



National University of Singapore
School of Electrical and Computer Engineering

Homework 2

Student:
Wanry Lin

Matriculation Number:
A0000001X

EE5904 Neural Network

May 10, 2023

Question 1

- (a) According to Steepest (Gradient) descent method, we can derive the update function as following:

$$x_{k+1} = x_k - \eta [2(x-1) + 400x(x^2 - y)]$$

$$y_{k+1} = y_k - \eta [200(y - x^2)]$$

so that we can obtain the update function and run the method with $\eta = 0.001$ to find the global minimum.

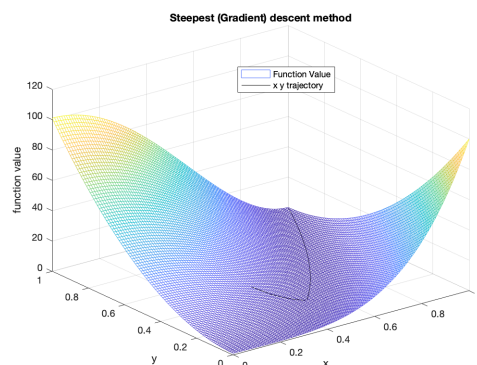


Figure 1: the function value and (x , y) trajectory

It runs 10928 times iteration before achieving the minimum. However, if the $\eta = 0.2$ it can't converge, which means the global minimum is not able to achieve. The iteration runs 10^6 times and then break.

- (b) According to Newton's method, we can derive the update function as following:

$$g(n) = \frac{\partial E(w)}{\partial w} = \begin{bmatrix} \frac{\partial E}{\partial x} \\ \frac{\partial E}{\partial y} \end{bmatrix} = \begin{bmatrix} 2(x-1) + 400(x^3 - xy) \\ 200(y - x^2) \end{bmatrix}$$

$$H(n) = \frac{\partial^2 E(w)}{\partial w^2} = \begin{bmatrix} \frac{\partial^2 E}{\partial x \partial x} & \frac{\partial^2 E}{\partial x \partial y} \\ \frac{\partial^2 E}{\partial y \partial x} & \frac{\partial^2 E}{\partial y \partial y} \end{bmatrix} = \begin{bmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

Then we can obtain the global minimum

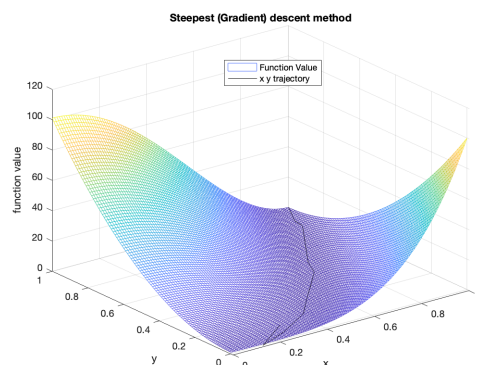
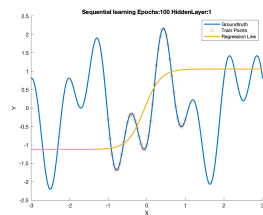


Figure 2: the function value and (x , y) trajectory

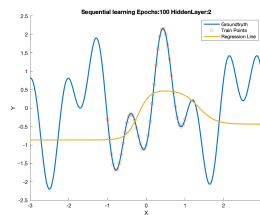
It runs just 11 times iteration before achieving the minimum, which shows its efficiency compared to Steepest (Gradient) descent method.

Question 2

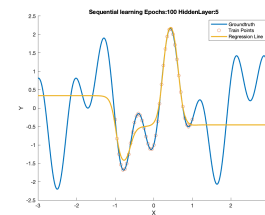
- (a) Use the sequential mode with BP algorithm and experiment with the following different hidden layer of the MLP: 1-n-1, where $n = 1, 2, 5, 10, 20, 50$. The training function used is trainlm. The number of epochs is 100. The results are shown as here.



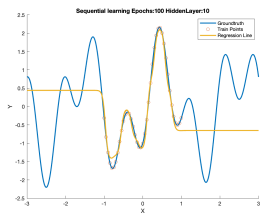
(a) 1 neuron hidden layer



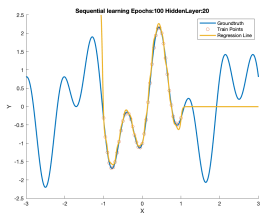
(b) 2 neuron hidden layer



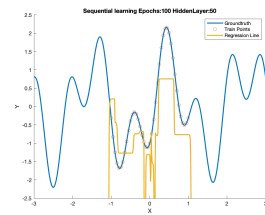
(c) 5 neuron hidden layer



(d) 10 neuron hidden layer



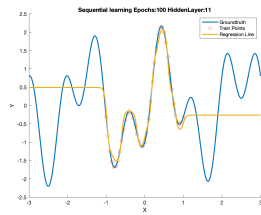
(e) 20 neuron hidden layer



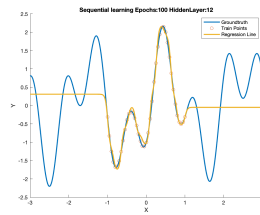
(f) 50 neuron hidden layer

Figure 3: 1-n-1 MLP

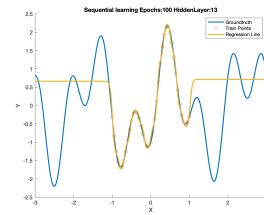
It is obvious that when $n = 1, 2, 3, 10$, the neural network is under-fitting. When $n = 20$, the neural network is proper fitting. When $n = 50$, the neural network is over-fitting. Because there is too many neurons in the hidden layer that is too sensitive to some points while ignore others. Meanwhile, it doesn't have ability to predict the values correctly out of $[-1, 1]$ which means it can't predict the value at $x = -3$ and $x = 3$.



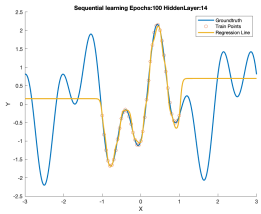
(a) 11 neuron hidden layer



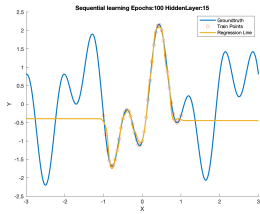
(b) 12 neuron hidden layer



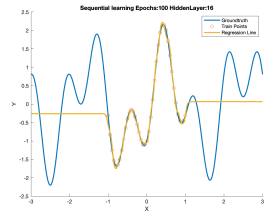
(c) 13 neuron hidden layer



(d) 14 neuron hidden layer



(e) 15 neuron hidden layer



(f) 16 neuron hidden layer

Figure 4: 1-n-1 MLP

It shows that the minimum of neuron is 13. Though the minimum of neuron should be 6, but when hidden neuron number over 13, the performance is perfect.

(b) Set train function as "trainlm", the result is shown here:

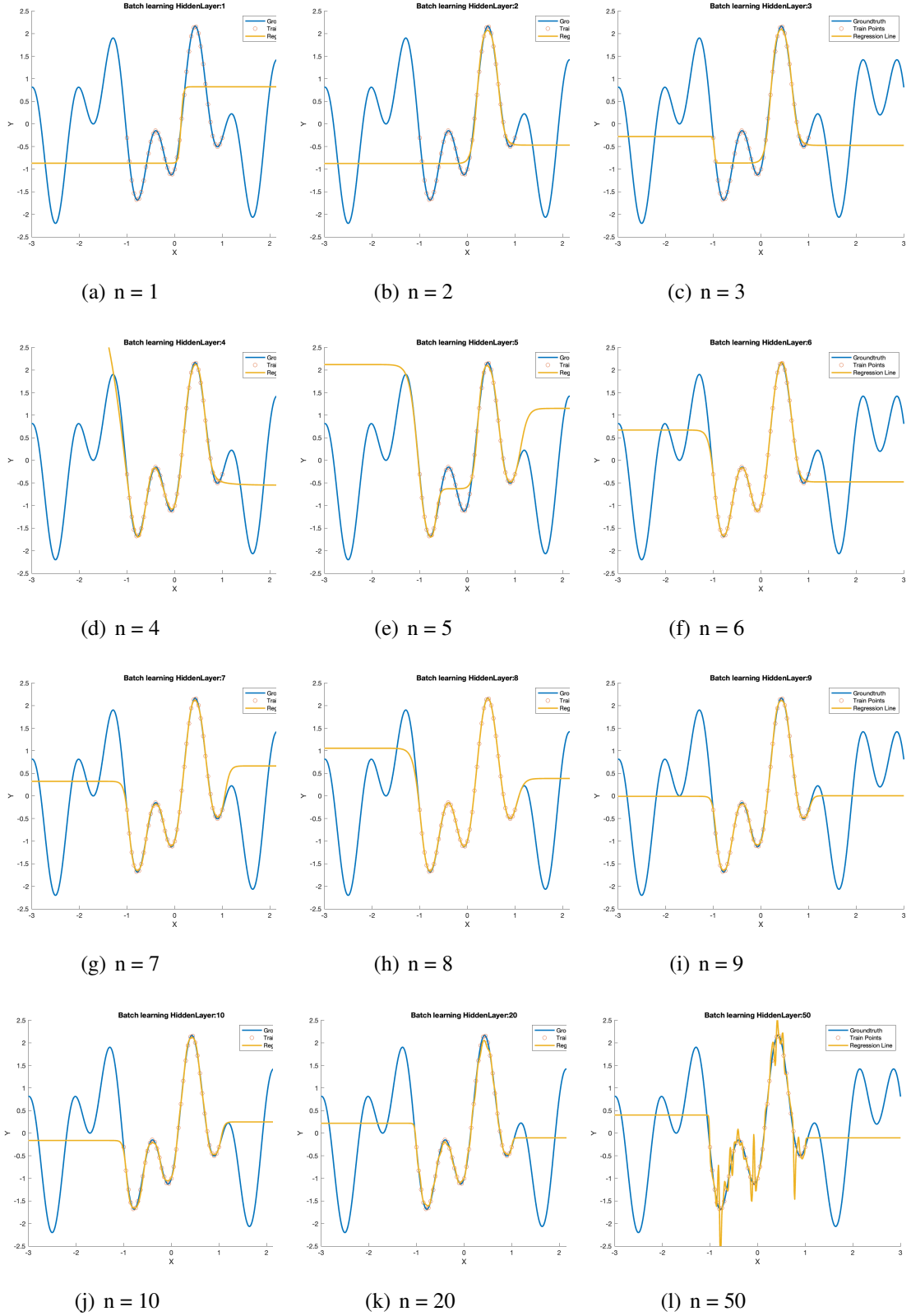


Figure 5: 1-n-1 MLP

It is obvious that when $n = 1, 2, 3, 4, 5$, the neural network is under-fitting. When $n = 6, 7, 8, 9, 10, 20$, the neural network is proper fitting. When $n = 50$, the neural network is over-fitting. Meanwhile, it doesn't have ability to predict the values correctly out

of $[-1,1]$ which means it can't predict the value at $x = -3$ and $x = 3$. **It can achieve proper fit at $n = 6$, which needs less hidden neuron than sequential learning.**

(c) Set train function as "trainbr", the result is shown here:

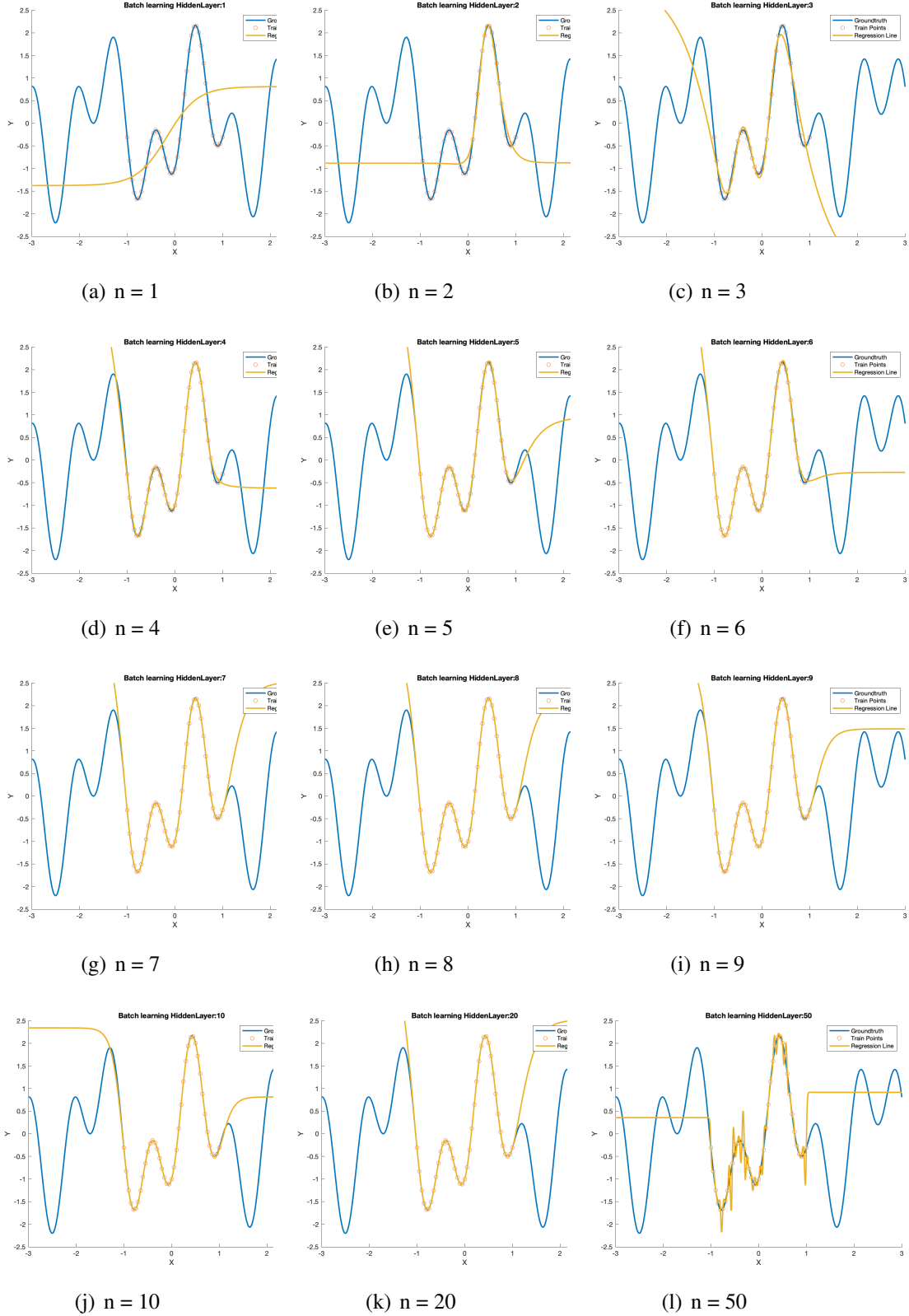


Figure 6: 1-n-1 MLP

It is obvious that when $n = 1,2,3,4,5,6$, the neural network is under-fitting. When $n = 7,8,9,10,20$, the neural network is proper fitting. When $n = 50$, the neural network is over-fitting. Meanwhile, it doesn't have ability to predict the values correctly out of $[-1,1]$ which means it can't predict the value at $x = -3$ and $x = 3$. **It can achieve proper fit at $n = 7$, which needs 1 more hidden neuron than "trainlm".**

Question 3

- (a) First read the image data and label before storing them in mat file for fast read. The performance figure is shown here:

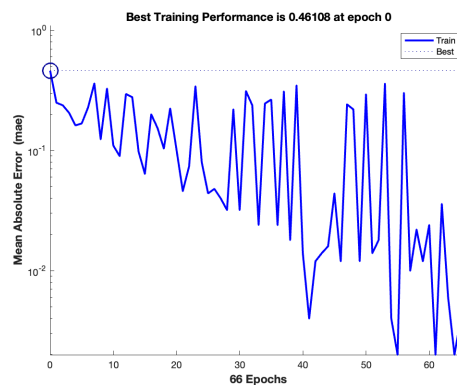


Figure 7: the performance of Rosenblatts perceptron

The accuracy of training set is 100% while test set is 68.86%.

- (b) The accuracy of naively downsampling image is shown here

Image size	256x256	128x128	64x64	32x32	PCA	sobel + 16x16
Train accuracy	100%	100%	100%	100%	100%	100%
Test accuracy	68.86%	68.26%	67.07%	65.87%	64.67%	68.26%

I use PCA to extract the feature of the image in 2 direction, row and column. The first 5 dimension of each direction is captured and reshaped as input of the perceptron. As the more image is compressed, the accuracy is lower.

The performance of each dimension reduction method is shown here:

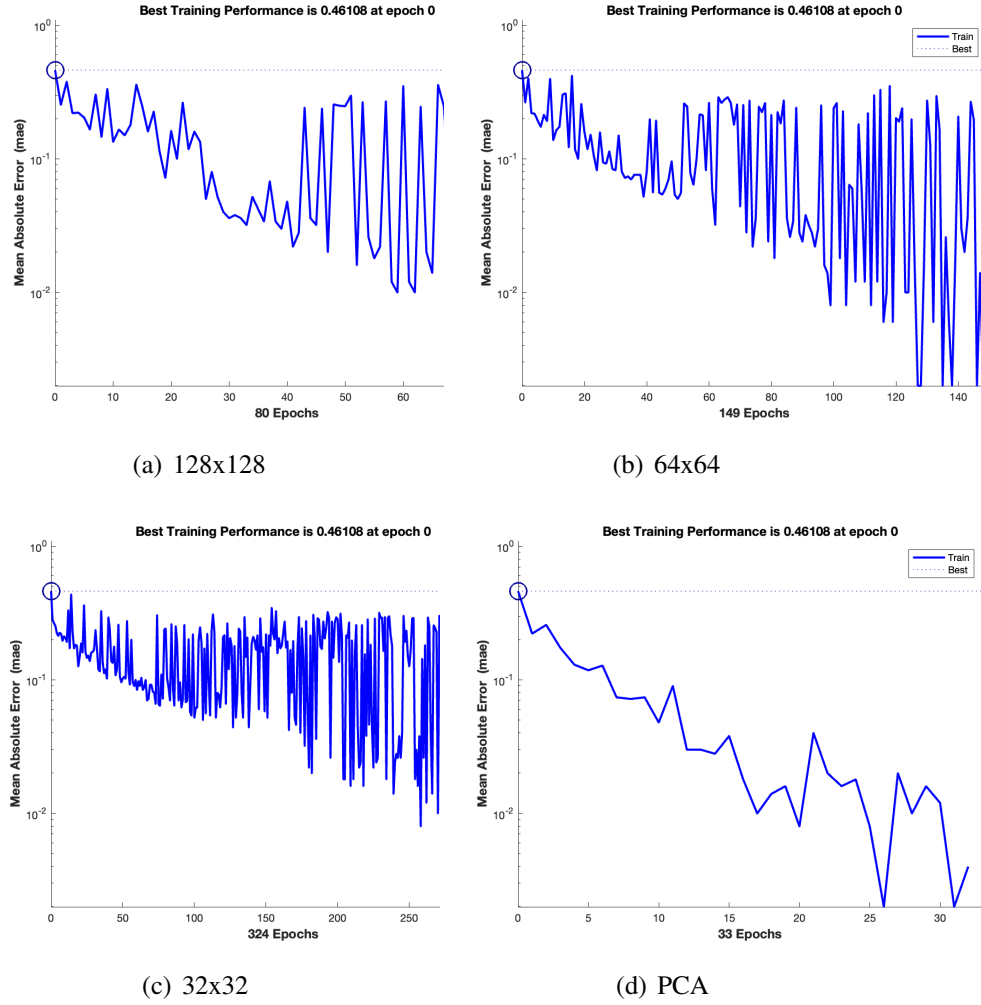


Figure 8: dimension reduction

- (c) In this part, to reduce the size of input data while keep the feature of the image. I use sobel kernel with 3x3 size to extract the edge of the image. Then, reduce the size of the image into 16x16. The hidden neuron number of MLP, therefore, is set to 256. The performance is shown here:

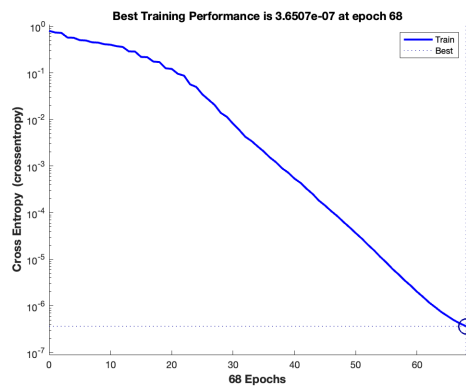


Figure 9: the performance of 1-196-1 MLP

The accuracy of training stage is 100% while for test is 73.08%.

- (d) It is overfitting. Because it achieves 100% in training set while much lower accuracy in test set, which means it learns the noise in training set as well.

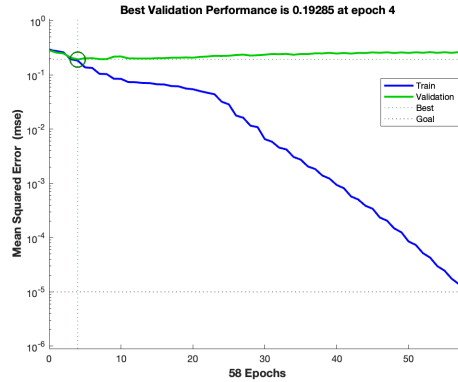


Figure 10: the performance of 1-196-1 MLP

It shows that it goes overfitting after epoch 4. I try to set the regularization at 0,0.25,0.5,0.75.

Regularization setting	Training accuracy	Test accuracy
0	67.19%	61.91%
0.25	78.36%	66.38%
0.5	72.38%	63.44%
0.75	92.00%	69.31%

Regularization truly prevents the network approaching overfitting but it doesn't help enhance the accuracy.

- (e) The accuracy of sequential learning is shown here:

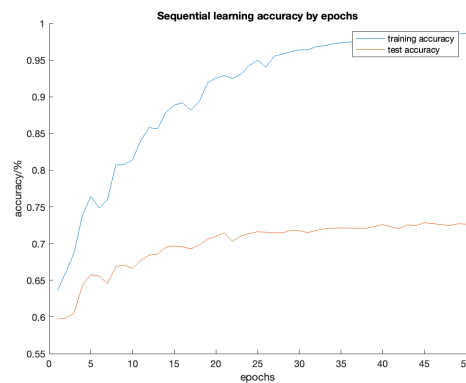


Figure 11: the performance of sequential learning

Finally, the training accuracy is 100% while the highest test accuracy is 72.84% at epoch 45. The performance of sequential learning is no better than batch mode learning while cost more expense in training. So batch mode training is better in my opinion.

(f) (1) In Question 3 (c), I use sobel kernel to extract the edge in the image and then reduce the size of image to 16x16, which achieves the accuracy about 73%. However, I try not using any dimension reduction method and input the original 256x256 image to the same MLP network, which achieves the accuracy about 71% at last. Therefore, it shows that the sobel kernel can enhance the strength of the feature in the image when reduce the dimension. This scheme improves my MLP network performance both in accuracy and training speed.

(2) I have tried sobel kernel with 3x3, 5x5,...25x25 size. I found the accuracy is highest, when the target size of image after reduction is 5 times of kernel patch size. For example, if the reduced image size is 16x16, sobel kernel size should be 3x3, since $16 \approx 15 = 3 \times 5$.

(3) I just tried sobel kernel with different size. It is obvious that it is not the most suitable one for this classifier. Maybe, the specified kernel can be generated by the Neural Network itself to divide the coast view and inside city view.

Appendix Matlab code

```
1 % EE5904 Neural Network
2 % Assignment 2
3
4 %% Question 1
5
6 % (a)
7 clc
8 clear all
9 close all
10 % initialization
11 % starting point
12 X = [];
13 Y = [];
14 X(1) = 0.5 * rand(1);
15 Y(1) = 0.5 * rand(1);
16 % learning rate
17 eta = 0.001;
18 % eta = 0.2;
19 % iteration counter
20 i = 0;
21 % cost function
22 syms x predicted_y
23 cost_f = (1-x)^2 + 100*(predicted_y-x^2)^2;
24 % cost trajectory
25 cost = [];
26 % Gradient descent
27 G_x = diff(cost_f,x);
28 G_y = diff(cost_f,predicted_y);
29
30 % update process
31 while true
32     i = i + 1;
33     % record cost
```

```

34     %      cost(i) = double(subs(cost_f,[x,y],[X(i),Y(i)]));
35     cost(i) = (1-X(i))^2 + 100*(Y(i)-X(i)^2)^2;
36     % stop when
37     if cost(i) < 1e-5
38         break
39     elseif i > 1e6
40         error("can not converge")
41     end
42     % update weights
43     %      X(i + 1) = X(i) - ...
44         eta*double(subs(G_x,[x,y],[X(i),Y(i)]));
45     %      Y(i + 1) = Y(i) - ...
46         eta*double(subs(G_y,[x,y],[X(i),Y(i)]));
47     X(i + 1) = X(i) - eta*(2*(X(i) - 1) + 400*X(i)*(X(i)^2 - ...
48         Y(i)));
49     Y(i + 1) = Y(i) - eta*(200*(Y(i) - X(i)^2));
50 end
51 % plot
52 % function value
53 [xaxis,yaxis]=meshgrid(0:0.01:1);
54 function_value = (1-xaxis).^2 + 100*(yaxis-xaxis.^2).^2;
55 mesh(xaxis,yaxis,function_value)
56 % x,y trajectory
57 hold on
58 F_value = (1-X).^2 + 100*(Y-X.^2).^2;
59 plot3(X,Y,F_value,'black')
60 hold off
61 grid on
62 xlabel("x")
63 ylabel("y")
64 zlabel("function value")
65 title("Steepest (Gradient) descent method")
66 legend("Function Value","x y trajectory")
67
68 % (b)
69 clc
70 clear all
71 close all
72 % initialization
73 % starting point
74 X = 0.5 * rand(1);
75 Y = 0.5 * rand(1);
76 % weights value
77 w = [X,Y];
78 % learning rate
79 % eta = 0.001;
80 eta = 0.2;
81 % iteration counter
82 i = 0;
83 % cost function
84 syms x predicted_y
85 cost_f = (1-x)^2 + 100*(predicted_y-x^2)^2;
86 % cost trajectory
87 cost = [];
88 % Gradient descent
89 G_x = diff(cost_f,x);
90 G_y = diff(cost_f,predicted_y);

```

```

89 G = [G_x,G_y];
90 % Hessian matrix
91 H = [diff(cost_f,x,2) diff(cost_f,x,predicted_y) ; ...
      diff(cost_f,predicted_y,x) diff(cost_f,predicted_y,2)];
92
93 % update process
94 while true
95     i = i + 1;
96     % record cost
97     %     cost(i) = double(subs(cost_f,[x,y],[X(i),Y(i)]));
98     X = w(i,1);
99     Y = w(i,2);
100    cost(i) = (1-X)^2 + 100*(Y-X^2)^2;
101    % stop when
102    if cost(i) < 1e-5
103        break
104    elseif i > 1e6
105        error("can not converge")
106    end
107    % update weights
108    %     w(i + 1,:) = w(i,:) - inv(subs(H,{x;y},{w(i,:)})) * ...
      subs(G,{x;y},{w(i,:)});
109    g = [(2*(X - 1) + 400*(X^3 - X*Y)); 200*(Y-X^2)];
110    h = [1200*X^2 - 400*Y + 2,-400*X; -400*X,200];
111    w(i + 1,:) = w(i,:) - (inv(h) * g)';
112 end
113
114 % plot
115 % function value
116 [xaxis,yaxis]=meshgrid(0:0.01:1);
117 function_value = (1-xaxis).^2 + 100*(yaxis-xaxis.^2).^2;
118 mesh(xaxis,yaxis,function_value)
119 % x,y trajectory
120 hold on
121 F_value = (1-w(:,1)).^2 + 100*(w(:,2)-w(:,1).^2).^2;
122 plot3(w(:,1),w(:,2),F_value,'black')
123 hold off
124 grid on
125 xlabel("x")
126 ylabel("y")
127 zlabel("function value")
128 title("Steepest (Gradient) descent method")
129 legend("Function Value","x y trajectory")
130
131 Z0 = 300;
132 Z1 = 54.7360 -26.3014i;
133 k = 2*pi;
134 l = 3/8;
135 Zin = Z0*(Z1 + i*Z0*tan(k*l))/(Z0 + i*Z1*tan(k*l));
136
137 %% Question 2
138
139 % (a)
140 clc
141 clear all
142 close all
143 % hyperparams
144 epochs = 100;

```

```

145 % data
146 train_x = -1:0.05:1;
147 train_y = 1.2*sin(pi*train_x) - cos(2.4*pi*train_x);
148 train_set = [train_x;train_y];
149
150 for i = [11:20]
151
152     % train
153     [net,train_accuracy] = Seq_mlp(i,train_set,epochs);
154     % test
155     test_x = -3:0.01:3;
156     truth_y = 1.2*sin(pi*test_x) - cos(2.4*pi*test_x);
157     predicted_y = net(test_x);
158
159     % plot
160     result = figure();
161     hold on
162     plot(test_x,truth_y,'-','LineWidth',2)
163     scatter(train_x,train_y)
164     plot(test_x,predicted_y,'-','LineWidth',2)
165     ylim([-2.5 2.5])
166     legend('Groundtruth','Train Points','Regression Line')
167     title(sprintf('Sequential learning Epochs:%d ...
        HiddenLayer:%d',epochs,i))
168     ylabel('Y')
169     xlabel('X')
170     hold off
171     % save image
172     saveas(result,sprintf('HiddenLayer%02d.png',i));
173 end
174
175 %% Q2 (b)&(c)
176 clc
177 clear all
178 close all
179
180 % training set
181 train_x = -1:0.05:1;
182 train_y = 1.2*sin(pi*train_x) - cos(2.4*pi*train_x);
183 train_set = [train_x;train_y];
184 % test set
185 test_x = -3:0.01:3;
186 truth_y = 1.2*sin(pi*test_x) - cos(2.4*pi*test_x);
187
188 % build NN with different hidden layer
189 for i = [1:10,20,50]
190     % Create Neuron Network
191     net = fitnet(i);
192
193     % net.trainFcn = 'trainlm';
194     net.trainFcn = 'trainbr';% 'trainlm' 'trainbr'
195     net.divideFcn = 'dividetrain'; % input for training only
196     net.performParam.regularization = 10e-6; % regularization ...
        strength
197
198     % Train the Network
199     [net,-] = train(net,train_x,train_y);
200

```

```

201     % Test
202     predicted_y = sim(net, test_x);
203     result = figure();
204     hold on
205     plot(test_x,truth_y,'-','LineWidth',2)
206     scatter(train_x,train_y)
207     plot(test_x,predicted_y,'-','LineWidth',2)
208     ylim([-2.5 2.5])
209     legend('Groundtruth','Train Points','Regression Line')
210     title(sprintf('Batch learning HiddenLayer:%d',i))
211     ylabel('Y')
212     xlabel('X')
213     hold off
214     % save image
215     saveas(result,sprintf('batchbr%02d.png',i));
216 end
217
218 %% Q3
219 % my matric number is A0260074M so I should in group 3
220 display(mod(74,4)+1);
221 % read and store image data for fast read
222 % set the label of coast is 1 and insidecity is 0
223 train_set = [];
224 test_set = [];
225 data_path = "/Users/wanrylin/Master Courses/EE ...
        5904/group_3/group_3";
226 % train set construction
227 train_path = strcat(data_path,"/train/");
228 namelist = dir(strcat(train_path,'*.jpg'));
229 img_num = length(namelist);
230 for i = 1:img_num
231     imgname = namelist(i).name;
232     tmp = strsplit(imgname, {'_', '.'});
233     label = str2num(tmp{2});
234     img = imread(strcat(train_path, imgname));
235     % every col is a combination of label(row 1) and image
236     vector = [label;img(:)];
237     train_set = [train_set,vector];
238 end
239 save("train_set.mat","train_set");
240
241 % test set construction
242 test_path = strcat(data_path,"/test/");
243 namelist = dir(strcat(test_path,'*.jpg'));
244 img_num = length(namelist);
245 for i = 1:img_num
246     imgname = namelist(i).name;
247     tmp = strsplit(imgname, {'_', '.'});
248     label = str2num(tmp{2});
249     img = imread(strcat(test_path, imgname));
250     vector = [label;img(:)];
251     test_set = [test_set,vector];
252 end
253 save("test_set.mat","test_set");
254
255 %% (a)
256 clc
257 clear all

```

```

258 close all
259
260 % read data
261 train_set = double(load('train_set.mat').train_set);
262 test_set = double(load('test_set.mat').test_set);
263
264 train_img = train_set(2:end,:);
265 train_label = train_set(1,:);
266 test_img = test_set(2:end,:);
267 test_label = test_set(1,:);
268
269 % single layer
270 net = perceptron();
271 net = configure(net,train_img,train_label);
272 % set network parameter
273 net.divideFcn = 'dividetrain';
274 % train
275 [net,tr]=train(net,train_img,train_label);
276
277 % compute accuracy
278 pred_label_train = net(train_img);
279 accu_train = 1 - mean(abs(pred_label_train-train_label));
280 pred_label_test = net(test_img);
281 accu_test = 1 - mean(abs(pred_label_test-test_label));
282 fprintf('accu_train: %.02f%%\n',accu_train*100)
283 fprintf('accu_val: %.02f%%\n',accu_test*100)
284
285 %% (b)
286 clc
287 clear all
288 close all
289
290 % read data
291 train_set = double(load('train_set.mat').train_set);
292 test_set = double(load('test_set.mat').test_set);
293
294 train_img = train_set(2:end,:);
295 train_label = train_set(1,:);
296 test_img = test_set(2:end,:);
297 test_label = test_set(1,:);
298
299 % % downsample image
300 % image_size = [32,32];
301 % len = length(train_label);
302 % train_img_re = [];
303 % for i = 1:len
304 %     image = reshape(train_img(:,i),[256,256]);
305 %     new_image = imresize(image,image_size);
306 %     train_img_re(:,i) = new_image(:);
307 % end
308 % len = length(test_label);
309 % test_img_re = [];
310 % for i = 1:len
311 %     image = reshape(test_img(:,i),[256,256]);
312 %     new_image = imresize(image,image_size);
313 %     test_img_re(:,i) = new_image(:);
314 % end
315

```

```

316 % % PCA
317 % dimension = 1;
318 % len = length(train_label);
319 % train_img_re = [];
320 % for i = 1:len
321 %     image = reshape(train_img(:,i),[256,256]);
322 %     row_feature = pca(image');
323 %     row_feature = row_feature(:,1:dimension);
324 %     col_feature = pca(image);
325 %     col_feature = col_feature(:,1:dimension);
326 %     train_img_re(:,i) = [row_feature(:);col_feature(:)];
327 % end
328 % len = length(test_label);
329 % test_img_re = [];
330 % for i = 1:len
331 %     image = reshape(test_img(:,i),[256,256]);
332 %     row_feature = pca(image');
333 %     row_feature = row_feature(:,1:dimension);
334 %     col_feature = pca(image);
335 %     col_feature = col_feature(:,1:dimension);
336 %     test_img_re(:,i) = [row_feature(:);col_feature(:)];
337 % end
338
339 % single layer
340 net = perceptron();
341 net = configure(net,train_img_re,train_label);
342 % set network parameter
343 net.divideFcn = 'dividetrain';
344 net.trainParam.epochs = 5000;
345 % net.trainFcn = "trainscg";
346 net.trainparam.goal = 1e-6;
347 % train
348 [net,tr]=train(net,train_img_re,train_label);
349
350 % compute accuracy
351 pred_label_train = net(train_img_re);
352 accu_train = 1 - mean(abs(pred_label_train-train_label));
353 pred_label_test = net(test_img_re);
354 accu_test = 1 - mean(abs(pred_label_test-test_label));
355 fprintf('accu_train: %.02f%%\n',accu_train*100)
356 fprintf('accu_val: %.02f%%\n',accu_test*100)
357
358 %% (c)
359 clc
360 clear all
361 close all
362
363 % read data
364 train_set = double(load('train_set.mat').train_set);
365 test_set = double(load('test_set.mat').test_set);
366
367 train_img = train_set(2:end,:);
368 train_label = train_set(1,:);
369 test_img = test_set(2:end,:);
370 test_label = test_set(1,:);
371
372 % 2d convolution
373 core_szie = 3;

```



```

374 dimension = 1;
375 [corex,corey]=sobel(core_szie);
376 image_size = [16,16];
377 len = length(train_label);
378 train_img_re = [];
379 for i = 1:len
380     image = reshape(train_img(:,i),[256,256]);
381     image_feature = 0.5*conv2(image,corex) + ...
382         0.5*conv2(image,corey);
383     image_feature = ...
384         image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core_szie-1)/2);
385     % imshow(image_feature)
386     new_image = imresize(image_feature,image_size);
387     % imshow(new_image)
388     train_img_re(:,i) = new_image(:);
389     % row_feature = pca(image_feature');
390     % row_feature = row_feature(:,1:dimension);
391     % col_feature = pca(image_feature);
392     % col_feature = col_feature(:,1:dimension);
393     % train_img_re(:,i) = [row_feature(:);col_feature(:)];
394 end
395 len = length(test_label);
396 test_img_re = [];
397 for i = 1:len
398     image = reshape(test_img(:,i),[256,256]);
399     image_feature = 0.5*conv2(image,corex) + ...
400         0.5*conv2(image,corey);
401     image_feature = ...
402         image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core_szie-1)/2);
403     new_image = imresize(image_feature,image_size);
404     test_img_re(:,i) = new_image(:);
405     % row_feature = pca(image_feature');
406     % row_feature = row_feature(:,1:dimension);
407     % col_feature = pca(image_feature);
408     % col_feature = col_feature(:,1:dimension);
409     % test_img_re(:,i) = [row_feature(:);col_feature(:)];
410 end
411 % MLP
412 net = patternnet(image_size(1)*image_size(2));
413 net.trainFcn = 'trainlm';
414 net.trainFcn = 'trainscg';% 'trainlm' 'trainbr'
415 net.divideFcn = 'dividetrain'; % input for training only
416 net.performFcn = 'mse';
417 net.trainParam.min_grad = 1e-9;
418 % net.performParam.regularization = 10e-6; % regularization ...
419     strength
420 % net = configure(net,train_img_re,train_label);
421 net = configure(net,train_img,train_label);
422 % train
423 % [net,tr]=train(net,train_img_re,train_label);
424 [net,tr]=train(net,train_img,train_label);
425 % compute accuracy
426 % pred_label_train = net(train_img_re);
427 pred_label_train = net(train_img);

```

```

427 accu_train = 1 - mean(abs(pred_label_train-train_label));
428 % pred_label_test = net(test_img_re);
429 pred_label_test = net(test_img);
430 accu_test = 1 - mean(abs(pred_label_test-test_label));
431 fprintf('accu_train: %.02f%%\n',accu_train*100)
432 fprintf('accu_val: %.02f%%\n',accu_test*100)
433
434 %% (d)
435 clc
436 clear all
437 close all
438
439 % read data
440 train_set = double(load('train_set.mat').train_set);
441 test_set = double(load('test_set.mat').test_set);
442
443 train_img = train_set(2:end,:);
444 train_label = train_set(1,:);
445 test_img = test_set(2:end,:);
446 test_label = test_set(1,:);
447
448 % 2d convolution
449 core_szie = 3;
450 dimension = 1;
451 [corex,corey]=sobel(core_szie);
452 image_size = [16,16];
453 len = length(train_label);
454 train_img_re = [];
455 for i = 1:len
456     image = reshape(train_img(:,i),[256,256]);
457     image_feature = 0.5*conv2(image,corex) + ...
458         0.5*conv2(image,corey);
459     image_feature = ...
460         image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core_szie-1)/2);
461     % imshow(image_feature)
462     new_image = imresize(image_feature,image_size);
463     % imshow(new_image)
464     train_img_re(:,i) = new_image(:);
465     % row_feature = pca(image_feature');
466     % row_feature = row_feature(:,1:dimension);
467     % col_feature = pca(image_feature);
468     % col_feature = col_feature(:,1:dimension);
469     % train_img_re(:,i) = [row_feature(:);col_feature(:)];
470 end
471 len = length(test_label);
472 test_img_re = [];
473 for i = 1:len
474     image = reshape(test_img(:,i),[256,256]);
475     image_feature = 0.5*conv2(image,corex) + ...
476         0.5*conv2(image,corey);
477     image_feature = ...
478         image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core_szie-1)/2);
479     new_image = imresize(image_feature,image_size);
480     test_img_re(:,i) = new_image(:);
481     % row_feature = pca(image_feature');
482     % row_feature = row_feature(:,1:dimension);
483     % col_feature = pca(image_feature);
484     % col_feature = col_feature(:,1:dimension);

```

```

481     %     test_img_re(:,i) = [row_feature(:);col_feature(:)];
482 end
483
484 % MLP
485 net = patternnet(image_size(1)*image_size(2));
486
487 %     net.trainFcn = 'trainlm';
488 net.trainFcn = 'trainscg';% 'trainlm' 'trainbr'
489 % net.divideFcn = 'dividetrain'; % input for training only
490 net.divideParam.trainRatio=0.8;
491 net.divideParam.valRatio=0.2;
492 net.divideParam.testRatio=0;
493 net.trainParam.max_fail = 2000;
494 net.trainParam.epochs = 5000;
495 net.performFcn = 'mse';
496 net.trainParam.min_grad = 1e-9;
497 net.trainParam.goal = 1e-3;
498 net.performParam.regularization = 0.15; % regularization strength
499
500 net = configure(net,train_img_re,train_label);
501 % train
502 [net,tr]=train(net,train_img_re,train_label);
503
504 % compute accuracy
505 pred_label_train = net(train_img_re);
506 accu_train = 1 - mean(abs(pred_label_train-train_label));
507 pred_label_test = net(test_img_re);
508 accu_test = 1 - mean(abs(pred_label_test-test_label));
509 fprintf('accu_train: %.02f%%\n',accu_train*100)
510 fprintf('accu_val: %.02f%%\n',accu_test*100)
511
512 %% (e)
513 clc
514 clear all
515 close all
516
517 % read data
518 train_set = double(load('train_set.mat').train_set);
519 test_set = double(load('test_set.mat').test_set);
520
521 train_img = train_set(2:end,:);
522 train_label = train_set(1,:);
523 test_img = test_set(2:end,:);
524 test_label = test_set(1,:);
525
526 % 2d convolution
527 core_szie = 3;
528 dimension = 1;
529 [corex,corey]=sobel(core_szie);
530 image_size = [16,16];
531 len = length(train_label);
532 train_img_re = [];
533 for i = 1:len
534     image = reshape(train_img(:,i),[256,256]);
535     image_feature = 0.5*conv2(image,corex) + ...
536         0.5*conv2(image,corey);
537     image_feature = ...
538         image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core

```

```

537     %     imshow(image_feature)
538     new_image = imresize(image_feature,image_size);
539     %     imshow(new_image)
540     train_img_re(:,i) = new_image(:);
541     %     row_feature = pca(image_feature');
542     %     row_feature = row_feature(:,1:dimension);
543     %     col_feature = pca(image_feature);
544     %     col_feature = col_feature(:,1:dimension);
545     %     train_img_re(:,i) = [row_feature(:);col_feature(:)];
546 end
547 len = length(test_label);
548 test_img_re = [];
549 for i = 1:len
550     image = reshape(test_img(:,i),[256,256]);
551     image_feature = 0.5*conv2(image,corex) + ...
552         0.5*conv2(image,corey);
553     image_feature = ...
554         image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core_szie-1)/2);
555     new_image = imresize(image_feature,image_size);
556     test_img_re(:,i) = new_image(:);
557     %     row_feature = pca(image_feature');
558     %     row_feature = row_feature(:,1:dimension);
559     %     col_feature = pca(image_feature);
560     %     col_feature = col_feature(:,1:dimension);
561     %     test_img_re(:,i) = [row_feature(:);col_feature(:)];
562 end
563 train_set = [train_label;train_img_re];
564 test_set = [test_label;test_img_re];
565 % sequential learning
566 epochs = 50;
567 [net,accu_train,accu_test] = ...
568     train_seq(256,train_set,test_set,epochs);
569 %plot
570 x = 1:epochs;
571 figure();
572 hold on
573 plot(x,accu_train);
574 plot(x,accu_test);
575 legend('training accuracy','test accuracy')
576 title('Sequential learning accuracy by epochs')
577 ylabel('accuracy/%')
578 xlabel('epochs')
579 hold off
580
581
582
583
584
585
586 function [net,accu_train] = Seq_mlp(n,train_data,epochs)
587 % Construct a 1-n-1 MLP and conduct sequential training.
588 %
589 % Args:
590 %     n: int, number of neurons in the hidden layer of MLP.

```

```

591 %   train_data: array, the training set data (train_num,2). 1 ...
      row is the x,2 row
592 %   is y
593 %   epochs: int, number of training epochs.
594 %
595 % Returns:
596 %
597 %   net: object, containg trained network.
598 %   accu_train: vector of (epochs, 1), containg the accuracy ...
      on training
599 %           set of each epoch during training.
600
601 % 1. Change the input to cell array form for sequential training
602 train_num = length(train_data);
603 x = num2cell(train_data(1,:), 1);
604 y = num2cell(train_data(2,:), 1);
605
606 % 2. Construct and configure the MLP
607 % using Levenberg-Marquardt backpropagation.
608 net = fitnet(n);
609
610 net.divideFcn = 'dividetrain'; % input for training only
611 net.trainParam.epochs = epochs;
612 net.trainFcn = 'traingdx'; % 'trainrp' 'traingdx'
613
614 accu_train = zeros(epochs,1); % record accuracy on training ...
      set of each epoch
615
616 % 3. Train the network in sequential mode
617 for i = 1 : epochs
618     display(['Epoch: ', num2str(i)])
619     idx = randperm(train_num); % shuffle the input
620     net = adapt(net, x(:,idx), y(:,idx));
621     pred_train = round(net(train_data(1,1:train_num))); % ...
      predictions on training set
622     accu_train(i) = 1 - ...
      mean(abs(pred_train-train_data(2,1:train_num)));
623 end
624 end
625
626 function y = pascal(k, n)
627 if k ≥ 0 && k ≤ n
628     y = factorial(n) / (factorial(n-k) * factorial(k));
629 else
630     y = 0;
631 end
632 end
633 function [sobel_x, sobel_y] = sobel(order)
634 sobel_x = zeros(order, order);
635 smooth = zeros(order, 1);
636 diff = zeros(order, 1);
637
638 for j = 1:order
639     smooth = pascal(j-1, order-1)';
640
641     for k = 1:order
642         diff = (pascal(k-1, order-2) - pascal(k-2, order-2))';
643

```

```

644         sobel_x(j, k) = smooth * diff;
645     end
646 end
647
648 sobel_y = -1 * sobel_x';
649 end
650
651 function [net, accu_train, accu_test] = ...
        train_seq(n, train_set, test_set, epochs )
652 % Construct a 1-n-1 MLP and conduct sequential training.
653 %
654 % Args:
655 % n: int, number of neurons in the hidden layer of MLP.
656 % images: matrix of (image_dim, image_num), containing possibly
657 % preprocessed image data as input.
658 % labels: vector of (1, image_num), containing corresponding ...
        label of
659 % each image.
660 % train_num: int, number of training images.
661 % val_num: int, number of validation images.
662 % epochs: int, number of training epochs.
663 %
664 % Returns:
665 % net: object, containing trained network.
666 % accu_train: vector of (epochs, 1), containing the accuracy ...
        on training
667 % set of each epoch during training.
668 % accu_val: vector of (epochs, 1), containing the accuracy on ...
        validation
669 % set of each epoch during training.
670 % 1. Change the input to cell array form for sequential ...
        training
671 train_img = train_set(2:end,:);
672 train_label = train_set(1,:);
673 test_img = test_set(2:end,:);
674 test_label = test_set(1,:);
675 images_c = num2cell(train_img, 1);
676 labels_c = num2cell(train_label, 1);
677 train_num = length(train_label);
678 test_num = length(test_label);
679 % 2. Construct and configure the MLP
680 net = patternnet(n);
681 net.divideFcn = 'dividetrain'; % input for training only
682 net.performParam.regularization = 0.25; % regularization ...
        strength
683 net.trainFcn = 'trainscg'; % 'trainrp' 'traingdx'
684 net.trainParam.epochs = epochs;
685 accu_train = zeros(epochs,1); % record accuracy on ...
        training set of each epoch
686 accu_test = zeros(epochs,1); % record accuracy on ...
        validation set of each epoch
687 % 3. Train the network in sequential mode
688 for i = 1 : epochs
689     display(['Epoch: ', num2str(i)])
690     idx = randperm(train_num); % shuffle the input
691     net = adapt(net, images_c(:,idx), labels_c(:,idx));
692     pred_label_train = net(train_img);

```

```
693         accu_train(i) = 1 - ...  
            mean(abs(pred_label_train-train_label));  
694         pred_label_test = net(test_img);  
695         accu_test(i) = 1 - mean(abs(pred_label_test-test_label));  
696     end  
697 end
```