



*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)

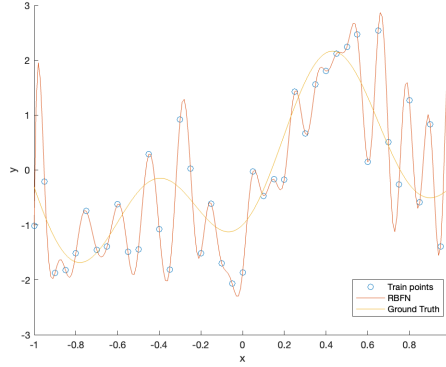


Figure 1: (a) The fitting result of RBFN by using training centers

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

(b)

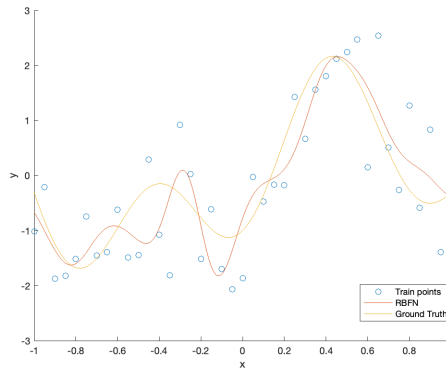


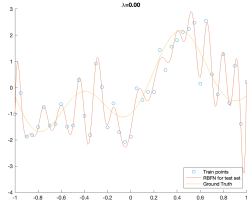
Figure 2: (b) The fitting result of RBFN by using 15 random centers

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.

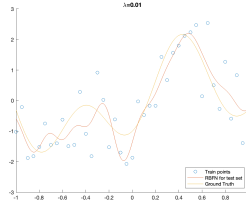
(c)  $\lambda$  is set to the values [0, 0.01, 0.1, 1, 10, 100]. The mini square error is used to evaluate the performance of the regression. The result is shown in the following table and figure.

Table 1: Fitting performance with different regularization

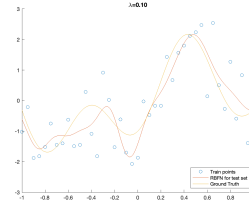
$\lambda$ MSE	0	0.01	0.1	1	10	100
train set	0.0389	0.4442	0.4583	0.4792	0.6491	1.3993
test set	0.6910	0.1572	0.1410	0.1136	0.2023	0.8647



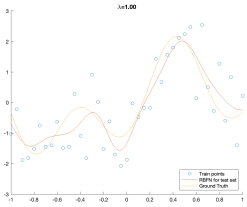
(a)  $\lambda = 0$



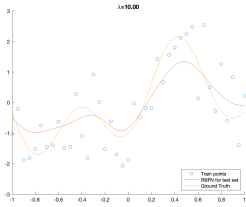
(b)  $\lambda = 0.01$



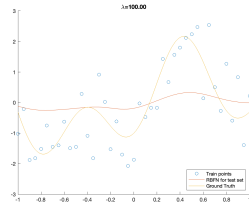
(c)  $\lambda = 0.1$



(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)

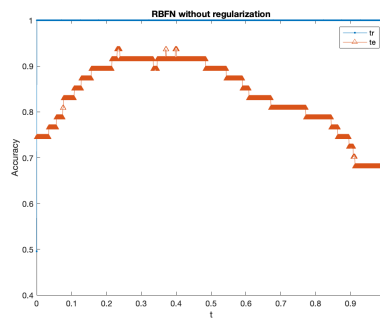


Figure 4: RBFN without regularization

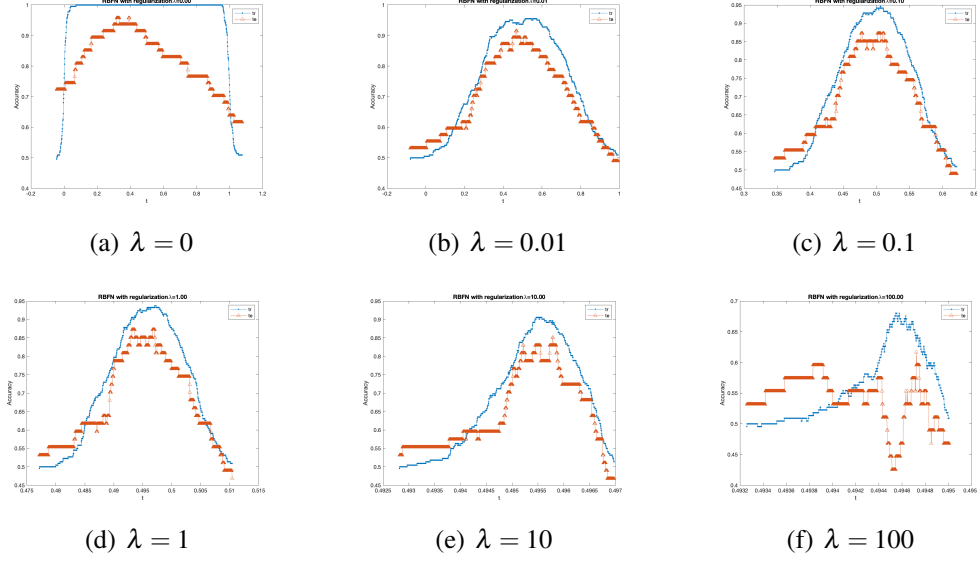


Figure 5: RBFN with different  $\lambda$  regularization

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,

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

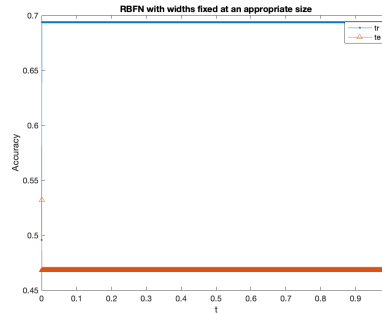


Figure 6: RBFN with widths fixed at an appropriate size

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  $\sigma$ .

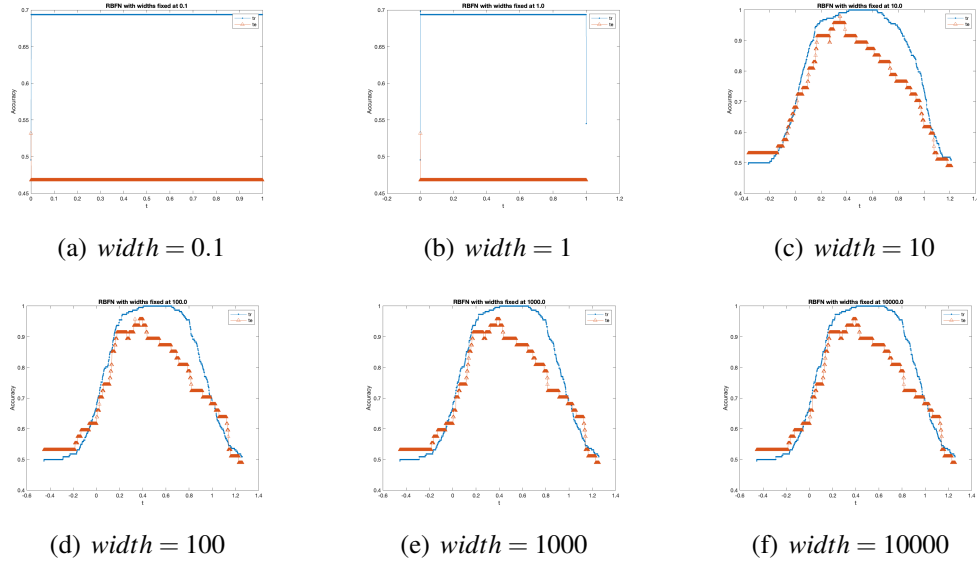


Figure 7: RBFN with different width

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)

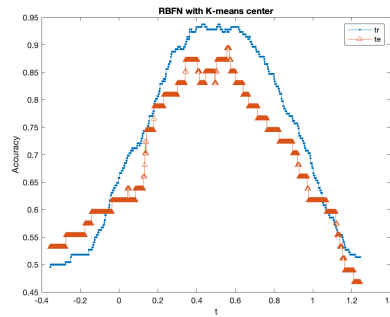


Figure 8: RBFN with 2 centers from K-means and widths = 10

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

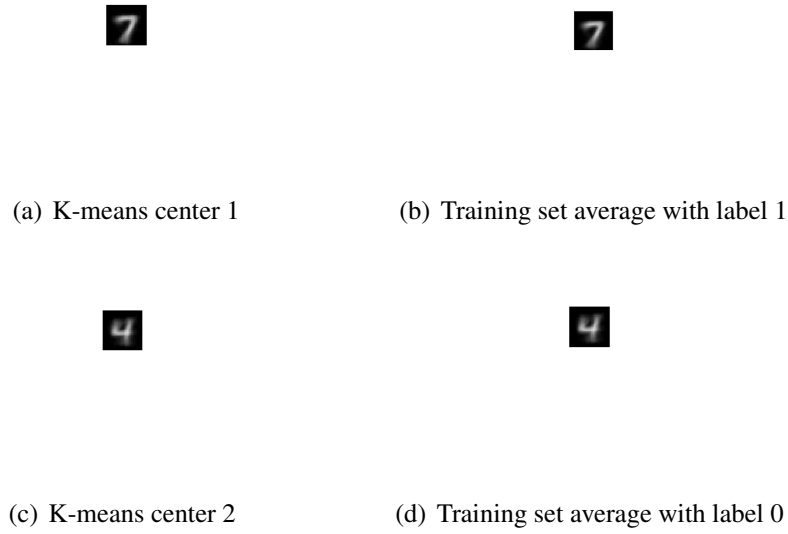


Figure 9: Visualization of centers

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)

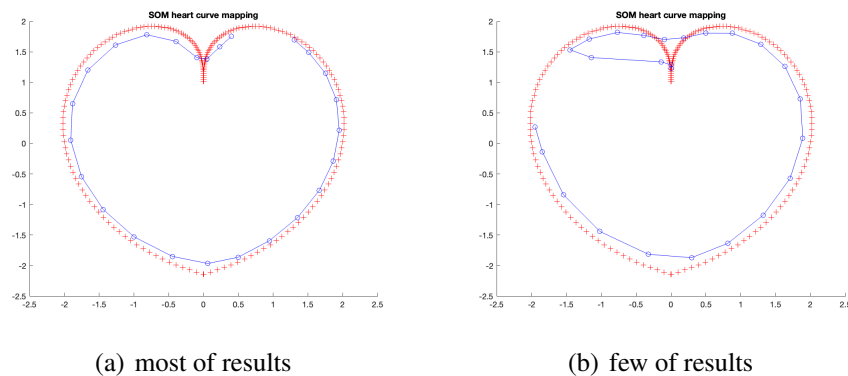
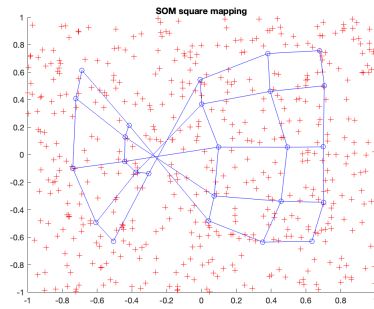


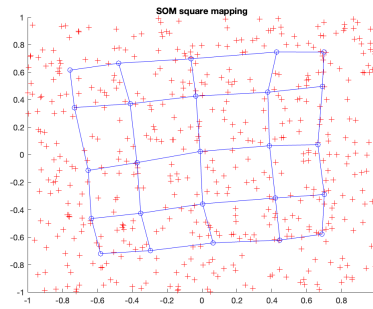
Figure 10: SOM for heart curve mapping

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

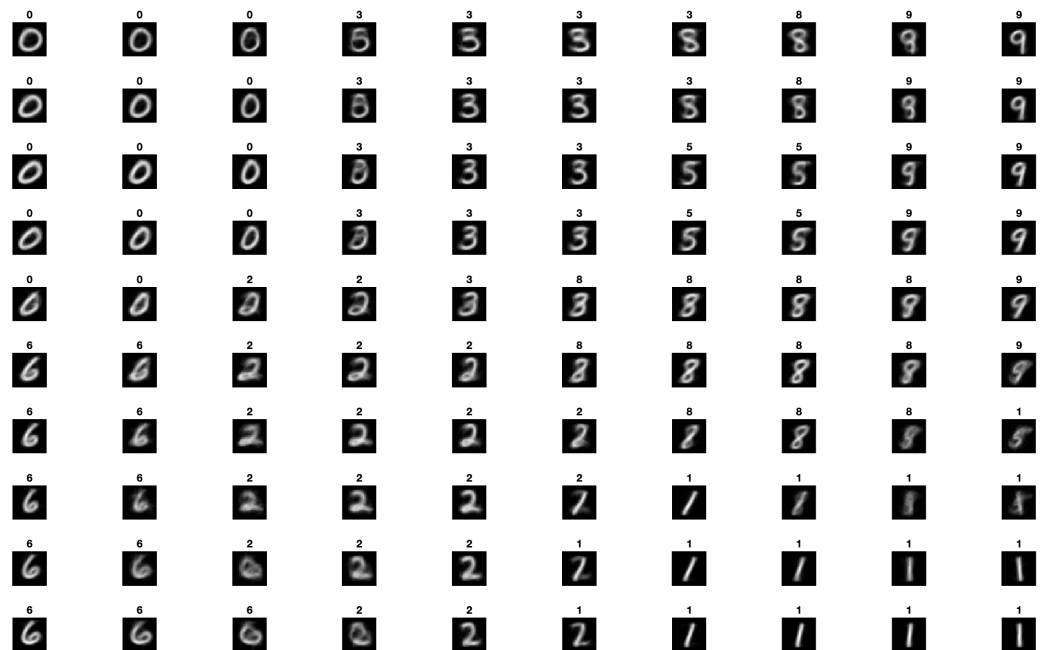


Figure 12: the semantic map of the trained SOM

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

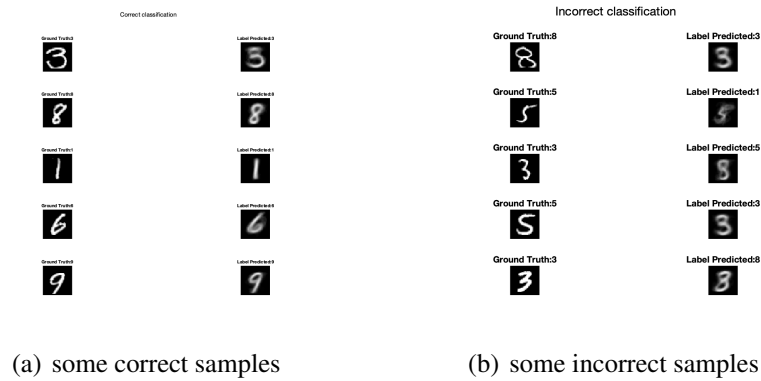


Figure 13: SOM classification visualization

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

## Appendix

```

1 % EE5904 Neural Network
2 % Assignment 3
3
4 %% Question 1
5
6 % (a)
7 clc
8 clear all
9 close all
10 % initialization data set
11 rng(5904); % random seed
12 train_x = -1:0.05:1;
13 train_y = ...
    1.2*sin(pi*train_x)-cos(2.4*pi*train_x)+randn(1,length(train_x));
14 test_x = -1:0.01:1;
15 test_y = 1.2*sin(pi*test_x)-cos(2.4*pi*test_x);
16 % RBF matrix
17 r = train_x' - train_x;
18 RBF = exp(-r.^2./(2*0.1^2));
19 w = RBF^-1*train_y';
20 % predict on test
21 r = test_x' - train_x;
22 RBF = exp(-r.^2./(2*0.1^2));
23 pred_test_y = (RBF*w)';
24 % plot the result
25 figure()

```



```

26 hold on
27 plot(train_x,train_y,'o')
28 plot(test_x,pred_test_y)
29 plot(test_x,test_y)
30 xlabel('x')
31 ylabel('y')
32 legend('Train points','RBFN','Ground Truth')
33 hold off
34
35 % (b)
36 % Choose centers
37 rand_center = datasample(train_x,15);
38 % Training stage
39 r = train_x' - rand_center;
40 RBF = exp(-r.^2./(2*0.1^2));
41 w = pinv(RBF)*train_y';
42 % Test stage
43 r = test_x' - rand_center;
44 RBF = exp(-r.^2./(2*0.1^2));
45 pred_test_y = (RBF*w)';
46 % plot result
47 figure()
48 hold on
49 plot(train_x,train_y,'o')
50 plot(test_x,pred_test_y)
51 plot(test_x,test_y)
52 xlabel('x')
53 ylabel('y')
54 legend('Train points','RBFN','Ground Truth')
55 hold off
56
57 % (c)
58 MSE_train = [];
59 MSE_test = [];
60 count = 1;
61 for lambda = [0,0.01,0.1,1,10,100]
62     % Training
63     r = train_x' - train_x;
64     RBF = exp(-r.^2./(2*0.1^2));
65     w = pinv(RBF'*RBF+lambda*eye(length(RBF)))*RBF'*train_y';
66     % Train set performance
67     pred_train_y = (RBF*w)';
68     MSE_train = [MSE_train,sum((pred_train_y - ...
        train_y).^2)/length(pred_train_y)];
69     % Test set performance
70     r = test_x' - train_x;
71     RBF = exp(-r.^2./(2*0.1^2));
72     pred_test_y = (RBF*w)';
73     MSE_test = [MSE_test,sum((pred_test_y - ...
        test_y).^2)/length(pred_test_y)];
74     % Plot
75     fig = figure();
76     hold on
77     plot(train_x,train_y,'o')
78     % plot(train_x,pred_train_y)
79     plot(test_x,pred_test_y)
80     plot(test_x,test_y)

```

```

81     legend('Train points','RBFN for test set','Ground ...
           Truth','Location','southeast')
82     title(join(['\lambda=',sprintf('%.2f',lambda)]))
83     hold off
84     saveas(fig,sprintf('q1_c%d.png',count))
85     count = count + 1;
86 end
87
88 % best lambda
89 for lambda = 1:10
90     % Training
91     r = train_x' - train_y;
92     RBF = exp(-r.^2./(2*0.1^2));
93     w =pinv(RBF'*RBF+lambda*eye(length(RBF)))*RBF'*train_y';
94     % Train set performance
95     pred_train_y = (RBF*w)';
96     MSE_train = [MSE_train,sum((pred_train_y - ...
           train_y).^2)/length(pred_train_y)];
97     % Test set performance
98     r = test_x' - train_y;
99     RBF = exp(-r.^2./(2*0.1^2));
100    pred_test_y = (RBF*w)';
101    MSE_test = [MSE_test,sum((pred_test_y - ...
           test_y).^2)/length(pred_test_y)];
102    % Plot
103    fig = figure();
104    hold on
105    plot(train_x,train_y,'o')
106    %     plot(train_x,pred_train_y)
107    plot(test_x,pred_test_y)
108    plot(test_x,test_y)
109    legend('Train points','RBFN for test set','Ground ...
           Truth','Location','southeast')
110    title(join(['\lambda=',sprintf('%d',lambda)]))
111    hold off
112    saveas(fig,sprintf('q1_c%d.png',count))
113    count = count + 1;
114 end
115 [min_MSE,min_idx] = min(MSE_test);
116
117 %% Question 2
118 % 7 4 classes
119
120 % (a)
121 clc
122 clear all
123 close all
124 % read the handwritten data
125 load('MNIST_database.mat')
126 column_no = 1;
127 tmp=reshape(train_data(:,column_no),28,28);
128 imshow(double(tmp));
129 close all
130 % find the location of classes 7,4
131 % train data
132 trainIdx = find(train_classlabel==7 | train_classlabel==4);
133 TrLabel = train_classlabel(trainIdx);
134 TrLabel(TrLabel==7) = 1;

```

```

135 TrLabel(TrLabel==4) = 0;
136 Train_Data = train_data(:,trainIdx);
137 % test data
138 testIdx = find(test_classlabel==7 | test_classlabel==4);
139 TeLabel = test_classlabel(testIdx);
140 TeLabel(TeLabel==7) = 1;
141 TeLabel(TeLabel==4) = 0;
142 Test_Data = test_data(:,testIdx);
143 deviation = 100;
144
145 % determine the weights of RBFN without regularization
146 % Training
147 distances = pdist(Train_Data'); % compute pairwise Euclidean ...
    distances
148 r = squareform(distances); % convert the pairwise distances ...
    into a distance matrix
149 RBF = exp(-r.^2./(2*deviation^2));
150 w =inv(RBF)*TrLabel';
151 TrPred = (RBF*w)';
152 % Prediction
153 r = dist(Test_Data',Train_Data);
154 RBF = exp(-r.^2./(2*deviation^2));
155 TePred = (RBF*w)';
156 % Plot
157 figure();
158 TrAcc = zeros(1,1000);
159 TeAcc = zeros(1,1000);
160 thr = zeros(1,1000);
161 TrN = length(TrLabel);
162 TeN = length(TeLabel);
163 for i = 1:1000
164     t = (max(TrPred)-min(TrPred)) * (i-1)/1000 + min(TrPred);
165     thr(i) = t;
166     TrAcc(i) = (sum(TrLabel(TrPred<t)==0) + ...
        sum(TrLabel(TrPred>=t)==1)) / TrN;
167     TeAcc(i) = (sum(TeLabel(TePred<t)==0) + ...
        sum(TeLabel(TePred>=t)==1)) / TeN;
168 end
169 plot(thr,TrAcc,'.- ',thr,TeAcc,'^-');legend('tr','te');
170 title('RBFN without regularization')
171 xlabel('t');
172 ylabel('Accuracy')
173
174 count = 2;
175 for lambda = [0,0.01,0.1,1,10,100]
176     % determine the weights of RBFN without regularization
177     % Training
178     distances = pdist(Train_Data');
179     r = squareform(distances);
180     RBF = exp(-r.^2./(2*deviation^2));
181     w =pinv(RBF'*RBF+lambda*eye(size(RBF,2)))*RBF'*TrLabel';
182     TrPred = (RBF*w)';
183     % Prediction
184     r = dist(Test_Data',Train_Data);
185     RBF = exp(-r.^2./(2*deviation^2));
186     TePred = (RBF*w)';
187     % Plot
188     fig = figure();

```

```

189     TrAcc = zeros(1,1000);
190     TeAcc = zeros(1,1000);
191     thr = zeros(1,1000);
192     TrN = length(TrLabel);
193     TeN = length(TeLabel);
194     for i = 1:1000
195         t = (max(TrPred)-min(TrPred)) * (i-1)/1000 + min(TrPred);
196         thr(i) = t;
197         TrAcc(i) = (sum(TrLabel(TrPred<t)==0) + ...
198                     sum(TrLabel(TrPred>=t)==1)) / TrN;
199         TeAcc(i) = (sum(TeLabel(TePred<t)==0) + ...
200                     sum(TeLabel(TePred>=t)==1)) / TeN;
201     end
202     plot(thr,TrAcc,'.- ',thr,TeAcc,'^');legend('tr','te');
203     title(join(['RBFN with ...
204                 regularization',sprintf('\\lambda=%.2f',lambda)]))
205     xlabel('t');
206     ylabel('Accuracy')
207     % saveas(fig,sprintf('q2_a%d.png',count))
208     count = count + 1;
209 end
210
211 % (b)
212 clc
213 close all
214 % Choose centers
215 rng(5904);
216 rand_center = datasample(Train_Data,100,2);
217 % Training
218 r = dist(Train_Data',rand_center);
219 deviation = sqrt(max(r,[],'all'))/sqrt(2*size(rand_center,2));
220 RBF = exp(-r.^2./(2*deviation^2));
221 w = pinv(RBF)*TrLabel';
222 TrPred = (RBF*w)';
223 % Prediction
224 r = dist(Test_Data',rand_center);
225 deviation = sqrt(max(r,[],'all'))/sqrt(2*size(rand_center,2));
226 RBF = exp(-r.^2./(2*deviation^2));
227 TePred = (RBF*w)';
228 % Plot
229 figure();
230 TrAcc = zeros(1,1000);
231 TeAcc = zeros(1,1000);
232 thr = zeros(1,1000);
233 TrN = length(TrLabel);
234 TeN = length(TeLabel);
235 for i = 1:1000
236     t = (max(TrPred)-min(TrPred)) * (i-1)/1000 + min(TrPred);
237     thr(i) = t;
238     TrAcc(i) = (sum(TrLabel(TrPred<t)==0) + ...
239                 sum(TrLabel(TrPred>=t)==1)) / TrN;
240     TeAcc(i) = (sum(TeLabel(TePred<t)==0) + ...
241                 sum(TeLabel(TePred>=t)==1)) / TeN;
242 end
243 plot(thr,TrAcc,'.- ',thr,TeAcc,'^');legend('tr','te');
244 title('RBFN with widths fixed at an appropriate size')
245 xlabel('t');
246 ylabel('Accuracy')
247

```

```

242 count = 2;
243 for deviation = [0.1,1,10,100,1000,10000]
244     % Training
245     r = dist(Train_Data',rand_center);
246     RBF = exp(-r.^2./(2*deviation^2));
247     w =pinv(RBF)*TrLabel';
248     TrPred = (RBF*w)';
249     % Prediction
250     r = dist(Test_Data',rand_center);
251     RBF = exp(-r.^2./(2*deviation^2));
252     TePred = (RBF*w)';
253     % Plot
254     fig = figure();
255     TrAcc = zeros(1,1000);
256     TeAcc = zeros(1,1000);
257     thr = zeros(1,1000);
258     TrN = length(TrLabel);
259     TeN = length(TeLabel);
260     for i = 1:1000
261         t = (max(TrPred)-min(TrPred)) * (i-1)/1000 + min(TrPred);
262         thr(i) = t;
263         TrAcc(i) = (sum(TrLabel(TrPred<t)==0) + ...
264             sum(TrLabel(TrPred>=t)==1)) / TrN;
265         TeAcc(i) = (sum(TeLabel(TePred<t)==0) + ...
266             sum(TeLabel(TePred>=t)==1)) / TeN;
267     end
268     plot(thr,TrAcc,'.- ',thr,TeAcc,'^-' );legend('tr','te');
269     title(join(['RBFN with widths fixed at ...
270         ',sprintf('%.1f',deviation)]));
271     xlabel('t');
272     ylabel('Accuracy')
273     % saveas(fig,sprintf('q2_b%d.png',count))
274     count = count + 1;
275 end
276
277 % (c)
278 clc
279 close all
280 % Choose centers using K means
281 k = 2;
282 [~, kcenters] = kmeans(Train_Data', k);
283 kcenters = kcenters';
284 % Training
285 r = dist(Train_Data',kcenters);
286 deviation = 10; % from q2_b
287 RBF = exp(-r.^2./(2*deviation^2));
288 w =pinv(RBF)*TrLabel';
289 TrPred = (RBF*w)';
290 % Prediction
291 r = dist(Test_Data',kcenters);
292 RBF = exp(-r.^2./(2*deviation^2));
293 TePred = (RBF*w)';
294 % Plot
295 figure();
296 TrAcc = zeros(1,1000);
297 TeAcc = zeros(1,1000);
298 thr = zeros(1,1000);

```

```

297 TrN = length(TrLabel);
298 TeN = length(TeLabel);
299 for i = 1:1000
300     t = (max(TrPred)-min(TrPred)) * (i-1)/1000 + min(TrPred);
301     thr(i) = t;
302     TrAcc(i) = (sum(TrLabel(TrPred<t)==0) + ...
                 sum(TrLabel(TrPred>=t)==1)) / TrN;
303     TeAcc(i) = (sum(TeLabel(TePred<t)==0) + ...
                 sum(TeLabel(TePred>=t)==1)) / TeN;
304 end
305 plot(thr,TrAcc,'.- ',thr,TeAcc,'^-' );legend('tr','te');
306 title('RBFN with K-means center')
307 xlabel('t');
308 ylabel('Accuracy')
309 % visualize center
310 figure();
311 title('K-means Centers 1')
312 imshow(reshape(kcenters(:,1),[28,28]));
313 figure();
314 title('K-means Centers 2')
315 imshow(reshape(kcenters(:,2),[28,28]));
316 % visualize training set average
317 figure();
318 title('Training set average 1')
319 imshow(reshape(mean(Train_Data(:,TrLabel==1),2),[28,28]));
320 figure();
321 title('Training set average 2')
322 imshow(reshape(mean(Train_Data(:,TrLabel==0),2),[28,28]));
323
324 %% Question 3
325
326 % (a)
327 clc
328 clear all
329 close all
330 rng(5904);
331 % Train data
332 t = linspace(-pi,pi,200);
333 trainX = [t.*sin(pi*sin(t)./t); 1-abs(t).*cos(pi*sin(t)./t)]; ...
           % 2x200 matrix, column-wise points
334 train_fig = figure();
335 plot(trainX(1,:),trainX(2,:),'+r');
336 close(train_fig);
337
338 % SOM
339 % initialization network
340 M = 1;
341 N = 25;
342 neurons = rand(2,N);
343 sigma0 = sqrt(M^2+N^2)/2;
344 iteration = 500;
345 eta0 = 0.1;
346 tau = iteration/log(sigma0); % I guess log in HW3 is ln
347 d0 = 1:N;
348 for epoch = 1:iteration
349     etan = eta0*exp(-epoch/iteration);
350     sigma = sigma0*exp(-epoch/tau);
351     for i = 1:size(trainX,2)

```

```

352 %         distance = sum(dist(trainX(:,i),neurons),1);
353         distance = sum((trainX(:,i) - neurons).^2,1);
354         [~,winner] = min(distance,[],2);
355         d = abs(d0-winner);
356         h = exp(-d.^2/(2*sigma^2));
357         % Update
358         neurons = neurons + etan*h.*(trainX(:,i) - neurons);
359     end
360 end
361
362 % plot the SOM result
363 figure();
364 hold on
365 plot(trainX(1,:),trainX(2,:),'+r');
366 plot(neurons(1,:),neurons(2,:),'o-b');
367 title("SOM heart curve mapping")
368 hold off
369
370 % (b)
371 clc
372 close all
373 trainX = rands(2,500); % 2x500 matrix, column-wise points
374 % SOM
375 % initialization network
376 rng(9045);
377 M = 5;
378 N = 5;
379 neurons = rand(2,M,N);
380 sigma0 = sqrt(M^2+N^2)/2;
381 iteration = 500;
382 eta0 = 0.1;
383 tau = iteration/log(sigma0); % I guess log in HW3 is ln
384 d0 = 1:N;
385 for epoch = 1:iteration
386     etan = eta0*exp(-epoch/iteration);
387     sigma = sigma0*exp(-epoch/tau);
388     for i = 1:size(trainX,2)
389         distance = squeeze(sum((trainX(:,i) - neurons).^2,1))';
390         [~,winner] = min(distance,[],'all','linear');
391         k = ceil(winner/5);
392         n = winner - (k-1)*5;
393         d_j = (d0 - n).^2;
394         d_i = (d0 - k).^2;
395         d_square = d_j' + d_i;
396         h = exp(-d_square.^2/(2*sigma^2));
397         h = permute(repmat(h,[1,1,2]),[3 2 1]);
398         % Update
399         neurons = neurons + etan*h.*(trainX(:,i) - neurons);
400     end
401 end
402
403 % plot the SOM result
404 figure();
405 hold on
406 plot(trainX(1,:),trainX(2,:),'+r');
407 for i = 1:5
408     for j = 1:5
409         % left and right neighbors

```

```

410         if i+1 ≤ 5
411             plot([neurons(1,i,j),neurons(1,i+1,j)], [neurons(2,i,j),neurons(2,i+1,j)
412         end
413         % top and bottom neighbors
414         if j+1 ≤ 5
415             plot([neurons(1,i,j),neurons(1,i,j+1)], [neurons(2,i,j),neurons(2,i,j+1)
416         end
417     end
418 end
419 title("SOM square mapping")
420 hold off
421
422 %% Question 3 c
423 % 7 4 classes
424
425 %(c)-1
426 clc
427 clear all
428 close all
429 rng(5904);
430 % read the handwritten data
431 load('MNIST_database.mat')
432 % find the location of classes 7,4
433 % train data
434 trainIdx = find(train_classlabel≠7 & train_classlabel≠4);
435 TrLabel = train_classlabel(trainIdx);
436 Train_Data = train_data(:,trainIdx);
437 % test data
438 testIdx = find(test_classlabel≠7 & test_classlabel≠4);
439 TeLabel = test_classlabel(testIdx);
440 Test_Data = test_data(:,testIdx);
441 % input data
442 Data = cat(2,Train_Data,Test_Data);
443 Label = cat(2,TrLabel,TeLabel);
444
445 % SOM
446 % initialization network
447 M = 10;
448 N = 10;
449 neurons = rand(size(Data,1),M,N);
450 sigma0 = sqrt(M^2+N^2)/2;
451 iteration = 1000;
452 eta0 = 0.1;
453 tau = iteration/log(sigma0); % I guess log in HW3 is ln
454 d0 = 1:N;
455 for epoch = 1:iteration
456     etan = eta0*exp(-epoch/iteration);
457     sigma = sigma0*exp(-epoch/tau);
458     for i = 1:size(Data,2)
459         distance = squeeze(sum((Data(:,i) - neurons).^2,1))';
460         [~,winner] = min(distance,[], 'all', 'linear');
461         k = ceil(winner/N);
462         n = winner - (k-1)*N;
463         d_j = (d0 - n).^2;
464         d_i = (d0 - k).^2;
465         d_square = d_j' + d_i;
466         h = exp(-d_square.^2/(2*sigma^2));
467         h = permute(repmat(h,[1,1,size(Data,1)]),[3 2 1]);

```



```

468         % Update
469         neurons = neurons + etan*h.*(Data(:,i) - neurons);
470     end
471 end
472
473 % map label using nearest neighbour
474 reshaped_neurons = reshape(neurons, [size(Data,1),1,100]);
475 true_label = [0,1,2,3,5,6,8,9];
476 for i = 1:8
477     label = true_label(i);
478     truth(i,:) = mean(Train_Data(:,TrLabel==label),2);
479 end
480 for i = 1:100
481     [idx, ~] = knnsearch(truth,reshaped_neurons(:,1,i)', 'K', 1);
482     neuron_label(i) = true_label(idx);
483 end
484
485 % plot SOM
486 fig = figure;
487 fig.WindowState = 'maximized' ;
488 title('semantic map')
489 for A = 1:100
490     subplot(10,10,A)
491     graph = reshape(reshaped_neurons(:,1,A), [28,28]);
492     imshow(graph);
493     title(sprintf('%d', neuron_label(A)))
494 end
495
496 %%
497 %(c)-1
498 clc
499 clear all
500 close all
501 rng(5904);
502 % read the handwritten data
503 load('MNIST_database.mat')
504 % find the location of classes 7,4
505 % train data
506 trainIdx = find(train_classlabel~=7 & train_classlabel~=4);
507 TrLabel = train_classlabel(trainIdx);
508 Train_Data = train_data(:,trainIdx);
509 % test data
510 testIdx = find(test_classlabel~=7 & test_classlabel~=4);
511 TeLabel = test_classlabel(testIdx);
512 Test_Data = test_data(:,testIdx);
513
514 % SOM
515 % initialization network
516 M = 10;
517 N = 10;
518 neurons = rand(size(Train_Data,1),M,N);
519 sigma0 = sqrt(M^2+N^2)/2;
520 iteration = 1000;
521 eta0 = 0.1;
522 tau = iteration/log(sigma0); % I guess log in HW3 is ln
523 d0 = 1:N;
524 for epoch = 1:iteration
525     etan = eta0*exp(-epoch/iteration);

```

```

526     sigma = sigma0*exp(-epoch/tau);
527     for i = 1:size(Train_Data,2)
528         distance = squeeze(sum((Train_Data(:,i) - ...
529             neurons).^2,1))';
529         [~,winner] = min(distance,[], 'all', 'linear');
530         k = ceil(winner/N);
531         n = winner - (k-1)*N;
532         d_j = (d0 - n).^2;
533         d_i = (d0 - k).^2;
534         d_square = d_j' + d_i;
535         h = exp(-d_square.^2/(2*sigma^2));
536         h = permute(repmat(h,[1,1,size(Train_Data,1)]),[3 2 1]);
537         % Update
538         neurons = neurons + etan*h.*(Train_Data(:,i) - neurons);
539     end
540 end
541
542 % map label using nearest neighbour
543 reshaped_neurons = reshape(neurons,[size(Train_Data,1),1,100]);
544 true_label = [0,1,2,3,5,6,8,9];
545 for i = 1:8
546     label = true_label(i);
547     truth(i,:) = mean(Train_Data(:,TrLabel==label),2);
548 end
549 for i = 1:100
550     [idx, ~] = knnsearch(truth,reshaped_neurons(:,1,i)', 'K', 1);
551     neuron_label(i) = true_label(idx);
552 end
553
554 % Test
555 % TePred
556 TePred = zeros(size(TeLabel));
557 counter_1 = 1;
558 counter_2 = 1;
559 for i = 1:size(Test_Data,2)
560     distance = squeeze(sum((Test_Data(:,i) - neurons).^2,1));
561     [~,winner] = min(distance,[], 'all', 'linear');
562     TePred(i) = neuron_label(winner);
563     % plot some correct samples
564     if TePred(i)==TeLabel(i) && counter_1 ≤ 5
565         figure(1)
566         sgttitle('Correct classification')
567         subplot(5,2,(counter_1-1)*2+1)
568         imshow(reshape(Test_Data(:,i),28,28))
569         title(sprintf('Ground Truth:%d',TeLabel(1,i)))
570         subplot(5,2,(counter_1-1)*2+2)
571         imshow(reshape(reshaped_neurons(:,winner),28,28))
572         title(sprintf('Label Predicted:%d',TePred(1,i)))
573         counter_1 = counter_1 + 1;
574     % plot some incorrect samples
575     elseif TePred(1,i)≠TeLabel(1,i) && counter_2 ≤ 5
576         figure(2)
577         sgttitle('Incorrect classification')
578         subplot(5,2,(counter_2-1)*2+1)
579         imshow(reshape(Test_Data(:,i),28,28))
580         title(sprintf('Ground Truth:%d',TeLabel(1,i)))
581         subplot(5,2,(counter_2-1)*2+2)
582         imshow(reshape(reshaped_neurons(:,winner),28,28))

```

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