Faculty of Engineering

Module	EE5904
Name	Zavier Ong Jin Jie
Matriculation No.	A0138993L

Group ID

Group Id = mod(93, 3) +1 = **1**

Q1 Rosenbrock's Valley Problem

Steepest (Gradient) descent

Number of iterations: 11411

Plot out the function value as it approaches the global minimum:

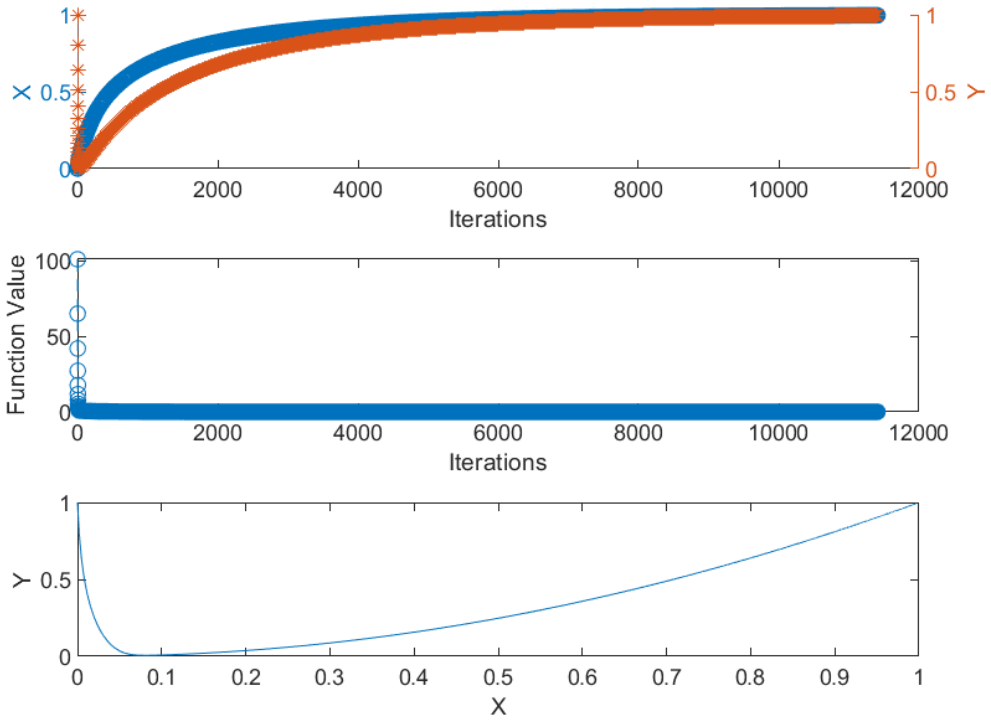

Figure 1. Plot of Q1a

What would happen if a larger rate (0.5) is used?

The function value would explode very quickly towards infinity. It would not be able to converge and reach the condition where the cost lower than the threshold.

	1	2
1	0	1
2	1	-99
3	-19999	9901
4	1.5997e+15	3.9995e+10
5	-8.1877e+47	2.5591e+32
6	1.0978e+146	6.7039e+97
7	-Inf	1.2051e+294
8	NaN	Inf
9	NaN	NaN
10	NaN	NaN
11	NaN	NaN
12	NaN	NaN
13	NaN	NaN
14	NaN	NaN

Figure 2. Learning rate of 0.5

As shown in Figure 2, we can see that the values of both X and Y explodes and quickly approaches infinity and negative infinity.

Newton's method

Number of iterations: 5

Plot out the function value as it approaches the global minimum:

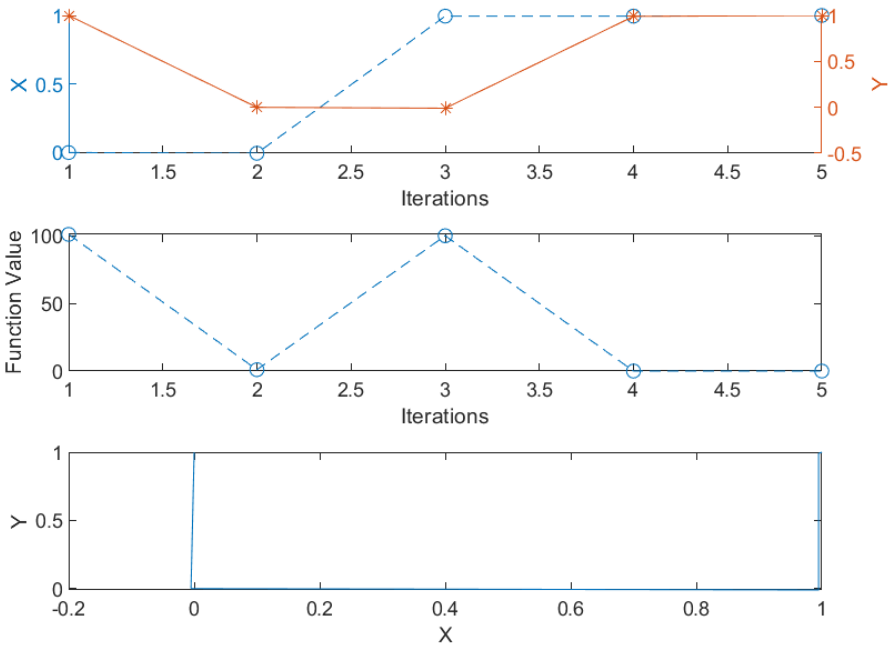

Figure 3. Plot of Q1b

Q2 Function Approximation

Sequential mode

Figure 4. Sequential mode MLP 1-n-1 plots (n = 1:10, 20, 50, 100)

Figure 4 above shows the plot of the outputs of the MLP in sequential mode for the test samples after training (orange) and comparing them to the desired output (blue). From left-to-right, up-to-down, is the order of the number of hidden neurons used in the MLP from 1 to 10, 20, 50 and 100.

Determine whether it is under-fitting, proper fitting, or over-fitting:

Under-Fitting	Proper Fitting	Over-Fitting
1, 2, 3, 4, 5, 6, 7, 8, 9	10, 20	50, 100

Identify the minimal number of hidden neurons from the experiments:

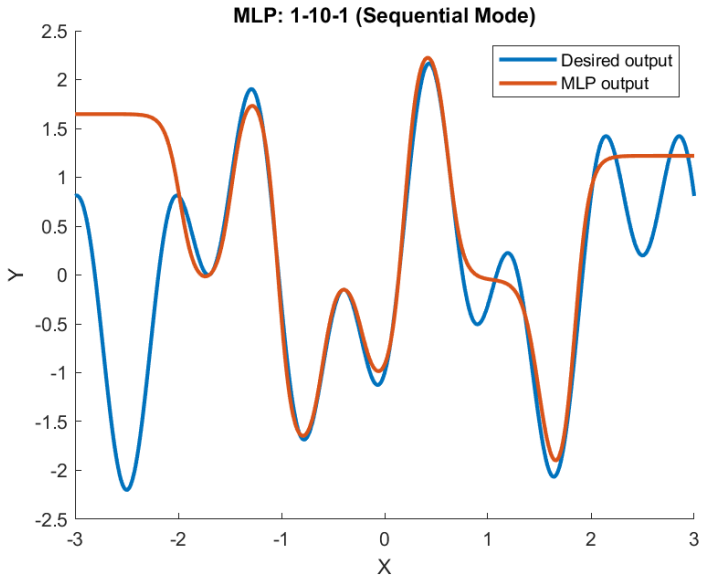

Figure 5. MLP 1-10-1

As seen in Figure 5, looking at the blue curve from x ranging from -2 to 2, we can count that there are 10 segments for this target function. Following Figure 4's set of plots, we can see that the MLP output start to become very similar to the desired output when the hidden neuron count is at 10. Therefore, it is consistent with the guideline given in the lecture slides.

Compute outputs of MLP when x=-3 and x=3

```
(x = -3)  desired: 8.090170e-01 output: 1.587239e+00
(x = 3)   desired: 8.090170e-01 output: 1.190783e+00
```

Figure 6. Output of 1-10-1 Sequential MLP when x=-3 and 3.

As seen in both Figure 5 and 6, MLP is no longer able to make reasonable predictions outside of the domain of the input limited by the training set.

Batch mode (trainlm)

Figure 7 Batch mode MLP (trianlm) 1-n-1 plots (n = 1:10, 20, 50 100)

Similar to Figure 4, Figure 7 above shows the plot of the outputs of the MLP in batch mode (trainlm) for the test samples after training (orange) and comparing them to the desired output (blue). From left-to-right, up-to-down, is the order of the number of hidden neurons used in the MLP from 1 to 10, 20, 50 and 100.

Determine whether it is under-fitting, proper fitting, or over-fitting:

Under-Fitting	Proper Fitting	Over-Fitting
1, 2, 3, 4, 5, 6, 7, 8	9, 10, 20, 50	100

Identify the minimal number of hidden neurons from the experiments: 9.

Compute outputs of MLP when x=-3 and x=3

```
(x = -3) desired: 8.090170e-01 output: 9.457045e-01
(x = 3)  desired: 8.090170e-01 output: 2.226221e+00
```

Figure 8 Output of 1-10-1 Batch mode MLP (trainlm) when x=-3 and 3.

As seen in Figure 8, MLP is no longer able to make reasonable predictions outside of the domain of the input limited by the training set.

Batch mode (trainbr)

Figure 9 Batch mode MLP (trianbr) 1-n-1 plots (n = 1:10, 20, 50 100)

Similar to Figure 4, Figure 9 above shows the plot of the outputs of the MLP in batch mode (trainbr) for the test samples after training (orange) and comparing them to the desired output (blue). From left-to-right, up-to-down, is the order of the number of hidden neurons used in the MLP from 1 to 10, 20, 50 and 100.

Determine whether it is under-fitting, proper fitting, or over-fitting:

Under-Fitting	Proper Fitting	Over-Fitting
1, 2, 3, 4, 5, 6	7, 8, 9, 10, 20, 50, 100	-

Identify the minimal number of hidden neurons from the experiments: 7.

Compute outputs of MLP when x=-3 and x=3

```
(x = -3) desired: 8.090170e-01 output: 9.457045e-01
(x = 3)  desired: 8.090170e-01 output: 2.226221e+00
```

Figure 10 Output of 1-10-1 Batch mode MLP (trainbr) when x=-3 and 3.

As seen in Figure 10, MLP is no longer able to make reasonable predictions outside of the domain of the input limited by the training set.

Q3 Facial Attribute Recognition (Gender Classification)

a. Plot and analyse the label distribution

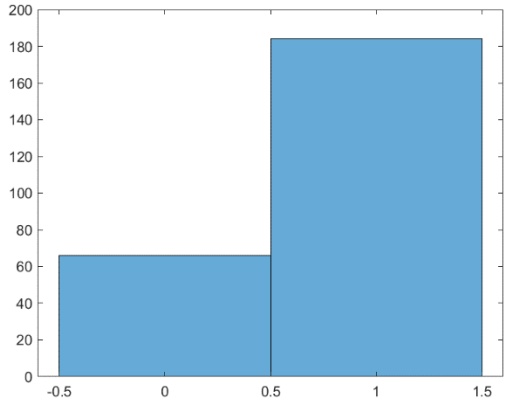

Figure 11 Test label distribution

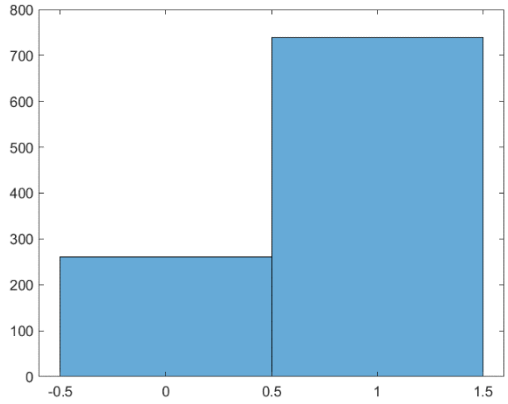

Figure 12 Train label distribution

According to the histogram in Figure 11, there are a total of 66 females (0) and 184 males (1) in the testing dataset.

According to the histogram in Figure 12, there are a total of 261 females (0) and 739 males (1) in the training dataset.

As seen in both distributions, we can see that the distributions are generally skewed towards the male population.

b. Rosenblatt's perceptron

```
Q3b. Rosenblatt's Perceptron
training accuracy: 1.000000, validation accuracy: 0.848000
```

Figure 13. Classification accuracy of Perceptron

As shown in Figure 13, we can see that the training accuracy is 100% while the validation accuracy is 84.8%. Even though the validation accuracy is quite high, it may be slightly overfitted to the training data.

c. Rosenblatt's perceptron with PCA

```
Q3c. Rosenblatt's Perceptron with PCA
training accuracy: 1.000000, validation accuracy: 0.740000
```

Figure 14. Classification accuracy of Perceptron of PCA

As shown in Figure 14, we can observe that the training accuracy is 100% while validation accuracy is only 74%. We can observe an almost 10% drop in accuracy when applying PCA before building the perceptron model as compared to just using perceptron to classify the dataset.

This could be because PCA is an algorithm that does not consider the prediction target. PCA will treat the features with large variances as important features. However, it may also be possible that features with large variances may have nothing to do with predicting the target. As a result, PCA may produce useless features as the main principal component and eliminate useful features when reducing dimensions in PCA.

d. MLP (batch mode)

```
Q3d.
MLP batch mode (1 hidden neurons)
training accuracy: 0.927000, validation accuracy: 0.848000
MLP batch mode (2 hidden neurons)
training accuracy: 0.934000, validation accuracy: 0.880000
MLP batch mode (3 hidden neurons)
training accuracy: 0.905000, validation accuracy: 0.872000
MLP batch mode (4 hidden neurons)
training accuracy: 0.931000, validation accuracy: 0.884000
MLP batch mode (5 hidden neurons)
training accuracy: 0.914000, validation accuracy: 0.868000
MLP batch mode (6 hidden neurons)
training accuracy: 0.907000, validation accuracy: 0.856000
MLP batch mode (7 hidden neurons)
training accuracy: 0.963000, validation accuracy: 0.880000
MLP batch mode (8 hidden neurons)
training accuracy: 0.960000, validation accuracy: 0.856000
MLP batch mode (9 hidden neurons)
training accuracy: 0.917000, validation accuracy: 0.888000
MLP batch mode (10 hidden neurons)
training accuracy: 0.938000, validation accuracy: 0.904000
MLP batch mode (20 hidden neurons)
training accuracy: 0.939000, validation accuracy: 0.860000
MLP batch mode (50 hidden neurons)
training accuracy: 0.936000, validation accuracy: 0.888000
MLP batch mode (100 hidden neurons)
training accuracy: 0.930000, validation accuracy: 0.880000
```

Figure 15 Batch mode MLP 1-n-1 plots (n = 1:10, 20, 50 100)

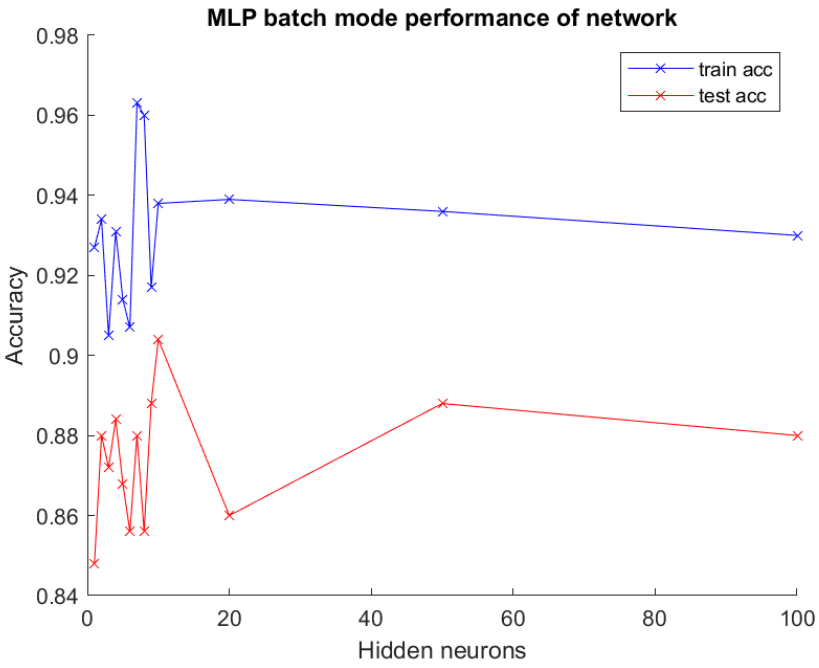

Figure 16. Plot of classification accuracy for batch mode MLP

As shown in Figure 15 and 16, most models can produce a relatively high validation accuracy. However, we can observe that 10 hidden neurons seem to produce the best result where the validation accuracy is the highest.

e. MLP (sequential mode)

```
Q3e.
MLP sequential mode (1 hidden neurons)
training accuracy: 0.739000, validation accuracy: 0.736000
MLP sequential mode (2 hidden neurons)
training accuracy: 0.762000, validation accuracy: 0.740000
MLP sequential mode (3 hidden neurons)
training accuracy: 0.864000, validation accuracy: 0.844000
MLP sequential mode (4 hidden neurons)
training accuracy: 0.803000, validation accuracy: 0.800000
MLP sequential mode (5 hidden neurons)
training accuracy: 0.875000, validation accuracy: 0.820000
MLP sequential mode (6 hidden neurons)
training accuracy: 0.862000, validation accuracy: 0.816000
MLP sequential mode (7 hidden neurons)
training accuracy: 0.862000, validation accuracy: 0.836000
MLP sequential mode (8 hidden neurons)
training accuracy: 0.879000, validation accuracy: 0.820000
MLP sequential mode (9 hidden neurons)
training accuracy: 0.879000, validation accuracy: 0.816000
MLP sequential mode (10 hidden neurons)
training accuracy: 0.910000, validation accuracy: 0.812000
MLP sequential mode (20 hidden neurons)
training accuracy: 0.915000, validation accuracy: 0.820000
MLP sequential mode (50 hidden neurons)
training accuracy: 0.939000, validation accuracy: 0.852000
MLP sequential mode (100 hidden neurons)
training accuracy: 0.976000, validation accuracy: 0.844000
```

Figure 17 Sequential mode MLP 1-n-1 plots (n = 1:10, 20, 50 100)

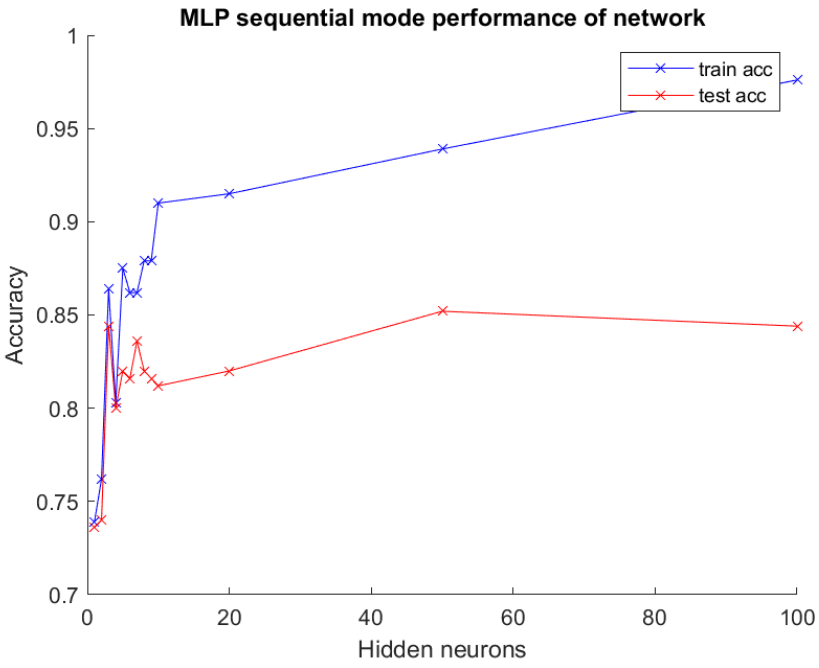

Figure 18 Plot of classification accuracy for sequential mode MLP

As shown in figure 17 and 18, we can observe that batch mode MLP is generally more accurate compared to sequential mode MLP. Though the training accuracy in sequential MLP is higher, its validation accuracy is lower than batch mode MLP peaking at about 85.2% while batch mode MLP peaks at 90.4%. In addition

to that, batch mode MLP is also much faster compared to sequential mode MLP. Therefore, I would recommend using batch mode MLP when training the network.

f. Justifying eye locations at certain location in all images

Yes. It is necessary to align by placing the eyes at the same locations. This is because one of the drawbacks of MLP is that it is not translation invariant, meaning that the MLP will react differently to an image and its shifted version. Therefore, when the eyes of the human images appear in a certain location of the image, the MLP will assume that the eyes will always appear in this section of the image.

Appendix

Q1a

```
clc;
clear;
close all;

x = 0;
y = 1;

cost = (1-x)^2 + 100*(y-x^2)^2;
max_epoch = 100000;
all_cost = [cost];
all_xy = [x,y;];
iteration = 0;
eta = 0.001; %learing_rate
threshold = 1e-5;

for epoch = 1:max_epoch
    iteration = iteration+1;
    %gradient descent
    old_x = x;
    old_y = y;
    x = old_x - eta*(2*(old_x-1) + 400*old_x*(old_x^2 -old_y));
    y = old_y - eta*(200*(old_y-old_x^2));
    cost = (1-x)^2 + 100*(y-x^2)^2;

    all_cost = [all_cost; cost];
    all_xy = [all_xy; x,y;];
    if (cost < threshold)
        break;
    end
end

num_iter = [1:iteration+1];
fprintf("Q1a. Number of iterations: %d\n", length(num_iter));
% plot
fig = figure();
subplot(3, 1, 1);
hold on;
yyaxis left;
plot(num_iter, all_xy(:,1),'--o');
ylabel('X');
yyaxis right;
plot(num_iter, all_xy(:,2),'*-');
ylabel('Y');
xlabel('Iterations');

subplot(3, 1, 2)
plot(num_iter,all_cost,'--o')
ylabel('Function Value')
xlabel('Iterations')

subplot(3, 1, 3)
plot(all_xy(:,1), all_xy(:, 2));
```

```
ylabel('Y');
xlabel('X');
saveas(fig,sprintf('q1_images/steepest_descent.png'));
```

Q1b

```
clc
clear
close all

x = 0;
y = 1;

cost = (1-x)^2 + 100*(y-x^2)^2;
max_epoch = 100000;
all_cost = [cost];
all_xy = [x,y;];
iteration = 0;
threshold = 1e-5;

for epoch = 1:max_epoch
    iteration = iteration+1;
    g = [(2*(x-1)+400*x*(x^2 -y)); (200*(y-x^2))];
    H = [(1200*x^2-400*y+2), (-400*x); (-400*x), 200];
    w = [x;y]- H\g; %inv(H)*g = H\g

    x = w(1);
    y = w(2);
    cost = (1-x)^2 + 100*(y-x^2)^2;
    all_cost = [all_cost; cost];
    all_xy = [all_xy; x,y;];
    if (cost < threshold)
        break;
    end
end

num_iter = [1:iteration+1];
fprintf("Q1a. Number of iterations: %d\n", length(num_iter));
% plot
fig = figure();
subplot(3, 1, 1);
hold on;
yyaxis left;
plot(num_iter, all_xy(:,1),'--o');
ylabel('X');
yyaxis right;
plot(num_iter, all_xy(:,2),'*-');
ylabel('Y');
xlabel('Iterations');

subplot(3, 1, 2)
plot(num_iter,all_cost,'--o')
ylabel('Function Value')
xlabel('Iterations')

subplot(3, 1, 3)
plot(all_xy(:,1), all_xy(:, 2));
ylabel('Y');
xlabel('X');
saveas(fig,sprintf('q1_images/newton.png'));
```

Q2a

```
clc
clear
close all

x_train = -2:0.05:2;
x_test = -3:0.01:3;

y_train = mlp_approx_fn(x_train);
y_test = mlp_approx_fn(x_test);

hidden_neurons = [1:10, 20, 50, 100];
%hidden_neurons = [10];
epochs = 200;
for i = hidden_neurons
    % training
    net = train_seq(i,x_train,y_train,81,epochs);
    % test
    y_out = net(x_test);
    % plot
    fig = figure();
    hold on
    plot(x_test,y_test,'-','LineWidth',2)
    plot(x_test,y_out,'-','LineWidth',2)
    ylim([-2.5 2.5])
    legend('Desired output', 'MLP output')
    title(sprintf('MLP: 1-%d-1 (Sequential Mode)', i))
    ylabel('Y')
    xlabel('X')
    hold off
    saveas(fig,sprintf('q2_images/sequential_%d.png',i));

    %outputs of MLP x=-3 and x-3
    if (i==10)
        %x=-3
        y_desired = mlp_approx_fn(-3);
        y_out = net(-3);
        fprintf('(x = -3) desired: %d output: %d\n', y_desired, y_out);
        %x=3
        y_desired = mlp_approx_fn(3);
        y_out = net(3);
        fprintf('(x = 3)  desired: %d output: %d\n', y_desired, y_out);
    end
end


function y = mlp_approx_fn(x)
    y = 1.2*sin(pi*x) - cos(2.4*pi*x);
end

function net = train_seq( n, images, labels, train_num, epochs )
    % 1. Change the input to cell array form for sequential training
    images_c = num2cell(images, 1);
    labels_c = num2cell(labels, 1);
```

```
    % 2. Construct and configure the MLP
    net = fitnet(n);
    net.divideFcn = 'dividetrain'; % input for training only
    net.performParam.regularization = 0.25; % regularization strength
    net.trainFcn = 'traingdx'; % 'trainrp' 'traingdx'
    net.trainParam.epochs = epochs;

    % 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));
    end
end
```

Q2b

```
clc
clear
close all

x_train = -2:0.05:2;
x_test = -3:0.01:3;

y_train = mlp_approx_fn(x_train);
y_test = mlp_approx_fn(x_test);

hidden_neurons = [1:10, 20, 50, 100];
%hidden_neurons = [10];
epochs = 10;
for i = hidden_neurons
    % training
    net = train_batch(i,x_train,y_train);
    % test
    y_out = net(x_test);
    % plot
    fig = figure();
    hold on
    plot(x_test,y_test,'-','LineWidth',2)
    plot(x_test,y_out,'-','LineWidth',2)
    ylim([-2.5 2.5])
    legend('Desired output', 'MLP output')
    title(sprintf('MLP: 1-%d-1 (Batch Mode (trainlm))', i))
    ylabel('Y')
    xlabel('X')
    hold off
    saveas(fig,sprintf('q2_images/lm_%d.png',i));

    %outputs of MLP x=-3 and x-3
    if (i==10)
        %x=-3
        y_desired = mlp_approx_fn(-3);
        y_out = net(-3);
        fprintf('(x = -3) desired: %d output: %d\n', y_desired, y_out);
        %x=3
        y_desired = mlp_approx_fn(3);
        y_out = net(3);
        fprintf('(x = 3)  desired: %d output: %d\n', y_desired, y_out);
    end
end


function y = mlp_approx_fn(x)
    y = 1.2*sin(pi*x) - cos(2.4*pi*x);
end

function net = train_batch( n, images, labels)
    % 1. Change the input to cell array form for sequential training
    images_c = num2cell(images, 1);
    labels_c = num2cell(labels, 1);
```

```
    % 2. Construct and configure the MLP
    net = feedforwardnet(n, 'trainlm');
    net = train(net, images_c, labels_c);
end
```

Q2c

```
clc
clear
close all

x_train = -2:0.05:2;
x_test = -3:0.01:3;

y_train = mlp_approx_fn(x_train);
y_test = mlp_approx_fn(x_test);

hidden_neurons = [1:10, 20, 50, 100];
%hidden_neurons = [10];
epochs = 10;
for i = hidden_neurons
    % training
    net = train_batch(i,x_train,y_train);
    % test
    y_out = net(x_test);
    % plot
    fig = figure();
    hold on
    plot(x_test,y_test,'-','LineWidth',2)
    plot(x_test,y_out,'-','LineWidth',2)
    ylim([-2.5 2.5])
    legend('Desired output', 'MLP output')
    title(sprintf('MLP: 1-%d-1 (Batch Mode (trainbr))', i))
    ylabel('Y')
    xlabel('X')
    hold off
    saveas(fig,sprintf('q2_images/br_%d.png',i));

    %outputs of MLP x=-3 and x-3
    if (i==10)
        %x=-3
        y_desired = mlp_approx_fn(-3);
        y_out = net(-3);
        fprintf('(x = -3) desired: %d output: %d\n', y_desired, y_out);
        %x=3
        y_desired = mlp_approx_fn(3);
        y_out = net(3);
        fprintf('(x = 3)  desired: %d output: %d\n', y_desired, y_out);
    end
end


function y = mlp_approx_fn(x)
    y = 1.2*sin(pi*x) - cos(2.4*pi*x);
end

function net = train_batch( n, images, labels)
    % 1. Change the input to cell array form for sequential training
    images_c = num2cell(images, 1);
    labels_c = num2cell(labels, 1);
```

```
    % 2. Construct and configure the MLP
    net = feedforwardnet(n, 'trainbr');
    net.trainParam.lr = 0.01;
    net = train(net, images_c, labels_c);
end
```

Q3

```
clc
clear
close all

%A0138993L
group_id = mod(93, 3)+1;
train_path = 'Face Database\TrainImages\';
test_path = 'Face Database\TestImages\';
x_ext = '*.jpg';
y_ext = '*.att';

train_x_jpg = dir(sprintf('%s%s', train_path, x_ext));
test_x_jpg = dir(sprintf('%s%s', test_path, x_ext));

train_y_att = dir(sprintf('%s%s', train_path, y_ext));
test_y_att = dir(sprintf('%s%s', test_path, y_ext));

%load img into grayscale
train_x = zeros(10201, length(train_x_jpg));
test_x = zeros(10201, length(test_x_jpg));

for i=1:length(train_x_jpg)
    img = imread(sprintf('%s%s', train_path, train_x_jpg(i).name));
    img_size = size(img);
    if (img_size(1) ~= 101 || img_size(2) ~= 101)
        img = imresize(img, [101, 101]);
    end
    grayscale = rgb2gray(img);
    train_x(:, i) = grayscale(:);
end
for i=1:length(test_x_jpg)
    img = imread(sprintf('%s%s', test_path, test_x_jpg(i).name));
    img_size = size(img);
    if (img_size(1) ~= 101 || img_size(2) ~= 101)
        img = imresize(img, [101, 101]);
    end
    grayscale = rgb2gray(img);
    test_x(:, i) = grayscale(:);
end

%extracting ground-truth labels
train_y = zeros(length(train_y_att),1);
test_y = zeros(length(test_y_att), 1);

for i=1:length(train_y)
    L = load(sprintf('%s%s', train_path, train_y_att(i).name));
    train_y(i,:) = L(group_id);
end
for i=1:length(test_y)
    L = load(sprintf('%s%s', test_path, test_y_att(i).name));
    test_y(i,:) = L(group_id);
end

%%%%%%%%%%%%%%%%%%%    3a    %%%%%%%%%%%%%%%%%%%
```

```
fig_test = figure();
h_test = histogram(test_y)
saveas(fig_test, 'test_label_distribution.png');
fig_train = figure();
h_train = histogram(train_y)
saveas(fig_train, 'train_label_distribution.png');

 train_y = train_y.';
 test_y = test_y.';
%%%%%%%%%%%%%%%%%%%   3b   %%%%%%%%%%%%%%%%%%%

net = perceptron;
%net.performFcn = 'mse';
[net, tr] = train(net, train_x, train_y);

%training accuracy
train_y_pred = net(train_x);
train_acc = mean(train_y_pred == train_y);
%test accuracy
test_y_pred = net(test_x);
test_acc = mean(test_y_pred == test_y);

fprintf("Q3b. Rosenblatt's Perceptron\ntraining accuracy: %f, validation
accuracy: %f\n", ...
    train_acc, test_acc);

%%%%%%%%%%%%%%%%%%%   3c   %%%%%%%%%%%%%%%%%%%
[coeff, score, ~, ~, ~, mu] = pca(train_x.');

net = perceptron;
net.performFcn = 'mse';
[net, tr] = train(net, score.', train_y);

train_y_pred = net(score.');
train_acc = mean(train_y_pred == train_y);

demean_test = test_x.'-mu;
score_test = demean_test*coeff;
test_y_pred = net(score_test.');
test_acc = mean(test_y_pred == test_y);

fprintf("Q3c. Rosenblatt's Perceptron with PCA\ntraining accuracy: %f,
validation accuracy: %f\n", ...
     train_acc, test_acc);

%%%%%%%%%%%%%%%%%%%   3d   %%%%%%%%%%%%%%%%%%%
fprintf("Q3d.\n");
train_acc_list = zeros(13, 1);
test_acc_list = zeros(13, 1);
hidden_neurons = [1:10, 20, 50, 100];
idx_count = 1;
for i = hidden_neurons
    net = patternnet(i);
    net = train(net, train_x, train_y);
```

```
    train_y_pred = net(train_x);
    train_y_pred = train_y_pred > 0.5;
    train_acc = mean(train_y_pred == train_y);
    train_acc_list(idx_count) = train_acc;

    test_y_pred = net(test_x);
    test_y_pred = test_y_pred > 0.5;
    test_acc = mean(test_y_pred == test_y);
    test_acc_list(idx_count) = test_acc;
    fprintf("MLP batch mode (%d hidden neurons)\ntraining accuracy: %f,
validation accuracy: %f\n", ...
        i, train_acc, test_acc);
    idx_count = idx_count+1;
end

fig = figure();
hold on
plot(hidden_neurons, train_acc_list, '-bx');
plot(hidden_neurons, test_acc_list, '-rx');
legend('train acc', 'test acc');
xlabel('Hidden neurons');
ylabel('Accuracy');
title('MLP batch mode performance of network');
saveas(fig, 'q3d_performance.png');

%%%%%%%%%%%%%%%%%%%   3e   %%%%%%%%%%%%%%%%%%
fprintf("Q3e.\n");
train_acc_list = zeros(13, 1);
test_acc_list = zeros(13, 1);
hidden_neurons = [1:10, 20, 50, 100];
idx_count = 1;
epochs = 10;
for i = hidden_neurons
    train_x_c = num2cell(train_x, 1);
    train_y_c = num2cell(train_y, 1);

    net = patternnet(i);
    net.divideFcn = 'dividetrain'; % input for training only
    net.performParam.regularization = 0.25; % regularization strength
    net.trainFcn = 'traingdx'; % 'trainrp' 'traingdx'
    net.trainParam.epochs = epochs;
    for j = 1 : epochs
        %display(['Epoch: ', num2str(j)])
        idx = randperm(1000); % shuffle the input
        net = adapt(net, train_x_c(:,idx), train_y_c(:,idx));
    end

    train_y_pred = net(train_x);
    train_y_pred = train_y_pred > 0.5;
    train_acc = mean(train_y_pred == train_y);
    train_acc_list(idx_count) = train_acc;

    test_y_pred = net(test_x);
    test_y_pred = test_y_pred > 0.5;
    test_acc = mean(test_y_pred == test_y);
    test_acc_list(idx_count) = test_acc;
```

```
    fprintf("MLP sequential mode (%d hidden neurons)\ntraining accuracy: %f,
validation accuracy: %f\n", ...
        i, train_acc, test_acc);
    idx_count = idx_count+1;
end

fig = figure();
hold on
plot(hidden_neurons, train_acc_list, '-bx');
plot(hidden_neurons, test_acc_list, '-rx');
legend('train acc', 'test acc');
xlabel('Hidden neurons');
ylabel('Accuracy');
title('MLP sequential mode performance of network');
saveas(fig, 'q3e_performance.png');
```