

**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



1. According to Steepest(Gradient) descent method



we should calculate the gradient vector at the beginning.



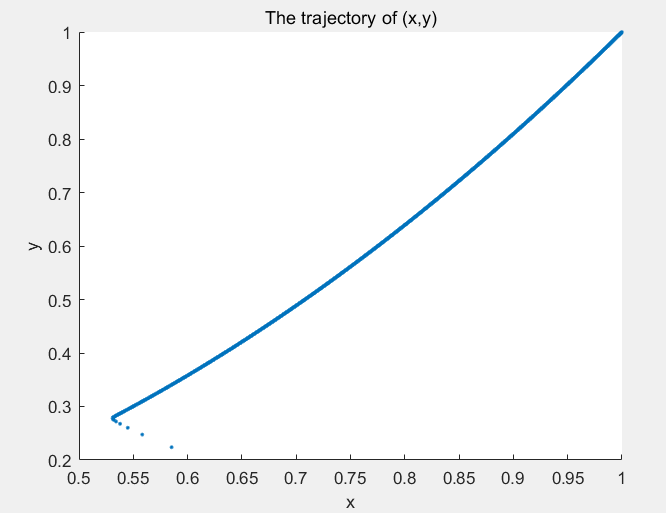


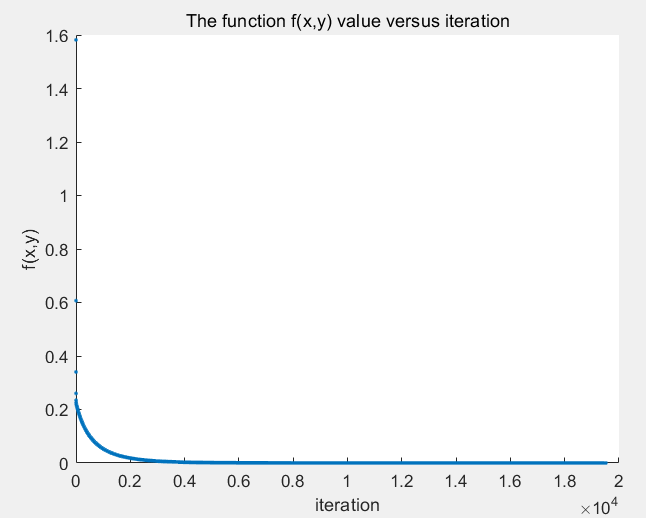
So the gradient vector 

Firstly, we set the stop criterion for the descent learning algorithm , 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 

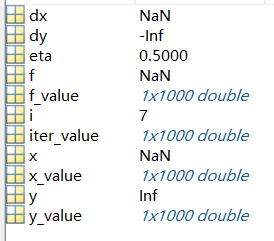
The trajectory of the two parameters x and y is shown.





We easily found the number of iteration is 19518, the value  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 



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**);** |

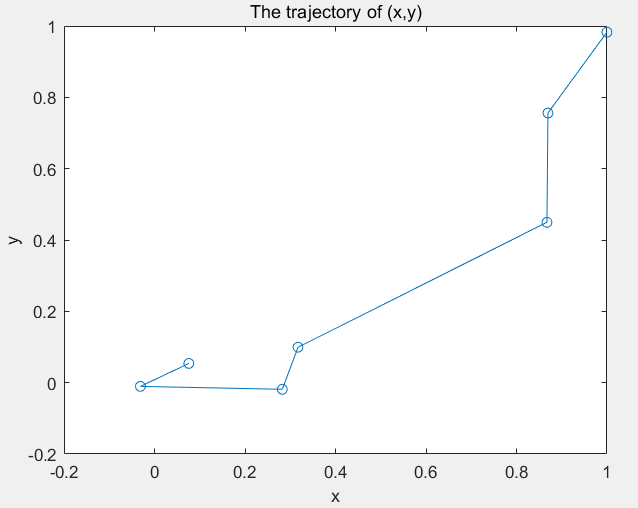
1. According to Newton’s method, 

Firstly, we calculate Hessian matrix,

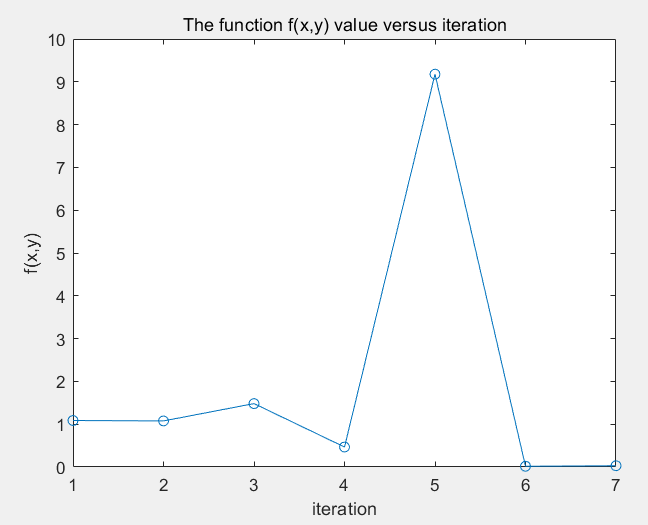
, , 



Firstly, we set the same stop criterion for the Newton’s learning algorithm , 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.



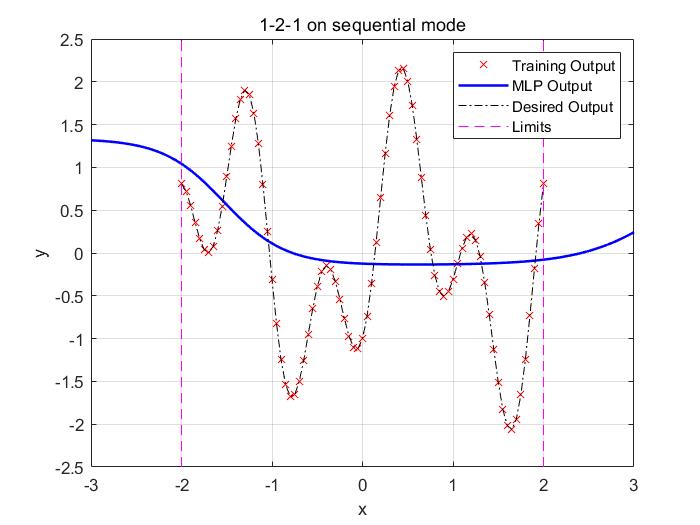
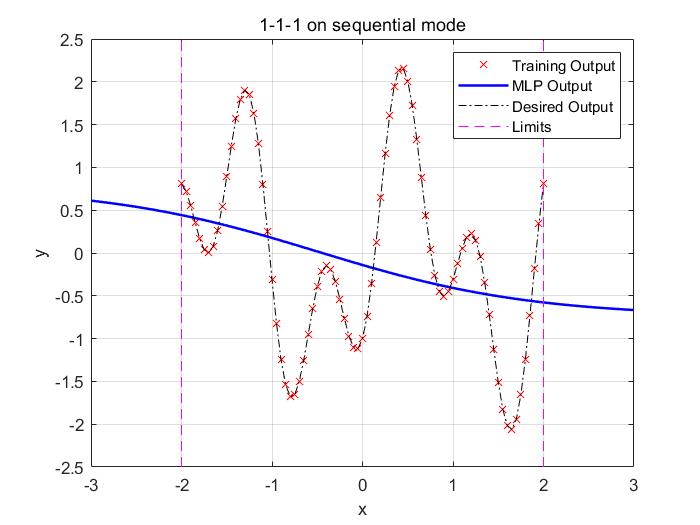
In conclusion, compared to Steepest gradient method, iterations from 19518 to 7, which decreases dramatically. It is obvious that Newton’s method is much faster.

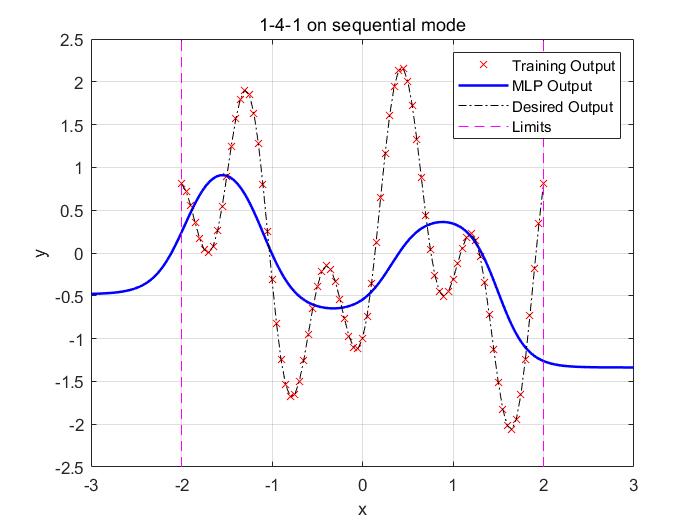
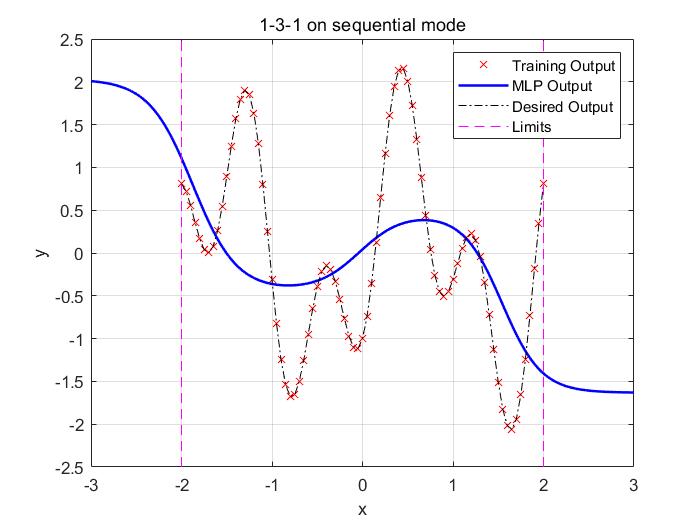
|  |
| --- |
| %init  close all**;**clear**;**clc**;**  %para  i**=**1**;**%iteration number  x\_value**=**zeros**(**1**,**1000**);**%init vector  y\_value**=**zeros**(**1**,**1000**);**  f\_value**=**zeros**(**1**,**1000**);**  iter\_value**=**zeros**(**1**,**1000**);**  x**=**rand**;**%random x,y  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)'); |

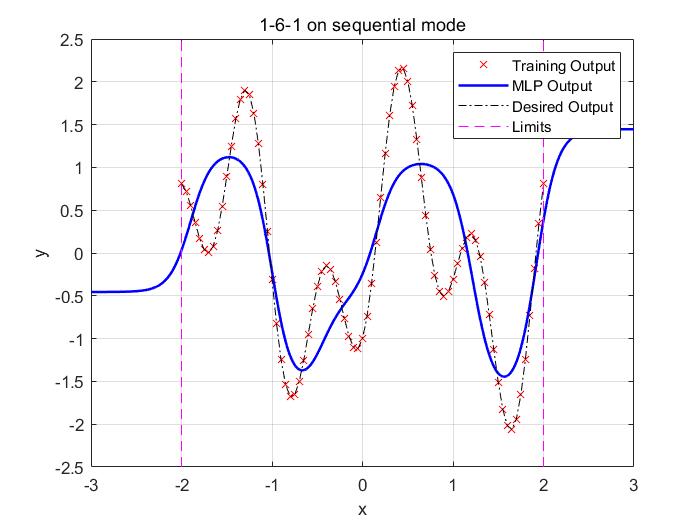
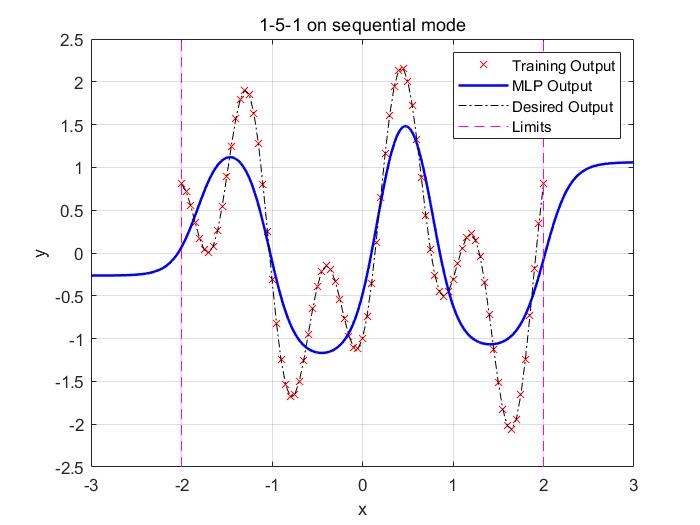
# Solution 2

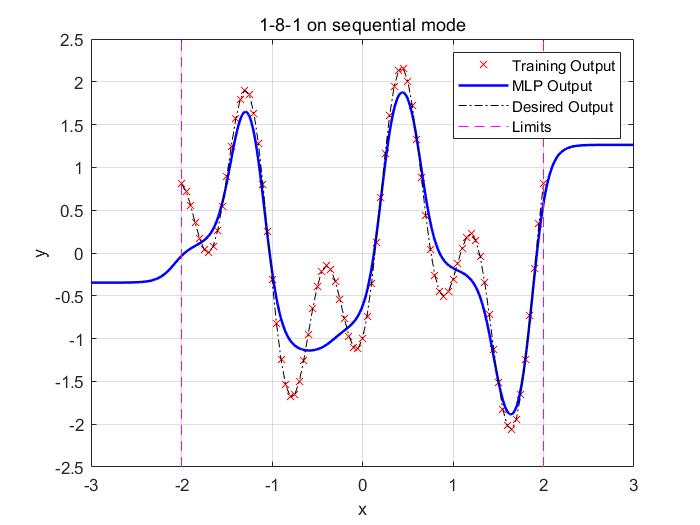
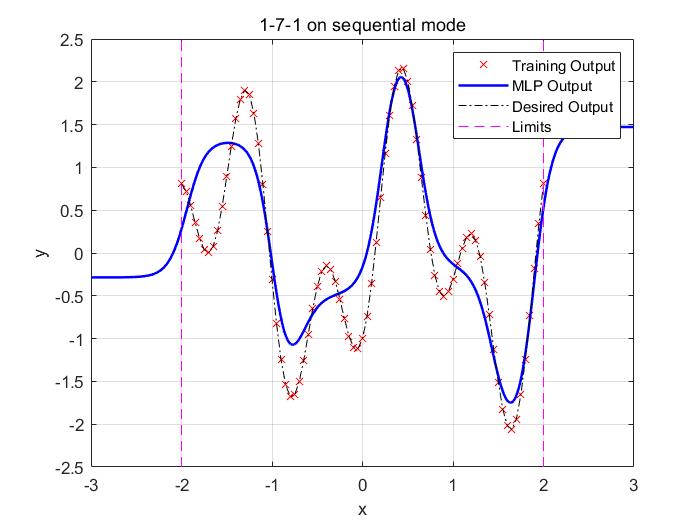
1. Sequential mode with BP algorithm

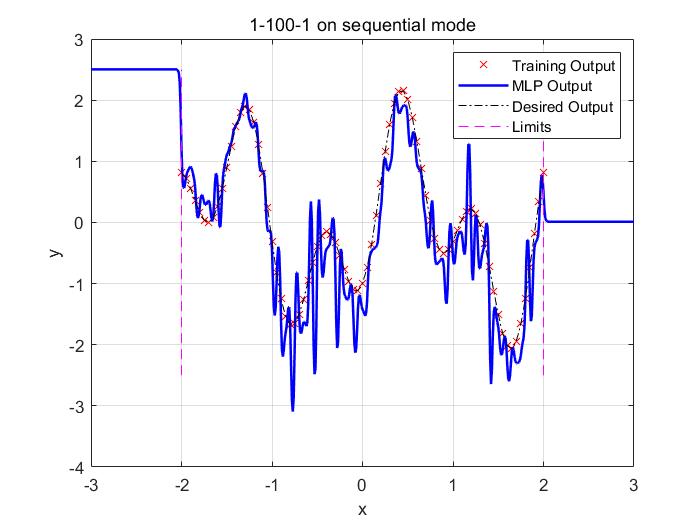
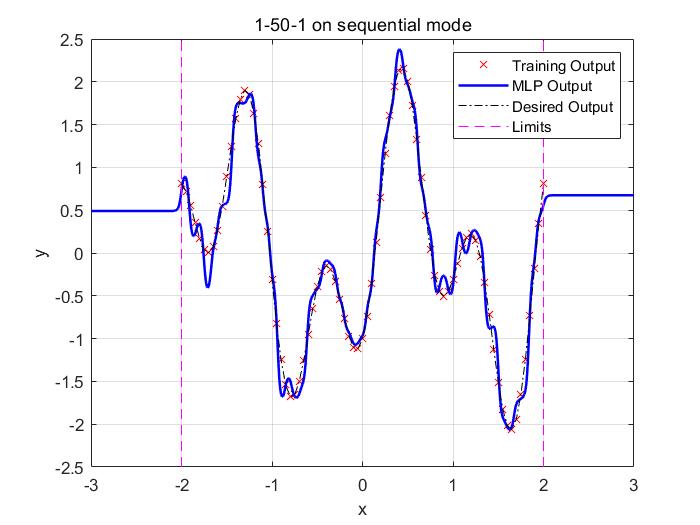
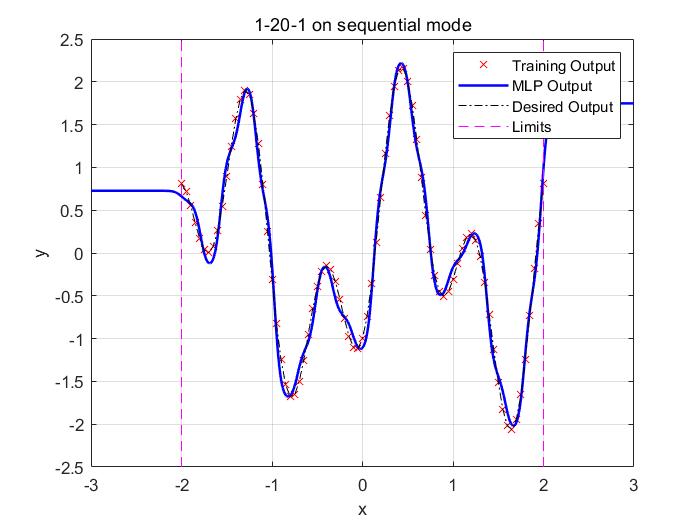
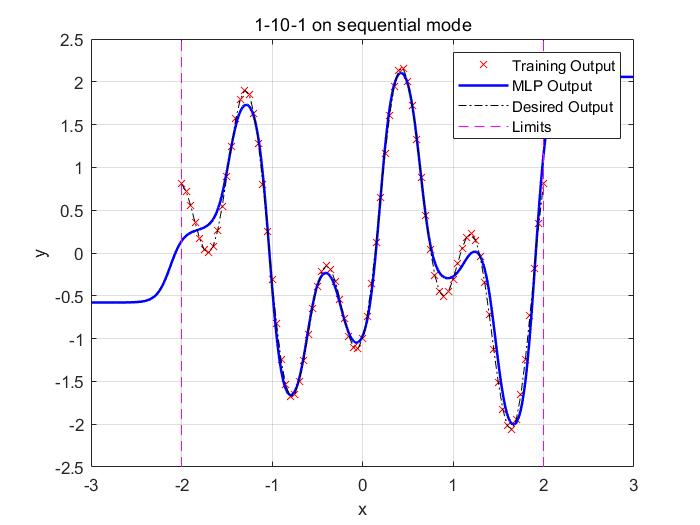
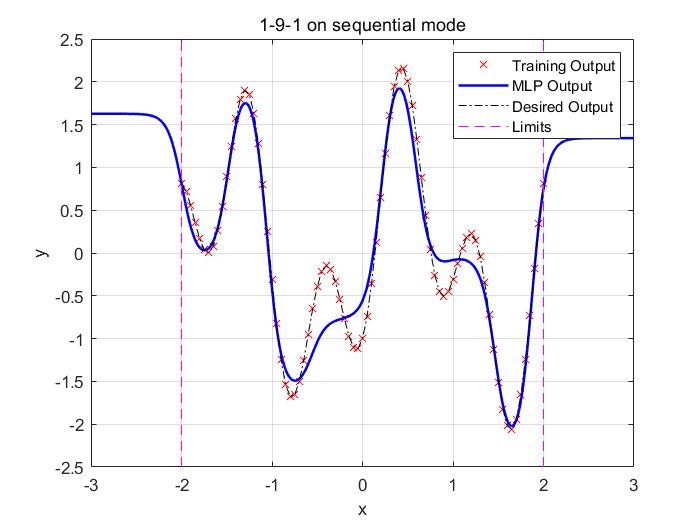
First way, by inputting different n (where n=1,2,3,4,5,6,7,8,9,10,20,50,100), we can get different structures of the MLP: 1-n-1. All the figures are shown as below:











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,





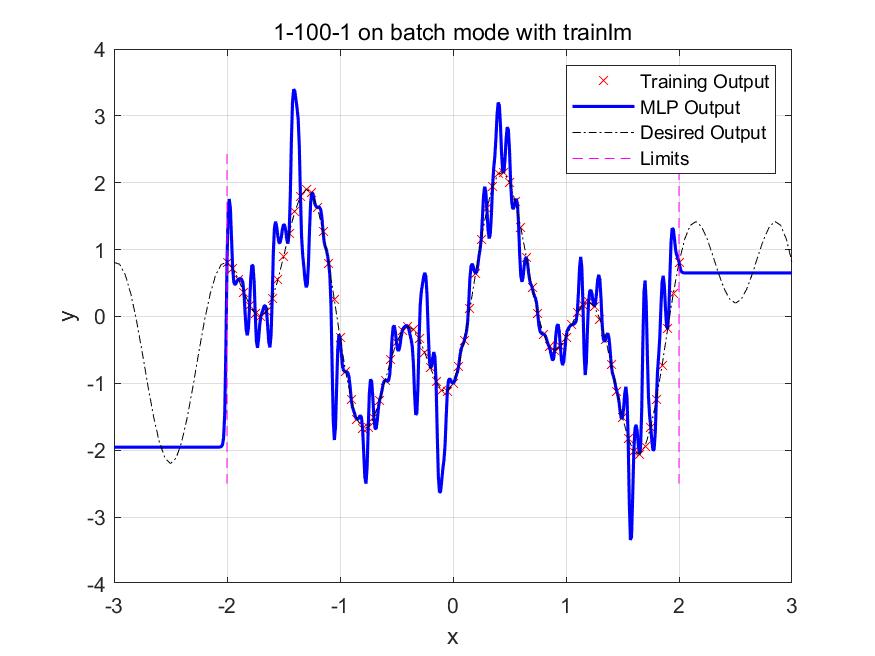
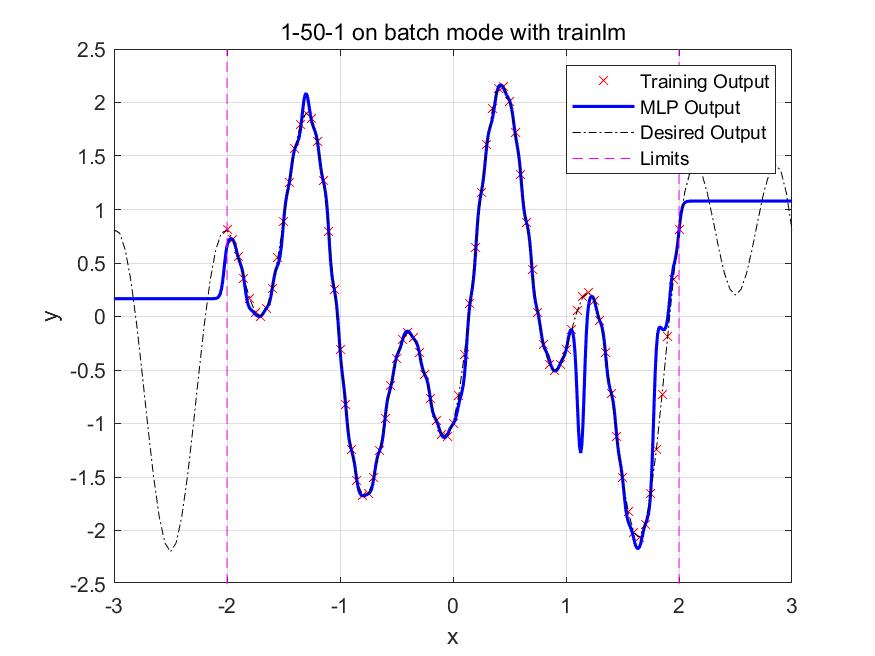
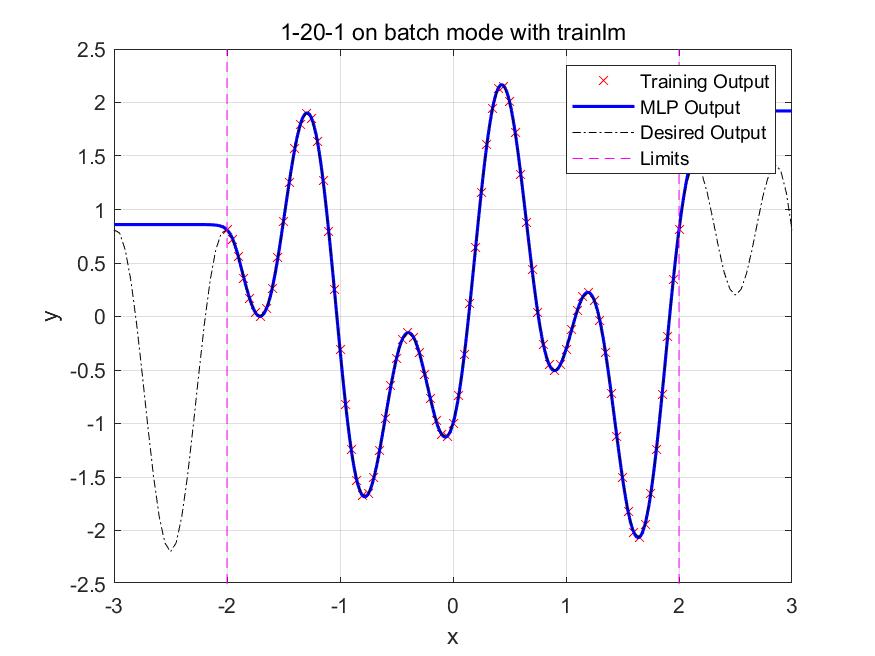
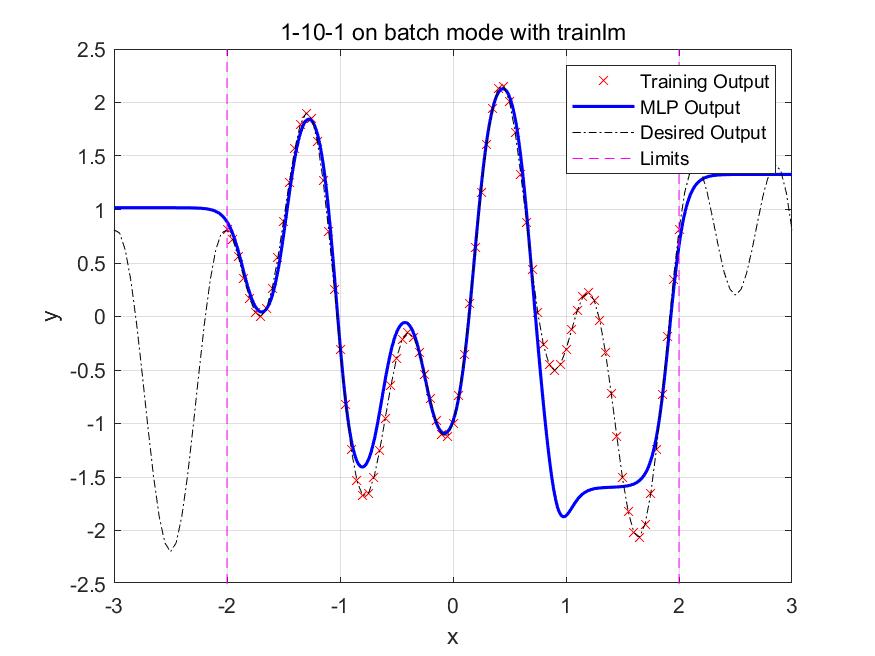
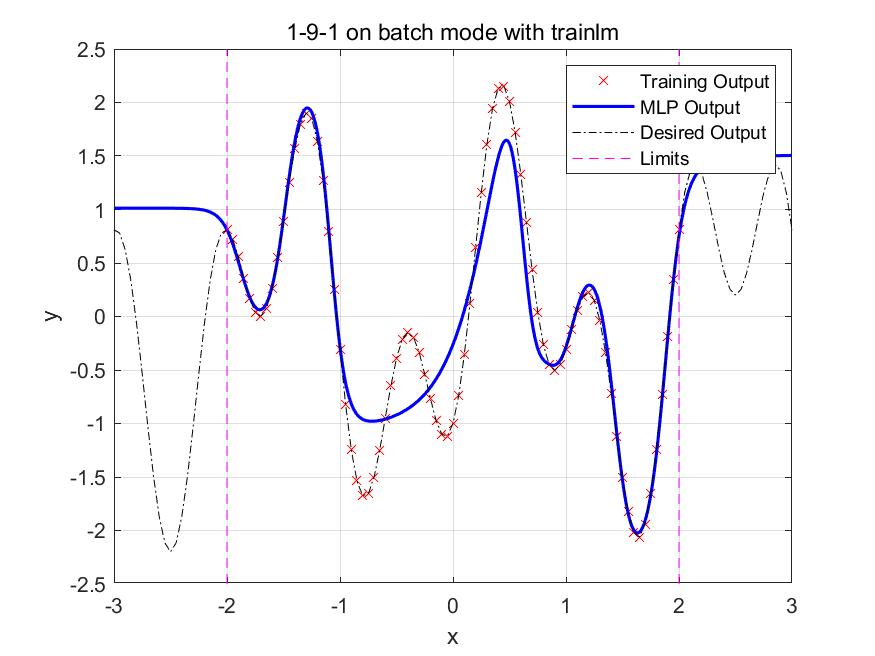
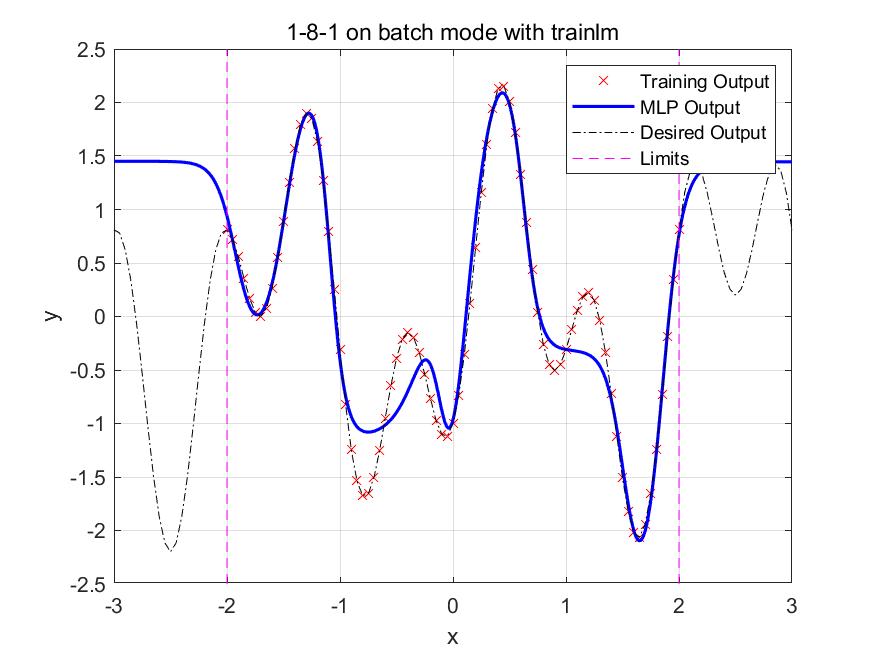
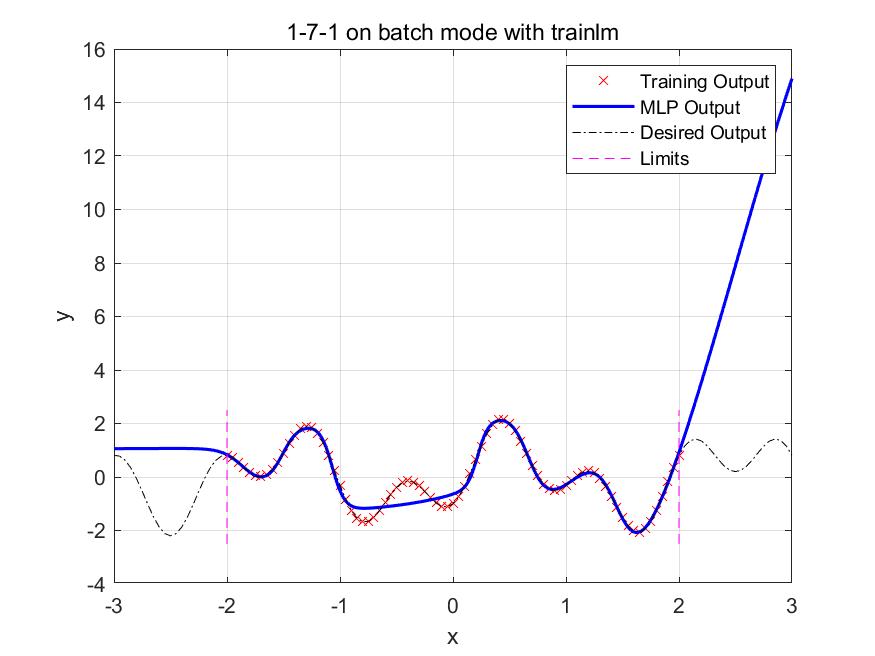
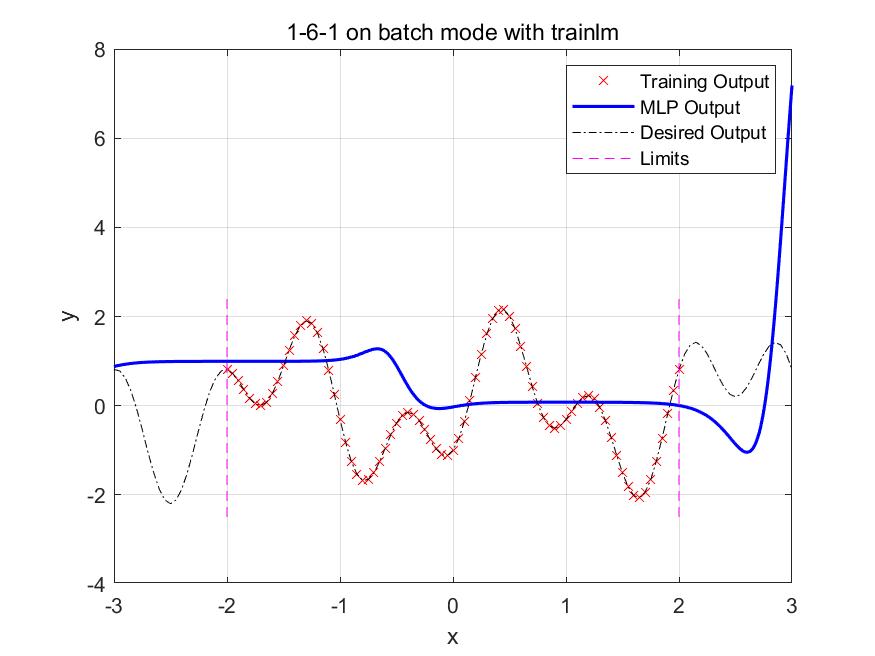
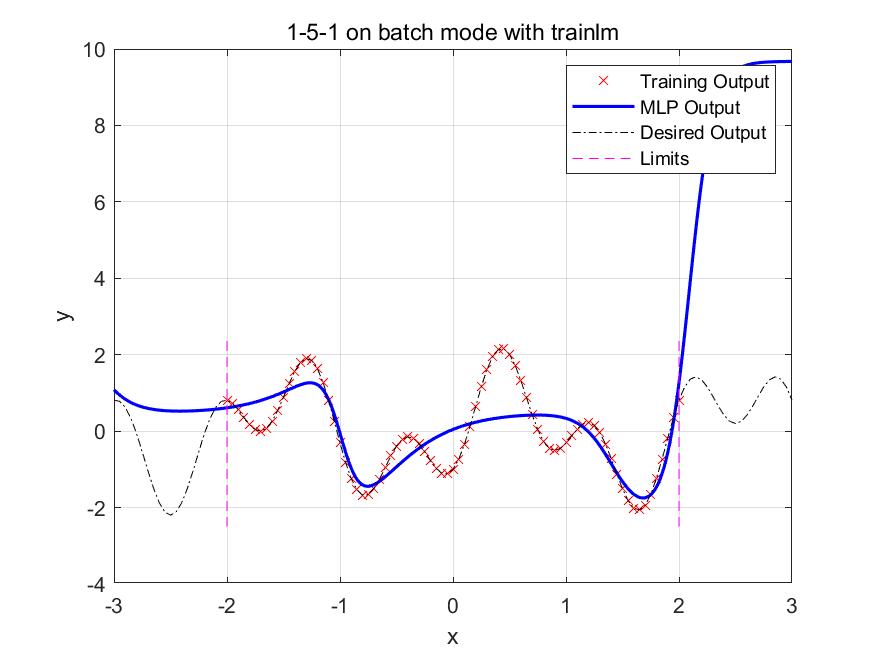
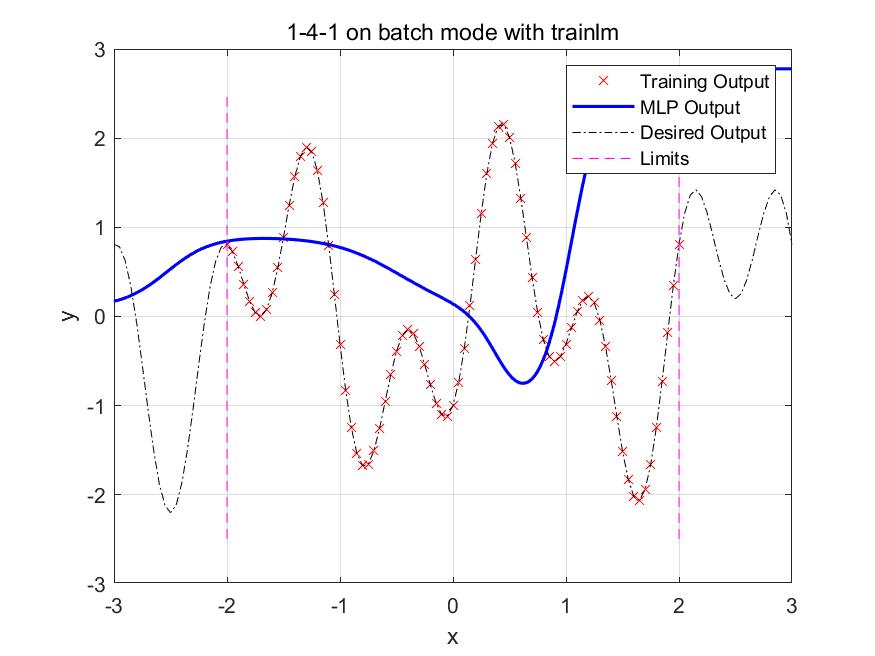
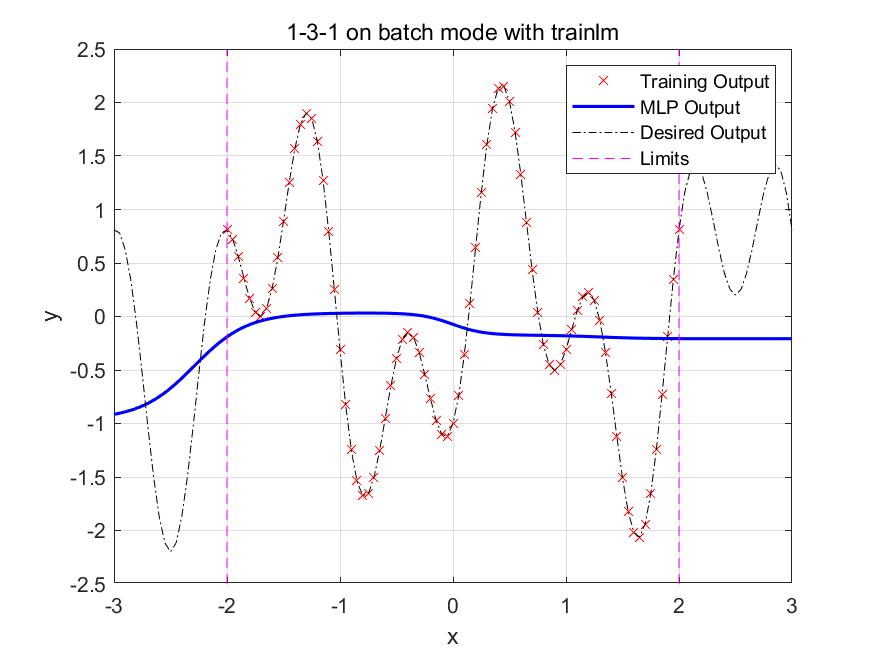
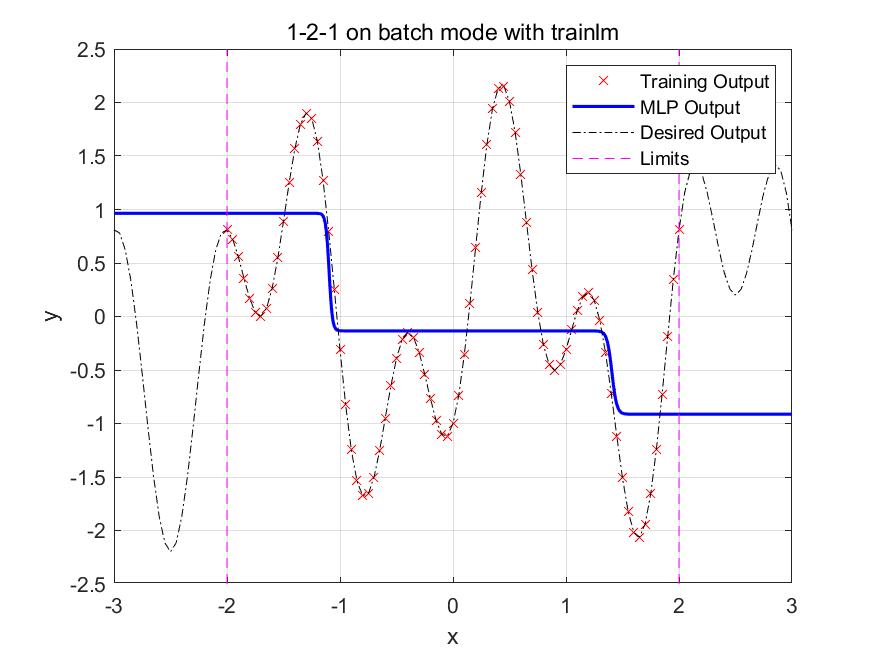
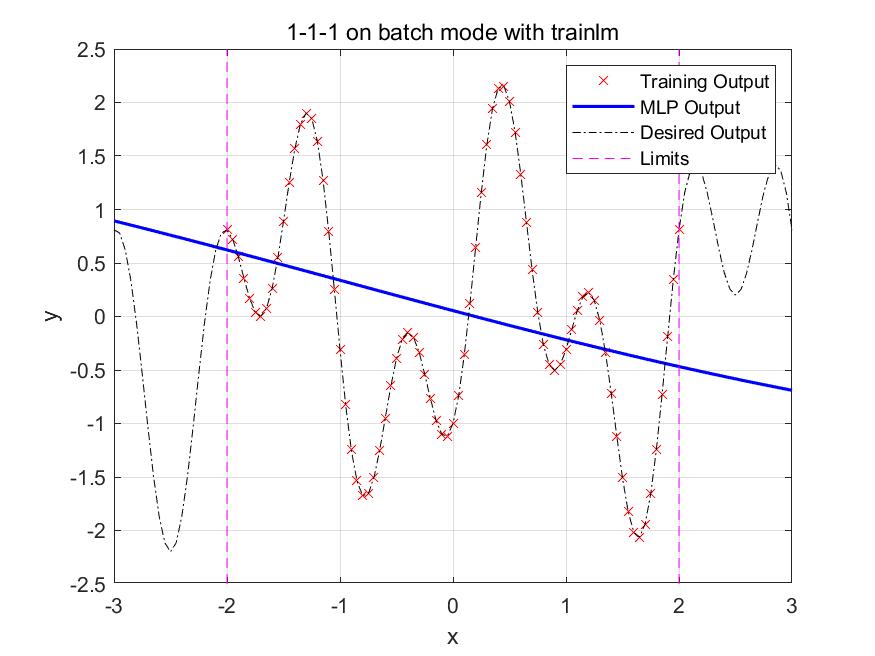
It seems that, if the new sample input is outside the domain of input of the training set, then the result is not correct.

|  |
| --- |
| 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** |

1. 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:



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,





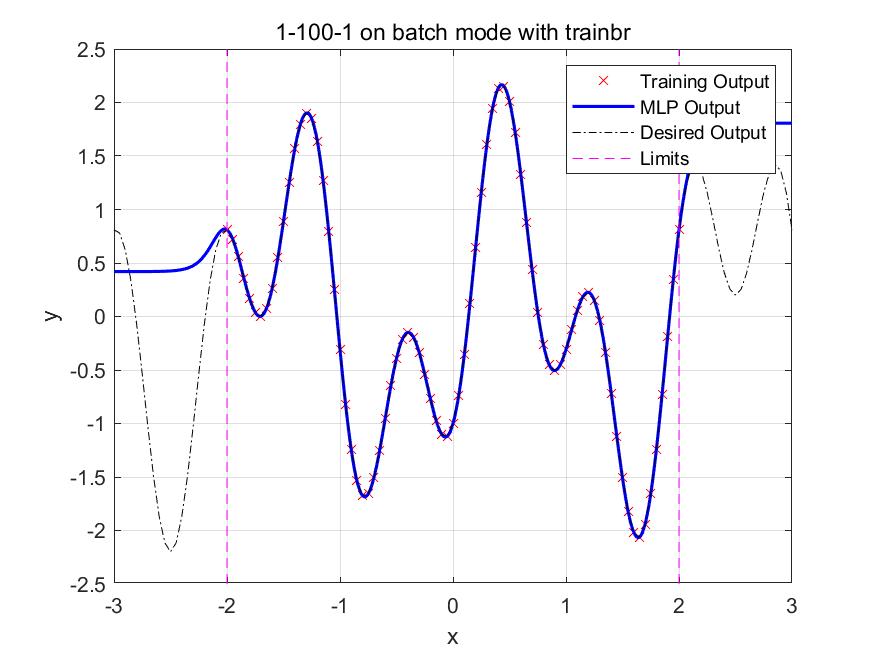
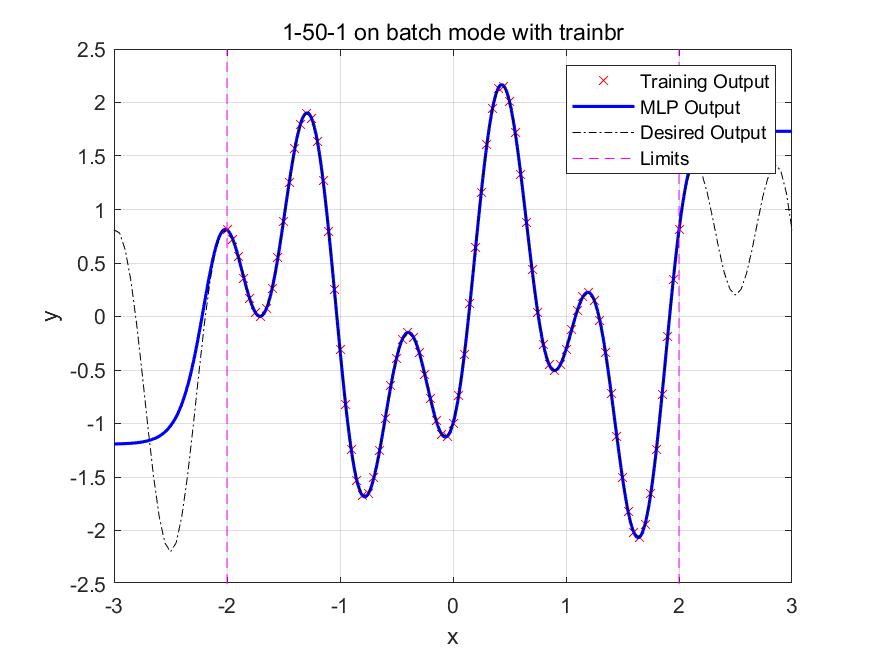
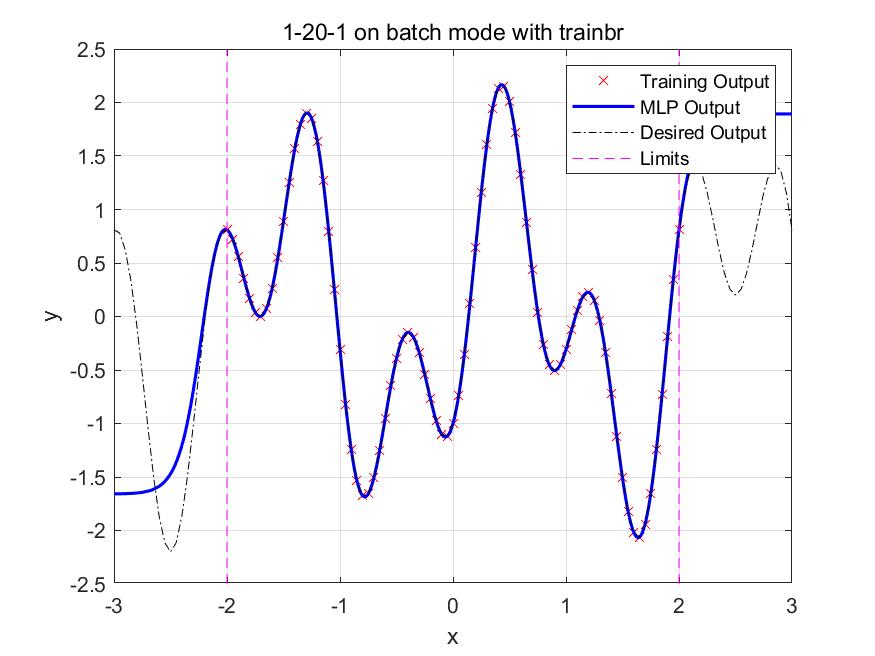
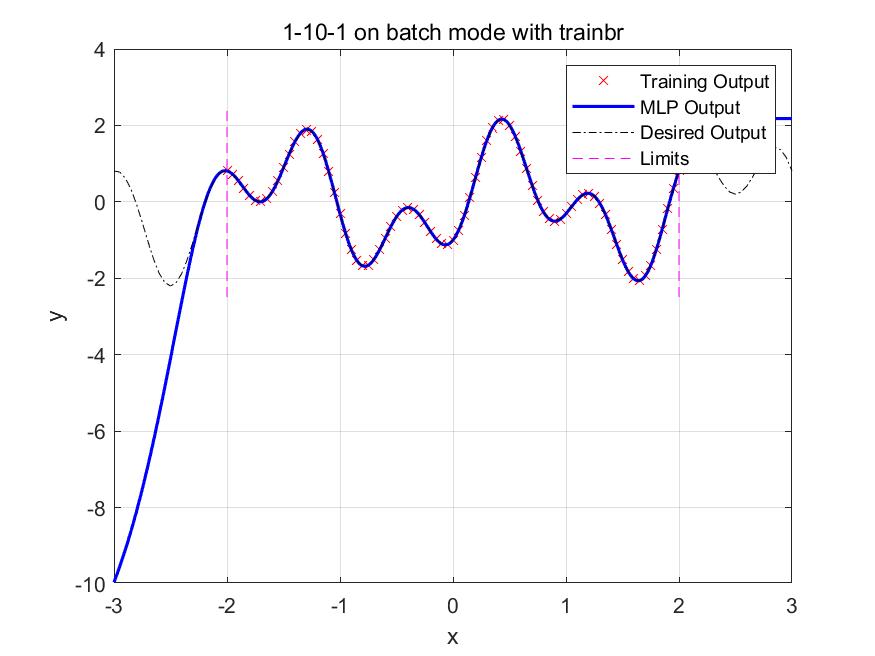
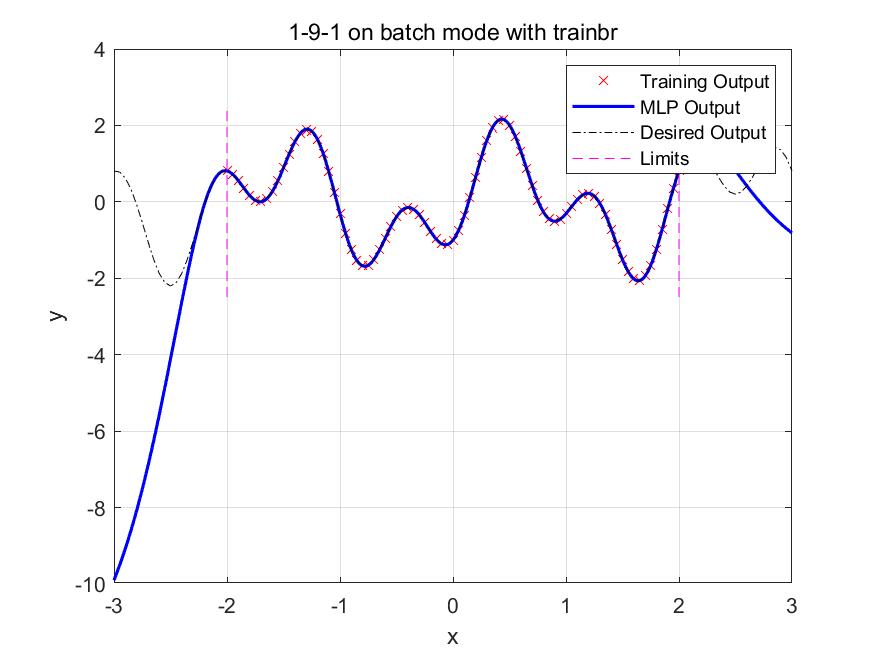
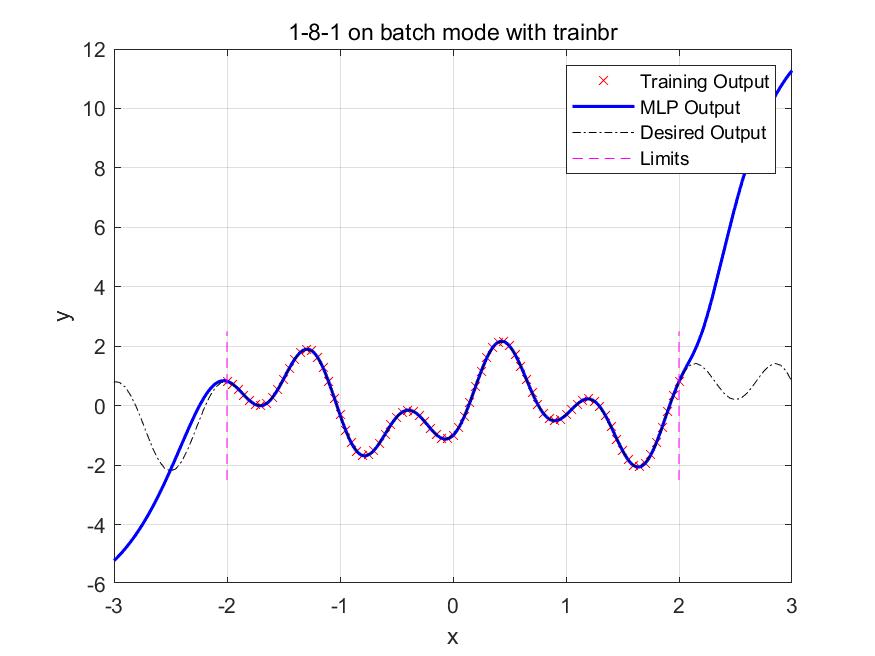
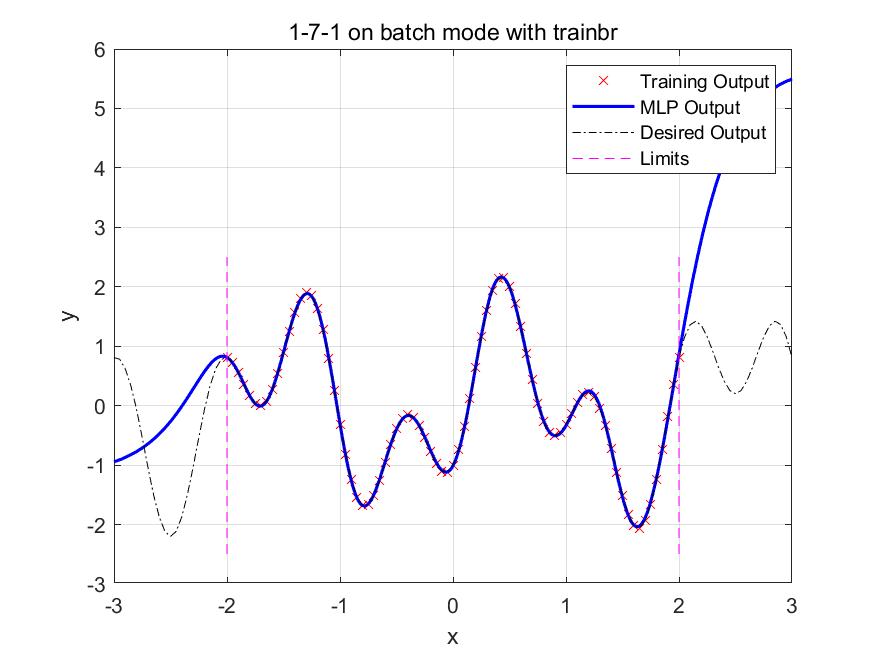
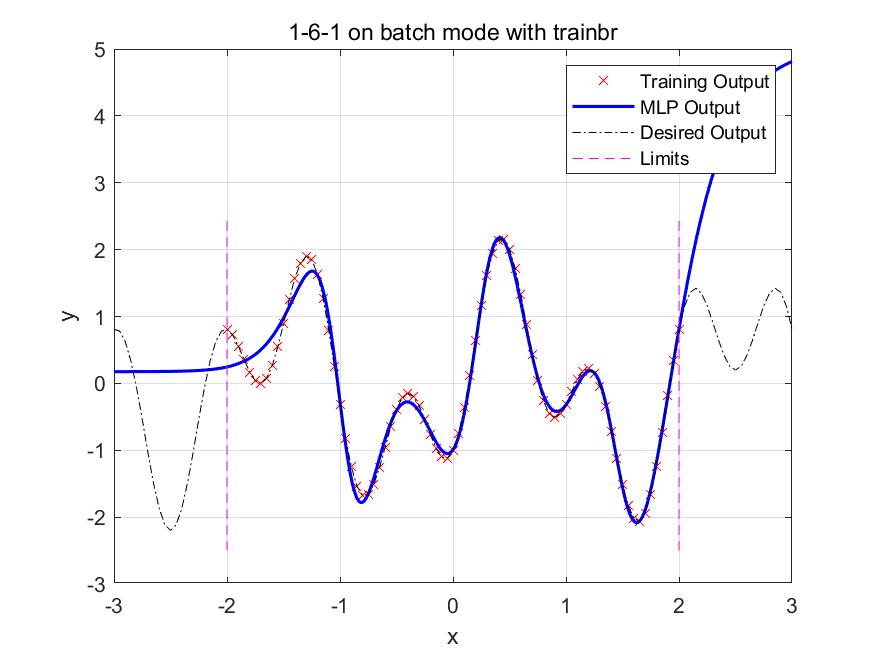
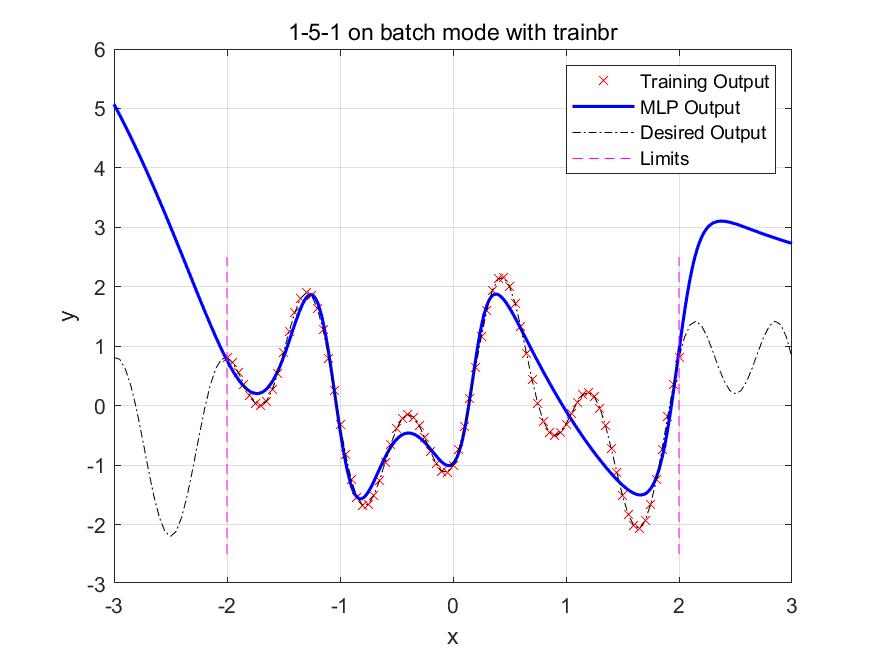
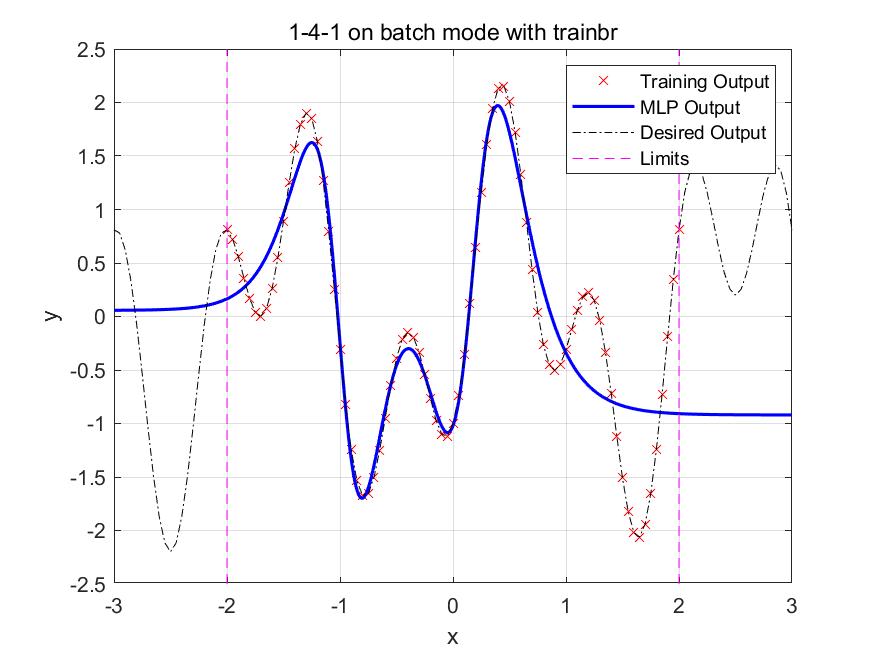
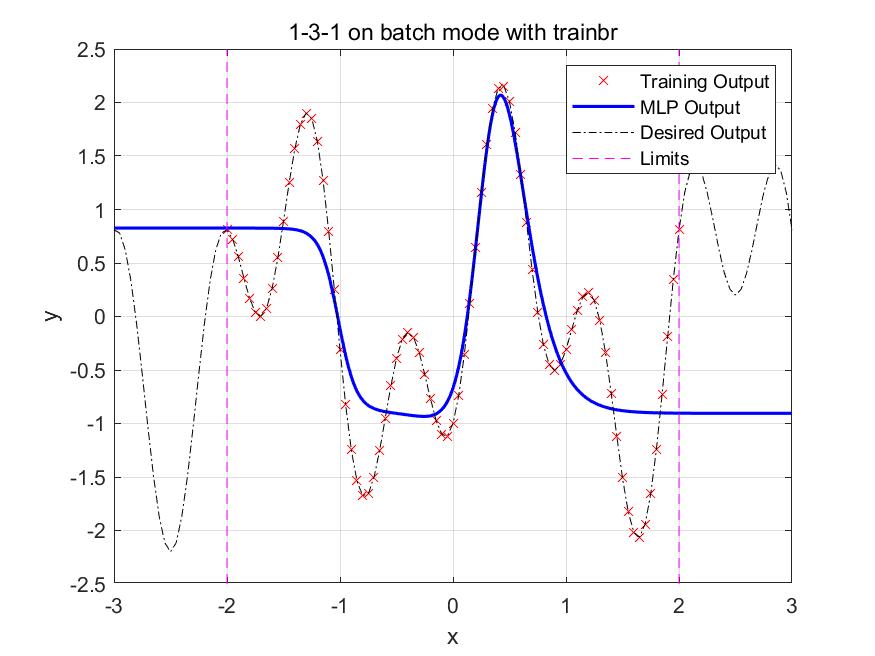
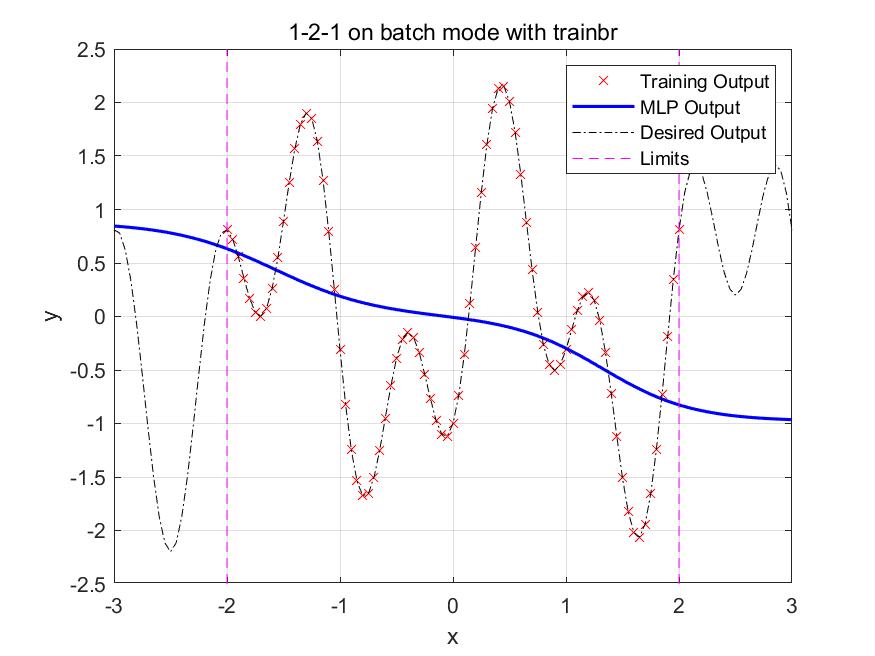
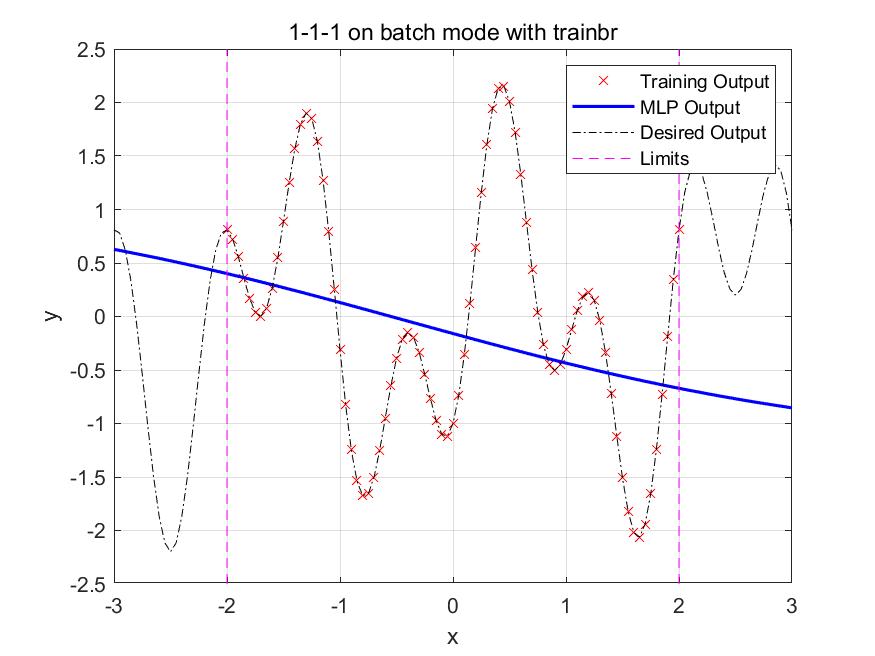
It also show that, if the new sample input is outside the domain of input of the training set, then the result is not correct.

|  |
| --- |
| 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 |

1. 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,





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 |

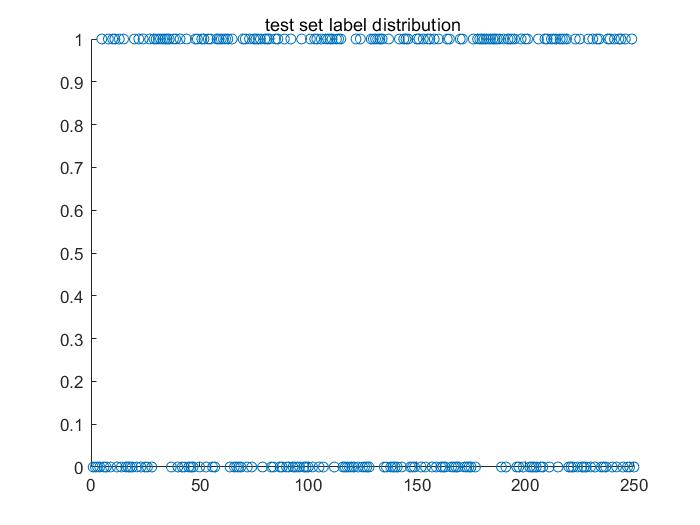
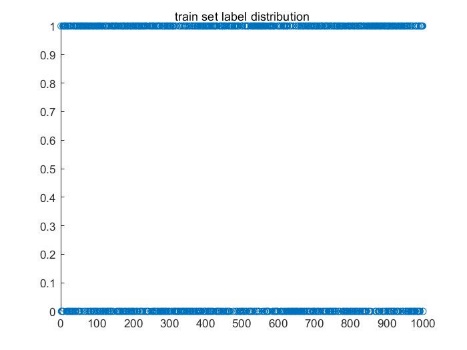
# Solution 3

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.

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

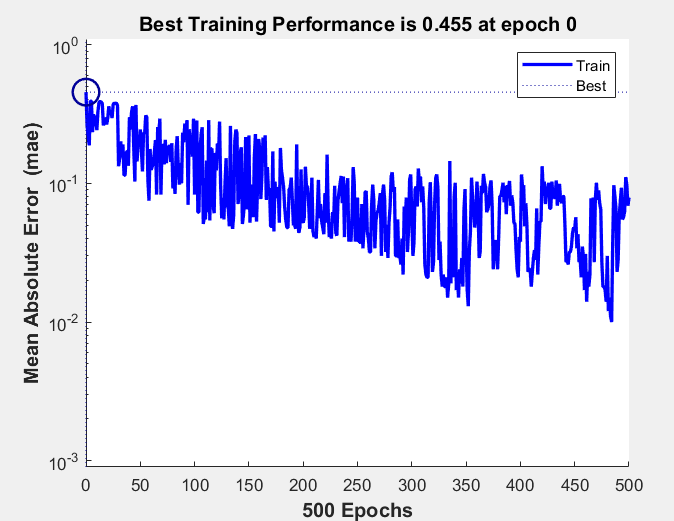
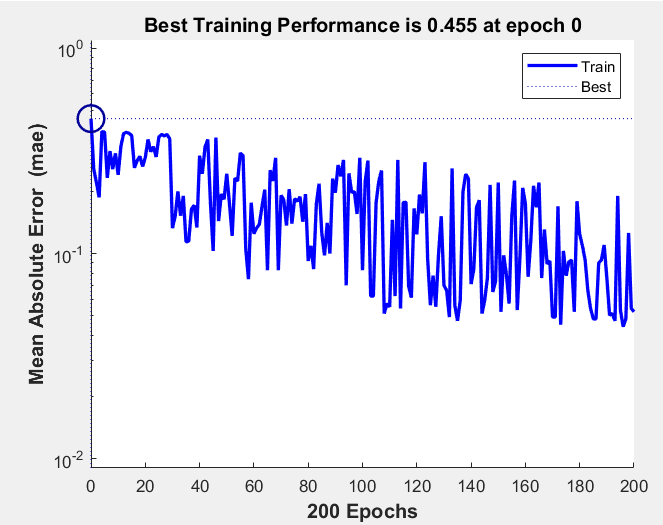
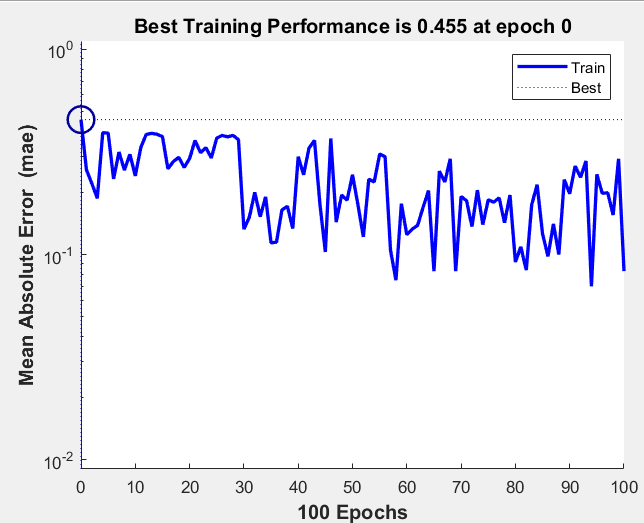
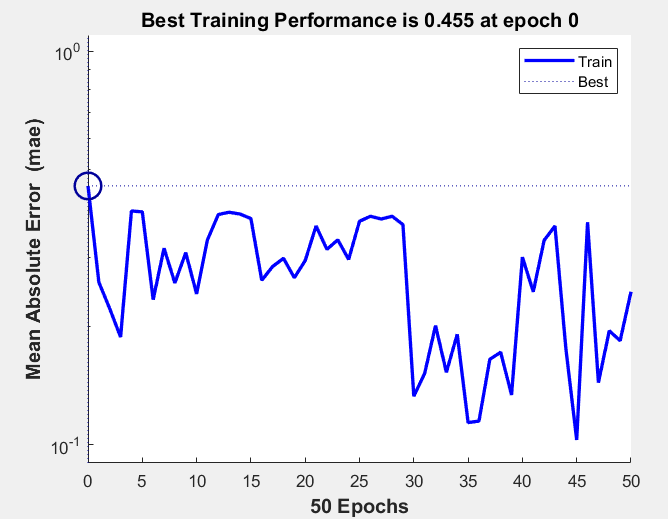


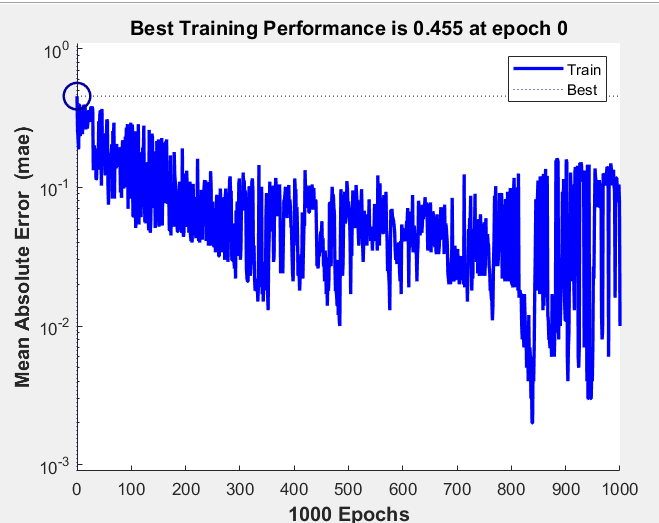
|  |
| --- |
| %init  close all**;**clear**;**clc**;**  %For Training  file\_dir0**=**'D:\2021\NUS-ECE\EE5904-神经网络\homework\HOMEWORK\_TWO\Face Database\TrainImages\'**;**%路径前缀  img\_list0**=**dir**(**strcat**(**file\_dir0**,**'\*.jpg'**));**  att\_list0**=**dir**(**strcat**(**file\_dir0**,**'\*.att'**));**  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"**);** |

1. 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% |

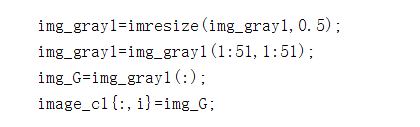




From the performance with different epochs, we can find that the perceptron can classify smile and non-smile. Especially for the test set, when we set 1000 epochs, the accuracy is close to 75.6%.

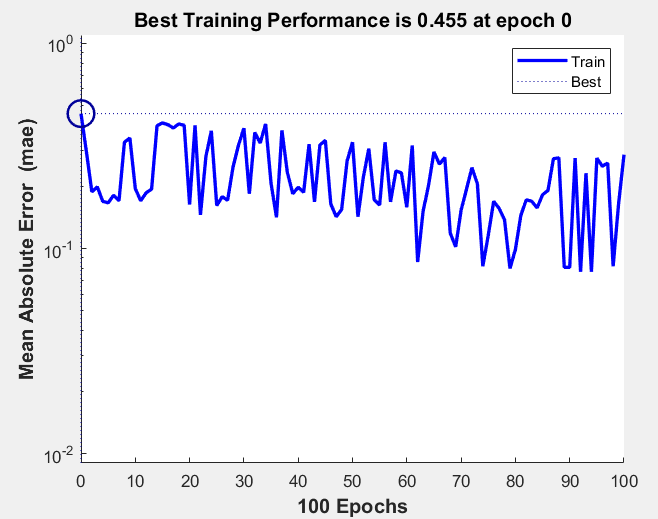
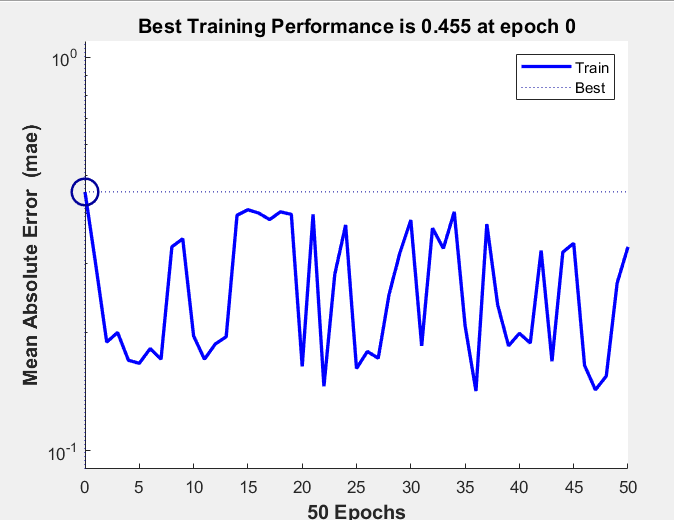
|  |
| --- |
| %init  close all**;**clear**;**clc**;**  %For Training  file\_dir0**=**'D:\2021\NUS-ECE\EE5904-神经网络\homework\HOMEWORK\_TWO\Face Database\TrainImages\'**;**%路径前缀  img\_list0**=**dir**(**strcat**(**file\_dir0**,**'\*.jpg'**));**  att\_list0**=**dir**(**strcat**(**file\_dir0**,**'\*.att'**));**  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**);** |

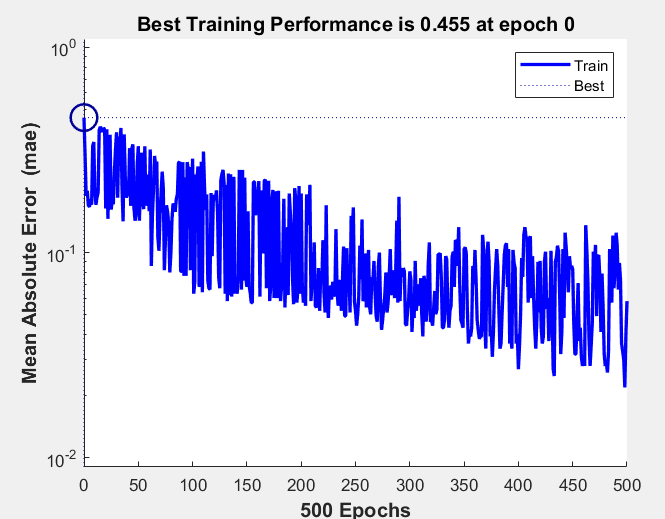
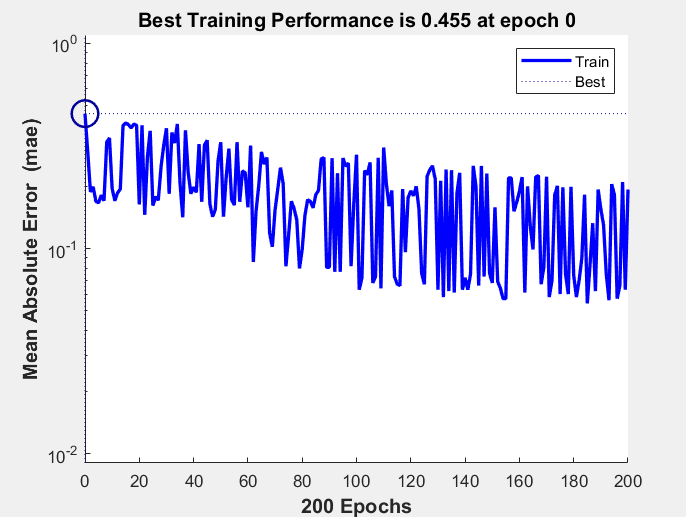
1. To finish this task, I just naively downsample the images. I use imresize function in this part. As we all know, we take the similar procedures in part b.



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% |

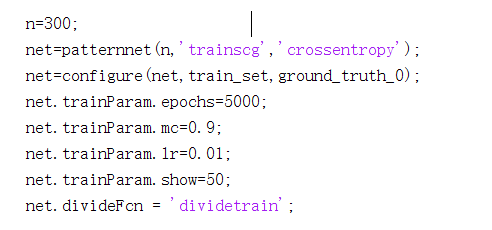




Through the analyze the result of this part ,we can find that if we use this downsample way to get small scale data, this accuracy would decrease a little. In conclusion, if we want to get more accurate result, we should better use raw data.

|  |
| --- |
| %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**);** |

1. 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.



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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**);** |

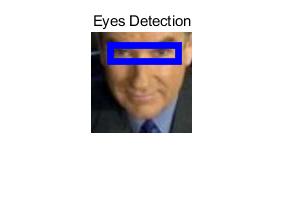
1. By guided by the example code in last lecture, I update the sequential code, and take similar way as part d. Because large epochs are very time-consuming in sequential method , so I set the learning rate 0.01 and epoch 100. Finally, the result is shown as below.

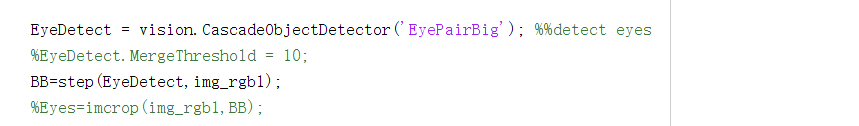
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Test-acc | 97.5% | 96.2% | 98.5% | 99.1% | 99.3% | 99.9% | 99.8% | 99.7% | 99.9% | 100% |
| Test-acc | 77.2% | 76.0% | 78.4% | 76.4% | 78.8% | 77.6%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

1. 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, , that specifies in pixels, the upper-left corner and size of a bounding box.





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**);** |