

|  |  |
| --- | --- |
| **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:

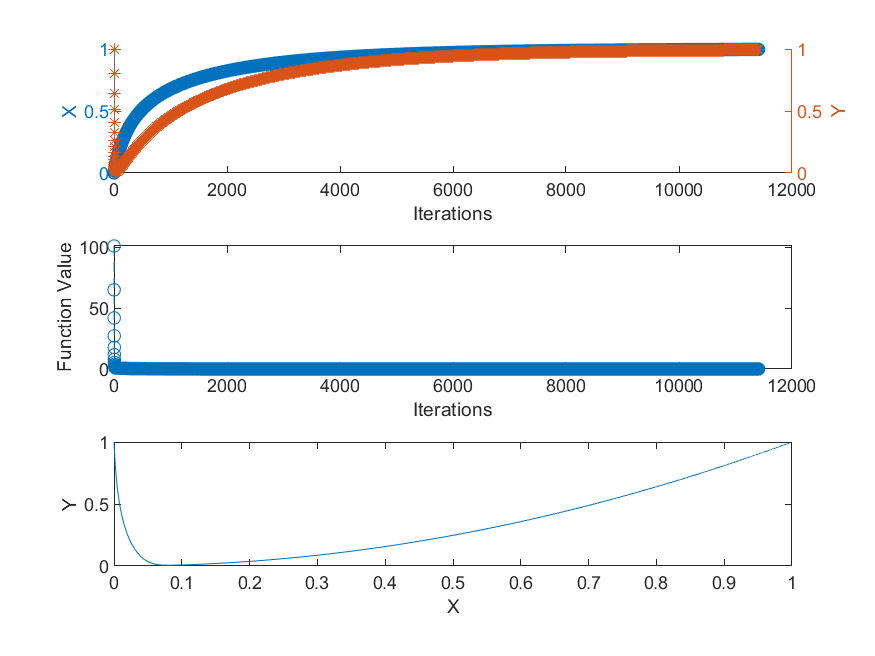


Figure 1. Plot of Q1a

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.

|  |  |
| --- | --- |
| 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:

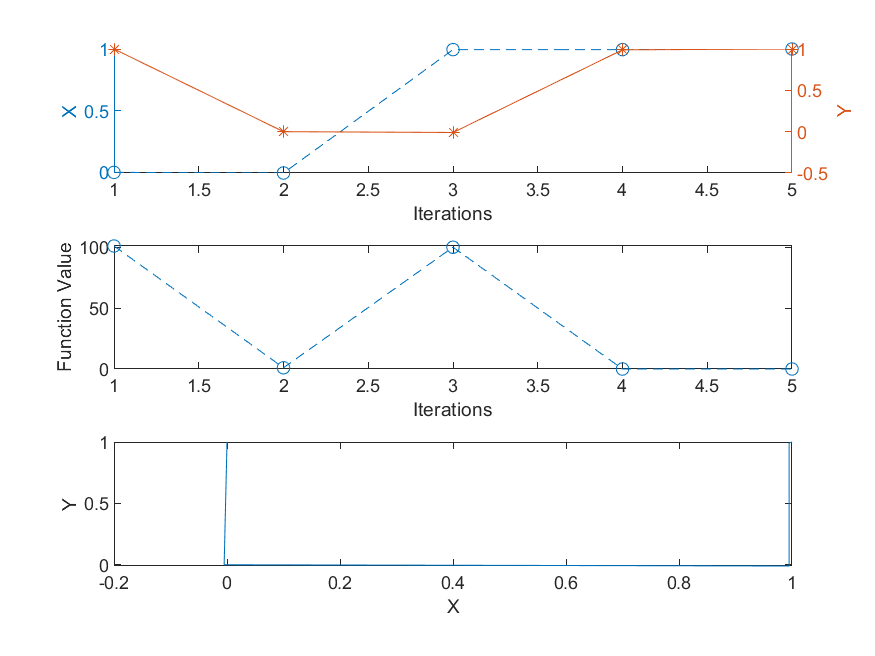


Figure 3. Plot of Q1b

# 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:

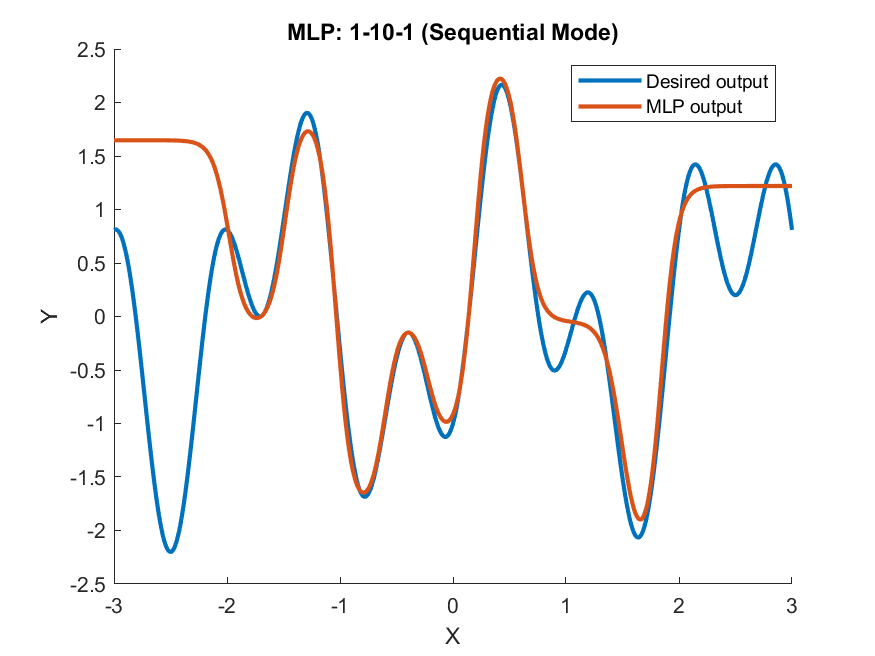


Figure 5. MLP 1-10-1

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



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



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



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)

## 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.

## Rosenblatt’s perceptron



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.

## Rosenblatt’s perceptron with PCA



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.

## MLP (batch mode)

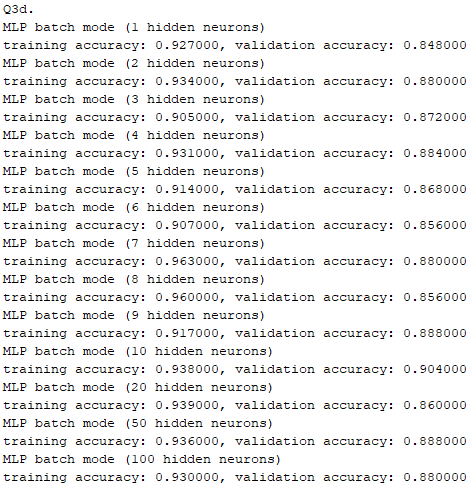


Figure 15 Batch mode MLP 1-n-1 plots (n = 1:10, 20, 50 100)

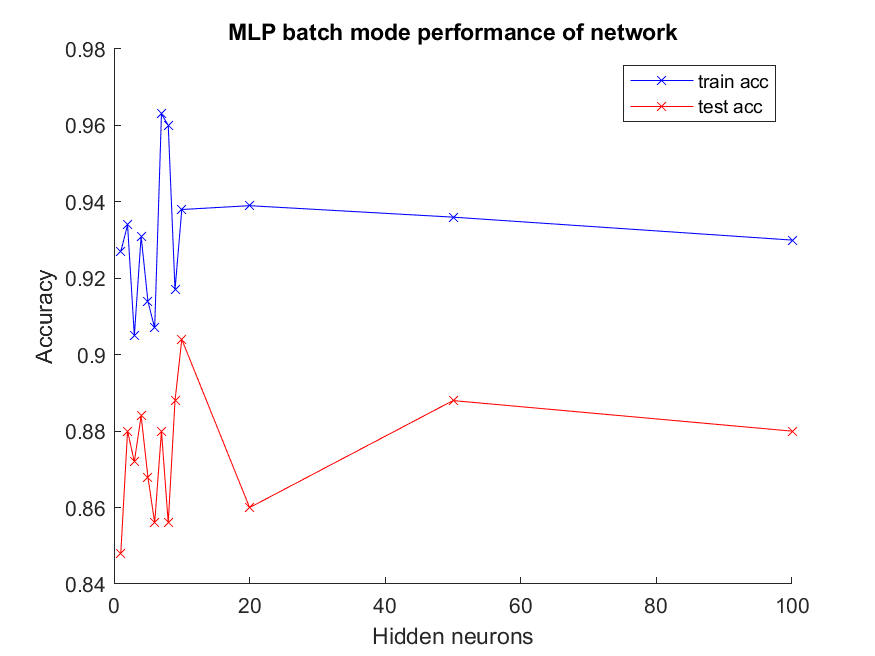


Figure 16. Plot of classification accuracy for batch mode MLP

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.

## MLP (sequential mode)

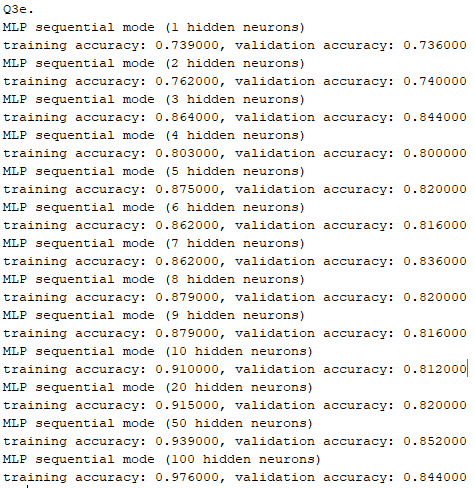


Figure 17 Sequential mode MLP 1-n-1 plots (n = 1:10, 20, 50 100)

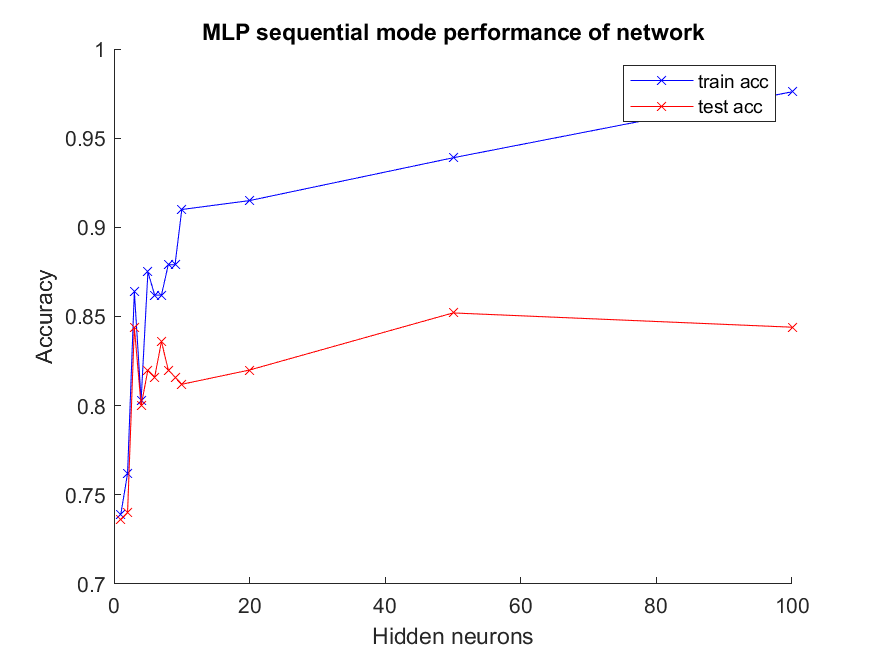


Figure 18 Plot of classification accuracy for sequential mode MLP

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.

## 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');