EE4305 Fuzzy/Neural Systems for Intelligent Robotics

Mini-project for part I: Neural Networks

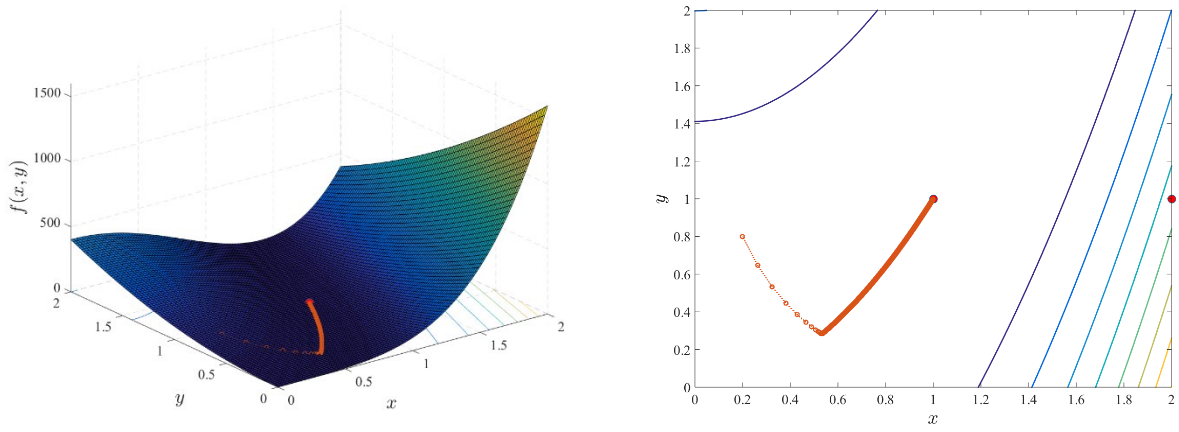
Q1 Rosenbrock's Valley Problem (10 Marks).

a) Steepest (Gradient) descent method:

The gradient vector is derived as

$$g = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 400x^3 - (400y - 2)x - 2 \\ 200y - 200x^2 \end{bmatrix}$$

When learning rate $\eta = 0.001$, the number of iterations is 13,755 and the final function value $f = 9.9975 \times 10^{-7}$ approaches the global minimum, i.e. 0, closely. The trajectory of $f(x, y)$ is plotted in Fig. 1. In addition, the function value versus iterations is given in Fig. 2, from which it can be clearly observed that $f(x, y)$ converges to zero rapidly in the first few iterations while progresses very slowly in the following iterations.



(a) Trajectory in 3D space with contour (b) Trajectory in 2D space with contour

Figure 1 Visualization of $f(x, y)$ trajectories when using gradient descent method.

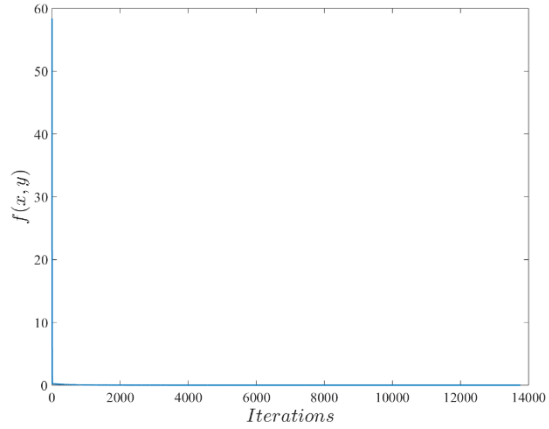


Figure 2 Function value versus iteration number when using gradient descent method.

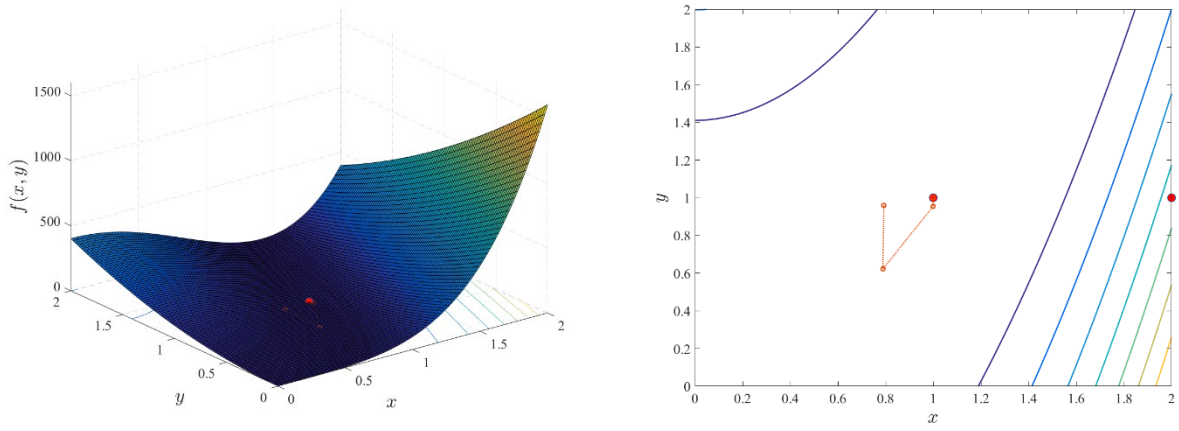
When the learning rate becomes 0.5, which is too large for the first-order Taylor approximation to be valid, the function value will not converge towards the global minimum but rather diverge to infinity.

b) Newton's method:

The Hessian matrix is derived as

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

The number of iterations is 4, and the function value approaches the global minimum with little error, i.e. $f = 1.4611 \times 10^{-7}$, when iteration stops. The trajectory of $f(x, y)$, in this case, is plotted in Fig. 3. Also, the function value versus iterations is displayed in Fig. 4, from which one can find that $f(x, y)$ converges to zero in just a few iterations.



(a) Trajectory in 3D space with contour (b) Trajectory in 2D space with contour

Figure 3 Visualization of $f(x, y)$ trajectories when using Newton's method.

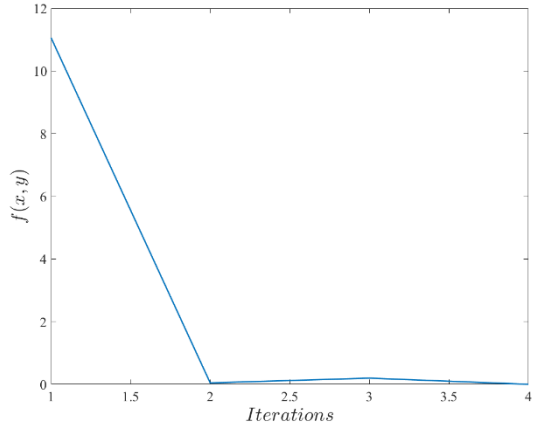


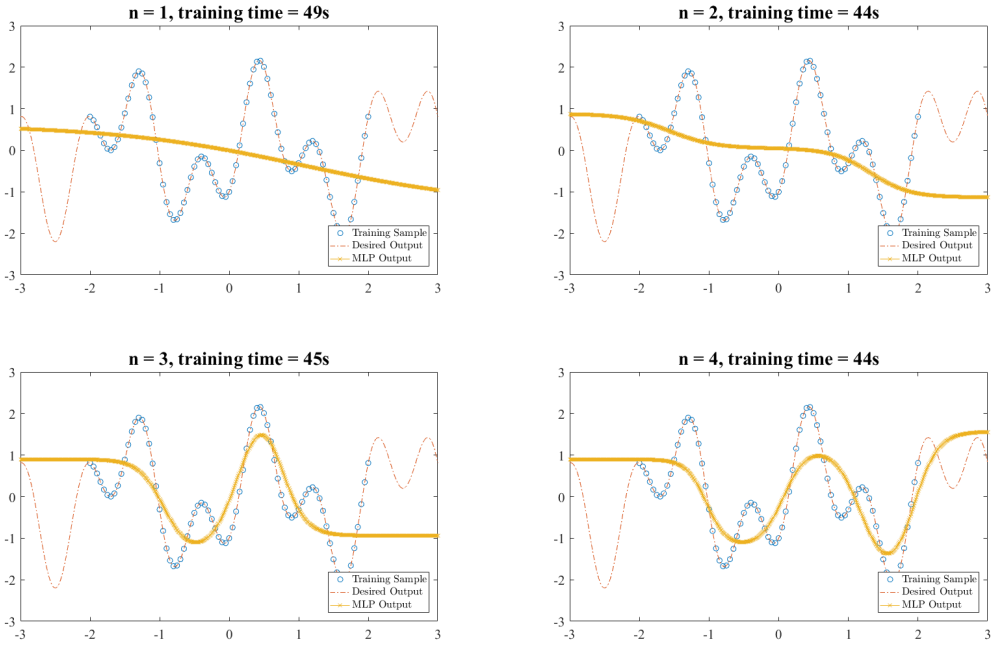
Figure 4 Function value versus iteration number when using Newton's method.

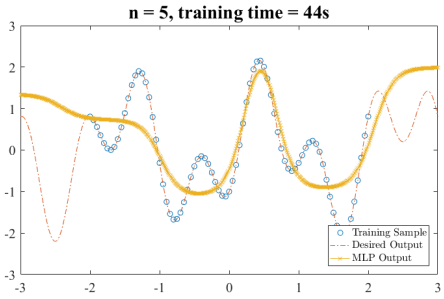
It is noted that the Newton's method, which is based on the second-order Taylor approximation, is much faster than the steepest descent method.

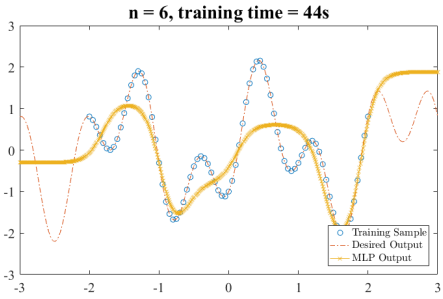
Q2 Function Approximation (20 Marks)

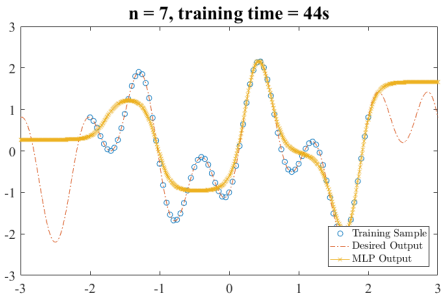
a) Sequential mode with BP algorithm:

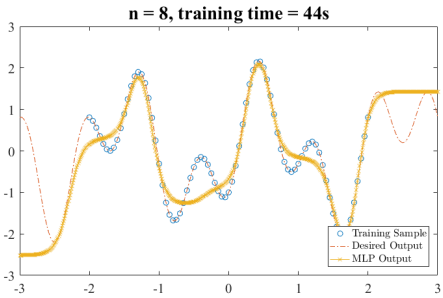
Experiment with different structures of MLP: 1-n-1 (where n = 1, 2, ..., 10, 20, 50, 100); the output of these sequentially trained MLPs upon test samples within [-2, 2] and test samples at $x = \pm 3$ are provided in the following figures.

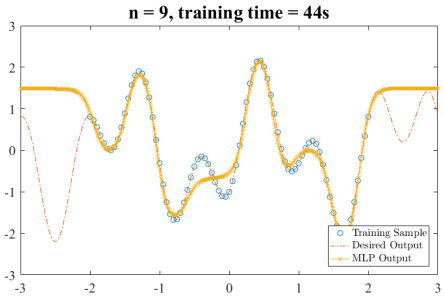


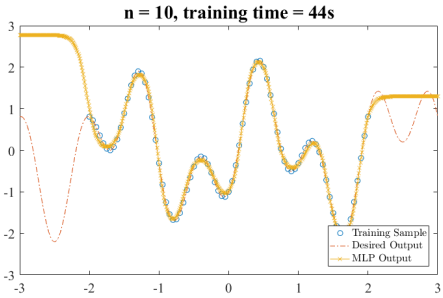

n = 5, training time = 44s
Training Sample
Desired Output
MLP Output

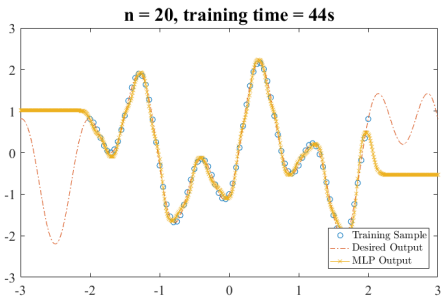

n = 6, training time = 44s
Training Sample
Desired Output
MLP Output

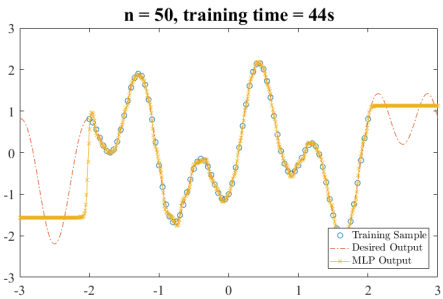

n = 7, training time = 44s
Training Sample
Desired Output
MLP Output

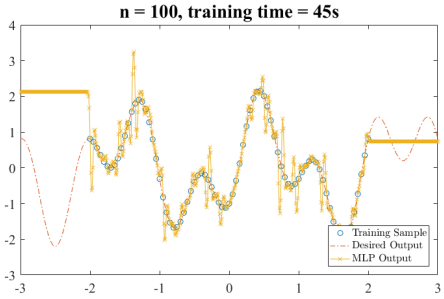

n = 8, training time = 44s
Training Sample
Desired Output
MLP Output


n = 9, training time = 44s
Training Sample
Desired Output
MLP Output


n = 10, training time = 44s
Training Sample
Desired Output
MLP Output


n = 20, training time = 44s
Training Sample
Desired Output
MLP Output


n = 50, training time = 44s
Training Sample
Desired Output
MLP Output



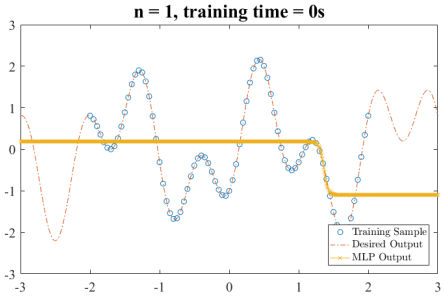
Fitting results are presented in the following Table, where 'UF', 'PF', and 'OF' stand for 'under-fitting', 'proper-fitting', and 'overfitting', respectively.

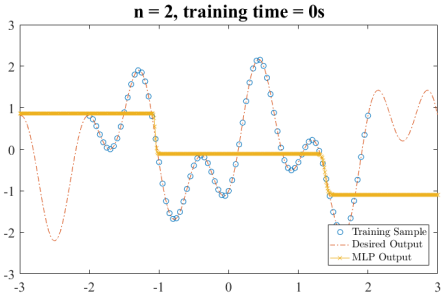
n	1	2	3	4	5	6	7	8	9	10	20	50	100
Performance	UF									PF			OF

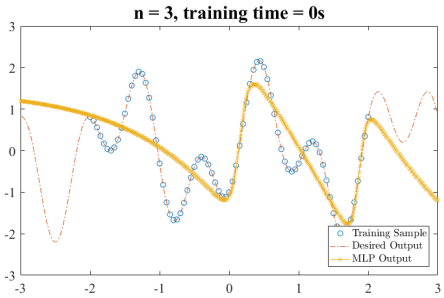
Through this Table, it is clear that the minimal number of required hidden neurons is 10, which is consistent with the guideline given in lecture slides since the investigated function exhibits 10 segments within domain [-2, 2]. Further, when hidden neurons exceed the minimal number too much, say 100, the obtained MLP becomes over-fitting. At last, **none** of these trained MLPs could make reasonable predictions outside of the input domain, e.g. when $x = \pm 3$. It is also noteworthy that the above MLPs are trained with learning rate $\eta = 0.003$, as the default 0.01 is too large for the MLP to converge under the sequential training mode.

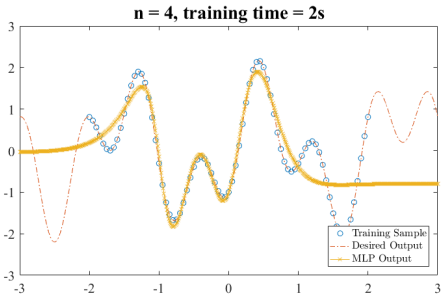
b) Batch mode with trainlm algorithm:

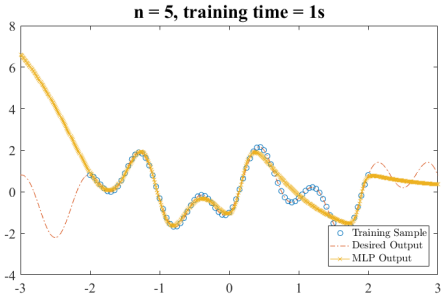
The same experiments are conducted again under batch mode (trainlm), and the obtained results are given as below.

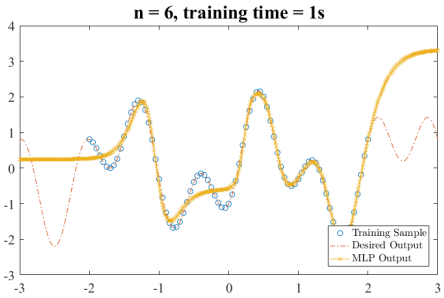

n = 1, training time = 0s
Training Sample
Desired Output
MLP Output

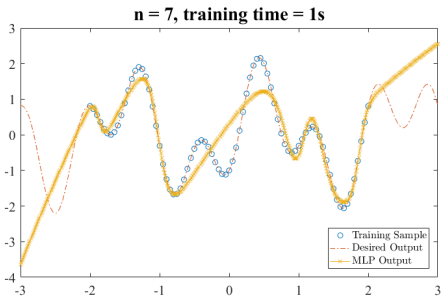

n = 2, training time = 0s
Training Sample
Desired Output
MLP Output

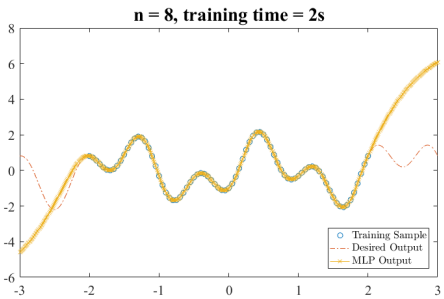

n = 3, training time = 0s
Training Sample
Desired Output
MLP Output

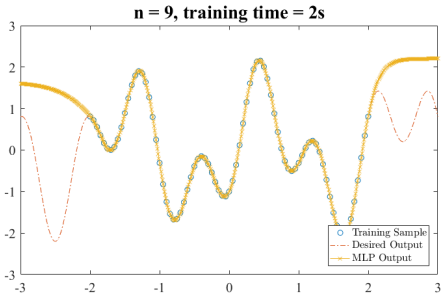

n = 4, training time = 2s
Training Sample
Desired Output
MLP Output

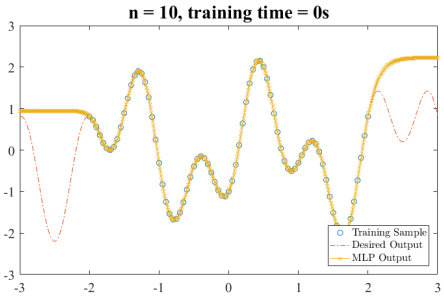

n = 5, training time = 1s
Training Sample
Desired Output
MLP Output

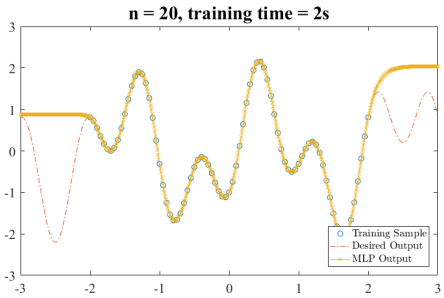

n = 6, training time = 1s
Training Sample
Desired Output
MLP Output

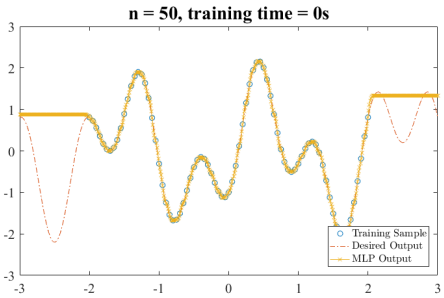

n = 7, training time = 1s
Training Sample
Desired Output
MLP Output

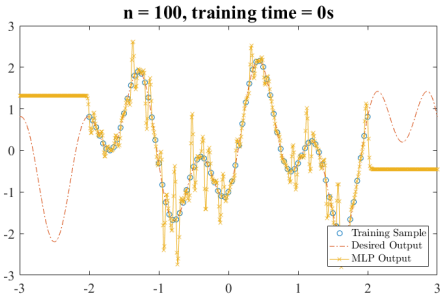

n = 8, training time = 2s
Training Sample
Desired Output
MLP Output











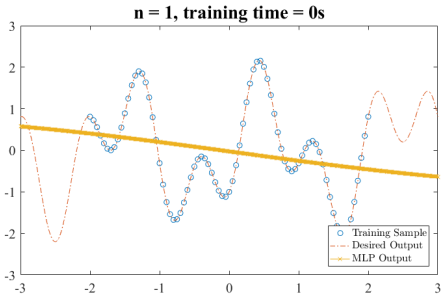
Fitting results are presented in the following table.

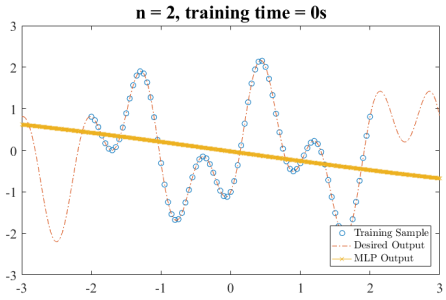
n	1	2	3	4	5	6	7	8	9	10	20	50	100
Performance	UF							PF					OF

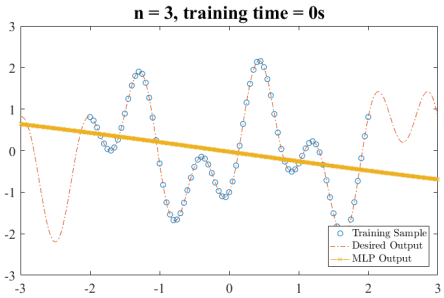
When using batch mode training, the minimal number of required hidden neurons reduces to 8. But still, **none** of these trained MLPs could make reasonable predictions outside of the input domain.

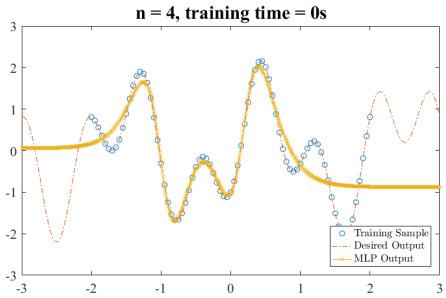
c) Batch mode with trainbr algorithm (Bayesian Regularization):

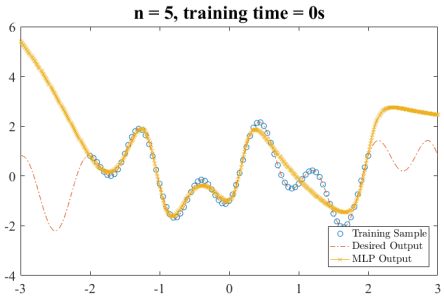
Training again under batch mode but using trainbr, the obtained results are provided in the following figures.

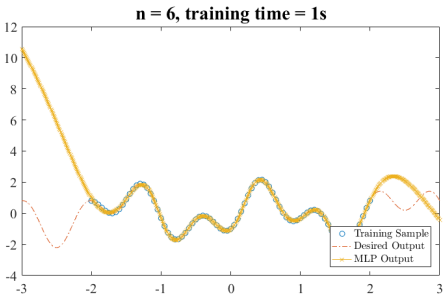


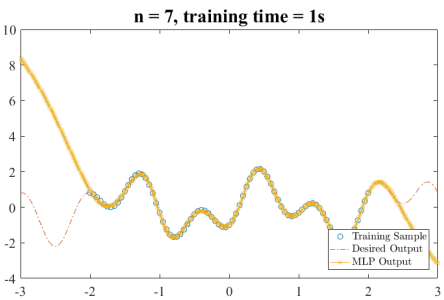


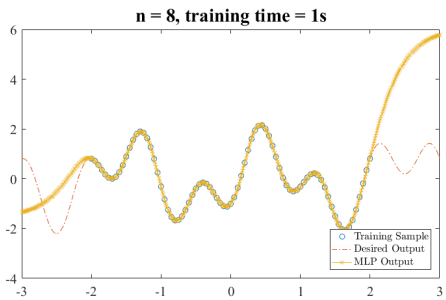


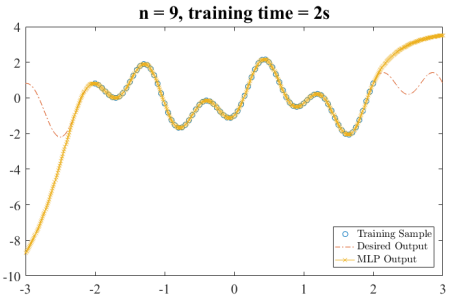


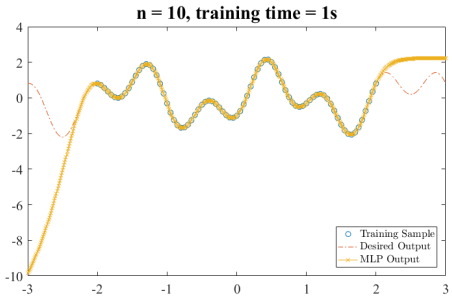


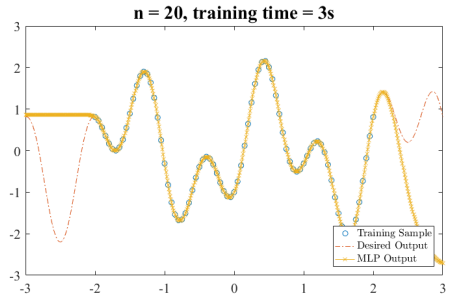


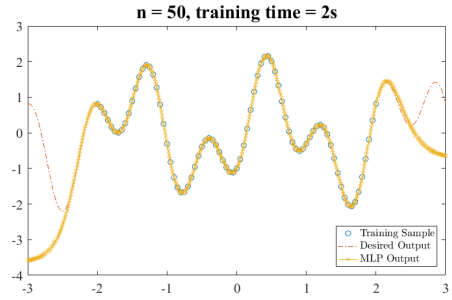


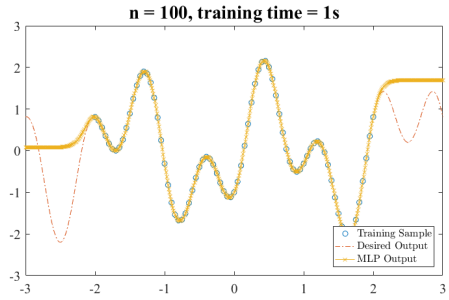












Fitting results are given as below.

n	1	2	3	4	5	6	7	8	9	10	20	50	100
Performance	UF					PF							

In this time, the minimal number of required hidden neurons further reduces to 4, which is not consistent with the guideline given in lecture slides. In addition, MLPs trained by 'trainbr' are still proper-fitting even when the hidden neurons exceed the minimal number a lot. These observations confirmed the effectiveness of **regularization** technique in preventing over-fitting and generalizing the trained MLPs. At last, still **none** of these trained MLPs could make reasonable predictions outside of the input domain.

Therefore, no matter how the MLP is trained, sequential or batch mode, it cannot make reasonable predictions outside the domain of training input. Additionally, when comparing the training time required by b, c) with a), it is clearly found that the training process of batch mode is much faster than that of the sequential mode.

Q3 Facial Attribute Recognition (40 Marks)

i) At first, it is noticed that not all the given images are of the same size 101*101 (some are 102*102). You may either crop or resize all the images into 101*101 for the following operations. Then, please consider the following implementation guidelines on preprocessing the 'images' data matrix, where we assume the matrix is of size (N = image number, D = image dimensionality).

- **Mean subtraction** is the most commonly applied operation in preprocessing. It involves subtracting the mean across every individual feature in the data and has the geometric interpretation of centering the cloud of data points around the origin along every dimension. In MATLAB, this operation could be implemented as: *images = images – repmat(mean(images), N, 1)*. With images specifically, for convenience it can be common to subtract a single value from all pixels: *images = images – mean(images(:))*.
- **Normalization** refers to normalizing the data dimensions so that they are of approximately the same scale. There are two common ways of achieving this normalization. One is to divide each dimension by its standard deviation, once it has been zero-centered: *images = images ./ repmat(std(images), N, 1)*. It only makes sense to apply this preprocessing if you have a reason to believe that different input features have different scales (or units), but they should be of approximately equal importance to the learning algorithm. 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 perform this additional preprocessing step. However, it is still recommended to take another form of normalization, where the minimal and maximal value along each dimension is rescaled to –1 and 1 respectively: *images = 2 * (images – repmat(min(images), N, 1)) ./ (repmat(max(images), N, 1) – repmat(min(images), N, 1)) – 1*. Please note that both the two methods are performed along each feature dimension (i.e. pixel location) respectively, NOT along each image. In addition, with Neural Network Toolbox in MATLAB, you can also utilize *mapstd()* or *mapminmax()* to implement these normalizations.

A common pitfall is to preprocess the training/test images based on the statistics (mean, standard derivation, etc.) that are calculated upon both the training and test sets. Remember that the test data are not available while training the networks! The correct way of preprocessing is to find out the statistics that you need based only upon the training set and 'apply' it to the test set, like:

```
[norm_train_matrix, parameters] = mapstd(training_matrix);
[norm_test_matrix] = mapstd('apply', test_matrix, parameters);
```

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

One hidden layer should be enough for this task. With increasing layers, the capacity (or expressiveness) of network is expanded; however, the overfitting problem becomes more prominent and you will encounter other critical issues such as gradient vanishing/exploding. Certain techniques have been developed to deal with these issues but beyond the scope of this class.

For a K class classification problem, you usually need K output neurons. However, since this task is a binary classification problem, the number of output neurons can be either one or two.

For this task, proper number of hidden neurons can be determined by

- Singular Value Decomposition
- Trial and error; or growing and pruning.

iii) Which activation function to use?

The activation function for hidden layer is '*tansig*'. The activation function for output layer is preferably '*logsig*' for the case of single output neuron and 'softmax' for double output neurons. These are the default settings of *patternnet()*.

iv) Robustness issue.

For MLP, due to randomly initialized weights and local minimums, it is more reasonable to train the model several times and average the final results.

In the following, we take group 1 (male vs. female) as an example to provide a reference solution.

The label distribution of training/test set is illustrated in Fig. 5, from which one can clearly observe that the dataset is **unbalanced**, i.e. the male images are almost three times larger the female's.

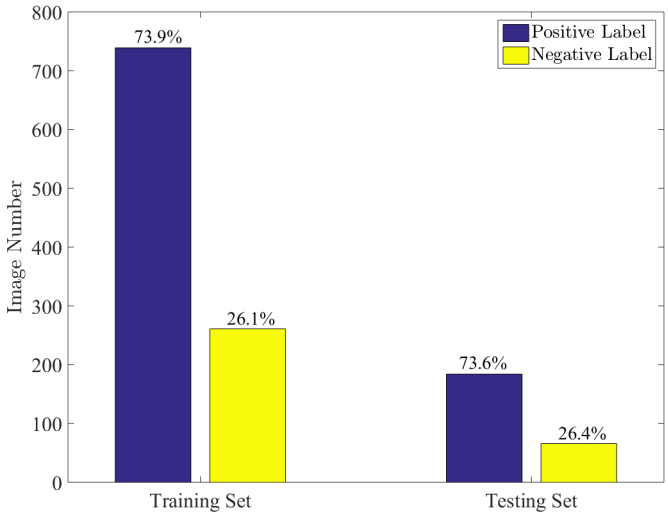


Figure 5 Label distribution of the training/test sets (positive: male; negative: female).

a) We now employ the Rosenblatt's perceptron to classify these images into two categories: male [0] versus female [1]. The images are firstly zero-centered and rescaled to [-1, 1]. Then, the MATLAB build-in function *perceptron()* is utilized to implement the single layer perceptron. After several trials, it turns out that the trained perceptron can successfully classify the training set with no errors (100% training accuracy) indicating the training set is **linearly separable**. However, this perceptron does not generalize well and the accuracy upon test set is merely approaching 85%.

Trails	1	2	3	4	5	6	7	8	9	10	**Avg.**
Train Acc.	100%										**100%**
Test Acc.	82.4%	84.8%	83.6%	84.4%	86.0%	86.0%	86.0%	83.6%	85.6%	85.2%	**84.76%**

b) The original input dimension is 10,201 (101*101), which may be redundant and leave space for reduction. Built upon such observation, PCA is further performed to reduce the dimension. In particular, only the first 512 principle components, in descending order of component variance, are preserved, which in total account for 99.27% of the entire variance. In Fig. 6, the first 12 principle components are visualized for illustration, accompanying with the percentage of the total variance explained by each of them. By projecting the original input onto these preserved components, the input dimension can be effectively reduced to 512. Please **note** that the principle components should be calculated based only on the training set and then apply to the test set.

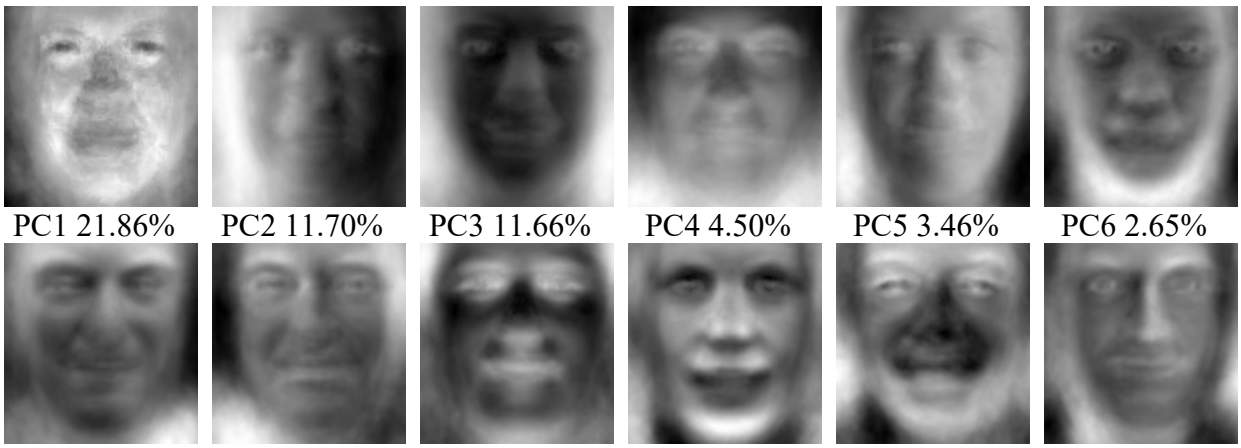


Figure 6 Visualization of the first 12 principle components.

(a) Original image (b) Reconstructed image

Figure 7 Comparison between the original image and reconstructed image based on 512 PCs.

Single layer perceptron in b) is retrained with the dimension reduced input. The corresponding performance is listed as below.

Trails	1	2	3	4	5	6	7	8	9	10	**Avg.**
Train Acc.	100%										**100%**
Test Acc.	84.4%	85.6%	84.0%	84.8%	85.2%	82.0%	86.8%	84.4%	85.2%	84.4%	**84.68%**

Compared with the results in b), it is evident that the obtained 512-dimensional data (20 times less than the original) are still informative and could achieve almost the same performance for classification. Therefore, they are also used in the following experiments for efficiency. In addition to PCA, you are also encouraged to try LDA that reduces the dimension in a supervised manner. PCA and LDA are both general dimensionality reduction techniques which could be applied to a wide range of tasks and guarantee a reasonable performance. However, in order to achieve a better result, you should conduct *feature engineering*, which attempts to represent the input by a set of discriminative features and thus is highly task-dependent. *Convolutional network* (ConvNet) is

your another option, which figures out these features through an end-to-end training approach instead of handcrafting them based on domain knowledge.

c) In this section, we apply MLPs of the 1-n-1 structure to the same task. Specifically, it can be easily implemented by the build-in function *patternnet()* with default settings. For the 1-n-1 MLPs, various hidden neurons numbers are tested, and the averaged results (each over 10 trials) are listed in following Table.

n	1	2	5	10	20	50	100
Avg. Train Acc.	100%						
Avg. Test Acc.	82.2%	81.9%	81.6%	82.0%	82.1%	82.2%	82.5%

As can be seen from the above table, it seems that no matter how large n is, the accuracy remains 100% upon the training set while around 82% for the test set. Such results could be explained by the fact that: In question a) and b), it has been confirmed that the training set is linearly separable, and thus in the MLP case, no matter how we set the number of hidden neurons, 100% training accuracy can be easily guaranteed. It is also noted that our model cannot gain any more insight on the underlaying patterns within the input space, once it achieves 100% accuracy on the training set. Therefore, the trained MLPs cannot generalize well to the test set, since the training set is too **small** to sufficiently cover the sample space and train a sophisticated model.

d) MLPs in c) are retrained under the sequential training protocol. The training epochs are restricted to 200 for time efficiency, and the obtained results are given as below.

n	1	2	5	10	20	50	100
Avg. Train Acc.	100%						
Avg. Test Acc.	82.2%	82.3%	82.2%	82.4%	82.4%	83.2%	82.6%

By comparing the results achieved through sequential training and batch training, we can clearly find that these two training modes yield a similar classification performance. However, as you may have noticed during the experiments, batch training is significantly faster than sequential training which is **time-consuming**. In the meanwhile, batch training is **memory-consuming**, since it has to load and process all the training data at once. Therefore, the choice between batch mode and sequential mode is task dependent: if the data set is small and we do not require online training an MLP, the batch mode is preferred (as in this task); while if the data set is quite large, you're recommended to conduct *mini-batch training*, which performs training based on a 'batch size' of samples at one time, and thus balances the time and memory consumption (batch/sequential training can be considered as the extreme cases of mini-batch training with batch_size = total_sample_number/1). There are also deeper considerations in choosing the batch size, and you're encouraged to find out them during practice.