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.

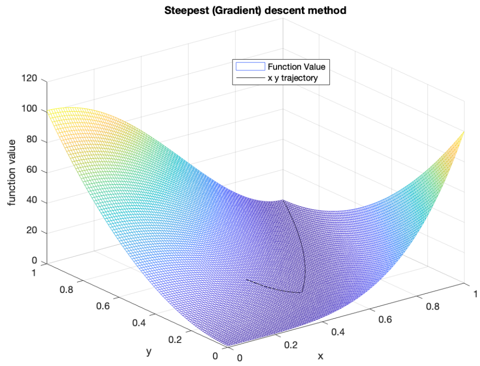


Figure 1: the function value and (x , y) trajectory

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

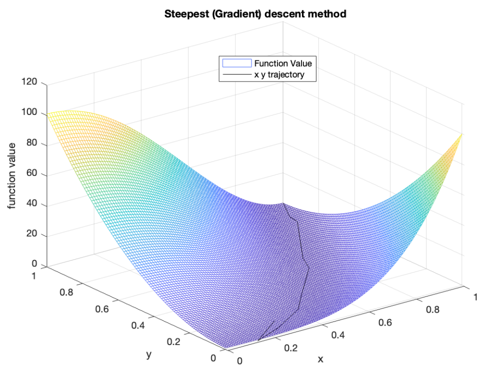
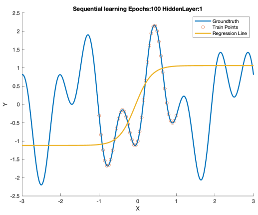


Figure 2: the function value and (x , y) trajectory

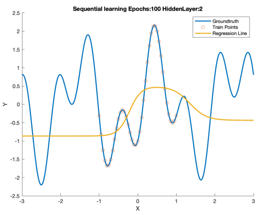
It runs just 11 times iteration before achieving the minimum, which shows its efficiency compared to Steepest (Gradient) descent method.

Question 2

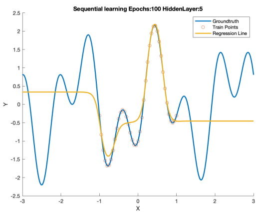
(a) Use the sequential mode with BP algorithm and experiment with the following different hidden layer of the MLP: 1-n-1 ,where n = 1, 2, 5, 10, 20, 50. The training function used is trainlm. The number of epochs is 100.
The results are shown as here.



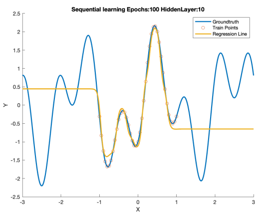
(a) 1 neuron hidden layer



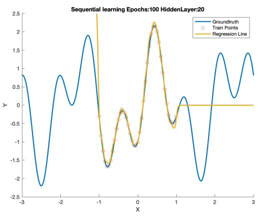
(b) 2 neuron hidden layer



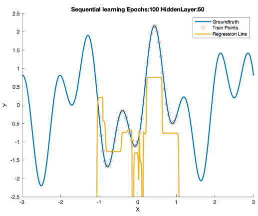
(c) 5 neuron hidden layer



(d) 10 neuron hidden layer



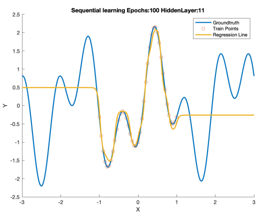
(e) 20 neuron hidden layer



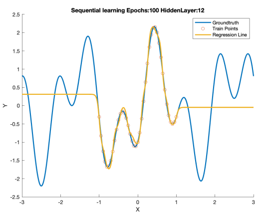
(f) 50 neuron hidden layer

Figure 3: 1-n-1 MLP

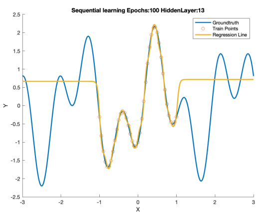
It is obvious that when n = 1,2,3,10, the neural network is under-fitting. When n = 20, the neural network is proper fitting. When n = 50, the neural network is over-fitting. Because there is too many neurons in the hidden layer that is too sensitive to some points while ignore others. Meanwhile, it 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.



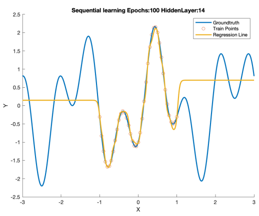
(a) 11 neuron hidden layer



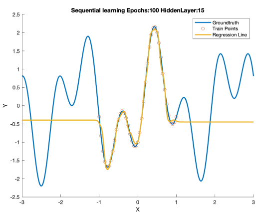
(b) 12 neuron hidden layer



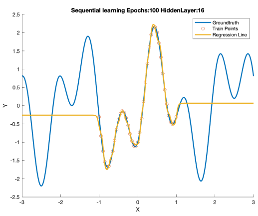
(c) 13 neuron hidden layer



(d) 14 neuron hidden layer



(e) 15 neuron hidden layer

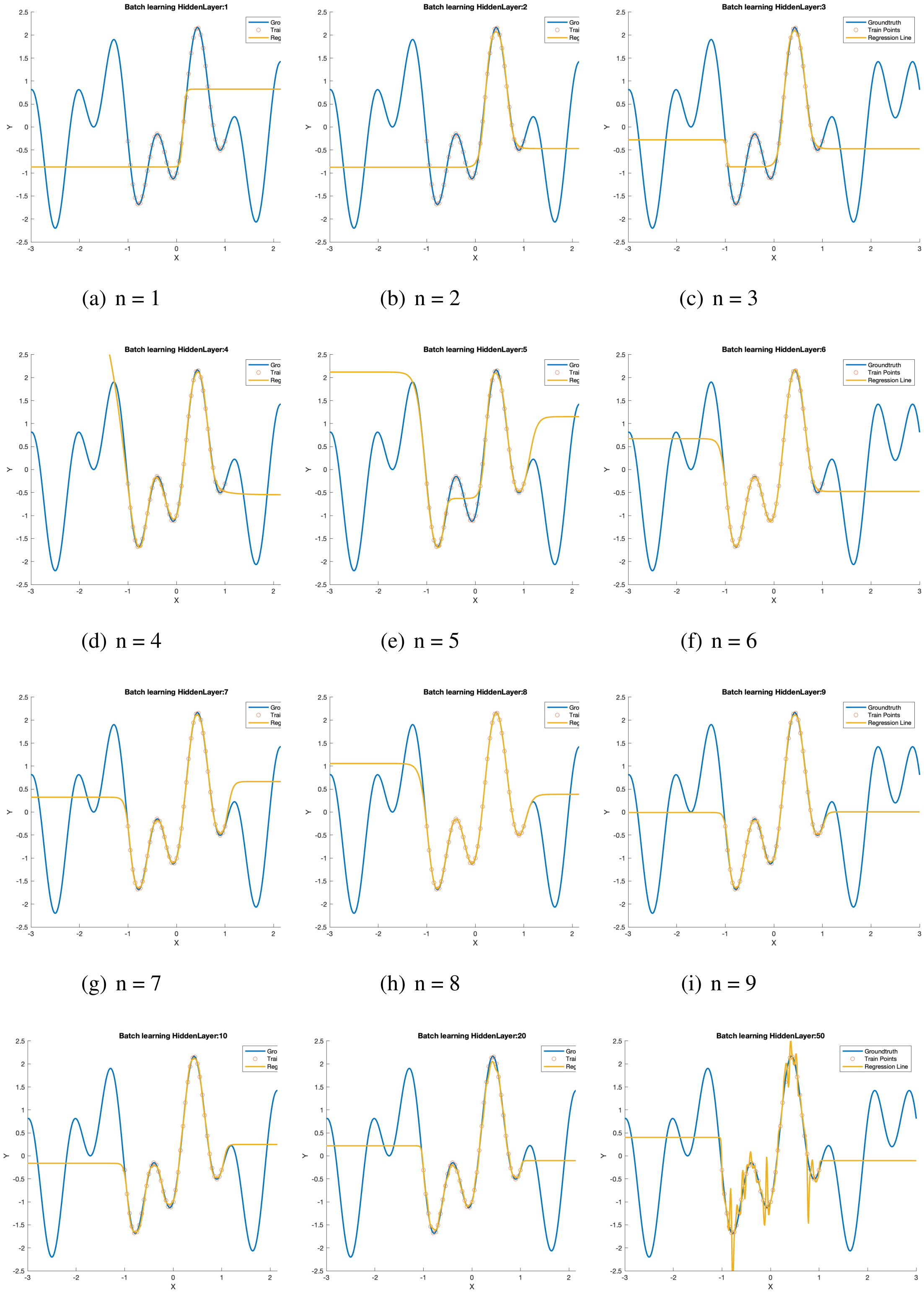


(f) 16 neuron hidden layer

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:



(a) n = 1 (b) n = 2 (c) n = 3

(d) n = 4 (e) n = 5 (f) n = 6

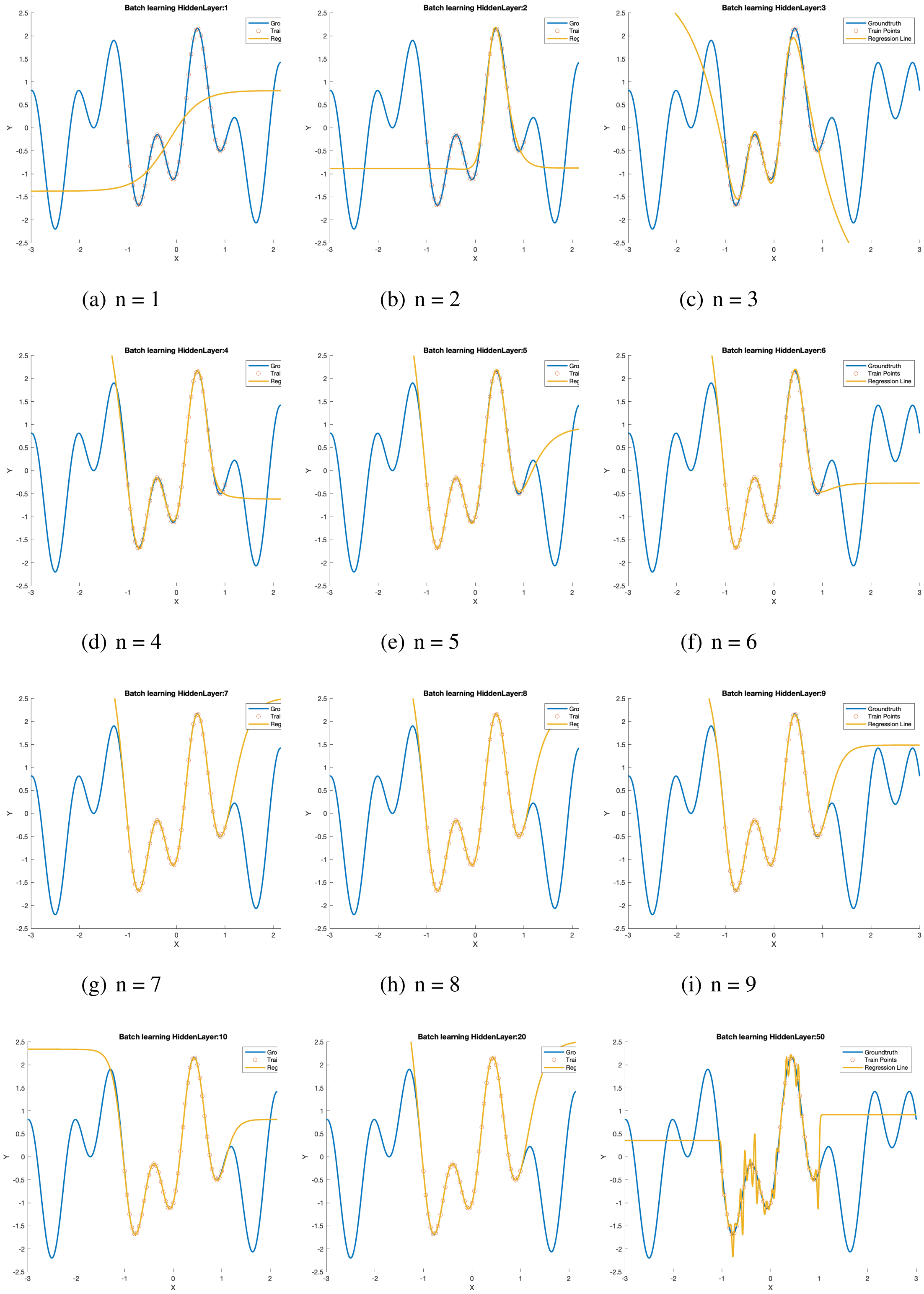
(g) n = 7 (h) n = 8 (i) n = 9

(j) n = 10 (k) n = 20 (l) n = 50

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:



(a) n = 1 (b) n = 2 (c) n = 3

(d) n = 4 (e) n = 5 (f) n = 6

(g) n = 7 (h) n = 8 (i) n = 9

(j) n = 10 (k) n = 20 (l) n = 50

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:

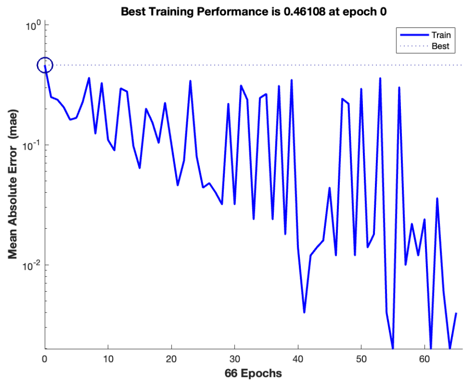


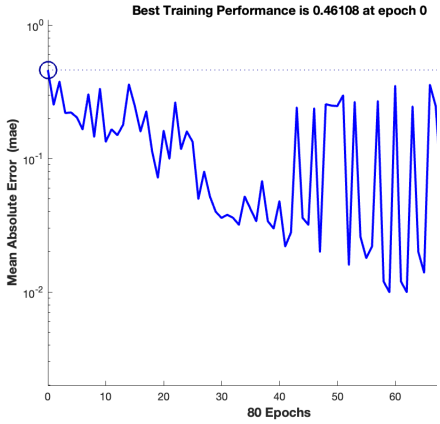
Figure 7: the performance of Rosenblatts perceptron

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

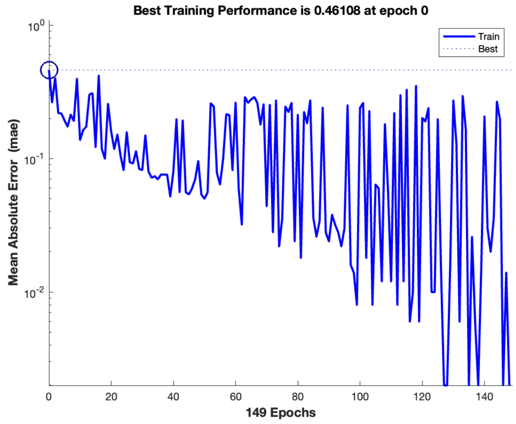
(b) The accuracy of naively downsampling image is shown here

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%

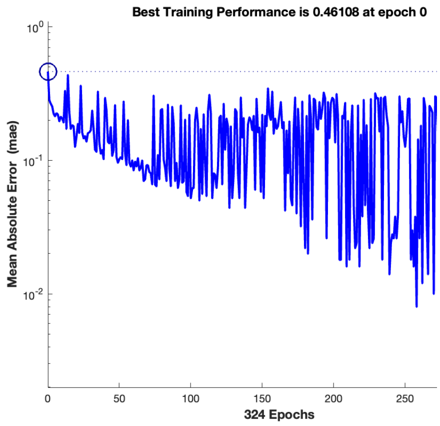
I use PCA to extract the feature of the image in 2 direction, row and column. The first 5 dimension of each direction is captured and reshaped as input of the perceptron. As the more image is compressed, the accuracy is lower.
The performance of each dimension reduction method is shown here:



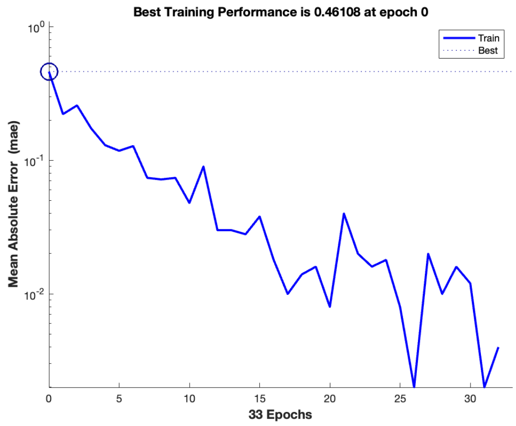
(a) 128x128



(b) 64x64



(c) 32x32



(d) PCA

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:

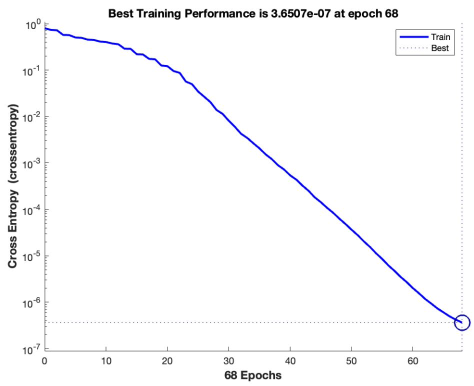


Figure 9: the performance of 1-196-1 MLP

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.

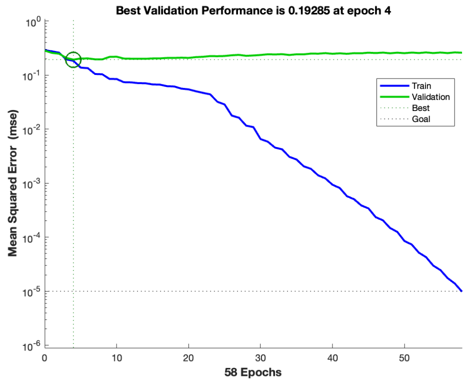


Figure 10: the performance of 1-196-1 MLP

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:

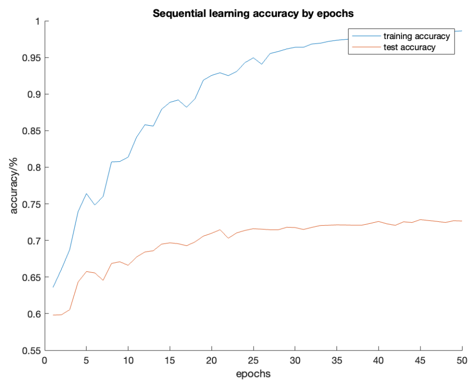


Figure 11: the performance of sequential learning

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 Question3 (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 $16 \approx 15 = 3x5$.

(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

```
% EE5904 Neural Network
% Assignment 2

%% Question 1

% (a)
clc
clear all
close all
% initialization
% starting point
X = [];
Y = [];
X(1) = 0.5 * rand(1);
Y(1) = 0.5 * rand(1);
% learning rate
eta = 0.001;
% eta = 0.2;
% iteration counter
i = 0;
% cost function
syms x predicted_y
cost_f = (1-x)^2 + 100*(predicted_y-x^2)^2;
% cost trajectory
cost = [];
% Gradient descent
G_x = diff(cost_f,x);
G_y = diff(cost_f,predicted_y);

% update process
while true
    i = i + 1;
    % record cost
```

```
    %     cost(i) = double(subs(cost_f,[x,y],[X(i),Y(i)]));
    cost(i) = (1-X(i))^2 + 100*(Y(i)-X(i)^2)^2;
    % stop when
    if cost(i) < 1e-5
        break
    elseif i > 1e6
        error("can not converge")
    end
    % update weights
    %     X(i + 1) = X(i) - ... eta*double(subs(G_x,[x,y],[X(i),Y(i)]));
    %     Y(i + 1) = Y(i) - ... eta*double(subs(G_y,[x,y],[X(i),Y(i)]));
    X(i + 1) = X(i) - eta*(2*(X(i) - 1) + 400*X(i)*(X(i)^2 - ... Y(i)));
    Y(i + 1) = Y(i) - eta*(200*(Y(i) - X(i)^2));
end

% plot
% function value
[xaxis,yaxis]=meshgrid(0:0.01:1);
function_value = (1-xaxis).^2 + 100*(yaxis-xaxis.^2).^2;
mesh(xaxis,yaxis,function_value)
% x,y trajectory
hold on
F_value = (1-X).^2 + 100*(Y-X.^2).^2;
plot3(X,Y,F_value,'black')
hold off
grid on
xlabel("x")
ylabel("y")
zlabel("function value")
title("Steepest (Gradient) descent method")
legend("Function Value","x y trajectory")

% (b)
clc
clear all
close all
% initialization
% starting point
X = 0.5 * rand(1);
Y = 0.5 * rand(1);
% weights value
w = [X,Y];
% learning rate
% eta = 0.001;
eta = 0.2;
% iteration counter
i = 0;
% cost function
syms x predicted_y
cost_f = (1-x)^2 + 100*(predicted_y-x^2)^2;
% cost trajectory
cost = [];
% Gradient descent
G_x = diff(cost_f,x);
G_y = diff(cost_f,predicted_y);
```

```
G = [G_x,G_y];
% Hessian matrix
H = [diff(cost_f,x,2) diff(cost_f,x,predicted_y) ; ...
    diff(cost_f,predicted_y,x) diff(cost_f,predicted_y,2)];

% update process
while true
    i = i + 1;
    % record cost
    %     cost(i) = double(subs(cost_f,[x,y],[X(i),Y(i)]));
    X = w(i,1);
    Y = w(i,2);
    cost(i) = (1-X)^2 + 100*(Y-X^2)^2;
    % stop when
    if cost(i) < 1e-5
        break
    elseif i > 1e6
        error("can not converge")
    end
    % update weights
    %     w(i + 1,:) = w(i,:) - inv(subs(H,{x;y},{w(i,:)})) * ...
        subs(G,{x;y},{w(i,:)});
    g = [(2*(X - 1) + 400*(X^3 - X*Y)); 200*(Y-X^2)];
    h = [1200*X^2 - 400*Y + 2,-400*X; -400*X,200];
    w(i + 1,:) = w(i,:) - (inv(h) * g)';
end

% plot
% function value
[xaxis,yaxis]=meshgrid(0:0.01:1);
function_value = (1-xaxis).^2 + 100*(yaxis-xaxis.^2).^2;
mesh(xaxis,yaxis,function_value)
% x,y trajectory
hold on
F_value = (1-w(:,1)).^2 + 100*(w(:,2)-w(:,1).^2).^2;
plot3(w(:,1),w(:,2),F_value,'black')
hold off
grid on
xlabel("x")
ylabel("y")
zlabel("function value")
title("Steepest (Gradient) descent method")
legend("Function Value","x y trajectory")

Z0 = 300;
Zl = 54.7360 -26.3014i;
k = 2*pi;
l = 3/8;
Zin = Z0*(Zl + i*Z0*tan(k*l))/(Z0 + i*Zl*tan(k*l));

%% Question 2

%(a)
clc
clear all
close all
% hyperparams
epochs = 100;
```

```
% data
train_x = -1:0.05:1;
train_y = 1.2*sin(pi*train_x) - cos(2.4*pi*train_x);
train_set = [train_x;train_y];

for i = [11:20]

    % train
    [net,train_accuracy] = Seq_mlp(i,train_set,epochs);
    % test
    test_x = -3:0.01:3;
    truth_y = 1.2*sin(pi*test_x) - cos(2.4*pi*test_x);
    predicted_y = net(test_x);

    % plot
    result = figure();
    hold on
    plot(test_x,truth_y,'-','LineWidth',2)
    scatter(train_x,train_y)
    plot(test_x,predicted_y,'-','LineWidth',2)
    ylim([-2.5 2.5])
    legend('Groundtruth','Train Points','Regression Line')
    title(sprintf('Sequential learning Epochs:%d ...
        HiddenLayer:%d',epochs,i))
    ylabel('Y')
    xlabel('X')
    hold off
    % save image
    saveas(result,sprintf('HiddenLayer%02d.png',i));
end

%% Q2 (b)&(c)
clc
clear all
close all

% training set
train_x = -1:0.05:1;
train_y = 1.2*sin(pi*train_x) - cos(2.4*pi*train_x);
train_set = [train_x;train_y];
% test set
test_x = -3:0.01:3;
truth_y = 1.2*sin(pi*test_x) - cos(2.4*pi*test_x);

% build NN with different hidden layer
for i = [1:10,20,50]
    % Create Neuron Network
    net = fitnet(i);

    %     net.trainFcn = 'trainlm';
    net.trainFcn = 'trainbr';% 'trainlm' 'trainbr'
    net.divideFcn = 'dividetrain'; % input for training only
    net.performParam.regularization = 10e-6; % regularization ...
        strength

    % Train the Network
    [net,¬] = train(net,train_x,train_y);

```

```
    % Test
    predicted_y = sim(net, test_x);
    result = figure();
    hold on
    plot(test_x,truth_y,'-','LineWidth',2)
    scatter(train_x,train_y)
    plot(test_x,predicted_y,'-','LineWidth',2)
    ylim([-2.5 2.5])
    legend('Groundtruth','Train Points','Regression Line')
    title(sprintf('Batch learning HiddenLayer:%d',i))
    ylabel('Y')
    xlabel('X')
    hold off
    % save image
    saveas(result,sprintf('batchbr%02d.png',i));
end

%% Q3
% my matric number is A0260074M so I should in group 3
display(mod(74,4)+1);
% read and store image data for fast read
% set the label of coast is 1 and insidecity is 0
train_set = [];
test_set = [];
data_path = "/Users/wanrylin/Master Courses/EE ...
    5904/group_3/group_3";
% train set construction
train_path = strcat(data_path,"/train/");
namelist = dir(strcat(train_path,'*.jpg'));
img_num = length(namelist);
for i = 1:img_num
    imgname = namelist(i).name;
    tmp = strsplit(imgname, {'_', '.'});
    label = str2num(tmp{2});
    img = imread(strcat(train_path, imgname));
    % every col is a combination of label(row 1) and image
    vector = [label;img(:)];
    train_set = [train_set,vector];
end
save("train_set.mat","train_set");

% test set construction
test_path = strcat(data_path,"/test/");
namelist = dir(strcat(test_path,'*.jpg'));
img_num = length(namelist);
for i = 1:img_num
    imgname = namelist(i).name;
    tmp = strsplit(imgname, {'_', '.'});
    label = str2num(tmp{2});
    img = imread(strcat(test_path, imgname));
    vector = [label;img(:)];
    test_set = [test_set,vector];
end
save("test_set.mat","test_set");

%% (a)
clc
clear all
```

```
close all

% read data
train_set = double(load('train_set.mat').train_set);
test_set = double(load('test_set.mat').test_set);

train_img = train_set(2:end,:);
train_label = train_set(1,:);
test_img = test_set(2:end,:);
test_label = test_set(1,:);

% single layer
net = perceptron();
net = configure(net,train_img,train_label);
% set network parameter
net.divideFcn = 'dividetrain';
% train
[net,tr]=train(net,train_img,train_label);

% compute accuracy
pred_label_train = net(train_img);
accu_train = 1 - mean(abs(pred_label_train-train_label));
pred_label_test = net(test_img);
accu_test = 1 - mean(abs(pred_label_test-test_label));
fprintf('accu_train: %.02f%%\n',accu_train*100)
fprintf('accu_val: %.02f%%\n',accu_test*100)

%% (b)
clc
clear all
close all

% read data
train_set = double(load('train_set.mat').train_set);
test_set = double(load('test_set.mat').test_set);

train_img = train_set(2:end,:);
train_label = train_set(1,:);
test_img = test_set(2:end,:);
test_label = test_set(1,:);

% % downsample image
% image_size = [32,32];
% len = length(train_label);
% train_img_re = [];
% for i = 1:len
%     image = reshape(train_img(:,i),[256,256]);
%     new_image = imresize(image,image_size);
%     train_img_re(:,i) = new_image(:);
% end
% len = length(test_label);
% test_img_re = [];
% for i = 1:len
%     image = reshape(test_img(:,i),[256,256]);
%     new_image = imresize(image,image_size);
%     test_img_re(:,i) = new_image(:);
% end

```

```
% % PCA
% dimension = 1;
% len = length(train_label);
% train_img_re = [];
% for i = 1:len
%     image = reshape(train_img(:,i),[256,256]);
%     row_feature = pca(image');
%     row_feature = row_feature(:,1:dimension);
%     col_feature = pca(image);
%     col_feature = col_feature(:,1:dimension);
%     train_img_re(:,i) = [row_feature(:);col_feature(:)];
% end
% len = length(test_label);
% test_img_re = [];
% for i = 1:len
%     image = reshape(test_img(:,i),[256,256]);
%     row_feature = pca(image');
%     row_feature = row_feature(:,1:dimension);
%     col_feature = pca(image);
%     col_feature = col_feature(:,1:dimension);
%     test_img_re(:,i) = [row_feature(:);col_feature(:)];
% end

% single layer
net = perceptron();
net = configure(net,train_img_re,train_label);
% set network parameter
net.divideFcn = 'dividetrain';
net.trainParam.epochs = 5000;
% net.trainFcn = "trainscg";
net.trainparam.goal = 1e-6;
% train
[net,tr]=train(net,train_img_re,train_label);

% compute accuracy
pred_label_train = net(train_img_re);
accu_train = 1 - mean(abs(pred_label_train-train_label));
pred_label_test = net(test_img_re);
accu_test = 1 - mean(abs(pred_label_test-test_label));
fprintf('accu_train: %.02f%%\n',accu_train*100)
fprintf('accu_val: %.02f%%\n',accu_test*100)

%% (c)
clc
clear all
close all

% read data
train_set = double(load('train_set.mat').train_set);
test_set = double(load('test_set.mat').test_set);

train_img = train_set(2:end,:);
train_label = train_set(1,:);
test_img = test_set(2:end,:);
test_label = test_set(1,:);

% 2d convolution
core_szie = 3;
```

```
dimension = 1;
[corex,corey]=sobel(core_szie);
image_size = [16,16];
len = length(train_label);
train_img_re = [];
for i = 1:len
    image = reshape(train_img(:,i),[256,256]);
    image_feature = 0.5*conv2(image,corex) + ...
        0.5*conv2(image,corey);
    image_feature = ...
        image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core
    %      imshow(image_feature)
    new_image = imresize(image_feature,image_size);
    %      imshow(new_image)
    train_img_re(:,i) = new_image(:);
    %      row_feature = pca(image_feature');
    %      row_feature = row_feature(:,1:dimension);
    %      col_feature = pca(image_feature);
    %      col_feature = col_feature(:,1:dimension);
    %      train_img_re(:,i) = [row_feature(:);col_feature(:)];
end
len = length(test_label);
test_img_re = [];
for i = 1:len
    image = reshape(test_img(:,i),[256,256]);
    image_feature = 0.5*conv2(image,corex) + ...
        0.5*conv2(image,corey);
    image_feature = ...
        image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core
    new_image = imresize(image_feature,image_size);
    test_img_re(:,i) = new_image(:);
    %      row_feature = pca(image_feature');
    %      row_feature = row_feature(:,1:dimension);
    %      col_feature = pca(image_feature);
    %      col_feature = col_feature(:,1:dimension);
    %      test_img_re(:,i) = [row_feature(:);col_feature(:)];
end

% MLP
net = patternnet(image_size(1)*image_size(2));

%      net.trainFcn = 'trainlm';
net.trainFcn = 'trainscg';% 'trainlm' 'trainbr'
net.divideFcn = 'dividetrain'; % input for training only
net.performFcn = 'mse';
net.trainParam.min_grad = 1e-9;
% net.performParam.regularization = 10e-6; % regularization ...
    strength

% net = configure(net,train_img_re,train_label);
net = configure(net,train_img,train_label);
% train
% [net,tr]=train(net,train_img_re,train_label);
[net,tr]=train(net,train_img,train_label);

% compute accuracy
% pred_label_train = net(train_img_re);
pred_label_train = net(train_img);
```

```
accu_train = 1 - mean(abs(pred_label_train-train_label));
% pred_label_test = net(test_img_re);
pred_label_test = net(test_img);
accu_test = 1 - mean(abs(pred_label_test-test_label));
fprintf('accu_train: %.02f%%\n',accu_train*100)
fprintf('accu_val: %.02f%%\n',accu_test*100)

%% (d)
clc
clear all
close all

% read data
train_set = double(load('train_set.mat').train_set);
test_set = double(load('test_set.mat').test_set);

train_img = train_set(2:end,:);
train_label = train_set(1,:);
test_img = test_set(2:end,:);
test_label = test_set(1,:);

% 2d convolution
core_szie = 3;
dimension = 1;
[corex,corey]=sobel(core_szie);
image_size = [16,16];
len = length(train_label);
train_img_re = [];
for i = 1:len
    image = reshape(train_img(:,i),[256,256]);
    image_feature = 0.5*conv2(image,corex) + ...
        0.5*conv2(image,corey);
    image_feature = ...
        image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core
    %      imshow(image_feature)
    new_image = imresize(image_feature,image_size);
    %      imshow(new_image)
    train_img_re(:,i) = new_image(:);
    %      row_feature = pca(image_feature');
    %      row_feature = row_feature(:,1:dimension);
    %      col_feature = pca(image_feature);
    %      col_feature = col_feature(:,1:dimension);
    %      train_img_re(:,i) = [row_feature(:);col_feature(:)];
end
len = length(test_label);
test_img_re = [];
for i = 1:len
    image = reshape(test_img(:,i),[256,256]);
    image_feature = 0.5*conv2(image,corex) + ...
        0.5*conv2(image,corey);
    image_feature = ...
        image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core
    new_image = imresize(image_feature,image_size);
    test_img_re(:,i) = new_image(:);
    %      row_feature = pca(image_feature');
    %      row_feature = row_feature(:,1:dimension);
    %      col_feature = pca(image_feature);
    %      col_feature = col_feature(:,1:dimension);
```

```
    %     test_img_re(:,i) = [row_feature(:);col_feature(:)];
end

% MLP
net = patternnet(image_size(1)*image_size(2));

%     net.trainFcn = 'trainlm';
net.trainFcn = 'trainscg';% 'trainlm' 'trainbr'
% net.divideFcn = 'dividetrain'; % input for training only
net.divideParam.trainRatio=0.8;
net.divideParam.valRatio=0.2;
net.divideParam.testRatio=0;
net.trainParam.max_fail = 2000;
net.trainParam.epochs = 5000;
net.performFcn = 'mse';
net.trainParam.min_grad = 1e-9;
net.trainParam.goal = 1e-3;
net.performParam.regularization = 0.15; % regularization strength

net = configure(net,train_img_re,train_label);
% train
[net,tr]=train(net,train_img_re,train_label);

% compute accuracy
pred_label_train = net(train_img_re);
accu_train = 1 - mean(abs(pred_label_train-train_label));
pred_label_test = net(test_img_re);
accu_test = 1 - mean(abs(pred_label_test-test_label));
fprintf('accu_train: %.02f%%\n',accu_train*100)
fprintf('accu_val: %.02f%%\n',accu_test*100)

%% (e)
clc
clear all
close all

% read data
train_set = double(load('train_set.mat').train_set);
test_set = double(load('test_set.mat').test_set);

train_img = train_set(2:end,:);
train_label = train_set(1,:);
test_img = test_set(2:end,:);
test_label = test_set(1,:);

% 2d convolution
core_szie = 3;
dimension = 1;
[corex,corey]=sobel(core_szie);
image_size = [16,16];
len = length(train_label);
train_img_re = [];
for i = 1:len
    image = reshape(train_img(:,i),[256,256]);
    image_feature = 0.5*conv2(image,corex) + ...
        0.5*conv2(image,corey);
    image_feature = ...
        image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core
```

```
    %     imshow(image_feature)
    new_image = imresize(image_feature,image_size);
    %     imshow(new_image)
    train_img_re(:,i) = new_image(:);
    %     row_feature = pca(image_feature');
    %     row_feature = row_feature(:,1:dimension);
    %     col_feature = pca(image_feature);
    %     col_feature = col_feature(:,1:dimension);
    %     train_img_re(:,i) = [row_feature(:);col_feature(:)];
end
len = length(test_label);
test_img_re = [];
for i = 1:len
    image = reshape(test_img(:,i),[256,256]);
    image_feature = 0.5*conv2(image,corex) + ...
        0.5*conv2(image,corey);
    image_feature = ...
        image_feature((core_szie+1)/2:end-(core_szie-1)/2,(core_szie+1)/2:end-(core
    new_image = imresize(image_feature,image_size);
    test_img_re(:,i) = new_image(:);
    %     row_feature = pca(image_feature');
    %     row_feature = row_feature(:,1:dimension);
    %     col_feature = pca(image_feature);
    %     col_feature = col_feature(:,1:dimension);
    %     test_img_re(:,i) = [row_feature(:);col_feature(:)];
end
train_set = [train_label;train_img_re];
test_set = [test_label;test_img_re];

% sequential learning
epochs = 50;
[net,accu_train,accu_test] = ...
    train_seq(256,train_set,test_set,epochs);

%plot
x = 1:epochs;
figure();
hold on
plot(x,accu_train);
plot(x,accu_test);
legend('training accuracy','test accuracy')
title('Sequential learning accuracy by epochs')
ylabel('accuracy/%')
xlabel('epochs')
hold off







function [net,accu_train] = Seq_mlp(n,train_data,epochs)
% Construct a 1-n-1 MLP and conduct sequential training.
%
% Args:
%   n: int, number of neurons in the hidden layer of MLP.
```

```
%   train_data: array, the training set data (train_num,2). 1 ... row is the x,2 row
%   is y
%   epochs: int, number of training epochs.
%
% Returns:
%
%   net: object, containg trained network.
%   accu_train: vector of (epochs, 1), containg the accuracy ... on training
%               set of each epoch during training.

% 1. Change the input to cell array form for sequential training
train_num = length(train_data);
x = num2cell(train_data(1,:), 1);
y = num2cell(train_data(2,:), 1);

% 2. Construct and configure the MLP
% using Levenberg-Marquardt backpropagation.
net = fitnet(n);

net.divideFcn = 'dividetrain'; % input for training only
net.trainParam.epochs = epochs;
net.trainFcn = 'traingdx'; % 'trainrp' 'traingdx'

accu_train = zeros(epochs,1); % record accuracy on training ... set of each epoch

% 3. Train the network in sequential mode
for i = 1 : epochs
    display(['Epoch: ', num2str(i)])
    idx = randperm(train_num); % shuffle the input
    net = adapt(net, x(:,idx), y(:,idx));
    pred_train = round(net(train_data(1,1:train_num))); % ... predictions on training set
    accu_train(i) = 1 - ...
        mean(abs(pred_train-train_data(2,1:train_num)));
end
end

function y = pascal(k, n)
if k ≥ 0 && k ≤ n
    y = factorial(n) / (factorial(n-k) * factorial(k));
else
    y = 0;
end
end
function [sobel_x, sobel_y] = sobel(order)
sobel_x = zeros(order, order);
smooth = zeros(order, 1);
diff = zeros(order, 1);

for j = 1:order
    smooth = pascal(j-1, order-1)';

    for k = 1:order
        diff = (pascal(k-1, order-2) - pascal(k-2, order-2))';

```

```
        sobel_x(j, k) = smooth * diff;
    end
end

sobel_y = -1 * sobel_x';
end

function [net, accu_train, accu_test] = ...
    train_seq(n,train_set,test_set,epochs )
% Construct a 1-n-1 MLP and conduct sequential training.
%
% Args:
% n: int, number of neurons in the hidden layer of MLP.
% images: matrix of (image_dim, image_num), containing possibly
% preprocessed image data as input.
% labels: vector of (1, image_num), containing corresponding ...
    label of
% each image.
% train_num: int, number of training images.
% val_num: int, number of validation images.
% epochs: int, number of training epochs.
%
% Returns:
% net: object, containing trained network.
% accu_train: vector of (epochs, 1), containing the accuracy ...
    on training
% set of each eopch during trainig.
% accu_val: vector of (epochs, 1), containing the accuracy on ...
    validation
% set of each eopch during trainig.
    % 1. Change the input to cell array form for sequential ...
        training
    train_img = train_set(2:end,:);
    train_label = train_set(1,:);
    test_img = test_set(2:end,:);
    test_label = test_set(1,:);
    images_c = num2cell(train_img, 1);
    labels_c = num2cell(train_label, 1);
    train_num = length(train_label);
    test_num = length(test_label);
    % 2. Construct and configure the MLP
    net = patternnet(n);
    net.divideFcn = 'dividetrain'; % input for training only
    net.performParam.regularization = 0.25; % regularization ...
        strength
    net.trainFcn = 'trainscg'; % 'trainrp' 'traingdx'
    net.trainParam.epochs = epochs;
    accu_train = zeros(epochs,1); % record accuracy on ...
        training set of each epoch
    accu_test = zeros(epochs,1); % record accuracy on ...
        validation set of each epoch
    % 3. Train the network in sequential mode
    for i = 1 : epochs
        display(['Epoch: ', num2str(i)])
        idx = randperm(train_num); % shuffle the input
        net = adapt(net, images_c(:,idx), labels_c(:,idx));
        pred_label_train = net(train_img);
```

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