NUS

National University of Singapore

Name : LUO ZIJIAN

Matric.No: A0224725H

MUSNET: E0572844

Subject: NEURAL NETWORKS

Assignment: HOMEWORK TWO

Solution 1

Based on the contents in question one, we can easily to get these conclusions

$$\begin{cases} 1-x=0 \\ y-x^2=0 \end{cases} \rightarrow \begin{cases} x=1 \\ y=1 \end{cases}$$

a) According to Steepest(Gradient) descent method

$$w(k+1)=w(k)-\eta g(k)$$

we should calculate the gradient vector at the beginning.

$$\frac{\partial f}{\partial x}=2(1-x)(-1)+200(y-x^2)(-2x)=400x^3-400xy+2x-2$$

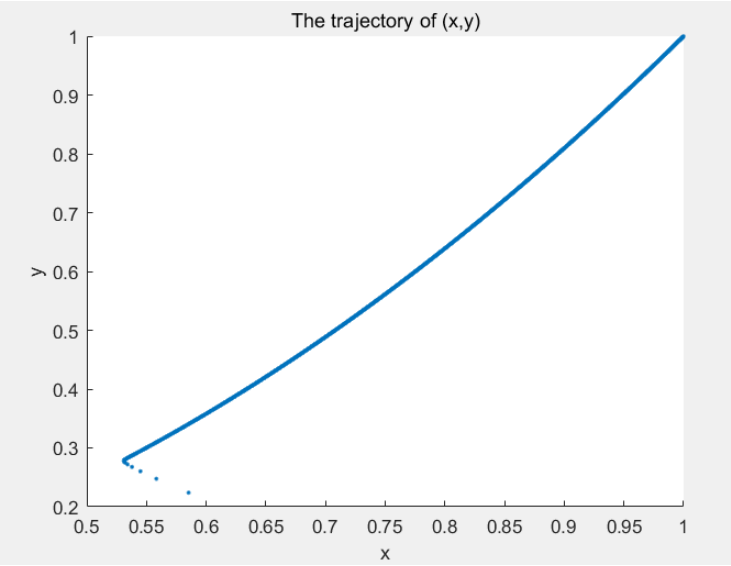
$$\frac{\partial f}{\partial y}=200(y-x^2)=200y-200x^2$$

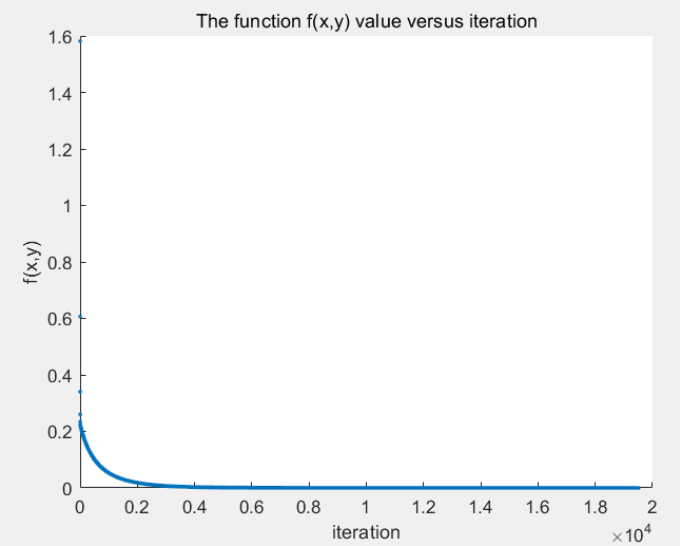
So the gradient vector $g(n)=\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}=\begin{bmatrix} 400x^3-400xy+2x-2 \\ 200y-200x^2 \end{bmatrix}$

Firstly, we set the stop criterion for the descent learning algorithm $\varepsilon=10^{-8}$, which means, if the Rosenbrock's Valley function satisfy the stop criterion, we should stop the iteration.

Secondly, for this experiment, we set the learning rate is $\eta=0.001$

The trajectory of the two parameters x and y is shown.





We easily found the number of iteration is 19518, the value $f = 9.9988 \times 10^{-9}$ approaches the global minimum and the iteration stops. Through this picture, we can notice that the f (x, y) is large at the first 2000 iterations, but after that, the function value decrease very slowly using the gradient descent method.

When we repeat this experiment with learning rate $\varepsilon = 0.5$

dx	NaN
dy	-Inf
eta	0.5000
f	NaN
f_value	*1x1000 double*
i	7
iter_value	*1x1000 double*
x	NaN
x_value	*1x1000 double*
y	Inf
y_value	*1x1000 double*

It turns out that the value of the function goes to infinity. As a result, we can conclude that a large learning rate will make the gradient descent algorithm fail.

```
%para
eta=0.5;%learning rate
i=1;%iteration number

x_value=zeros(1,1000);%init vector
y_value=zeros(1,1000);
f_value=zeros(1,1000);
iter_value=zeros(1,1000);

x=rand;%random x,y
```

```
x=double(x);
y=rand;
y=double(y);
f=(1-x).^2+100.*(y-x.^2).^2;

while f>0.00000001
    x_value(i)=x;%recording trajectory
    y_value(i)=y;
    f_value(i)=f;
    iter_value(i)=i;

    dx=400*x^3-400*x*y+2*x-2;%derivation
    dy=200*y-200*x^2;

    x=x-eta*dx;%update
    y=y-eta*dy;
    f=(1-x).^2+100.*(y-x.^2).^2;

    i=i+1;
end

%plot out (x,y) trajectory
figure(1);
scatter(x_value,y_value,'.');
hold on;
title('The trajectory of (x,y)');
xlabel('x');
ylabel('y');
%plot out function f value
figure(2);
scatter(iter_value,f_value,'.');
hold on;
title('The function f(x,y) value versus iteration');
xlabel('iteration');
ylabel('f(x,y)');
x=double(x);
y=double(y);
```

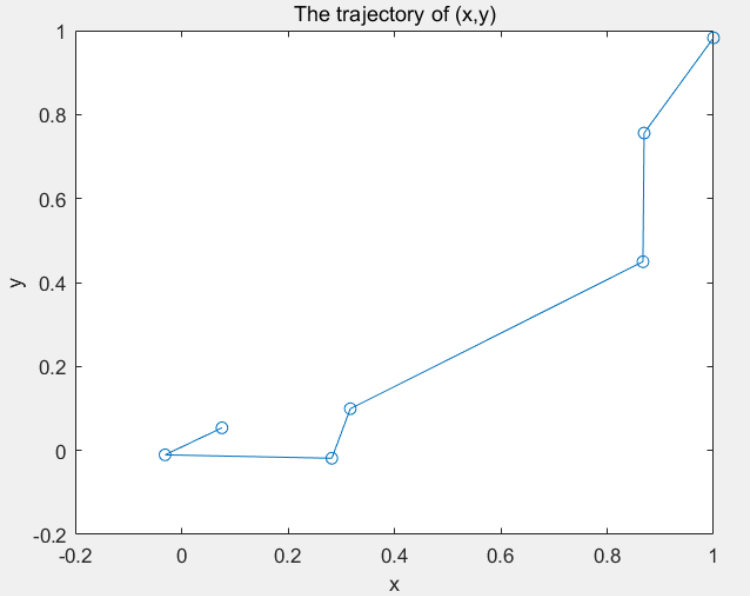
b) According to Newton's method, $\Delta w(n) = -H^{-1}(n)g(n)$

Firstly, we calculate Hessian matrix,

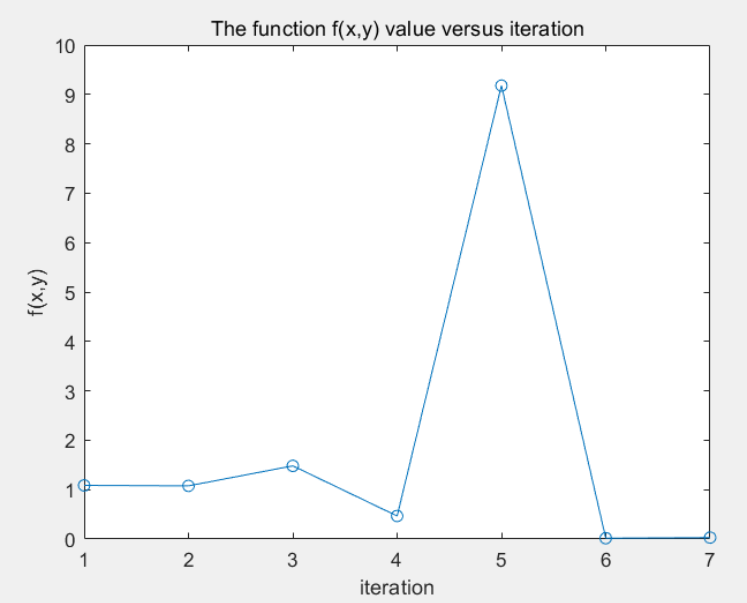
$$\frac{\partial^2 f}{\partial x^2} = 1200x^2 - 400y + 2, \quad \frac{\partial^2 f}{\partial y^2} = 200, \quad \frac{\partial^2 f}{\partial x \partial y} = -400x$$

$$H = \begin{bmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

Firstly, we set the same stop criterion for the Newton's learning algorithm $\varepsilon = 10^{-8}$, it turns out that only 7 iterations are needed to reach the same criterion. The trajectory of the (x, y) is shown.



The trajectory of the function value with the Newton's method is shown.



In conclusion, compared to Steepest gradient method, iterations from 19518 to 7, which decreases dramatically. It is obvious that Newton's method is much faster.

```
%init
close all;clear;clc;


%para
```

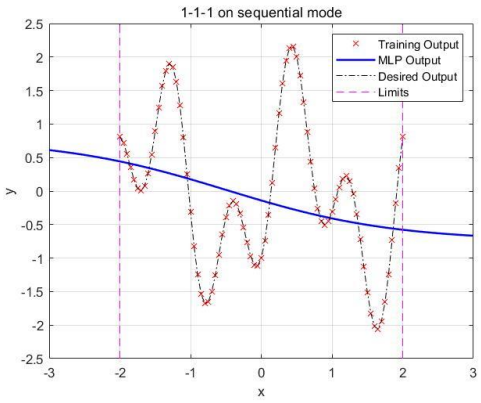
```
i=1;%iteration number

x_value=zeros(1,1000);%init vector
y_value=zeros(1,1000);
f_value=zeros(1,1000);
iter_value=zeros(1,1000);

x=rand;%random x,y
x=double(x);
y=rand;
y=double(y);

dx=400*x^3-400*x*y+2*x-2;%derivation
dy=200*y-200*x^2;
dxx=1200.*x.^2-400.*y+2;
dyy=200;
dxy=-400.*x;
f=(1-x).^2+100.*(y-x.^2).^2;
H=[dxx,dxy;dxy,dyy];

while f>0.00000001
    x_value(i)=x;%recording trajectory
    y_value(i)=y;
    f_value(i)=f;
    iter_value(i)=i;

    dx=400*x^3-400*x*y+2*x-2;%derivation
    dy=200*y-200*x^2;
    dxx=1200.*x.^2-400.*y+2;
    dyy=200;
    dxy=-400.*x;
    H=[dxx,dxy;dxy,dyy];
    h=inv(H);

    x=x-h(1,1).*dx-h(1,2).*dy;%update
    y=y-h(2,1).*dx-h(2,2).*dy;
    f=(1-x).^2+100.*(y-x.^2).^2;
    i=i+1;
end

%plot out (x,y) trajectory
figure(1);
x_value=x_value(1:i-1);
y_value=y_value(1:i-1);
```

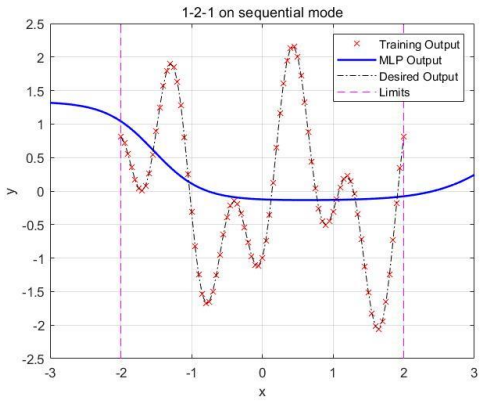
```
plot(x_value,y_value,'o-');
hold on;
title('The trajectory of (x,y)');
xlabel('x');
ylabel('y');
%plot out function f value
figure(2);
iter_value=iter_value(1:i-1);
f_value=f_value(1:i-1);
plot(iter_value,f_value,'o-');
hold on;
title('The function f(x,y) value versus iteration');
xlabel('iteration');
ylabel('f(x,y)');
```

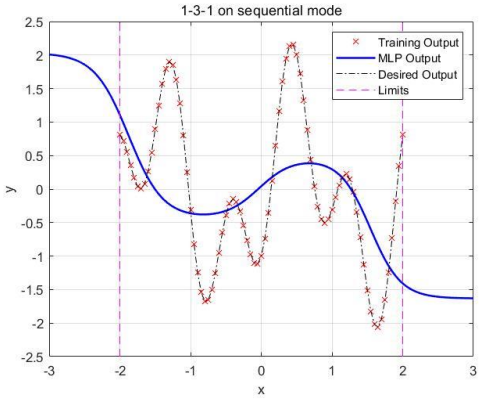
Solution 2

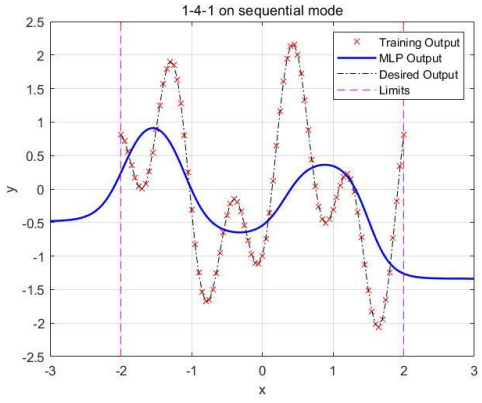
a) Sequential mode with BP algorithm

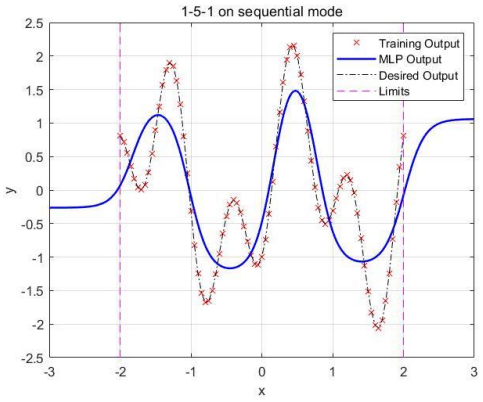
First way, by inputting different n (where n=1,2,3,4,5,6,7,8,9,10,20,50,100), we can get different structures of the MLP: 1-n-1. All the figures are shown as below:

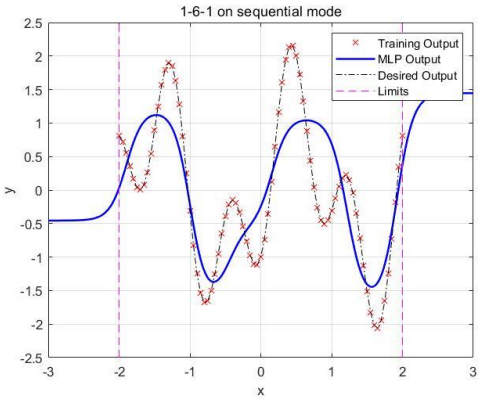


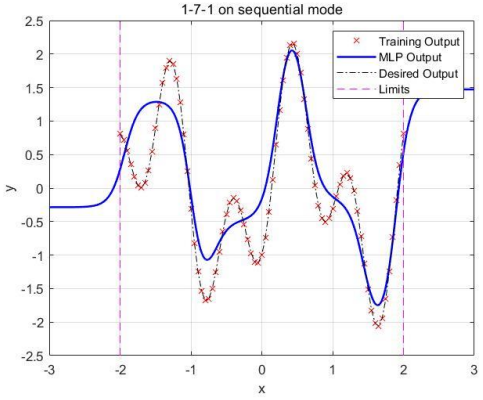


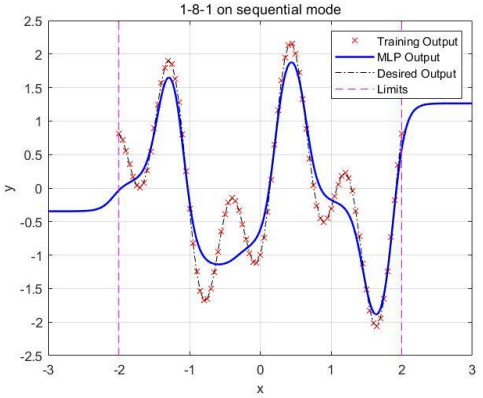


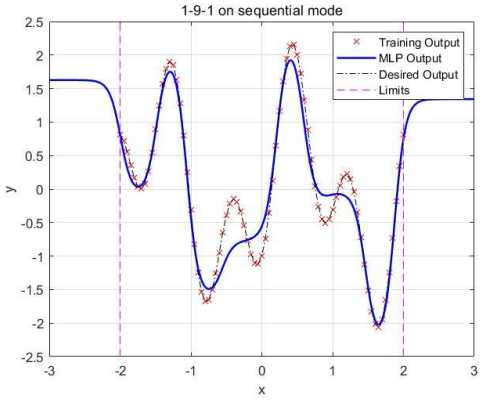


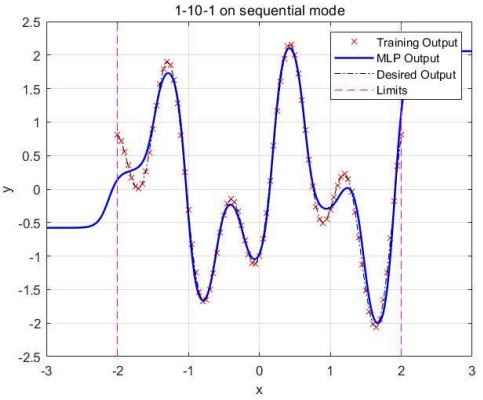

1-5-1 on sequential mode
Training Output
MLP Output
Desired Output
Limits
x
y

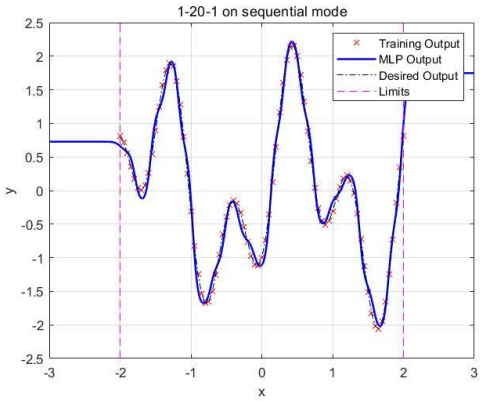

1-6-1 on sequential mode
Training Output
MLP Output
Desired Output
Limits
x
y

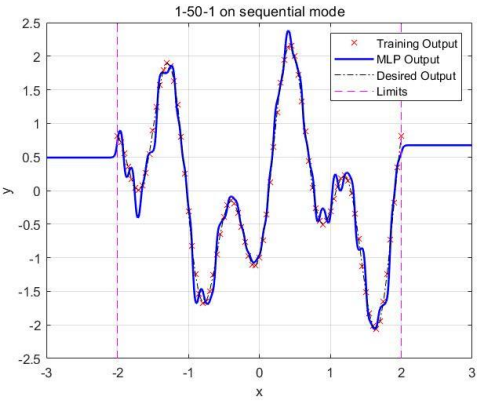

1-7-1 on sequential mode
Training Output
MLP Output
Desired Output
Limits
x
y

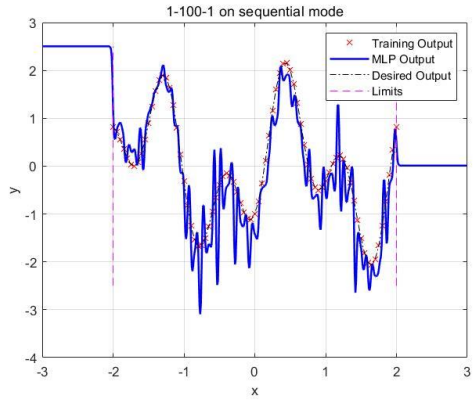

1-8-1 on sequential mode
Training Output
MLP Output
Desired Output
Limits
x
y


1-9-1 on sequential mode
Training Output
MLP Output
Desired Output
Limits
x
y


1-10-1 on sequential mode
Training Output
MLP Output
Desired Output
Limits
x
y


1-20-1 on sequential mode
Training Output
MLP Output
Desired Output
Limits
x
y


1-50-1 on sequential mode
Training Output
MLP Output
Desired Output
Limits
x
y



Through the analyze of the result in these pictures, we can easily conclude that

	Under-fitting	Proper fitting	Over-fitting
n	1-10	20,50	100

Next we use the n=20 MLP to output when x equals 3 and -3,

$$x = -3 \rightarrow y_{ground\_truth} = 0.8441, y_{MLP} = 1.321$$

$$x = 3 \rightarrow y_{ground\_truth} = 0.809, y_{MLP} = 0.604$$

It seems that, if the new sample input is outside the domain of input of the training set, then the result is not correct.

```
syms x y;
y = 1.2*sin(pi*x)-cos(2.4*pi*x);

training_set_input(:) = -2:0.05:2;
training_set_output(1,:) = eval(subs(y,x,training_set_input(1,:)));
train_num=size(training_set_input,2);

test_set_input(:) = -2:0.01:2;
test_set_output(1,:) = eval(subs(y,x,test_set_input(1,:)));

desired_input(:) = -3:0.05:3;
desired_output(:) = eval(subs(y,x,desired_input(1,:)));

for n = [1:10,20,50,100]
    [ net, accu_train, accu_val ] = train_seq(n,
training_set_input, training_set_output,train_num , 0, 150);
    disp(n)
    results = sim(net, -3:0.01:3);
    figure
    plot(training_set_input,training_set_output,'rx') %training
output
```

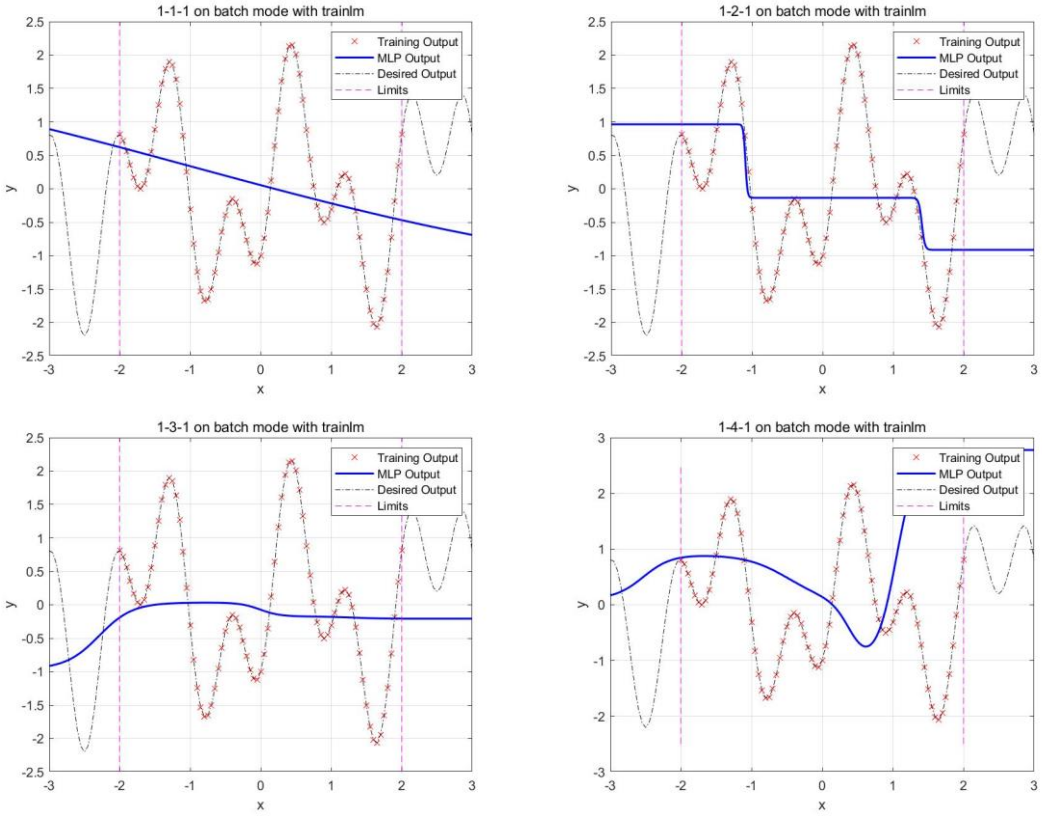
```
    hold on
    plot(-
3:0.01:3,results(1,:),'b','LineWidth',1.5);              %mlp
output
    plot(desired_input,desired_output,'k-.');      %desired output
    line([-2 -2], [-2.5 2.5],'Color','magenta', 'LineStyle','--');
    line([2 2], [-2.5 2.5],'Color','magenta', 'LineStyle','--');
    legend('Training Output', 'MLP Output', 'Desired Output',
'Limits');
    title(strcat("1-", num2str(n), "-1 on sequential mode"));
    xlabel('x');
    ylabel('y');
    grid
    saveas(gcf,num2str(n) ,'jpg');
end

function [ net, accu_train, accu_val ] = train_seq( n, images,
labels, train_num, val_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;
net.inputWeights{1,1}.learnParam.lr = 0.003;
net.layerWeights{2,1}.learnParam.lr = 0.003;
net.biases{1}.learnParam.lr = 0.003;
net.biases{2}.learnParam.lr = 0.003;
accu_train = zeros(epochs,1); % record accuracy on training set of
each epoch
accu_val = 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_train = round(net(images(:,1:train_num))); % predictions
on training set
```

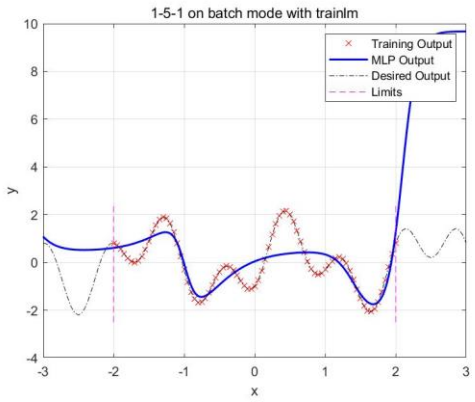
```
    accu_train(i) = 1 - mean(abs(pred_train-labels(1:train_num)));
    pred_val = round(net(images(:,train_num+1:end))); % predictions
on validation set
    accu_val(i) = 1 - mean(abs(pred_val-labels(train_num+1:end)));
    %disp(sim(net,0))
end

end
```

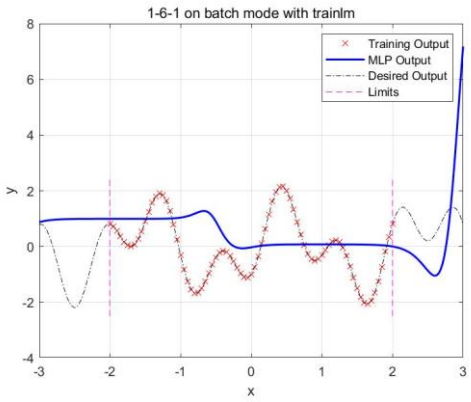
b) batch mode with “trainlm” algorithm

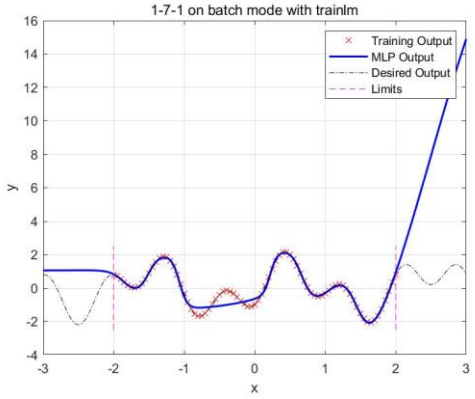
In this case, we use the batch mode training with “trainlm” algorithm.

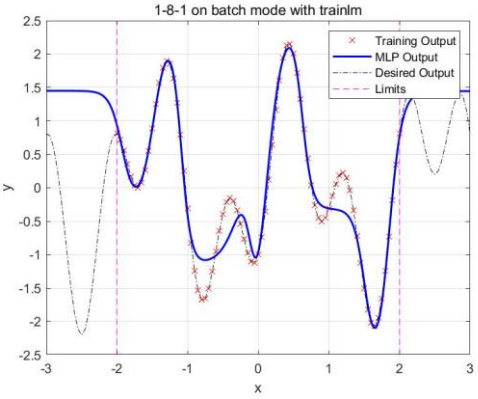
Using same pre-processing procedures, by inputting different n (where n=1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50, 100), we can get different structures of the MLP: 1-n-1. All the figures are shown as below:

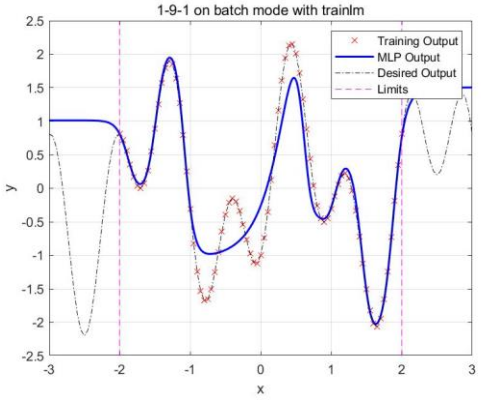


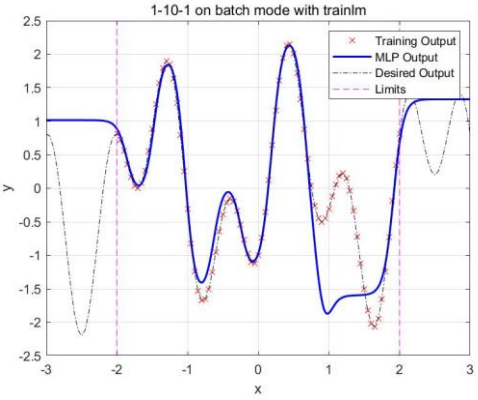

1-5-1 on batch mode with trainlm
Training Output
MLP Output
Desired Output
Limits
x
y

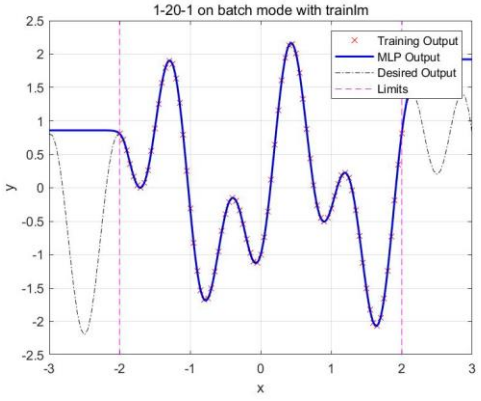

1-6-1 on batch mode with trainlm
Training Output
MLP Output
Desired Output
Limits
x
y

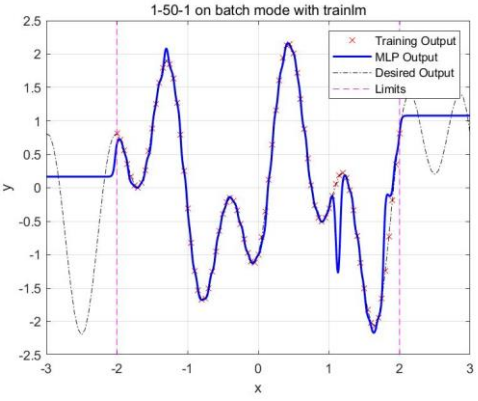

1-7-1 on batch mode with trainlm
Training Output
MLP Output
Desired Output
Limits
x
y

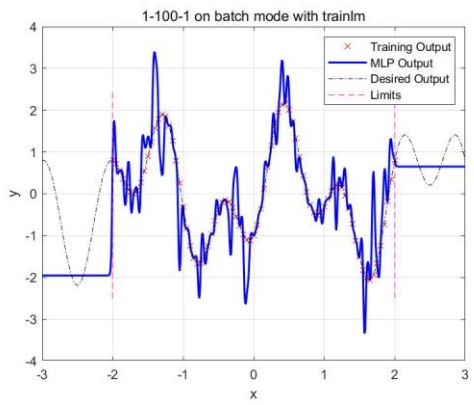

1-8-1 on batch mode with trainlm
Training Output
MLP Output
Desired Output
Limits
x
y


1-9-1 on batch mode with trainlm
Training Output
MLP Output
Desired Output
Limits
x
y


1-10-1 on batch mode with trainlm
Training Output
MLP Output
Desired Output
Limits
x
y


1-20-1 on batch mode with trainlm
Training Output
MLP Output
Desired Output
Limits
x
y


1-50-1 on batch mode with trainlm
Training Output
MLP Output
Desired Output
Limits
x
y



Through the analyze of the result in these pictures, we can easily conclude that

	Under-fitting	Proper fitting	Over-fitting
n	1-10	20,50	100

Next we use the n=20 MLP to output when x equals 3 and -3,

$$x = -3 \rightarrow y_{ground\_truth} = 0.841, y_{MLP} = 0.793$$

$$x = 3 \rightarrow y_{ground\_truth} = 0.809, y_{MLP} = 1.813$$

It also show that, if the new sample input is outside the domain of input of the training set, then the result is not correct.

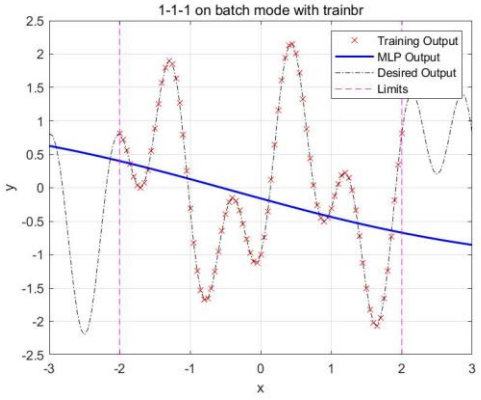
```
close all
clear
clc

syms x y;
y = 1.2*sin(pi*x)-cos(2.4*pi*x);

training_set_input(:) = -2:0.05:2;
training_set_output(1,:) = eval(subs(y,x,training_set_input(1,:)));
test_set_input(:) = -2:0.01:2;
test_set_output(1,:) = eval(subs(y,x,test_set_input(1,:)));
desired_input(:) = -3:0.05:3;
desired_output(:) = eval(subs(y,x,desired_input(1,:)));

x = training_set_input;
t = training_set_output;

% Choose a Training Function
trainFcn = 'trainlm';

% Create a Fitting Network
for n = [1:10,20,50,100]
```

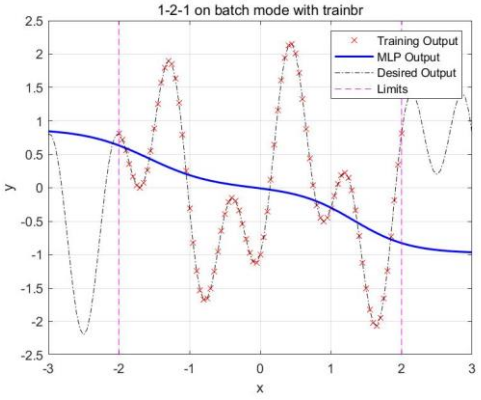
```
hiddenLayerSize = n;
net = fitnet(hiddenLayerSize,trainFcn);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 5/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);

% View the Network
%view(net)
%figure;
%plotfit(net,x,t)


results = sim(net, -3:0.01:3);
figure
plot(training_set_input,training_set_output,'rx') %training output
hold on
plot(-3:0.01:3,results(1,:),'b','LineWidth',1.5);              %mlp
output
plot(desired_input,desired_output,'k-.');      %desired output
line([-2 -2], [-2.5 2.5],'Color','magenta', 'LineStyle','--');
line([2 2], [-2.5 2.5],'Color','magenta', 'LineStyle','--');
legend('Training Output', 'MLP Output', 'Desired Output',
'Limits');
title(strcat("1-", num2str(n), "-1 on batch mode with trainlm"));
xlabel('x');
ylabel('y');
grid
saveas(gcf,num2str(n) ,'jpg');
end
```

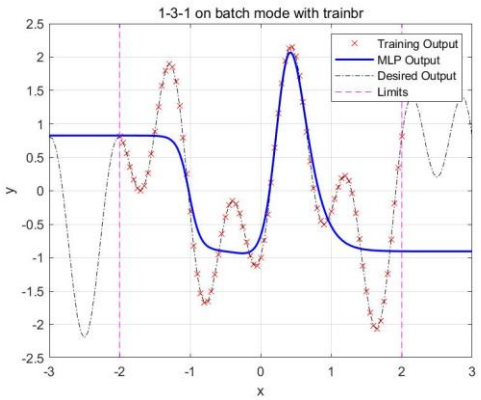
c) batch mode with “trainbr” algorithm

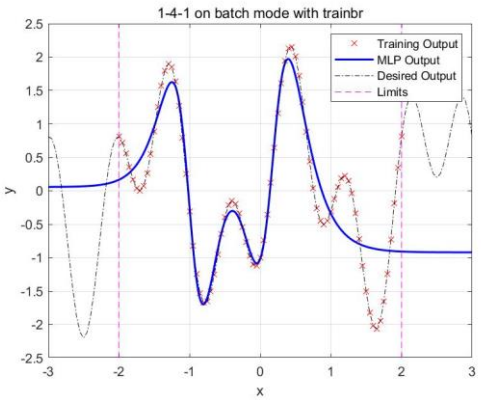
In this case, we use the batch mode training with “trainbr” algorithm.

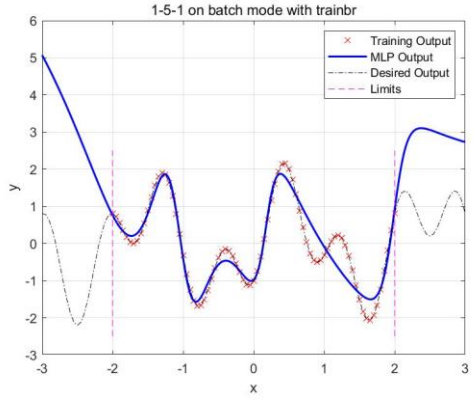
Using same pre-processing procedures, by inputting different n (where n=1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50, 100), we can get different structures of the MLP: 1-n-1. All the figures are shown as below:

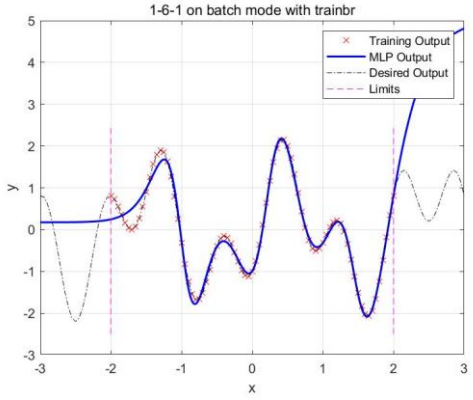


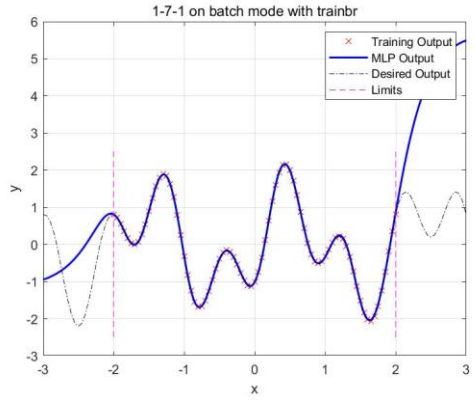


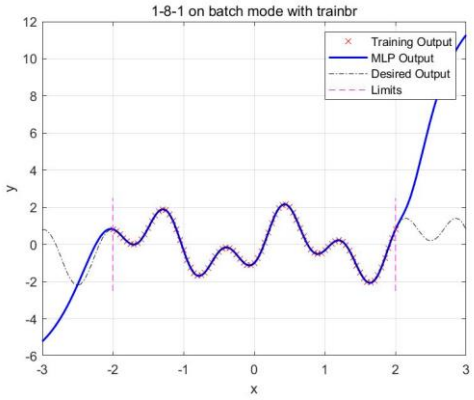


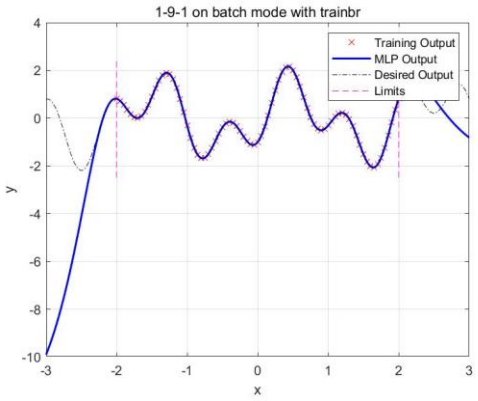


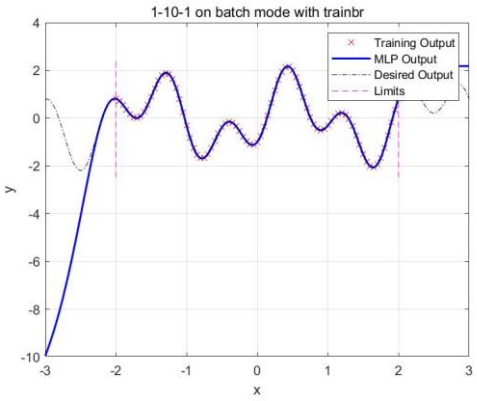


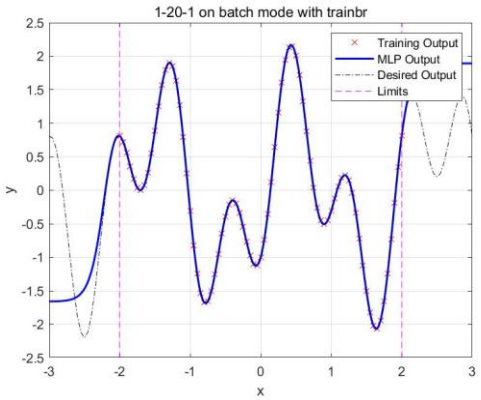


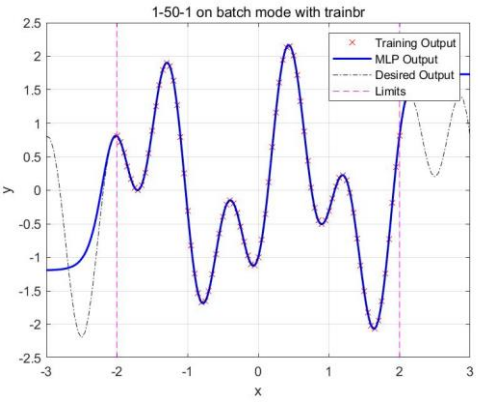


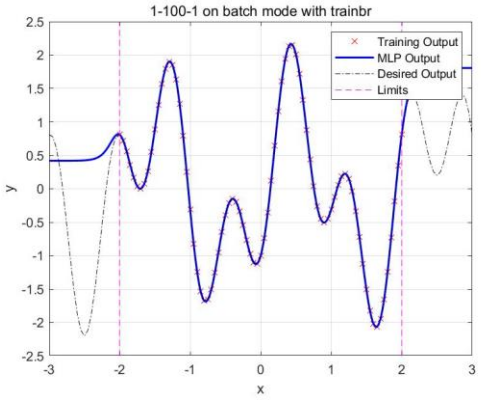












Through the analyze of the result in these pictures, we can easily conclude that

	Under-fitting	Proper fitting	Over-fitting
n	1-6	7,8,9,10,20,50,100	None

Next we use the n=20 MLP to output when x equals 3 and -3,

$$x = -3 \rightarrow y_{ground\_truth} = 0.841, y_{MLP} = -1.658$$

$$x = 3 \rightarrow y_{ground\_truth} = 0.809, y_{MLP} = 1.835$$

It also show that, if the new sample input is outside the domain of input of the training set, then the result is not correct.

```
close all
clear
clc

syms x y;
y = 1.2*sin(pi*x)-cos(2.4*pi*x);

training_set_input(:) = -2:0.05:2;
training_set_output(1,:) = eval(subs(y,x,training_set_input(1,:)));
test_set_input(:) = -2:0.01:2;
test_set_output(1,:) = eval(subs(y,x,test_set_input(1,:)));
desired_input(:) = -3:0.05:3;
desired_output(:) = eval(subs(y,x,desired_input(1,:)));

x = training_set_input;
t = training_set_output;

% Choose a Training Function
trainFcn = 'trainbr';

% Create a Fitting Network
for n = [1:10,20,50,100]
hiddenLayerSize = n;
net = fitnet(hiddenLayerSize,trainFcn);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 5/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
```

```
% View the Network
%view(net)
%figure;
%plotfit(net,x,t)


results = sim(net, -3:0.01:3);
figure
plot(training_set_input,training_set_output,'rx') %training output
hold on
plot(-3:0.01:3,results(1,:),'b','LineWidth',1.5);              %mlp
output
plot(desired_input,desired_output,'k-.');      %desired output
line([-2 -2], [-2.5 2.5],'Color','magenta', 'LineStyle','--');
line([2 2], [-2.5 2.5],'Color','magenta', 'LineStyle','--');
legend('Training Output', 'MLP Output', 'Desired Output',
'Limits');
title(strcat("1-", num2str(n), "-1 on batch mode with trainbr"));
xlabel('x');
ylabel('y');
grid
saveas(gcf,num2str(n) ,'jpg');
end
```

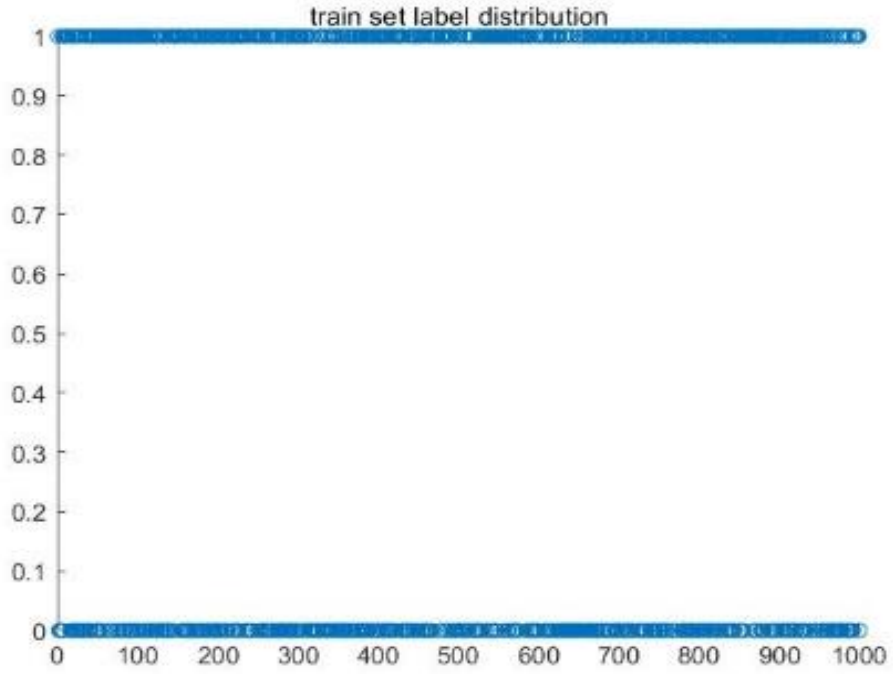
Solution 3

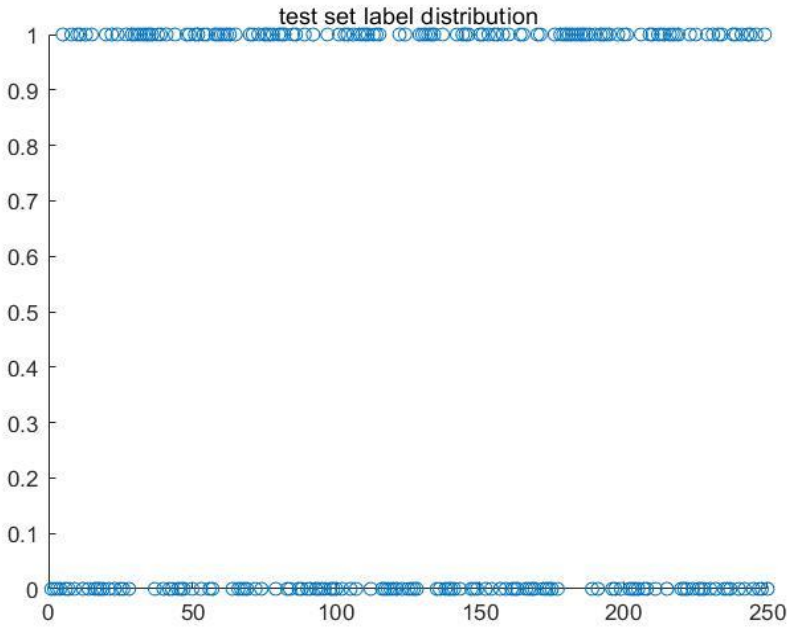
Matric No. A0224725H. So my group[mod(25,3)+1=2] is 2.(smile or non-smile)

When we are processing the images, we should pre-process images. Following the guides in lectures, we can get the ground truth of train set and test set through the .att files. And for the raw data from images, we use the standard processing methods from .jpg files.

In order to reduce the repeating work in later part, I save these raw data as .mat files in Matlab working space. So it is easy to know what I do in separate part.

a) For the label distribution, I got the ground truth from train set and test set. And plot these pictures as below.





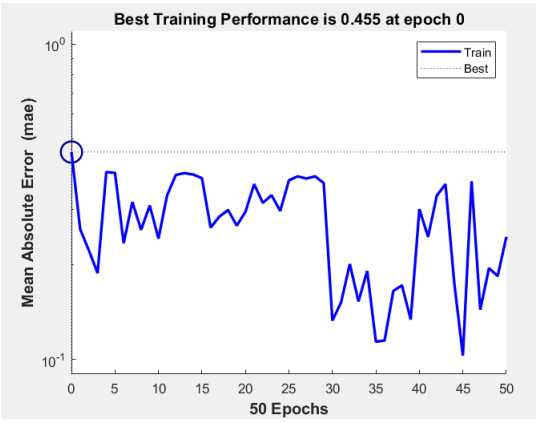

```
%init
close all;clear;clc;

%For Training
file_dir0='D:\2021\NUS-ECE\EE5904-神经网络
\homework\HOMEWORK_TWO\Face Database\TrainImages\';%路径前缀
img_list0=dir(strcat(file_dir0,'*.jpg'));
att_list0=dir(strcat(file_dir0,'*.att'));
```

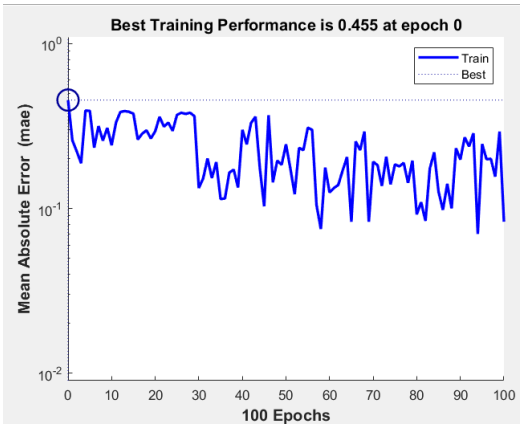
```
image_c0=cell(1,1000);
ground_truth_0=zeros(1,1000);

for i=1:1000  %For Training%
    img_name0 = strcat(file_dir0,img_list0(i).name);%得到完整的路径
    att_name0 = strcat(file_dir0,att_list0(i).name);
    img_rgb0=imread(num2str(img_name0));%将每张图片读出 rgb
    img_gray0=rgb2gray(img_rgb0);
    img_gray0=img_gray0(1:101,1:101);
    img_G=img_gray0(:);
    image_c0{:,i}=img_G;

    %ground_truth for training
    L=load(att_name0);
    ground_truth_number=L(2);
    ground_truth_0(:,i)=ground_truth_number;
end

%For Testing
file_dir1='D:\2021\NUS-ECE\EE5904-神经网络
\homework\HOMEWORK_TWO\Face Database\TestImages\';%路径前缀
img_list1=dir(strcat(file_dir1,'*.jpg'));
att_list1=dir(strcat(file_dir1,'*.att'));
image_c1=cell(1,250);
ground_truth_1=zeros(1,250);

for i=1:250  %For Testing%
    img_name1 = strcat(file_dir1,img_list1(i).name);%得到完整的路径
    att_name1 = strcat(file_dir1,att_list1(i).name);
    img_rgb1=imread(num2str(img_name1));%将每张图片读出 rgb
    img_gray1=rgb2gray(img_rgb1);
    img_gray1=img_gray1(1:101,1:101);
    img_G=img_gray1(:);
    image_c1{:,i}=img_G;

    %ground_truth for Testing
    L=load(att_name1);
    ground_truth_number=L(2);
    ground_truth_1(:,i)=ground_truth_number;
end

image_mat0=cell2mat(image_c0);
image_mat1=cell2mat(image_c1);
train_set=double(image_mat0);
```

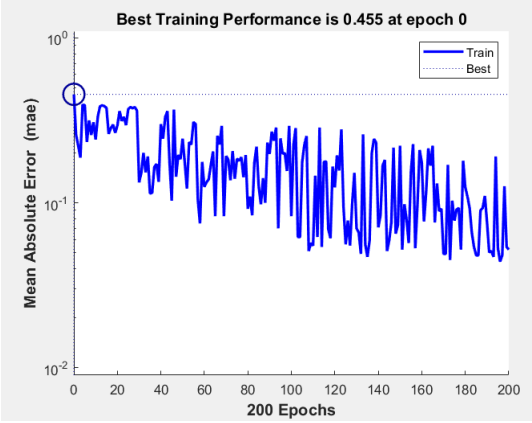
```
test_set=double(image_mat1);

figure(1)
train_length=size(ground_truth_0,2);
train_seq=1:train_length;
scatter(train_seq,ground_truth_0,'o');
title("train set label distribution");
figure(2)
test_length=size(ground_truth_1,2);
test_seq=1:test_length;
scatter(test_seq,ground_truth_1,'o');
title("test set label distribution");
```

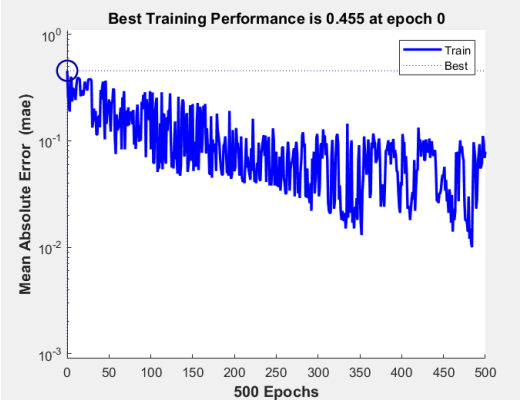
b) Apply Rosenblatt's perceptron (single layer perceptron) to this task.
Firstly, I use epoch =50, learning rate = 0.01, and change the epochs in later training. Here are the results

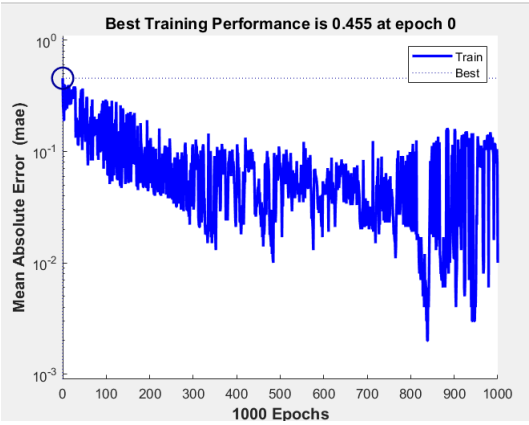
	accuracy_train	accuracy_test
Epoch =50	75.6%	66.0%
Epoch =100	91.7%	73.2%
Epoch =200	94.8%	74.8%
Epoch =500	96.1%	75.6%
Epoch=1000	99.0%	75.6%











From the performance with different epochs, we can find that the perceptron can classify smile and non-smile. Especially for the test set, when we set 1000 epochs, the accuracy is close to 75.6%.

```
%init
close all;clear;clc;

%For Training
file_dir0='D:\2021\NUS-ECE\EE5904-神经网络
\homework\HOMEWORK_TWO\Face Database\TrainImages\';%路径前缀
img_list0=dir(strcat(file_dir0,'*.jpg'));
att_list0=dir(strcat(file_dir0,'*.att'));
image_c0=cell(1,1000);
ground_truth_0=zeros(1,1000);

for i=1:1000  %For Training%
    img_name0 = strcat(file_dir0,img_list0(i).name);%得到完整的路径
    att_name0 = strcat(file_dir0,att_list0(i).name);
    img_rgb0=imread(num2str(img_name0));%将每张图片读出 rgb
    img_gray0=rgb2gray(img_rgb0);
    img_gray0=img_gray0(1:101,1:101);
    img_G=img_gray0(:);
    image_c0{:,i}=img_G;
```

```
    %ground_truth for training
    L=load(att_name0);
    ground_truth_number=L(2);
    ground_truth_0(:,i)=ground_truth_number;
end

%For Testing
file_dir1='D:\2021\NUS-ECE\EE5904-神经网络
\homework\HOMEWORK_TWO\Face Database\TestImages\';%路径前缀
img_list1=dir(strcat(file_dir1,'*.jpg'));
att_list1=dir(strcat(file_dir1,'*.att'));
image_c1=cell(1,250);
ground_truth_1=zeros(1,250);

for i=1:250  %For Testing%
    img_name1 = strcat(file_dir1,img_list1(i).name);%得到完整的路径
    att_name1 = strcat(file_dir1,att_list1(i).name);
    img_rgb1=imread(num2str(img_name1));%将每张图片读出 rgb
    img_gray1=rgb2gray(img_rgb1);
    img_gray1=img_gray1(1:101,1:101);
    img_G=img_gray1(:);
    image_c1{:,i}=img_G;

    %ground_truth for Testing
    L=load(att_name1);
    ground_truth_number=L(2);
    ground_truth_1(:,i)=ground_truth_number;
end

image_mat0=cell2mat(image_c0);
image_mat1=cell2mat(image_c1);
train_set=double(image_mat0);
test_set=double(image_mat1);

save('train_set.mat','train_set');
save('test_set.mat','test_set');
save('train_ground_truth.mat','ground_truth_0');
save('test_ground_truth.mat','ground_truth_1');

%set network
net=perceptron('hardlim','learnpn');
net.trainParam.epochs=50;
net.trainParam.show=50;
```

```
net.trainParam.lr=0.01;
net.divideFcn = 'dividetrain';
net.performParam.regularization = 0.1;

[net,tr]=train(net,train_set,ground_truth_0);
output_test=sim(net,test_set);

%accuracy
output_train=sim(net,train_set);
test=0;
train=0;
for i=1:250
    value=abs(output_test(i)-ground_truth_1(i));
    if(value<0.5)
        test=test+1;
    end
end
for i=1:1000
    value=abs(output_train(i)-ground_truth_0(i));
    if(value<0.5)
        train=train+1;
    end
end

accuracy_train=sprintf('accuracy_train= %0.1f%%',train/10);
disp(accuracy_train);
accuracy_test=sprintf('accuracy_test= %0.1f%%',test/2.5);
disp(accuracy_test);

plotperform(tr);
```

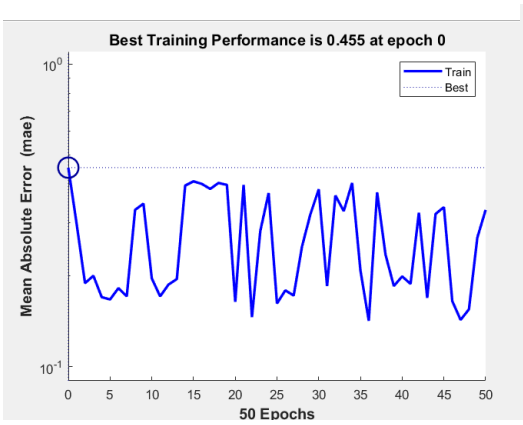
c) To finish this task, I just naively downsample the images. I use imresize function in this part. As we all know, we take the similar procedures in part b.

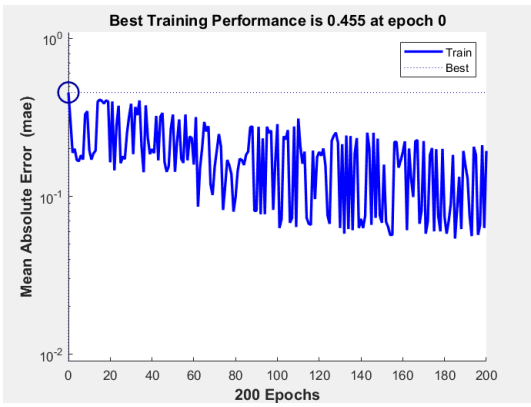
```
img_gray1=imresize(img_gray1,0.5);
img_gray1=img_gray1(1:51,1:51);
img_G=img_gray1(:);
image_c1{:,i}=img_G;
```

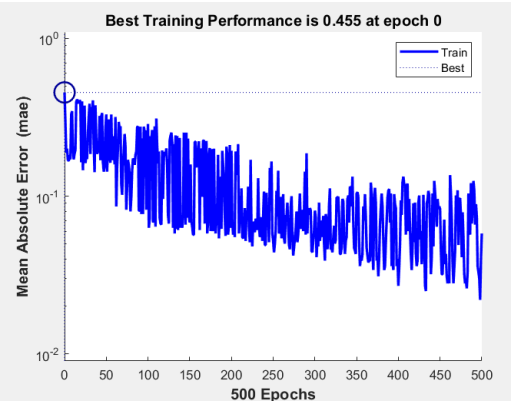
Firstly, I use epoch =50, learning rate = 0.01, and change the epochs

	accuracy_train	accuracy_test
Epoch =50	67.0%	62.8%
Epoch =100	71.3%	65.6%

Epoch =200	80.6%	68.0%
Epoch =500	94.2%	71.6%







Through the analyze the result of this part ,we can find that if we use this downsample way to get small scale data, this accuracy would decrease a little. In conclusion, if we want to get more accurate result, we should better use raw data.

```
%init
close all;clear;clc;


%For Training
file_dir0='D:\2021\NUS-ECE\EE5904-神经网络
\homework\HOMEWORK_TWO\Face Database\TrainImages\';%路径前缀
img_list0=dir(strcat(file_dir0,'*.jpg'));
att_list0=dir(strcat(file_dir0,'*.att'));
image_c0=cell(1,1000);
image_pca0=cell(1,1000);
ground_truth_0=zeros(1,1000);

for i=1:1000  %For Training%
    img_name0 = strcat(file_dir0,img_list0(i).name);%得到完整的路径
    att_name0 = strcat(file_dir0,att_list0(i).name);
    img_rgb0=imread(num2str(img_name0));%将每张图片读出 rgb
```

```
    img_gray0=rgb2gray(img_rgb0);

    %img_gray0=im2double(img_gray0); %pca alogorithm%
    %img_gray0=pca(img_gray0);
    %img_gray0=img_gray0(1:101,1:100);
    %img_G=img_gray0(:);
    %image_pca0{:,i}=img_G;

    img_gray0=imresize(img_gray0,0.5); %simply downsample%
    img_gray0=img_gray0(1:51,1:51);
    img_G=img_gray0(:);
    image_c0{:,i}=img_G;

    %ground_truth for training
    L=load(att_name0);
    ground_truth_number=L(2);
    ground_truth_0(:,i)=ground_truth_number;
end

%For Testing
file_dir1='D:\2021\NUS-ECE\EE5904-神经网络
\homework\HOMEWORK_TWO\Face Database\TestImages\';%路径前缀
img_list1=dir(strcat(file_dir1,'*.jpg'));
att_list1=dir(strcat(file_dir1,'*.att'));
image_c1=cell(1,250);
image_pca1=cell(1,250);
ground_truth_1=zeros(1,250);

for i=1:250  %For Testing%
    img_name1 = strcat(file_dir1,img_list1(i).name);%得到完整的路径
    att_name1 = strcat(file_dir1,att_list1(i).name);
    img_rgb1=imread(num2str(img_name1));%将每张图片读出 rgb

    img_gray1=rgb2gray(img_rgb1);

    %img_gray1=im2double(img_gray1);%pca alogorithm%
    %img_gray1=pca(img_gray1);
    %img_gray1=img_gray1(1:101,1:100);
    %img_G=img_gray1(:);
    %image_pca1{:,i}=img_G;

    img_gray1=imresize(img_gray1,0.5);
    img_gray1=img_gray1(1:51,1:51);
```

```
    img_G=img_gray1(:);
    image_c1{:,i}=img_G;

    %ground_truth for Testing
    L=load(att_name1);
    ground_truth_number=L(2);
    ground_truth_1(:,i)=ground_truth_number;
end

image_mat0=cell2mat(image_c0);
image_mat1=cell2mat(image_c1);
%image_mat0=cell2mat(image_pca0);
%image_mat1=cell2mat(image_pca1);
train_set=double(image_mat0);
test_set=double(image_mat1);

save('train_set.mat','train_set');
save('test_set.mat','test_set');

%init train/test label
train_label=zeros(1,1000);
test_label=zeros(1,250);
x0=1:1000;
x1=1:250;
train_std=std(ground_truth_0);
test_std=std(ground_truth_1);

%set network
net=perceptron('hardlim','learnpn');
net.trainParam.epochs=500;
net.trainParam.show=50;
net.trainParam.lr=0.01;
net.divideFcn = 'dividetrain';
net.performParam.regularization = 0.1;

[net,tr]=train(net,train_set,ground_truth_0);
output_test=sim(net,test_set);

%accuracy
output_train=sim(net,train_set);
test=0;
train=0;
for i=1:250
    value=abs(output_test(i)-ground_truth_1(i));
```

```
    if(value<0.5)
        test=test+1;
    end
end
for i=1:1000
    value=abs(output_train(i)-ground_truth_0(i));
    if(value<0.5)
        train=train+1;
    end
end

accuracy_train=sprintf('accuracy_train= %0.1f%%',train/10);
disp(accuracy_train);
accuracy_test=sprintf('accuracy_test= %0.1f%%',test/2.5);
disp(accuracy_test);

plotperform(tr);
```

d) To finish this task, I just change the number of n in 1-n-1 structure. I set the learning rate is 0.01, and the result is shown as below.

```
net=patternnet(n,'trainscg','crossentropy');
net=configure(net,train_set,ground_truth_0);
net.trainParam.epochs=5000;
net.trainParam.mc=0.9;
net.trainParam.lr=0.01;
net.trainParam.show=50;
net.divideFcn = 'dividetrain';
```

n	1	2	3	4	5	6	7	8	9	10
Train-acc	98.4%	99.8%	100%	100%	100%	100%	100%	100%	100%	100%
Test-acc	74.0%	73.6%	77.6%	76.0%	76.0%	78.8%	76.0%	78.4%	79.2%	76.8%
epoch	1577	867	371	383	464	308	207	164	215	132
n	15	20	30	40	50	60	100	150	200	300
Train-acc	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Test-acc	76.8%	78.0%	79.2%	78.8%	80.0%	79.2%	76.4%	77.2%	77.2%	76.8%
epoch	146	87	82	74	74	71	71	68	73	72

According to the result, we can find that the accuracy of training set can reach 100%, as long as n more than 3. And when n larger than 8, the accuracy of test set keeps at 78%. Compared with perceptron method, it can get better performance by using MLP for this question.

```
%init
close all;clear;clc;


load('train_set.mat');
load('test_set.mat');
load('train_ground_truth');
load('test_ground_truth');

n=300;
net=patternnet(n,'trainscg','crossentropy');
net=configure(net,train_set,ground_truth_0);
net.trainParam.epochs=5000;
net.trainParam.mc=0.9;
net.trainParam.lr=0.01;
net.trainParam.show=50;
net.divideFcn = 'dividetrain';

[net,tr]=train(net,train_set,ground_truth_0);
output_test=sim(net,test_set);

%accuracy
output_train=sim(net,train_set);
test=0;
train=0;

for i=1:250
    value=abs(output_test(i)-ground_truth_1(i));
    if(value<0.5)
        test=test+1;
    end
end
for i=1:1000
    value=abs(output_train(i)-ground_truth_0(i));
    if(value<0.5)
        train=train+1;
    end
end
```

```
accuracy_train=sprintf('accuracy_train= %0.1f%%',train/10);
disp(accuracy_train);
accuracy_test=sprintf('accuracy_test= %0.1f%%',test/2.5);
disp(accuracy_test);

plotperform(tr);
```

e) By guided by the example code in last lecture, I update the sequential code, and take similar way as part d. Because large epochs are very time-consuming in sequential method , so I set the learning rate 0.01 and epoch 100. Finally, the result is shown as below.

n	1	2	3	4	5	6	7	8	9	10
Test-acc	97.5%	96.2%	98.5%	99.1%	99.3%	99.9%	99.8%	99.7%	99.9%	100%
Test-acc	77.2%	76.0%	78.4%	76.4%	78.8%	77.6%	78.0%	76.8%	74.0%	76.8%

According to the result, we can find that the accuracy of training set can reach 100%, as long as n more than 10. And when n larger than 5, the accuracy of test set keeps at 78%. Compared with batch mode method, we can find these two modes can get similar performance.

```
load('train_set.mat');
load('test_set.mat');
load('train_ground_truth');
load('test_ground_truth');

n=10;
[net,accu_train,accu_test]=train_seq(n,train_set,ground_truth_0,1000,0,100);

%validate
output_test=sim(net,test_set);

%accuracy
output_train=sim(net,train_set);
test=0;
train=0;
for i=1:250
    value=abs(output_test(i)-ground_truth_1(i));
```

```
    if(value<0.5)
        test=test+1;
    end
end
for i=1:1000
    value=abs(output_train(i)-ground_truth_0(i));
    if(value<0.5)
        train=train+1;
    end
end

accuracy_train=sprintf('accuracy_train= %0.1f%%',train/10);
disp(accuracy_train);
accuracy_test=sprintf('accuracy_test= %0.1f%%',test/2.5);
disp(accuracy_test);

function [ net, accu_train, accu_val ] = train_seq( n, images,
labels, train_num, val_num, 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 training
% 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
images_c = num2cell(images, 1);
labels_c = num2cell(labels, 1);
% 2. Construct and configure the MLP
net = patternnet(n);
net.divideFcn = 'dividetrain';            % input for training only
net.performParam.regularization = 0.1;    % regularization strength
net.trainFcn = 'trainrp';                % 'trainrp' 'traingdx'
```

```
net.trainParam.epochs = epochs;
net.inputWeights{1,1}.learnParam.lr = 0.003;
net.layerWeights{2,1}.learnParam.lr = 0.003;
net.biases{1}.learnParam.lr = 0.002;
net.biases{2}.learnParam.lr = 0.002;
accu_train = zeros(epochs,1);              % record accuracy on
training set of each epoch
accu_val = 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_train = round(net(images(:,1:train_num)));     %
predictions on training set
    accu_train(i) = 1 - mean(abs(pred_train-labels(1:train_num)));
    pred_val = round(net(images(:,train_num+1:end)));   %
predictions on validation set
    accu_val(i) = 1 - mean(abs(pred_val-labels(train_num+1:end)));
    %disp(accu_train(i))
end
end
```

f) When I process this part, I searched in Mathwork.com to get eyes detection tools. Therefore, I take this method. And the BB is an *M*-by-4 element matrix. Each row of the output matrix contains a four-element vector, $[x, y, width, height]$, that specifies in pixels, the upper-left corner and size of a bounding box.

```
EyeDetect = vision.CascadeObjectDetector('EyePairBig'); %%detect eyes
%EyeDetect.MergeThreshold = 10;
BB=step(EyeDetect,img_rgb1);
%Eyes=imcrop(img_rgb1,BB);
```

I use this method to get the BB information from all images, and then I did the average result.

	Train set	Test set
x	17.9629	17.9321
y	13.9989	13.9502
width	67.2944	67.1629
height	16.4247	16.4163

It is easy to get the conclusion that all the images are ready aligned by placing eyes at a certain location. Therefore, I think it is necessary to do so. My ideas are based on the detection box. When we set detection box at a certain position, we can detect the key feature at a right way.

```
%init
close all;clear;clc;

%For Training
file_dir0='D:\2021\NUS-ECE\EE5904-神经网络
\homework\HOMEWORK_TWO\Face Database\TrainImages\';%路径前缀
img_list0=dir(strcat(file_dir0,'*.jpg'));
eyes_train_data=cell(1,1000);
eyes_train_BB=cell(1,1000);

for i=1:1000  %For Training%
    img_name0 = strcat(file_dir0,img_list0(i).name);%得到完整的路径
    img_rgb0=imread(num2str(img_name0));%将每张图片读出 rgb

    EyeDetect =
vision.CascadeObjectDetector('EyePairBig'); %%detect eyes
    %EyeDetect.MergeThreshold = 10;
    BB=step(EyeDetect,img_rgb0);
    eyes_train_BB{:,i}=BB;
end

%For Testing
file_dir1='D:\2021\NUS-ECE\EE5904-神经网络
\homework\HOMEWORK_TWO\Face Database\TestImages\';%路径前缀
img_list1=dir(strcat(file_dir1,'*.jpg'));
eyes_test_data=cell(1,250);
eyes_test_BB=cell(1,250);
```

```
for i=1:250  %For Testing%
    img_name1 = strcat(file_dir1,img_list1(i).name);%得到完整的路径
    img_rgb1=imread(num2str(img_name1));%将每张图片读出 rgb

    EyeDetect =
vision.CascadeObjectDetector('EyePairBig'); %%detect eyes
    %EyeDetect.MergeThreshold = 10;
    BB=step(EyeDetect,img_rgb1);
    eyes_test_BB{:,i}=BB;
end

test_BB_update=cell(1,221);
j=1;
for i =1:length(eyes_test_BB)
    if isempty(eyes_test_BB{i})==0
        test_BB_update{:,j}=[eyes_test_BB{i}]';
        j=j+1;
    end
end

train_BB_update=cell(1,890);
j=1;
for i =1:length(eyes_train_BB)
    if isempty(eyes_train_BB{i})==0
        train_BB_update{:,j}=[eyes_train_BB{i}]';
        j=j+1;
    end
end

BB_test=cell2mat(test_BB_update);
BB_train=cell2mat(train_BB_update);
mean_test=mean(BB_test,2);
mean_train=mean(BB_train,2);
```