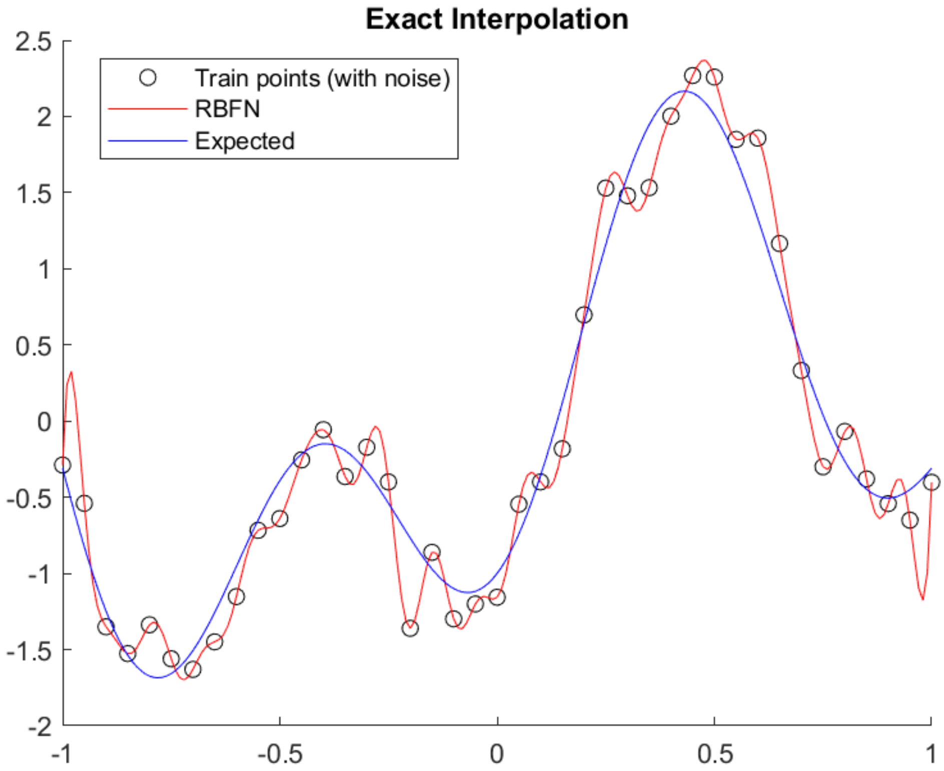


Figure 1. Exact Interpolation

```
Mean Square Error on test: 0.067415
```

Figure 2. MSE on test set

As observed in Figure 2, the Mean Square Error on the test set 0.067. From Figure 1, it can be observed that the RBFN is overfitting as it follows the training points present with noise very closely. As a result, leading to poor fitting results on the test set.

1b. Fixed Centers Selected at Random

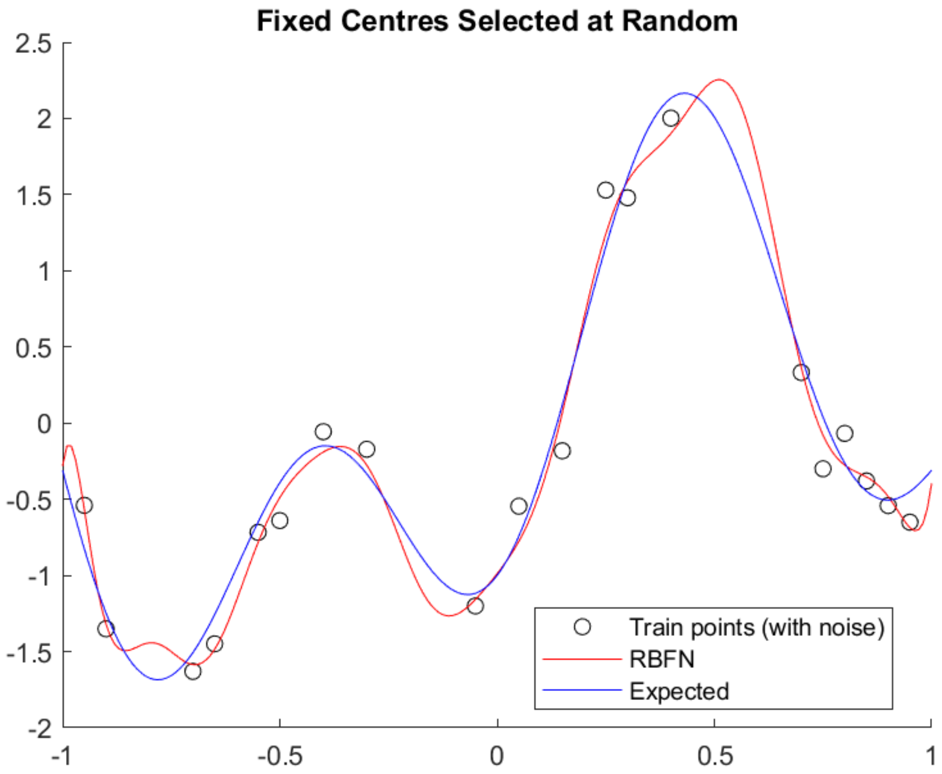


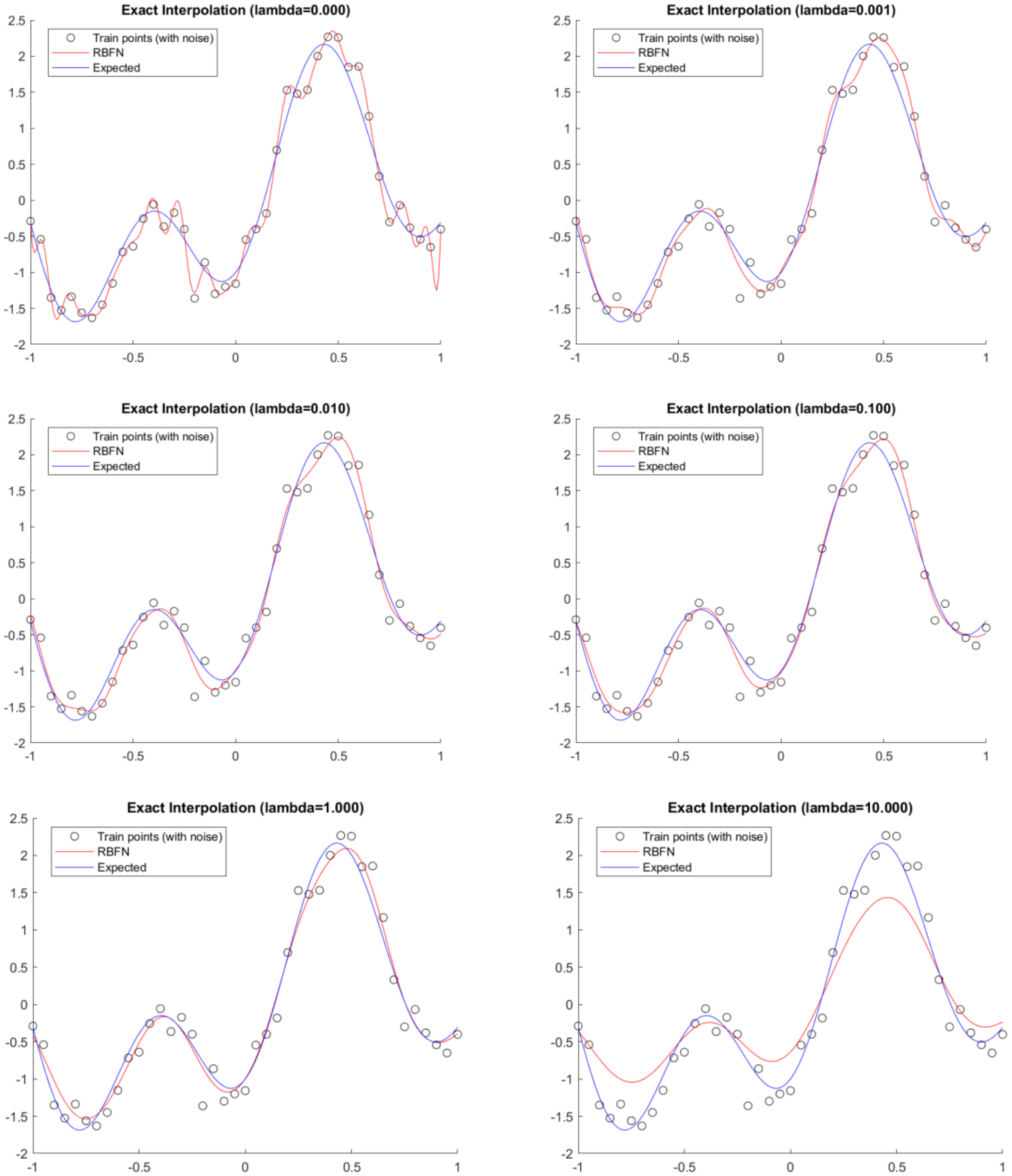
Figure 3. Fixed Centers Selected at Random

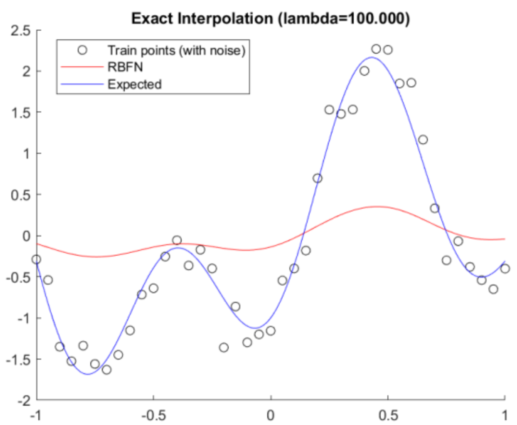
```
Mean Square Error on test: 0.027274
```

Figure 4. MSE on test set

Comparing Figure 4 and Figure 2, it is observed that the MSE on the test set is reduced. In Figure 3, the RBFN is able to fit to the test set better as compared to the exact interpolation method. Though noise is still present in the data, as only 20 centers are randomly selected among the sampling points, this strategy can reduce the degree of overfitting.

1c. Exact Interpolation with regularization






```
Mean Square Error on test (lambda = 0.000): 0.059834
Mean Square Error on test (lambda = 0.001): 0.024742
Mean Square Error on test (lambda = 0.010): 0.021024
Mean Square Error on test (lambda = 0.100): 0.017379
Mean Square Error on test (lambda = 1.000): 0.012527
Mean Square Error on test (lambda = 10.000): 0.140537
Mean Square Error on test (lambda = 100.000): 0.856718
```

Figure 5. Exact Interpolations with varying regularization factors

Figure 5 depicts the approximate performances of the RBFN using the same centers and widths determined in part a) while applying different regularization factors ranging from 0 to 100 at magnitudes of 10 (0, 0.001, 0.010, 0.1, 1, 10, 100). As shown in the bottom left corner in Figure 5, when regularization factor (lambda) is 1, it produces the least error and best performance of the RBFN. By observing the MSE errors as well as the plots, when lambda is small, the RBFN would over fit towards the training data, resulting to poor performance. On the flip side, when lambda is large, the smoothness constraint would dominate, leading to under fitting and ultimately decreasing performance of the RBFN.

Q2. Handwritten Character Classification with RBFN

2a. Exact Interpolation method

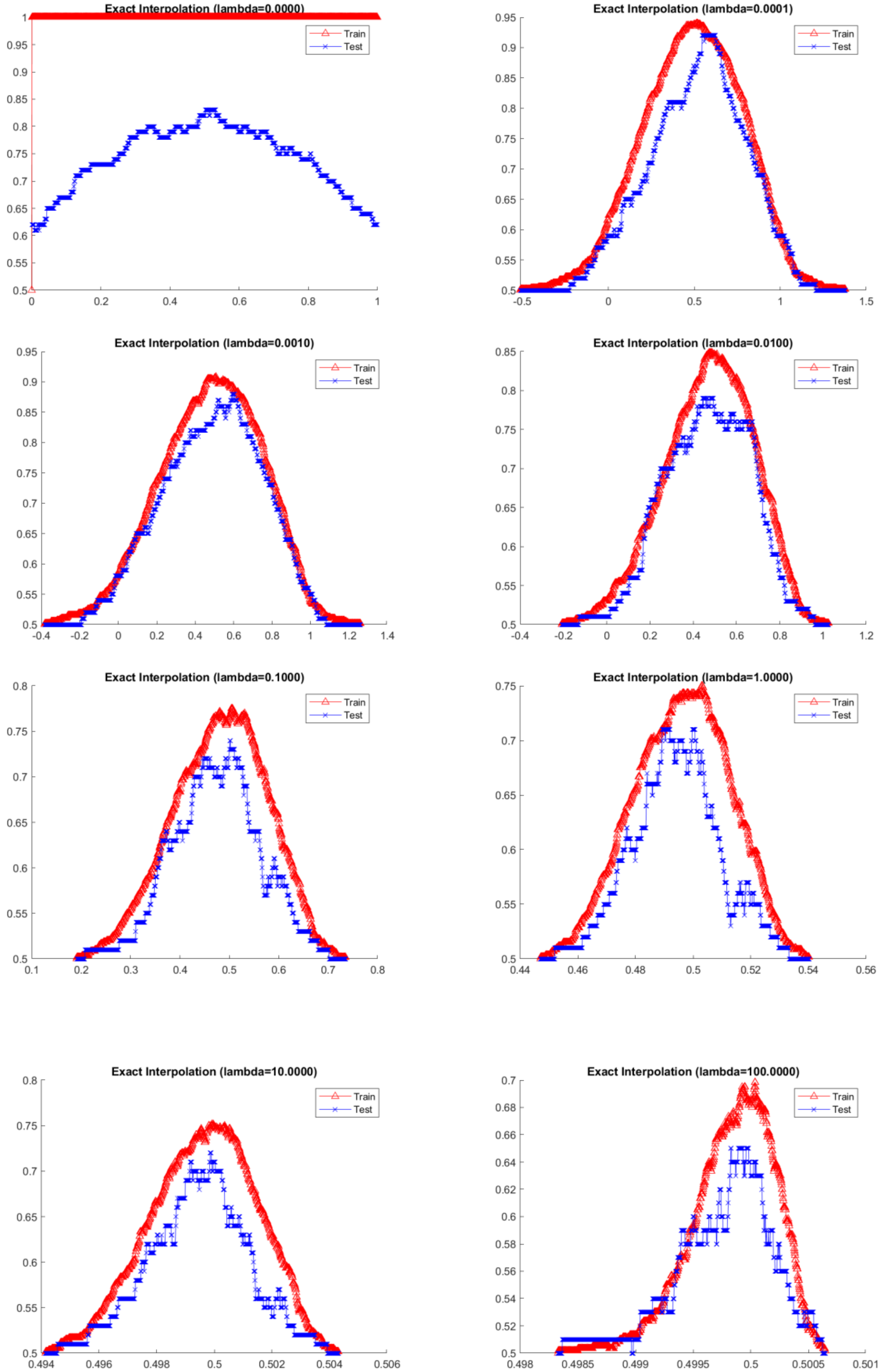


Figure 6. Exact Interpolation with varying regularization factors

Figure 6 depicts the exact interpolation method with varying regularization factors (lambda) on the training set. The first plot of Figure 6 (0,0) displays the performance of the RBFN without regularization. It is observed that the peak accuracy is at about 85%, suggesting that the RBFN could be over fitted to the model. After comparing the performances of the different RBFN with different regularization factors, RBFN with a lambda of 0.001 produces the best results. As lambda increases, the performance consistently deteriorates on both the training and test set.

2b. Fixed Centers Selected at Random

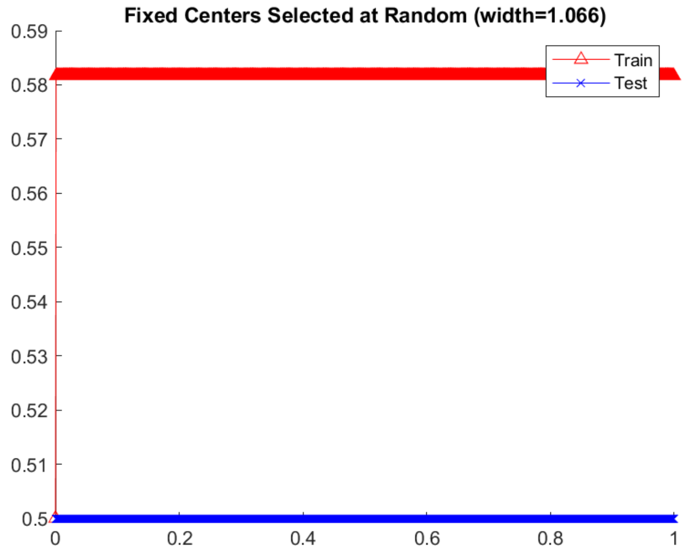


Figure 7. Width fixed at appropriate size

The sigma of fixed centers can be found by:

$$\sigma_i = \frac{d_{max}}{\sqrt{2M}} = 1.066$$

Based on Figure 7, we can see that the individual RBFs have bad performance as it is too peaked, which suggests that the center points are redundant and M should have a smaller value.

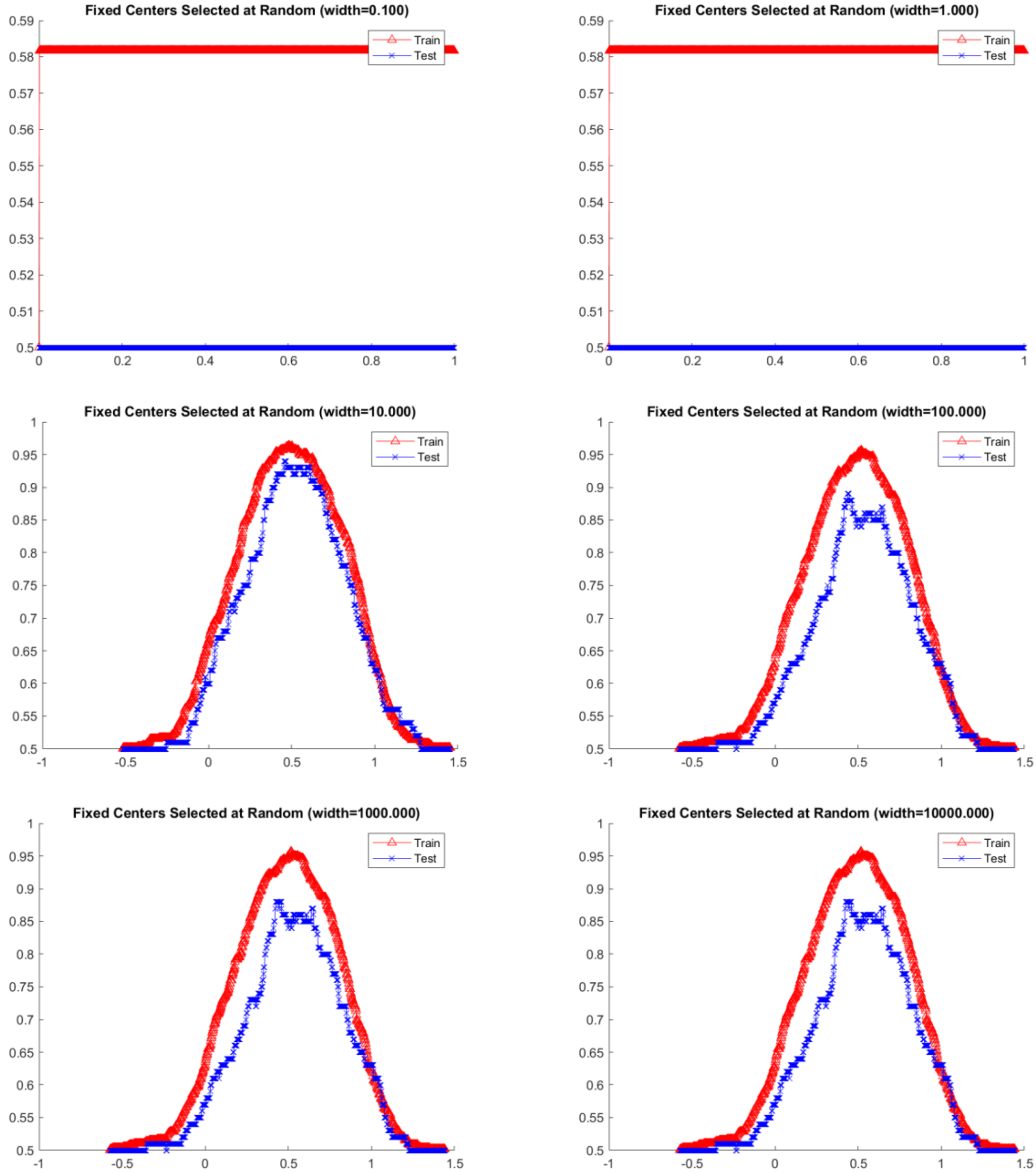


Figure 8. Fixed Centers Selected at Random with varying widths

Observing Figure 8, we can see that widths of 0.1 and 1 produces similar results to Figure 7, suggesting that the RBFs are too peaked while widths of 100, 1000 and 10000 are a little flat producing relatively poorer performances as compared to width 10. In this setting, the width of 10 seems to produce the best results with highest accuracy.

2c. K-Mean Clustering

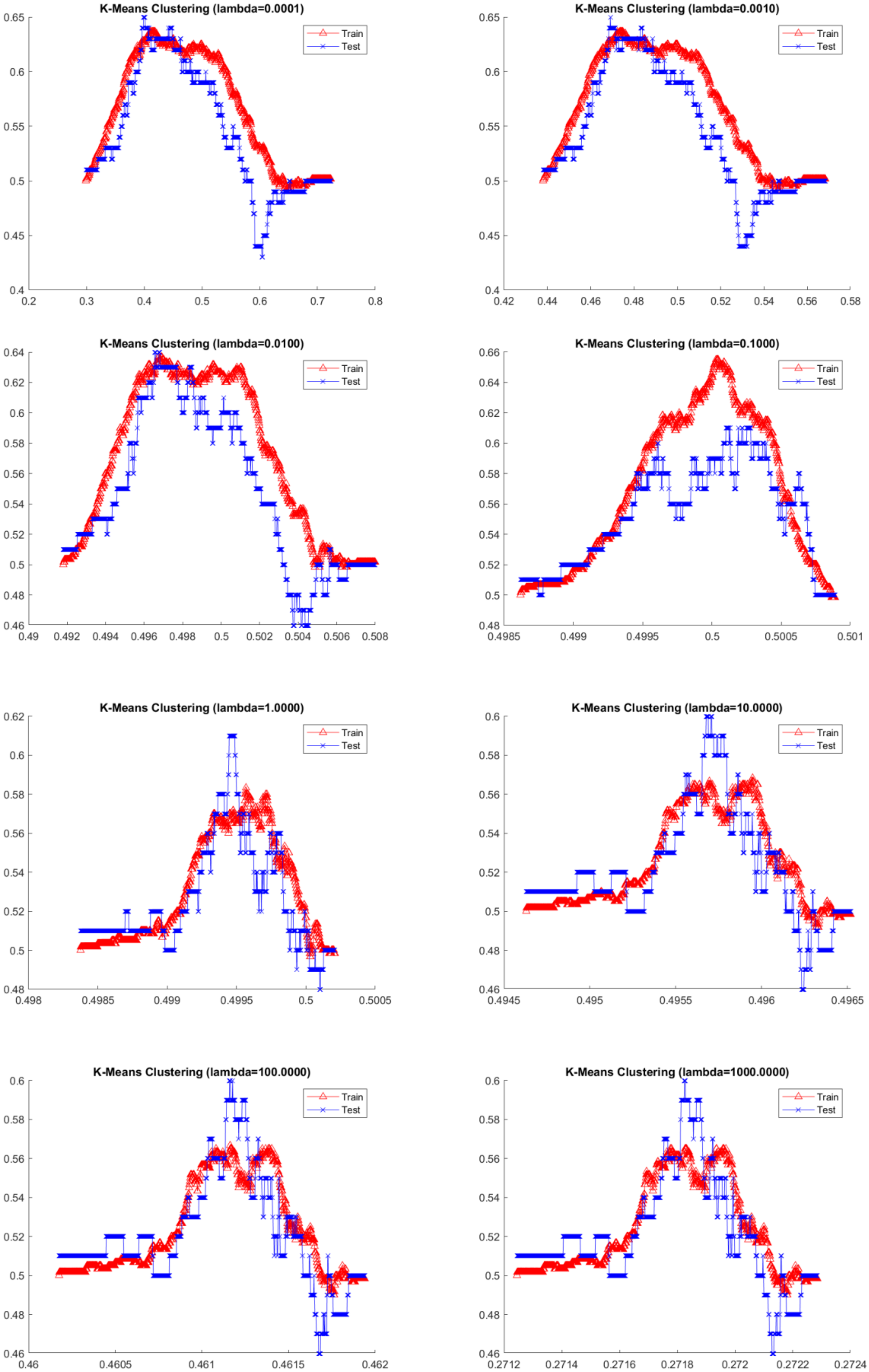


Figure 9. K-mean Clustering with varying regularization factors

As shown in Figure 9, different regularization factors were attempted to achieve a better result to no avail. However, it seems like the best accuracy would be when lambda is 1.

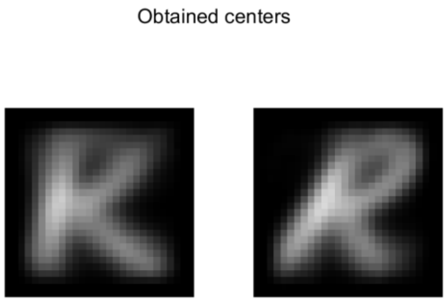


Figure 10. Visualization of K-mean centers

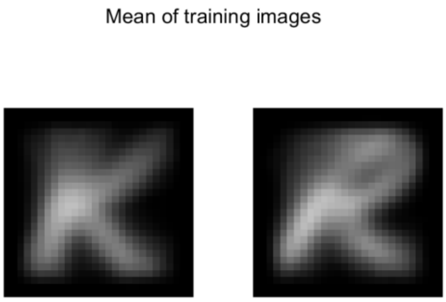


Figure 11. Visualization of Mean of training images

Comparing Figure 10 (Obtained Centers) and Figure 11 (Mean training images), we can see both images resembles the letters ‘K’ and ‘R’. However, it is interesting to note that the obtained center for letter ‘K’ has an additional faded line demarcated in Figure 10 by a red box. This additional line causes the letter ‘K’ to become somewhat similar to the letter ‘R’ as well, resulting in an inaccurate performance of the RBFN and K-means centers.

Q3. Self-Organizing Map

3a. SOM mapping to 1D output layer to 'sinusoid curve'

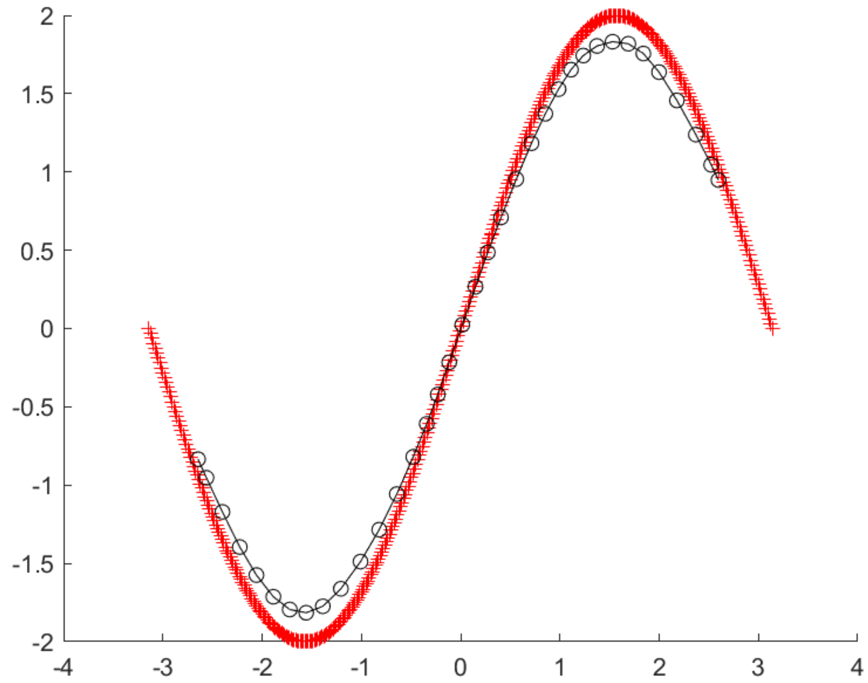


Figure 12. Visualization of SOM neurons trained by sinusoid curve

3b. SOM mapping to 2D output layer to 'circle'

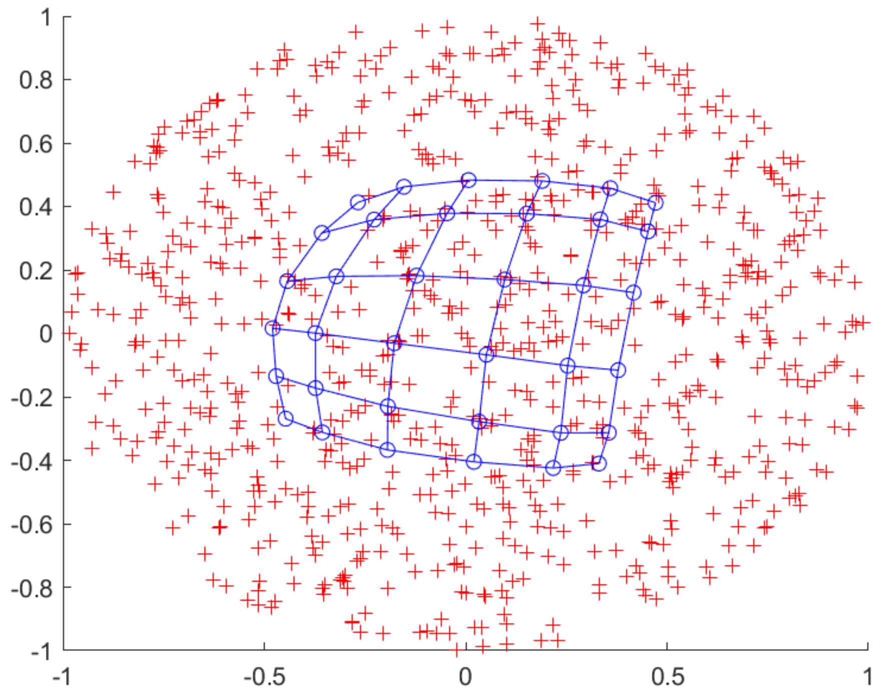


Figure 13. Visualization of SOM neurons trained by 'circle' data

3c. SOM that cluster and classifies handwritten characters

c-1)

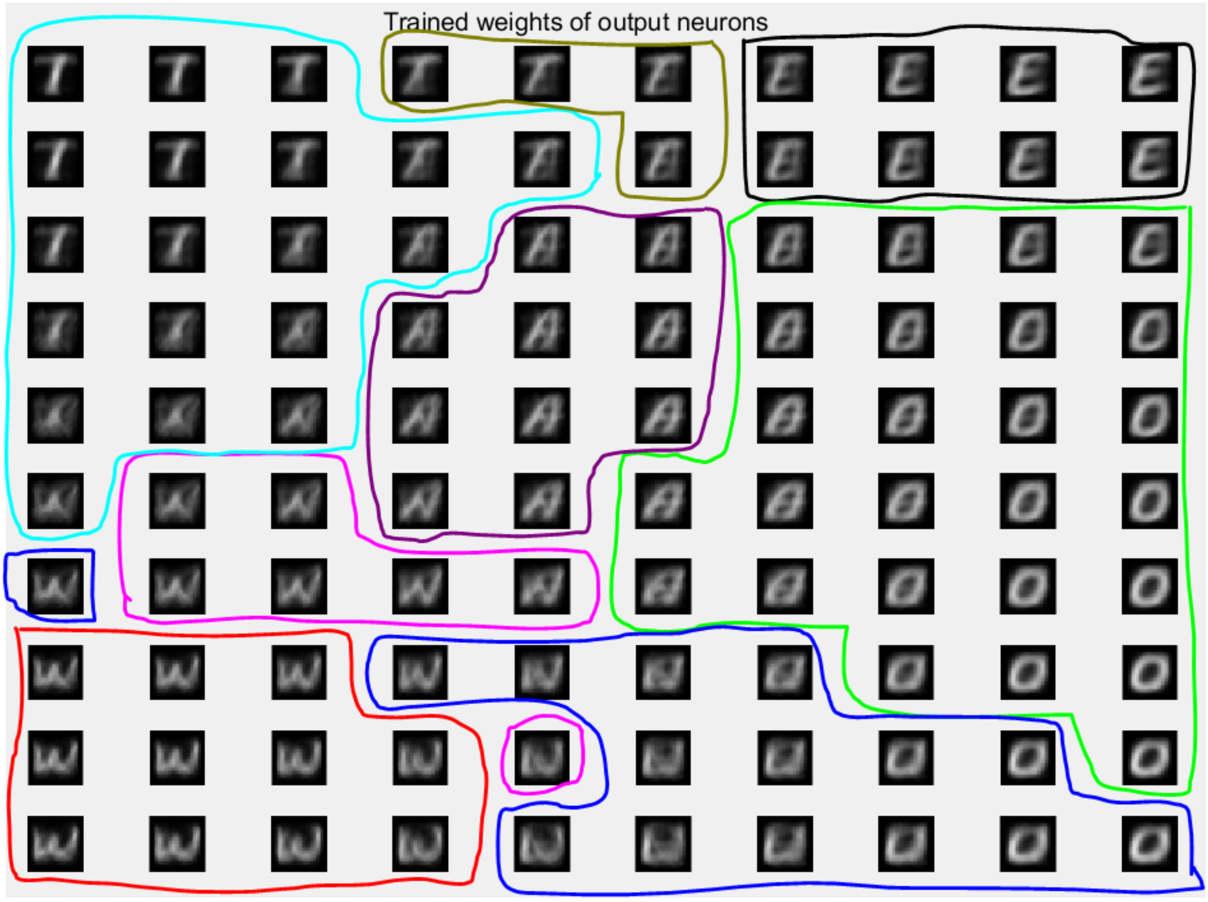


Figure 14. Corresponding Semantic Map of trained SOM

Label(Char)	0 (N)	1 (E)	2 (U)	4 (A)	5 (L)	6 (T)	7 (W)	8 (O)
Color	pink	black	blue	purple	cyan	olive	red	green

As shown in Figure 14, the trained SOM forms a semantic map where the similar samples belonging to the same label are mapped close together and dissimilar ones apart. For example, we can see the letters ‘U’ and ‘W’ are mapped close together as they both have a downwards curve. However, the letters ‘W’ and ‘E’ are mapped far apart are they do not have much similarities. In addition, neurons that are near the bounders drawn in Figure 14 tend to be ambiguous, containing multiple letters overlapped against each other.

c-2)

```
Classification accuracy on test set: 0.6050
```

Figure 15. Classification accuracy on test set

The classification accuracy on the test set is relatively low. This could be because the recommended iteration is set to 1000. However, the total number of training points are 2400. Since at each iteration, we would update the weights based on only 1 training point, there would be another 1400 sample points (or even more considering how the points are sampled). As a result, depending on the sample points, the output neurons would be skewed towards the label that has been chosen at random more.

This can be shown in Figure 14, where there are only 4 neurons that labelled 6 (‘T’) while there are 26 neurons that are labelled as 8 (‘O’). This would suggest that when classifying test images, it would be more likely that the trained SOM would classify the letter ‘O’ correctly while letter ‘T’ incorrectly.

There are 2 possible methods to minimize this issue:

1. Ensure that all labels are being sampled uniformly when updating the weights.
2. Increase the number of iterations to allow uniform distribution of sampled labels.

Appendix

Q1a

```
clc
clear
close all

%for reproducibility
rng(3);
% train data
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 data
test_x = -1:0.01:1;
test_y = 1.2*sin(pi*test_x) - cos(2.4*pi*test_x);

% exact interpolation on train
% no need to square or sqrt since x has only 1 value
r = abs(train_x'-train_x);
% gaussian rbf
phi = exp( (r.^2) / (-2*((0.1)^2)) );
w = phi \ train_y';

%predict y
r_test = abs(test_x' - train_x);
phi = exp( (r_test.^2) / (-2*((0.1)^2)) );
y_predict = (phi*w)';

fig = figure();
hold on
plot(train_x,train_y,'ok')
plot(test_x,y_predict, 'r')
plot(test_x,test_y, 'b')
legend('Train points (with noise)','RBFN','Expected', 'Location', 'Best')
title('Exact Interpolation')
saveas(fig, 'q1a.png')
hold off

%MSE of test
mse_test = sum((y_predict-test_y).^2)/size(y_predict, 2);
fprintf('Mean Square Error on test: %f\n', mse_test);
```

Q1b

```
clc
clear
close all

%for reproducibility
rng(3);
% train data
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 data
test_x = -1:0.01:1;
test_y = 1.2*sin(pi*test_x) - cos(2.4*pi*test_x);

% randomly select 20 centers
m = 20;
center_idx = randperm(size(train_x, 2));
mew_x = train_x(center_idx(1:m));
mew_y = train_y(center_idx(1:m));

% no need to square or sqrt since x has only 1 value
r = abs(train_x'-mew_x);
dist_cen = abs(mew_x'-mew_x);
% maximum dist between chosen centers
dmax = max(dist_cen, [], 'all');
% rbf
phi = exp( -(m/dmax^2) * r.^2 );
w = phi \ train_y';

%predict y
r_test = abs(test_x' - mew_x);
% rbf
phi = exp( -(m/dmax^2) * r_test.^2 );
y_predict = (phi*w)';

fig = figure();
hold on
plot(mew_x,mew_y,'ok')
plot(test_x,y_predict, 'r')
plot(test_x,test_y, 'b')
legend('Train points (with noise)','RBFN','Expected', 'Location', 'Best')
title('Fixed Centres Selected at Random')
saveas(fig, 'q1b.png')
hold off

%MSE of test
mse_test = sum((y_predict-test_y).^2)/size(y_predict, 2);
fprintf('Mean Square Error on test: %f\n', mse_test);
```

Q1c

```
clc
clear
close all

%for reproducibility
rng(3);
% train data
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 data
test_x = -1:0.01:1;
test_y = 1.2*sin(pi*test_x) - cos(2.4*pi*test_x);

r_factors = [0, 0.001, 0.01, 0.1, 1, 10, 100];
for i = r_factors
    % exact interpolation on train
    % no need to square or sqrt since x has only 1 value
    lambda = i;
    r = abs(train_x'-train_x);
    % gaussian rbf
    phi = exp( (r.^2) / (-2*((0.1)^2)) );
    % applying regularization method to determine new weights
    w = pinv((phi'*phi) + lambda*eye(size(phi, 2))) * (phi'*train_y');

    %predict y
    r_test = abs(test_x' - train_x);
    phi = exp( (r_test.^2) / (-2*((0.1)^2)) );
    y_predict = (phi*w)';

    fig = figure();
    hold on
    plot(train_x,train_y,'ok')
    plot(test_x,y_predict, 'r')
    plot(test_x,test_y, 'b')
    legend('Train points (with noise)','RBFN','Expected', 'Location',
'Best')
    title(sprintf('Exact Interpolation (lambda=%.3f)', i))
    saveas(fig, sprintf('q1c_%.3f.png', i))
    hold off

    %MSE of test
    mse_test = sum((y_predict-test_y).^2)/size(y_predict, 2);
    fprintf('Mean Square Error on test (lambda = %0.3f): %f\n', i,
mse_test);
end
```

Q2a

```
clc
clear
close all

% Matric A0138993L
% Classes chosen: 9 and 3
load('characters10.mat');

train_idx = find(train_label == 3 | train_label == 9);
% 9 --> 1 and 3 --> 0
TrLabel = train_label(train_idx);
TrLabel(TrLabel == 9) = 1;
TrLabel(TrLabel == 3) = 0;

train_x = train_data(train_idx, :);
% normalizing train data
train_x = mat2gray(train_x(:,:));

test_idx = find(test_label == 3 | test_label == 9);
TeLabel = test_label(test_idx);
TeLabel(TeLabel == 9) = 1;
TeLabel(TeLabel == 3) = 0;

test_x = test_data(test_idx, :);
% normalizing test data
test_x = mat2gray(test_x(:,:));

r_factors = [0, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100];
sd= 100;
for i = r_factors
    % exact interpolation on train
    lambda = i;
    r_train = pdist2(train_x, train_x, 'squaredeuclidean');
    % gaussian rbf
    phi = exp( r_train / (-2*((sd)^2)) );
    % applying regularization method to determine new weights
    if (i == 0)
        w = phi \ TrLabel;
    else
        w = ((phi'*phi) + lambda*eye(size(phi, 2))) \ (phi'*TrLabel);
    end

    % TrPred
    TrPred = (phi*w)';

    %TePred
    r_test = pdist2(test_x, train_x, 'squaredeuclidean');
    %gaussian rbf
    phi = exp( r_test / (-2*((sd)^2)) );
    TePred = (phi*w)';

    fig = figure();
    TrAcc = zeros(1,1000);
    TeAcc = zeros(1,1000);
    thr = zeros(1,1000);
    TrN = length(TrLabel);
    TeN = length(TeLabel);
    for j = 1:1000
```

```
        t = (max(TrPred)-min(TrPred)) * (j-1)/1000 + min(TrPred);
        thr(j) = t;
        TrAcc(j) = (sum(TrLabel(TrPred<t)==0) + sum(TrLabel(TrPred>=t)==1))
/ TrN;
        TeAcc(j) = (sum(TeLabel(TePred<t)==0) + sum(TeLabel(TePred>=t)==1))
/ TeN;
    end
    hold on
    plot(thr, TrAcc, '-^r');
    plot(thr, TeAcc, '-xb');
    legend('Train','Test');
    title(sprintf('Exact Interpolation (lambda=%.4f)', i))
    saveas(fig,sprintf('q2a_lambda_%.4f.png',i))
    hold off
end
```

Q2b

```
clc
clear
close all

% Matric A0138993L
% Classes chosen: 9 and 3
load('characters10.mat');

%imshow(reshape(train_data(2997,:), [28,28]));
train_idx = find(train_label == 3 | train_label == 9);
% 9 --> 1 and 3 --> 0
TrLabel = train_label(train_idx);
TrLabel(TrLabel == 9) = 1;
TrLabel(TrLabel == 3) = 0;

train_x = train_data(train_idx, :);
% normalizing train data
train_x = mat2gray(train_x(:,:));

test_idx = find(test_label == 3 | test_label == 9);
TeLabel = test_label(test_idx);
TeLabel(TeLabel == 9) = 1;
TeLabel(TeLabel == 3) = 0;

test_x = test_data(test_idx, :);
% normalizing test data
test_x = mat2gray(test_x(:,:));

% randomly select 100 centers
m = 100;
% seed for reproducibility
rng(3)
center_idx = randperm(size(train_x, 1));
selected_train_x = train_x(center_idx(1:m), :);
selected_TrLabel = TrLabel(center_idx(1:m), :);

r_train = pdist2(train_x, selected_train_x, 'squaredeuclidean');
dist_cen = pdist2(selected_train_x, selected_train_x, 'squaredeuclidean');
dmax_squared = max(dist_cen, [], 'all');
sigma = sqrt(dmax_squared/(2*m));

vary_width = [sigma, 0.1, 1, 10, 100, 1000, 10000];
for i = vary_width
    width = i;
    phi = exp( -(r_train / (2*(width^2))) );
    w = phi \ TrLabel;

    %TrPred
    TrPred = (phi*w);

    %TePred
    r_test = pdist2(test_x, selected_train_x, 'squaredeuclidean');
    phi = exp ( -(r_test / (2*(width^2))) );
    TePred = (phi*w);

    fig = figure();
    TrAcc = zeros(1,1000);
```

```
    TeAcc = zeros(1,1000);
    thr = zeros(1,1000);
    TrN = length(TrLabel);
    TeN = length(TeLabel);
    for j = 1:1000
        t = (max(TrPred)-min(TrPred)) * (j-1)/1000 + min(TrPred);
        thr(j) = t;
        TrAcc(j) = (sum(TrLabel(TrPred<t)==0) + sum(TrLabel(TrPred>=t)==1))
/ TrN;
        TeAcc(j) = (sum(TeLabel(TePred<t)==0) + sum(TeLabel(TePred>=t)==1))
/ TeN;
    end
    hold on
    plot(thr, TrAcc, '-^r');
    plot(thr, TeAcc, '-xb');
    legend('Train','Test');
    title(sprintf('Fixed Centers Selected at Random (width=%.3f)', i))
    saveas(fig,sprintf('q2b_lambda_%.3f.png',i))
    hold off
end
```

Q2c

```
clc
clear
close all

% Matric A0138993L
% Classes chosen: 9 and 3
load('characters10.mat');

train_idx = find(train_label == 3 | train_label == 9);
% 9(K) --> 1 and 3(R) --> 0
TrLabel = train_label(train_idx);
TrLabel(TrLabel == 9) = 1;
TrLabel(TrLabel == 3) = 0;

train_x = train_data(train_idx, :);
% normalizing train data
train_x = mat2gray(train_x(:,:));

test_idx = find(test_label == 3 | test_label == 9);
TeLabel = test_label(test_idx);
TeLabel(TeLabel == 9) = 1;
TeLabel(TeLabel == 3) = 0;

test_x = test_data(test_idx, :);
% normalizing test data
test_x = mat2gray(test_x(:,:));

rng(3);
k = 2;
center_idx = randperm(size(train_x, 1));
curr_cen = train_x(center_idx(1:k), :);
old_cen = zeros(size(curr_cen));

%K means clustering
while ~isequal(curr_cen, old_cen)
    old_cen = curr_cen;
    % assignment
    distance = pdist2(old_cen, train_x);
    [~, label] = min(distance, [], 1);
    % updating
    curr_cen(1,:) = mean(train_x(label==1, :), 1);
    curr_cen(2,:) = mean(train_x(label==2, :), 1);
end

%obtained centers
fig = figure();
sgtitle('Obtained centers');
subplot(121);
imshow(reshape(curr_cen(1,:), [28,28]));
subplot(122);
imshow(reshape(curr_cen(2,:), [28,28]));
saveas(fig, 'q2c_k_centers.png');
%mean of training image
fig = figure();
sgtitle('Mean of training images');
subplot(121);
imshow(reshape(mean(train_x(TrLabel==1, :), 1), [28,28]));
subplot(122);
```

```
imshow(reshape(mean(train_x(TrLabel==0, :), 1), [28,28]));
saveas(fig, 'q2c_mean_imgs.png');

%training
r_factors = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000];
sigma = 100;
for i=r_factors
    lambda = i;
    r_train = pdist2(train_x, curr_cen, 'squaredeuclidean');
    phi = exp( r_train / (-2*((sigma)^2)) );
    w = ((phi'*phi) + lambda*eye(size(phi, 2))) \ (phi'*TrLabel);
    %TrPred
    TrPred = (phi*w)';

    %TePred
    r_test = pdist2(test_x, curr_cen, 'squaredeuclidean');
    %gaussian rbf
    phi = exp( r_test / (-2*((sigma)^2)) );
    TePred = (phi*w)';

    fig = figure();
    TrAcc = zeros(1,1000);
    TeAcc = zeros(1,1000);
    thr = zeros(1,1000);
    TrN = length(TrLabel);
    TeN = length(TeLabel);
    for j = 1:1000
        t = (max(TrPred)-min(TrPred)) * (j-1)/1000 + min(TrPred);
        thr(j) = t;
        TrAcc(j) = (sum(TrLabel(TrPred<t)==0) + sum(TrLabel(TrPred>=t)==1))
/ TrN;
        TeAcc(j) = (sum(TeLabel(TePred<t)==0) + sum(TeLabel(TePred>=t)==1))
/ TeN;
    end
    hold on
    plot(thr, TrAcc, '-^r');
    plot(thr, TeAcc, '-xb');
    legend('Train','Test');
    title(sprintf('K-Means Clustering (lambda=%.4f)', i))
    saveas(fig,sprintf('q2c_lambda_%.4f.png',i))
    hold off
end
```

Q3a

```
clc
clear
close all

%training points sampled from sine curve
x = linspace(-pi, pi, 400);
train_x = [x; 2*sin(x)]; %2x400 matrix

%SOM
T = 600;
N = 1;
M = 36;
lr0 = 0.1;
sigma0 = sqrt(M^2*N^2) / 2;
tau = T / log(sigma0);
weights = rand(2, 36);
for n = 1:T
    lr = lr0*exp(-n/T);
    sigma = sigma0*exp(-n/tau);
    %sample input vector
    i = randperm(400, 1);
    %determine winner
    distance = sum((train_x(:,i) - weights).^2,1);
    [~, winner] = min(distance, [], 2);
    neuron_position = (1:36);
    d = abs(neuron_position - winner);
    h = exp(-d.^2/(2*sigma^2));
    % Update
    weights = weights + lr*h.*(train_x(:,i) - weights);
end

fig = figure();
hold on
plot(train_x(1,:), train_x(2,:), '+r');
plot(weights(1,:), weights(2,:), '-ok');
hold off
saveas(fig, 'q3a.png');
```

Q3b

```
clc
clear
close all

rng(43)
X = randn(800,2);
s2 = sum(X.^2,2);
train_x = (X.*repmat(1*(gammainc(s2/2,1).^(1/2))./sqrt(s2),1,2))';

%SOM
T = 600;
lr0 = 0.1;
sigma0 = sqrt(6^2*6^2) / 2;
tau = T / log(sigma0);
weights = rand(2, 6, 6);

for n = 1:T
    lr = lr0*exp(-n/T);
    sigma = sigma0*exp(-n/tau);
    %sample input vector
    i = randperm(800, 1);
    %determine winner
    distance = squeeze(sum((train_x(:,i) - weights).^2,1))';
    [~,winner] = min(distance,[],'all','linear');
    [col, row] = ind2sub(size(distance), winner);

    %get time-carying neighborhood function
    neuron_position = (1:6);
    d_j = (neuron_position - col).^2;
    d_i = (neuron_position - row).^2;
    dji = d_j' + d_i;
    h = exp(-dji./(2*sigma^2));
    % Update
    h = permute(repmat(h,[1,1,2]),[3 2 1]);
    weights = weights + lr*h.*(train_x(:,i) - weights);
end

fig = figure();
hold on
plot(train_x(1,:), train_x(2,:), '+r');
weights_1 = squeeze(weights(1, :, :));
weights_2 = squeeze(weights(2, :, :));
for i = 1:6
    plot(weights_1(i,:), weights_2(i,:), 'bo-');
    plot(weights_1(:,i), weights_2(:,i), 'bo-');
end
hold off
saveas(fig, 'q3b.png');
```

Q3c1

```
clc
clear
close all
% Matric A0138993L
% Classes chosen: 9 and 3
load('characters10.mat');
% set seed for reproducibility
rng(234);

train_idx = find(train_label ~= 3 & train_label ~= 9);
TrLabel = train_label(train_idx);
train_x = train_data(train_idx, :);
% normalizing train data
train_x = mat2gray(train_x(:,:))';

T = 1000;
lr0 = 0.1;
sigma0 = sqrt(10^2*10^2) / 2;
tau = T / log(sigma0);
weights = rand(784, 10, 10);

for n = 1:T
    lr = lr0*exp(-n/T);
    sigma = sigma0*exp(-n/tau);
    %sample input vector
    i = randperm(2400, 1);
    %determine winner
    distance = squeeze(sum((train_x(:,i) - weights).^2,1))';
    [~,winner] = min(distance,[],'all','linear');
    [col, row] = ind2sub(size(distance), winner);

    %get time-carying neighborhood function
    neuron_position = (1:10);
    d_j = (neuron_position - col).^2;
    d_i = (neuron_position - row).^2;
    dji = d_j' + d_i;
    h = exp(-dji./(2*sigma^2));
    % Update
    h = permute(repmat(h,[1,1,784]),[3 2 1]);
    weights = weights + lr*h.*(train_x(:,i) - weights);
end

fig = figure();
fig.Position = [100 100 1200 800];
sgtitle('Trained weights of output neurons');
marked_neuron = zeros(10);

for i = 1:size(weights, 2)
    for j = 1:size(weights, 3)
        distance = squeeze(sum((train_x(:,:)-weights(:,i,j)).^2, 1))';
        [~, win_idx] = min(distance, [], 'all', 'linear');
        winner_label = TrLabel(win_idx);
        marked_neuron(i, j) = winner_label;
        subplot(10,10, ((i-1)*10+j));
        imshow(reshape(weights(:,i,j), 28, 28));
    end
end
saveas(fig, 'q3c1.png');
```

Q3c2

```
clc
clear
close all
% Matric A0138993L
% Classes not chosen: 9 and 3
load('characters10.mat');

%set seed for reproducibility
rng(234);
train_idx = find(train_label ~= 3 & train_label ~= 9);
TrLabel = train_label(train_idx);
train_x = train_data(train_idx, :);
% normalizing train data
train_x = mat2gray(train_x(:,:))';

test_idx = find(test_label ~= 3 & test_label ~= 9);
TeLabel = test_label(test_idx);
test_x = test_data(test_idx, :);
% normalizing test data
test_x = mat2gray(test_x(:,:))';

T = 1000;
lr0 = 0.1;
sigma0 = sqrt(10^2*10^2) / 2;
tau = T / log(sigma0);
weights = rand(784, 10, 10);

for n = 1:T
    lr = lr0*exp(-n/T);
    sigma = sigma0*exp(-n/tau);
    %sample input vector
    i = randperm(2400, 1);
    %determine winner
    distance = squeeze(sum((train_x(:,i) - weights).^2,1))';
    [~,winner] = min(distance,[],'all','linear');
    [col, row] = ind2sub(size(distance), winner);
    %get time-carying neighborhood function
    neuron_position = (1:10);
    d_j = (neuron_position - col).^2;
    d_i = (neuron_position - row).^2;
    dji = d_j' + d_i;
    h = exp(-dji./(2*sigma^2));
    % Update
    h = permute(repmat(h,[1,1,784]),[3 2 1]);
    weights = weights + lr*h.*(train_x(:,i) - weights);
end

marked_neuron = zeros(10);

for i = 1:size(weights, 2)
    for j = 1:size(weights, 3)
        distance = squeeze(sum((train_x(:,:)-weights(:,i,j)).^2, 1))';
        [~, win_idx] = min(distance, [], 'all', 'linear');
        winner_label = TrLabel(win_idx);
        marked_neuron(i, j) = winner_label;
    end
end
```

```
TePred = zeros(size(TeLabel, 1), 1);
for i = 1:size(test_x, 2)
    distance = squeeze(sum((test_x(:,i)-weights).^2, 1))';
    [~, win_idx] = min(distance, [], 'all', 'linear');
    [col, row] = ind2sub(size(distance), win_idx);
    TePred(i, 1) = marked_neuron(row, col);
end

TeAcc = sum(TePred == TeLabel)/size(test_x, 2);
fprintf('Classification accuracy on test set: %.4f\n', TeAcc);
```