Q1 Rosenbrock's Valley Problem (15 Marks)

You should give the formulas for the gradient and the Hessian you have computed. Otherwise, if you only show the plots and codes whereas you don't compute and show the two formulas explicitly, 5 points will be deducted.

a) Global minimum

It is straightforward to figure out that the global minimum is

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

b) Steepest (Gradient) descent method

The gradient of the given function can be computed as

$$\nabla f(x,y)=\begin{bmatrix} 400x^3-400xy+2x-2 & 200(y-x^2) \end{bmatrix}^T \tag{2}$$

The stop criterion for the gradient descent learning algorithm is set to be $\varepsilon=10^{-10}$, that is, when $abs(f)\le\varepsilon$ is true, we stop the iteration and record the iteration number. For the MATLAB experiment, the starting point is randomly initialized to be $\begin{bmatrix}0.5434 & 0.2784\end{bmatrix}^T$.

First, set the learning rate to be $\eta=0.001$, and it turns out that the gradient descent algorithm takes 25262 iterations to reach the stop criterion. The trajectory of the two parameters x and y is shown in Figure 1.

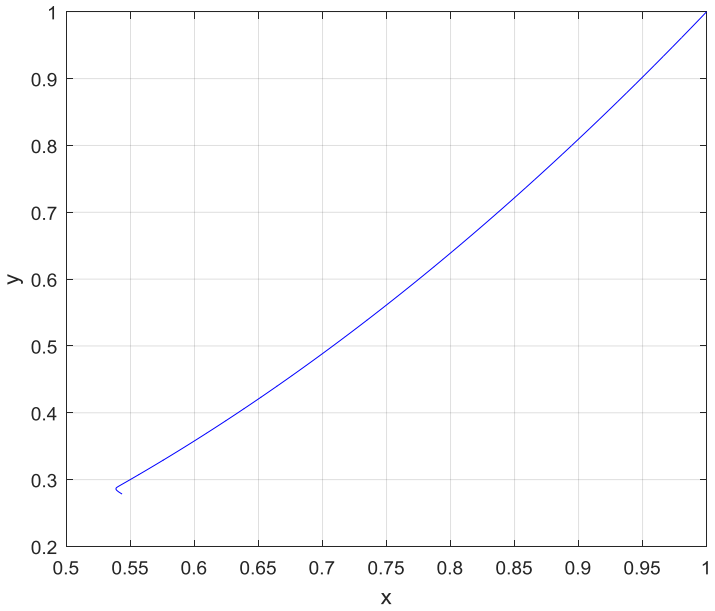

Figure 1 Trajectory of (x, y) for $\eta=0.001$ with gradient descent

The final value for parameters are $x=0.999990008114697$ and $y=0.999979976345482$ after **25262** iterations. The corresponding trajectory of the function value is given in Figure 2. As can be seen, after the first 2500 iterations, the function value continues to decrease very slowly using the gradient descent method. This is due to the inherent feature of the Rosenbrock's valley problem, where the global minimum is inside a long, narrow and parabolic shaped flat valley. This kind of

terrain will make the gradient descent algorithm converge very slowly.

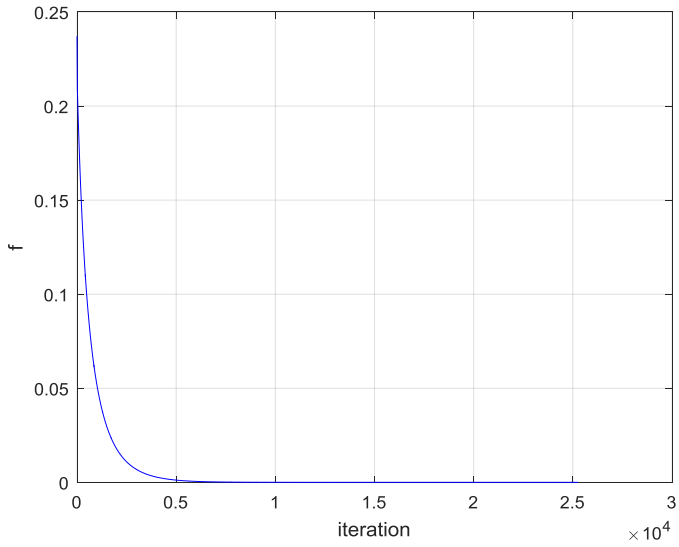

Figure 2 Trajectory of the function value for $\eta = 0.001$ with gradient descent

Now repeat the above experiment with a larger learning rate $\eta = 1$. It turns out the MATLAB program is terminated abruptly because the value of the function goes to infinity. This shows that an inappropriately large learning rate will make the gradient descent algorithm diverge and fail.

c) Newton's method

Now try the Newton's method. The update law for Newton's method is

$$\begin{bmatrix} x & y \end{bmatrix}^T = \begin{bmatrix} x & y \end{bmatrix}^T - H_{f(x,y)}^{-1} \nabla f(x,y) \tag{3}$$

where $H_{f(x,y)}$ is the Hessian of the given function. It is computed as

$$H_{f(x,y)} = \begin{bmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{bmatrix} \tag{4}$$

With the same starting point and stop criterion as the previous gradient descent method, we then try to find the minimum using the Newton's method. It turns out that only **10** iterations are needed to reach the same stop criterion. The trajectory of the $\begin{pmatrix} x & y \end{pmatrix}$ is illustrated in Figure 3.

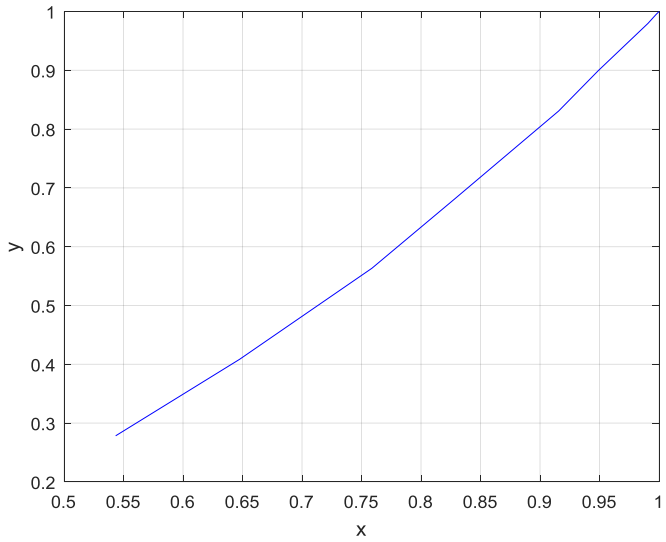

Figure 3 Trajectory of (x, y) using Newton's method

The trajectory of the function value with the Newton's method is given in Figure 4.

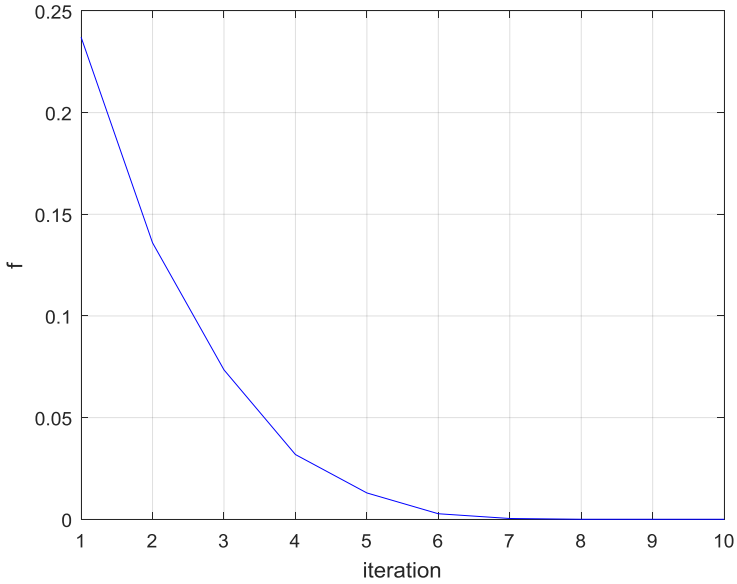

Figure 4 Trajectory of the function value with Newton's method

The final value for the parameters are $x = 0.999999974596114$ and $y = 0.999999948684952$. It is obvious that Newton's method is much faster than the gradient descent algorithm since it takes just 10 iterations while gradient descent method takes 25262 iterations. However, Newton's method is faster at the cost of computing the Hessian matrix, which may be quite complicated in some cases.

Q2 Function Approximation (20 Marks)

In this problem, we use a 3-layer MLP to approximate the given function. The function to be approximated is drawn in Figure 5. According to the geometric guide in our lecture, the minimal number of hidden neurons for a good approximation of this given function is about 10. In the following, we will try to validate this conclusion.

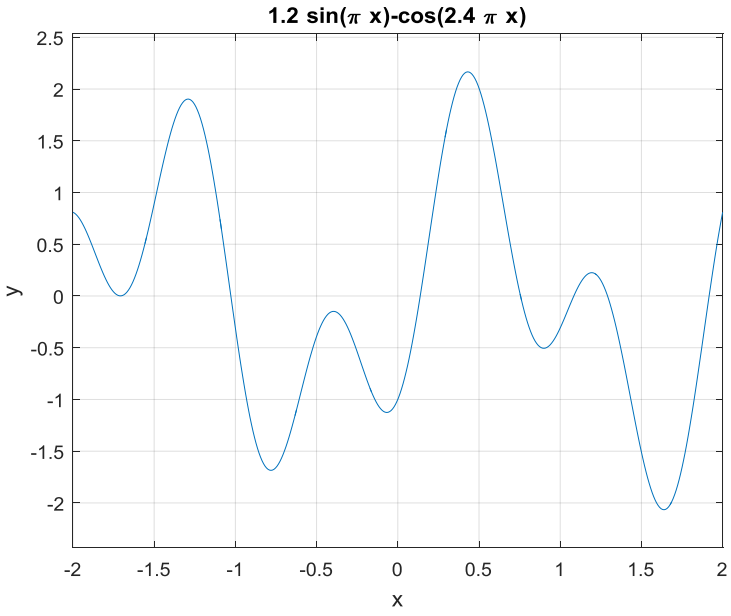

Figure 5 the function to be approximated

For all the following 3 sub problems, the maximum epochs for training is set to be 1000.

a) sequential mode with BP algorithm

First we try the sequential mode (incremental training in MATLAB), which can be done using the *adapt* function with sequential input (MATLAB 2018a)[1]. The fitting result is shown in Figure 6. (Note that if you have used different training settings, your result may be slightly different.)

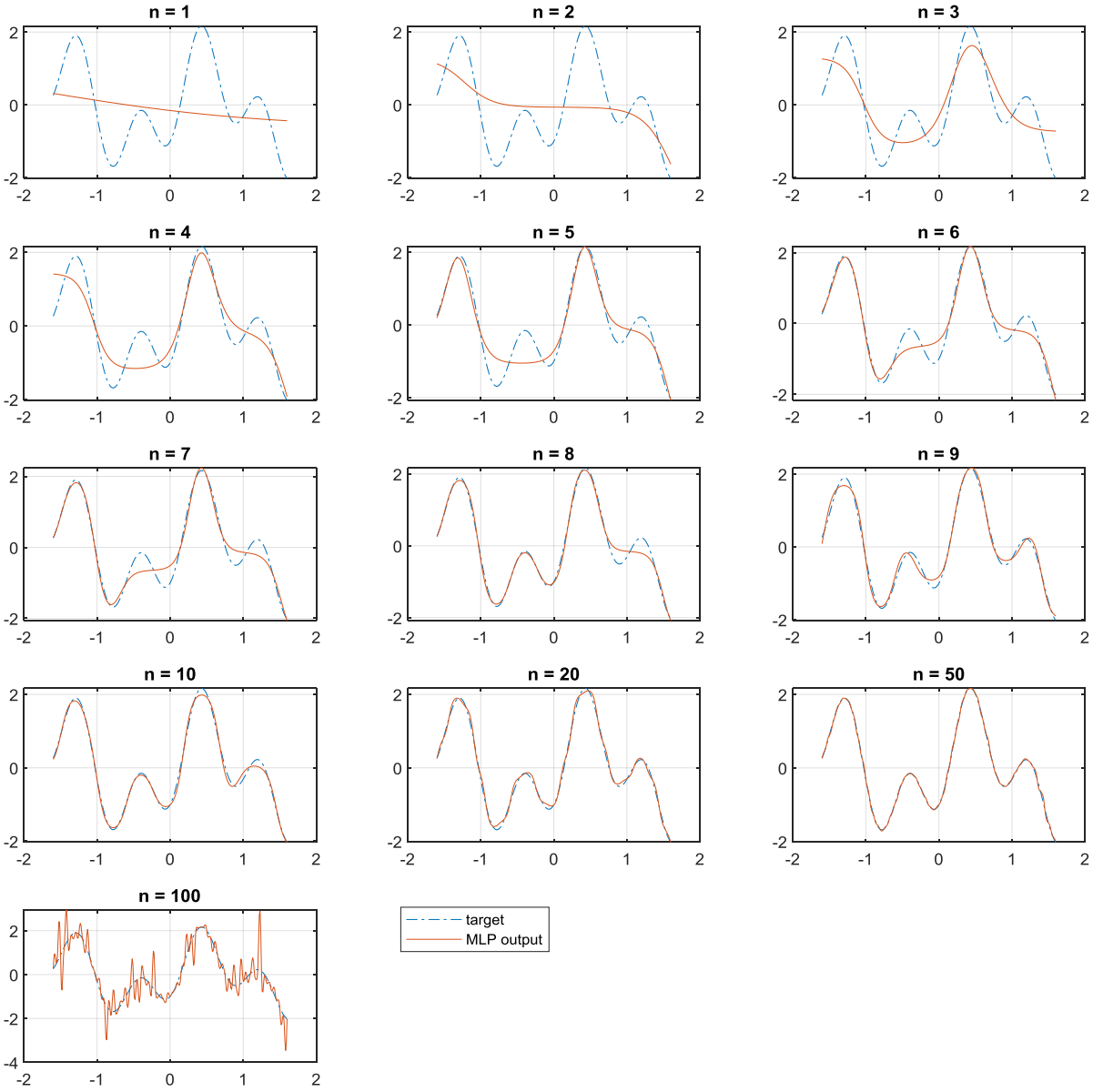

Figure 6 Target output and MLP output for the test set (sequential mode)

It should be noted that we get the above result in Figure 6 with the learning rate as 0.003. After some tests, it can be found that the default learning rate given in MATLAB, 0.01, seems too large for the sequential mode training in this problem, and the network cannot converge well with 0.01.

The fitting status for this problem using the sequential training mode is summarized in Table 1.

[1] http://www.mathworks.com/help/nnet/ug/neural-network-training-concepts.html

Table 1 Fitting status with the sequential training mode

	Under-fitting	Proper fitting	Over-fitting
n	1 - 9	10, 20, 50	100

(Note: here $n = 10$ is not good enough. Thus, you can also say $n = 10$ leads to underfitting.)

Next we use the $n = 10$ MLP to compute the outputs for $x = -3$ and $x = 3$ in order to check its extrapolation ability, and the result is:

$$x = -3 \rightarrow y_t = 0.809, y_{MLP} = 0.0530 \quad x = 3 \rightarrow y_t = 0.809, y_{MLP} = -2.4164 \tag{5}$$

As can be seen, if the new sample input is outside the domain of input of the training set, then the trained MLP can only give a very poor predication. (You can also try an MLP with a different *n*. The result will be the same.)

b) batch mode with trainlm algorithm

In this case, we use the batch mode training with the trainlm algorithm. The result is given in Figure 7 (on the next page).

The fitting status for this problem using the trainlm batch mode is summarized in Table 2.

Table 2 Fitting status with the batch training mode, trainlm

	Under-fitting	Proper fitting	Over-fitting
n	1 - 6	8-10, 20, 50	100

Compared with the sequential mode, we can see that when *n* = 8, a reasonable fitting performance is achieved with the trainlm batch mode, while it is obviously under-fitting with the sequential training mode.

Now we use the $n = 10$ MLP to compute the outputs for $x = -3$ and $x = 3$, the result is:

$$x = -3 \rightarrow y_t = 0.809, y_{MLP} = -5.0567 \quad x = 3 \rightarrow y_t = 0.809, y_{MLP} = 3.2343 \tag{6}$$

It is obvious that the MLP cannot extrapolate well. Thus, MLP cannot make reasonable predictions outside the input domain of the training set.

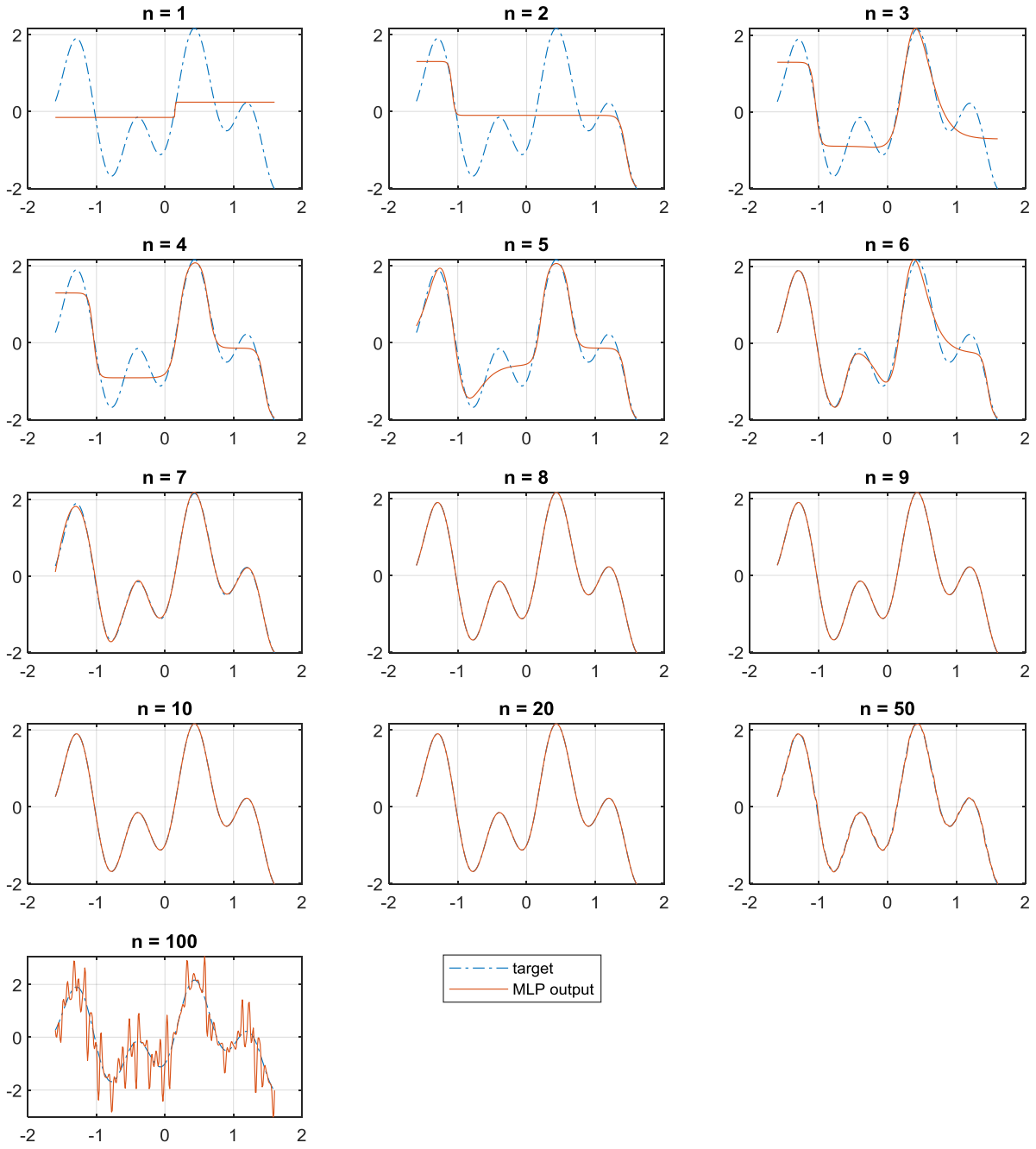

Figure 7 Target output and MLP output for the test set (batch mode, trainlm)

c) batch mode with trainbr algorithm

Use the trainbr algorithm to repeat the above batch mode training procedures. The fitting result is shown in Figure 8 (on the next page).

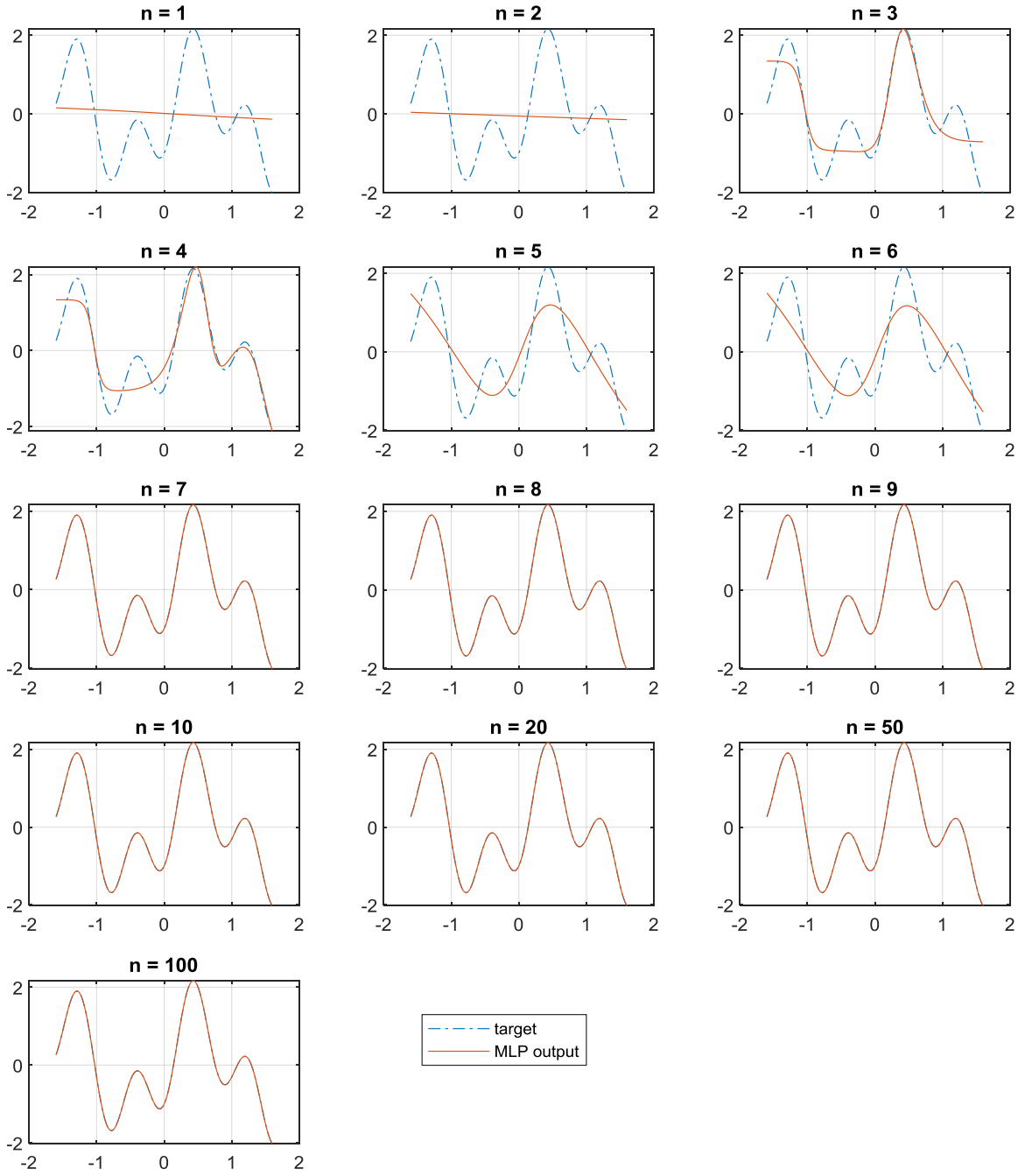

Figure 8 Target output and MLP output for the test set (batch mode, trainbr)

The fitting status for this problem using the batch trainbr mode is summarized in Table 3.

Table 3 Fitting status with the batch training mode, trainbr

	Under-fitting	Proper fitting	Over-fitting
n	1 - 6	7 - 10, 20, 50, 100	none

Compared with the above trainlm batch mode, this trainbr batch method can achieve a better fitting performance. However, you may find that the result produced by the trainbr batch mode may be odd sometimes, and you may need to run the code for multiple times to get the one similar to Figure 8.

As expected, the MLP trained with trainbr cannot handle extrapolation neither. The predications of $x=-3$ and $x=3$ with the $n=10$ MLP are given in equation (7).

$$x=-3\rightarrow y_t=0.809, y_{MLP}=1.6931 \quad x=3\rightarrow y_t=0.809, y_{MLP}=3.1540 \qquad (7)$$

Therefore, no matter how the MLP is trained with the sequential or batch mode, it cannot make reasonable predictions outside the domain of the training set input.

In MATLAB Neural Network Toolbox, batch training is significantly faster and can produce smaller predication errors than sequential training[2]. You may realize this fact during your training.

Q3 Image Classification (40 Marks)

For this task, though there are no detailed requirements like Q2, please do NOT only list your MATLAB code and a single number denoting the accuracy. Please give more details about your network design, the training process, and use tables/figures to better show your results. Besides, give your comments and analysis based on your observations. Otherwise, a 5 – 10 point penalty may be applied on a case by case basis. Finally, if you don't explicitly specify which group of data you are dealing with, 3 points will be deducted.

Since you may have different tasks for different matriculation numbers, at first please consider the following general implementation guidelines.

i) Preprocess the images

Because we only use feedforward MLP in this task instead of the more popular convolutional neural networks for image classification, the images are first converted into grayscale ones using the *rgb2gray* function in MATLAB. After that, the 2D image matrix is flattened into a 1D vector using the *reshape* function. Thus, at the end we have turned each image into a 1D numeric vector of size 1024 which is labelled 0 or 1 for binary classification.

ii) Normalization of the input data

For traditional tabular data, we usually normalize the data using either of the following two methods.

- Standardize. You can normalize the data using the following standard formula

$$x^n=\frac{x-\mu}{\sigma} \qquad (8)$$

where μ is the mean, σ is the standard deviation of the *m* training set samples and x^n is the normalized data. This kind of normalization is also called standardization or variance normalization.

- Map into $[-1,1]$. Input normalization can also be performed by map the input data into $[-1,1]$ with this formula

$$x^n=-1+2\frac{x-x_{\min}}{x_{\max}-x_{\min}} \qquad (9)$$

where x^n is the normalized data and $x_{\max}$, $x_{\min}$ denote the maximum and minimum of the *m*

[2] Mathworks. Neural Network Toolbox User's Guide (R2015b), 3-17, 2015.

training samples.

Please note that in most cases all the two normalizations are performed along each feature dimension respectively, NOT along each sample. Besides, with Neural Network Toolbox in MATLAB, you can just set the preprocess function to be *mapstd* or *mapminmax* to achieve the above two kinds of normalization.

However, with images specifically, for convenience it can be common to subtract a single value from all pixels. Besides, in case of images, the relative scales of pixels are already approximately equal (and in range from 0 to 255), so it is not strictly necessary to further normalizing the image data by dividing the standard variance or scaling into [-1, 1]. For more details, please refer to the online material of the Stanford CS231n course[3].

iii) How many hidden layers? How many output neurons?

Usually one hidden layer is enough. For a *K* class classification problem, you usually need *K* output neurons. However, since this task is a two-class problem, the number of output neurons can be either one or two.

iv) Which activation function to use?

The activation functions for the hidden neurons are tansig. The activation function for the output neuron is preferably logsig since it is pattern-classification problem.

v) Robustness issue

For MLP, due to random initial weights, it is more reasonable to run the program several times and average the final results.

In the following, we take the group 0 (airplane vs. cat) for example to give a reference solution. Note that, according to the requirements of this problem, the training will take 900 images and the test set contains 100 images. NOTE: for the following explorations, you may have different observations/results due to different experiment settings, for example, various training algorithms, learning rates, and the number of hidden neurons. Once you can analyze and justify your own results, you will get the full marks.

a) Rosenblatt's perceptron

Now we use the single layer perceptron to classify the images into two categories: airplane and cat. In MATLAB, the *perceptron* function can be used to implement a single layer perceptron with a fixed learning rate 1 and the normalized perceptron weight and bias learning function *learnpn*. (In this question, for comparison with question b), no normalization of the image data is applied.)

The initial weights are set to be all 0. The max number of epochs is set to be 300. The recognition accuracy is summarized in Table 4.

Table 4 Result of cat/airplane classification using the Rosenblatt's perceptron

	Total sample number	Classification errors	Classification accuracy
Training set	900	438	51.33%

[3] http://cs231n.github.io/neural-networks-2/#datapre

Test set	100	49	51.00%

From Table 4 we can see that, due to its simple structure, the perceptron cannot classify cats and airplanes. The classification accuracy is close to 50%, i.e., close to random guess. Next, we will try more complicated multi-layer perceptron (MLP) for this classification problem to see whether a better result can be obtained.

b) Rosenblatt's perceptron with standardization

Standardize the images and repeat the perceptron training in a) again. Note that you should first subtract the global mean and then divide the data by the global standard deviation such that the normalized (standardized) data have a zero mean and a unit variance. However, since the explanation in this question is not clear enough, no points are deducted even if you use the global variance instead. The results are given below.

Table 5 Result of cat/airplane classification using the Rosenblatt's perceptron after standardization

	Total sample number	Classification errors	Classification accuracy
Training set	900	114	87.33%
Test set	100	41	59.00%

Obviously, the standardization operation has improved the classification accuracy on both the training and the test set. However, the classification accuracy on the test set is very low compared with the one of the training set, which implies the classification capacity of the perceptron cannot generalize well to the test set. By such standardization preprocessing, we have zero-center the data and then make them unit standard variance. The data after standardization usually helps the learning of a classifier.

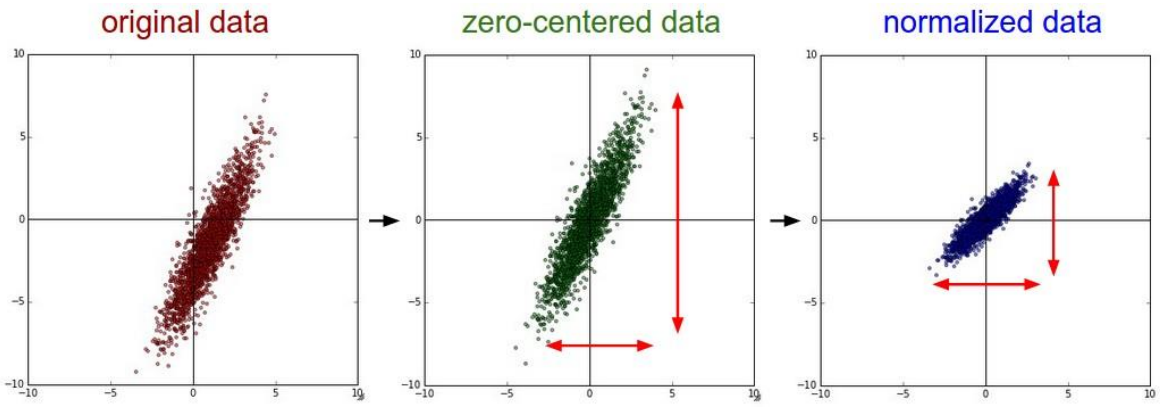

Figure 9 Common data preprocessing pipeline. Left: Original toy, 2-dimensional input data. Middle: The data is zero-centered by subtracting the mean in each dimension. The data cloud is now centered around the origin. Right: Each dimension is additionally scaled by its standard deviation. The red lines indicate the extent of the data - they are of unequal length in the middle, but of equal length on the right. (Source: cs231n)

In the following training using MLP, we will feed it the standardized data for better performance. NOTE: the mean and the standard variance are computed using only the training set, but they are applied to both the training and test set.

c) Multi-layer perceptron

In this question, we apply the MLP for airplane-vs-cat classification using batch mode of learning. For a 3-layer MLP with n hidden neurons, various n's are tested for cat-airplane classification and the result is shown in Table 6. The maximum number of epochs is 500. However, as you may have observed, MATLAB will terminate the training in advance once the gradient is smaller than the threshold $1e^{-6}$ by default. Such early stopping is usually caused by the perfect classification of the training set, i.e., no further improvement can be made.

Table 6 Classification errors for the training set and test set with various n's

		n=1	n=2	n=5	n=10	n=20	n=50	n=100
Classification errors	training	16	15	0	0	0	0	0
	test	42	40	45	34	31	32	27

As can be seen from the above table, it seems that starting from $n = 5$ the training set can achieve zero classification errors. However, the errors on the test set are still large, which indicates severe overfitting.

d) Multi-layer perceptron with regularization

- Overfitting detection

Let's take the number of hidden neurons $n = 50$. From a) we know that the current MLP must be overfitting. To determine when the overfitting begins, we need to inspect the loss of the validation set during training. This can be done by MATLAB easily. The key is to divide the dataset into two parts, training and validation, according to a certain ratio. The corresponding code is given below.

```
n = 50;
epochs = 500;
net = patternnet(n);
net.divideFcn = 'divideblock';
net.divideParam.trainRatio = 0.9;
net.divideParam.valRatio = 0.1;
net.divideParam.testRatio = 0;
net.trainParam.max_fail = 10;
net.trainParam.epochs = epochs;
```

Then using the neural network training GUI or the *plotperform* function, we can show the training and the validation performance as the epochs increase (Figure 10).

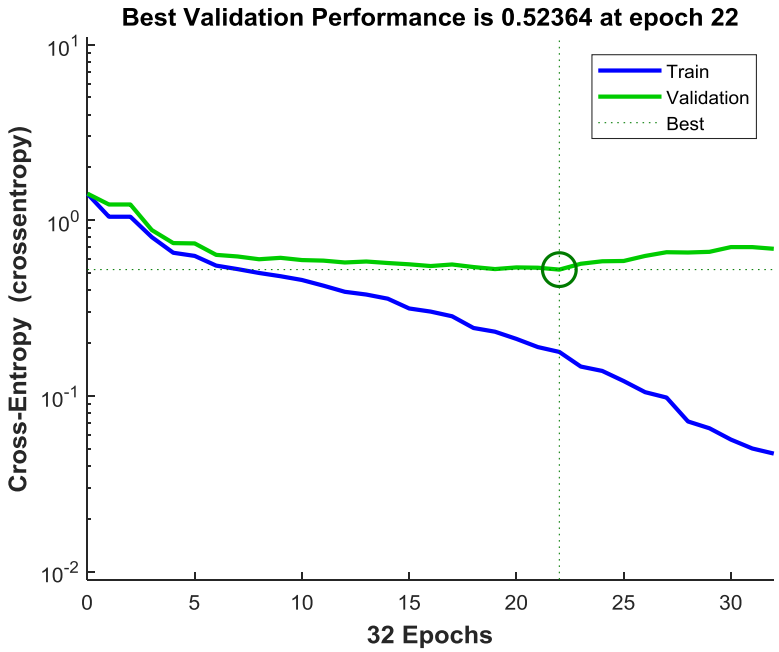

Figure 10 Training and validation performance for n=50

As we can see, overfitting begins after epoch 22. The best validation performance on epoch 22 is about 75 / 100 = 75%, and the training performance on epoch 22 is 712 / 900 = 79%.

- Regularization

To test the role of effectiveness of regularization, we can repeat the experiment in c). Take $n = 50$ and set the maximum number of epochs 500. Then, enable regularization by net.performParam.regularization = 0.6. It turns out that the training process will stop after about 90 epochs due to the small gradient limitation. As expected, the classification error can still reach 0 if enough epochs are allowed. Besides, the number of errors on the test set is not stable in multiple trials, varying from 25 to 30. Compared with the results obtained without regularization in Table 6, regularization can slightly improve the generalization perform here, though the test set classification accuracy is still unsatisfactory, only about 70%~75%. (Depending on your task and network structure, you may have different observations here. Just give your conclusion based on your actual results.)

e) Sequential mode training

In this question, the MLP is trained using the sequential mode. The result is given in Table 7. The learning rate is set to 0.01. The network is trained up to 500 epochs.

Table 7 Classification errors for the training set and test set with various *n*'s (sequential mode)

		n=1	*n*=2	*n*=5	*n*=10	*n*=20	*n*=50	*n*=100
Classification errors	training	215	70	19	0	0	0	0
	test	37	45	40	40	34	30	27

Comparing the results of the sequential mode (Table 7) and batch mode (Table 6), we can see that these two modes give very similar classification performance for the test set, though the batch mode outperforms the sequential mode for the training set when the number of hidden neurons is

small.

As we have mentioned before, in MATLAB Neural Network Toolbox, batch training is significantly faster and can produce smaller predication errors than sequential training[4]. Hence, if we need to do offline training for an MLP in MATLAB, the batch mode is preferred obviously. In practice, if the training set is very large, for instance, composed of thousands of images, then a mini-batch training mode may be preferred.

f) Possible improvements

There are no standard answers to this question. In the research field of computer vision, many techniques/tricks have been proposed to boost the image classification performance using neural networks, especially deep convolutional neural networks (CNN). Following are some examples.

- Collect more data. If possible, this should be the most effective way to improve the performance of neural networks. More images for training will usually boost the classification performance.
- Data augmentation. We can augment the dataset by flipping, rotating, or cropping the images in order to get more training examples and make the network adapt better to different views of an object.
- Dimension reduction. We can first reduce the dimension of the inputs by performing PCA.
- Use full color information. Note that we have transformed the RGB images into grayscale at the very beginning. It may be better if we use all the information from the three-color channels. However, in this case the image vector has size 1024 * 3, and thus dimension reduction is preferred before feeding these data into neural networks.
- Use other activation functions**.** Currently in deep CNN for image classification, the mostly widely adopted activation function of the hidden layers is ReLU[5]. We may also try ReLU instead of tansig in our MLP.
- Deep neural networks. We can try more hidden layers along with deep learning techniques like batch normalization and dropout.

About Python, PyTorch, and TensorFlow

It is OK if you prefer Python for this assignment using frameworks like PyTorch and TensorFlow. However, please pay attention to the requirements for Q3 a) and b), which state you should apply Rosenblatt's perceptron. There is no built-in support for perceptron in the aforementioned Python frameworks, and thus you have to write your own algorithm to implement a perceptron according to lecture 2. Some students simply use a fully-connected linear layer followed by a sigmoid layer. Then, train the network using gradient descent, sgd, or adam. Unfortunately, what you get in this way is NOT a perceptron but a logistic regression classifier. Consequently, 5 points will be deducted if you

[4] Mathworks. Neural Network Toolbox User's Guide (R2015b), 3-17, 2015

[5] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

do so. A perceptron is characterized by a hard limit and it has its own learning algorithm (see lecture 2 for details).