Student name: Jin Hoontae (A0243155L)

Question 1

a) Show that it has a global minimum at (x, y) = (1, 1) where f(x, y) = 0

The Rosenbrock's Valley function is written as:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2 \quad [1]$$

By expanding the function, we obtain:

$$f(x,y) = 1 - 2x + x^2 + 100(y^2 - 2x^2y + x^4)$$

$$f(x,y) = 100x^4 + x^2 + 100y^2 - 200x^2y - 2x + 1 \quad [2]$$

The gradients of the function with respect to x and y are:

$$\frac{df}{dx} = 400x^3 + 2x - 400xy - 2 \quad [3]$$

$$\frac{df}{dy} = 200y - 200x^2 \quad [4]$$

A global minimum of the function can be found when each gradient (**Eq 3** and **Eq4**) is equal to 0. Since there are no local minimal in the function and the question asks us to show if the global minimum is located at (1,1) coordinates, we can examine by putting the given values into the gradient equations obtained above. Hence, by hand-calculation, we obtain:

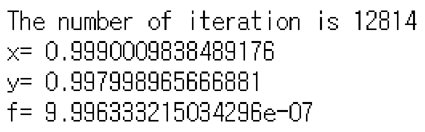
$$\frac{df}{dx} = (400 \cdot 1) + 2(1) - (400 \cdot 1 \cdot 1) - 2 = 0$$

$$\frac{df}{dy} = (200 \cdot 1) - (200 \cdot 1) = 0$$

As can be observed from the above calculation, each gradient function becomes 0 when the given coordinate values are used. Thus, it is true that the Rosenbrock's Valley function has a global minimum at (1, 1).

b-1) Find the global minimum using steepest (Gradient) descent method

When the learning rate was 0.001, it took 12,814 iterations for the function to reach the global minimum coordinates as shown in **Fig 1**. As demonstrated in **Q1-a** where the global minimum is reached when x and y values are 1.0, the values of x and t obtained by using the steepest descent method are also extremely close to 1.0 as shown below. The output value (f) is also close to zero. Hence, the results show that the steepest descent method could find the correct values for the function given.



```
The number of iteration is 12814
x= 0.9990009838489176
y= 0.997998965666881
f= 9.996333215034296e-07
```

Figure 1. The details of the values for the global minimum of the function (when lr=0.001)

As shown in **Fig 2.b**, the values of x and y approach towards the ideal global minimum point (1,1) of the function as enough iterations are attempted. Likewise, as the values of x and y reach the global minimum, the function value also becomes nearly 0.

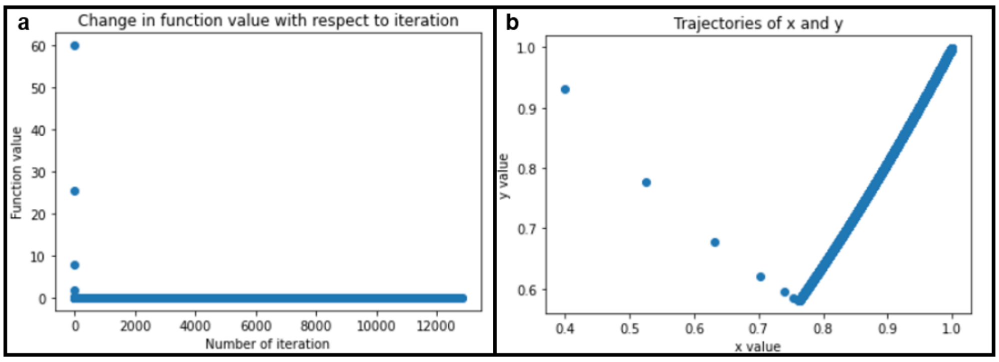
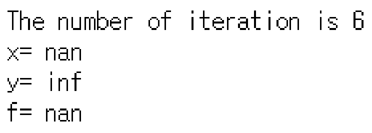


Figure 2. Plots for the function value change and trajectories of x and y for the global minimum

b-1) When the learning rate is 1.0

In contrast, when the learning rate was 1.0, the method was not able to compute the appropriate values as shown in **Fig 3**. This means that the Taylor approximation does not work when the learning rate value is too big, thereby failing the convergence performance.



```
The number of iteration is 6
x= nan
y= inf
f= nan
```

Figure 3. The details of the values for the global minimum of the function (when lr=1)

As shown in **Fig 4**, invalid values were obtained for x, y and f values due the learning rate being too big.

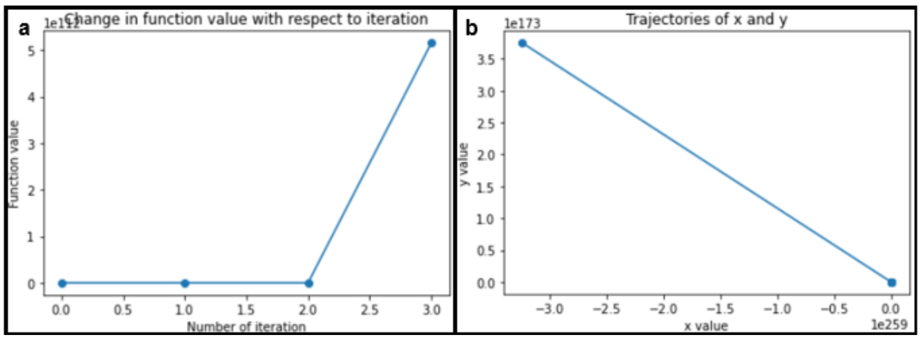
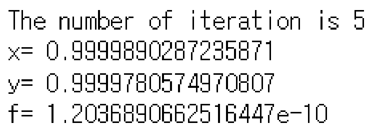


Figure 4. Plots for the function value change and trajectories of x and y for the global minimum

C) Newton's method

To utilize the Newton's method, $\frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial y^2}$ and $\frac{\partial^2 f}{\partial xy}$ were further calculated to construct the Hessian matrix. With the given learning rate (0.001) and the Newton's method, the number of iterations decreased significantly from 12,814 to 5 as shown in **Fig 5**. Furthermore, the values of x,y and f became closer to 0, as compared to those obtained by the steepest descent method.



```
The number of iteration is 5
x= 0.9999890287235871
y= 0.9999780574970807
f= 1.2036890662516447e-10
```

Figure 3. The details of the values for the global minimum of the function using Newton's method

Finally, **Figure 6** proves that the Newton's method reaches the global minimum much faster.

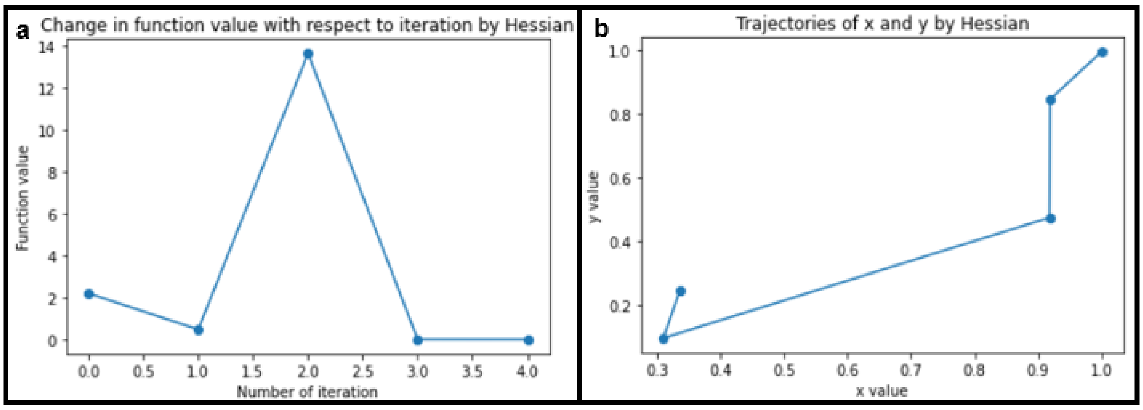


Figure 4. Plots for the function value change and trajectories of x and y for the global minimum by Newton's method

Question 2

a) Use the sequential mode with BP algorithm and experiment with the following different structures of the MLP: 1-n-1 (where n = 1, 2, ..., 10, 20, 50, 100).

a-1) Compare the test samples after training to the desired outputs.

The function given in this question was: $y = 1.2\sin(\pi x) - \cos(2.4\pi x)\ for\ x \in [-1.6, 1.6]$. The training dataset containing the inputs (-1.6 to 1.6 with a uniform step length of 0.5) and the computed outputs were fed into the ANN model for training. After training the model, a test sample containing inputs (-1.6 to 1.6 with a uniform step length of 0.01) was used to interpret how the predicted outputs would look depending on the number of hidden neurons. In total, 13 different hidden neuron groups (1 to 10, 20, 50 and 100 hidden neurons) were tested to examine the fitting level. **Fig 5**. shows that the number of neurons in the hidden layer significantly affects the prediction accuracy.

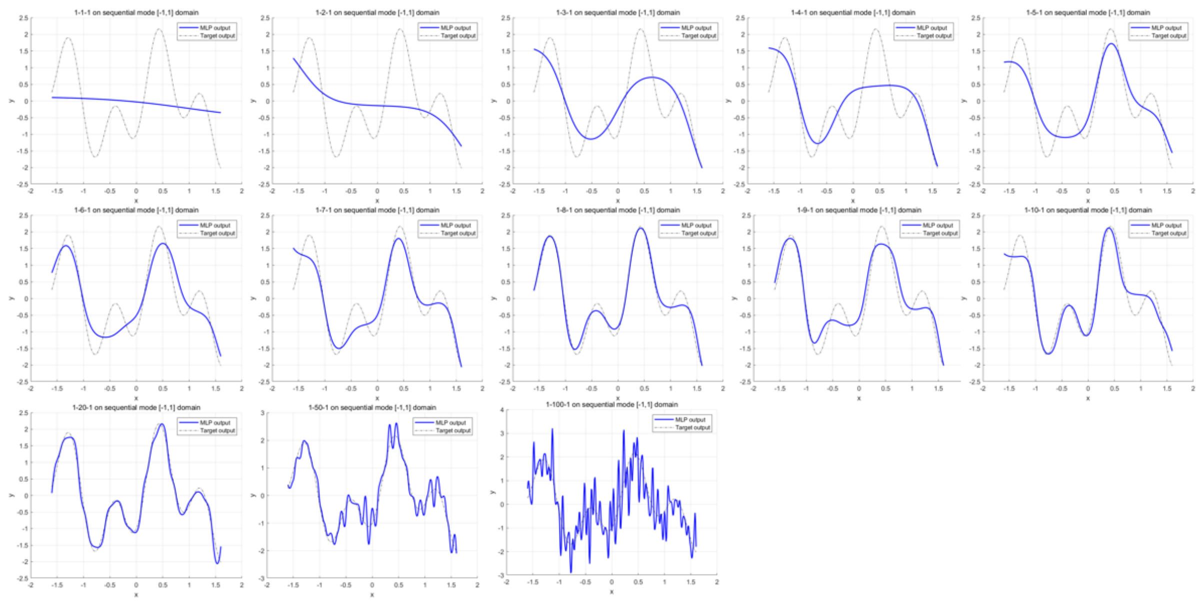


Figure 5. Regression plots with respect to the number of hidden neurons

According to the guideline given in the lecture note, the minimum number of hidden neurons for the given function is 8. As shown in the figure, the model fails to plot a desirable graph when the number of the neurons is 1 to 7. Another group of the hidden neurons (8, 9, 10 and 20) seems to predict the desired outputs fairly precisely. However, although (1-8-1) and (1-20-1) managed to compute the desired outputs with a low relative error, the other ones (1-9-1 and 1-10-1) computed less accurate outputs. Nevertheless, since they were plotted better than the first seven plots (1 to 7), it was concluded that they fall into the proper-fitting group. Finally, the over-fitting behavior could be observed in the last hidden neuron group (1-50-1 and 1-100-1). **Table 1** summarizes the results obtained to examine the fitting types.

Table 1. Fitting type summarization for the given hidden neuron groups

Fitting type	ANN structure
Under-Fitting	(1-1-1), (1-2-1), (1-3-1), (1-4-1), (1-5-1), (1-6-1), (1-7-1)
Proper-Fitting	(1-8-1), (1-9-1), (1-10-1), (1-20-1)
Over-Ftting	(1-50-1), (1-100-1)

* (1-8-1) is the minimum number of hidden neurons required to compute the desired outputs which is the same as the guideline given in the lecture note

a-2) Compute the outputs of the MLP when x=-3 and +3 and see if the MLP can make reasonable predictions outside of the domain of the input limited by the training set.

Further experiment was conducted to examine if the constructed model would be able to compute the outputs that had not been experimented/trained before. Hence, new test inputs (-3 to 3 with a uniform step length of 0.01) were fed into the model. **Fig. 6** demonstrates that the constructed ANN model is not capable of predicting target outputs (that are out of the domain) correctly at all if it has not been trained in advance.

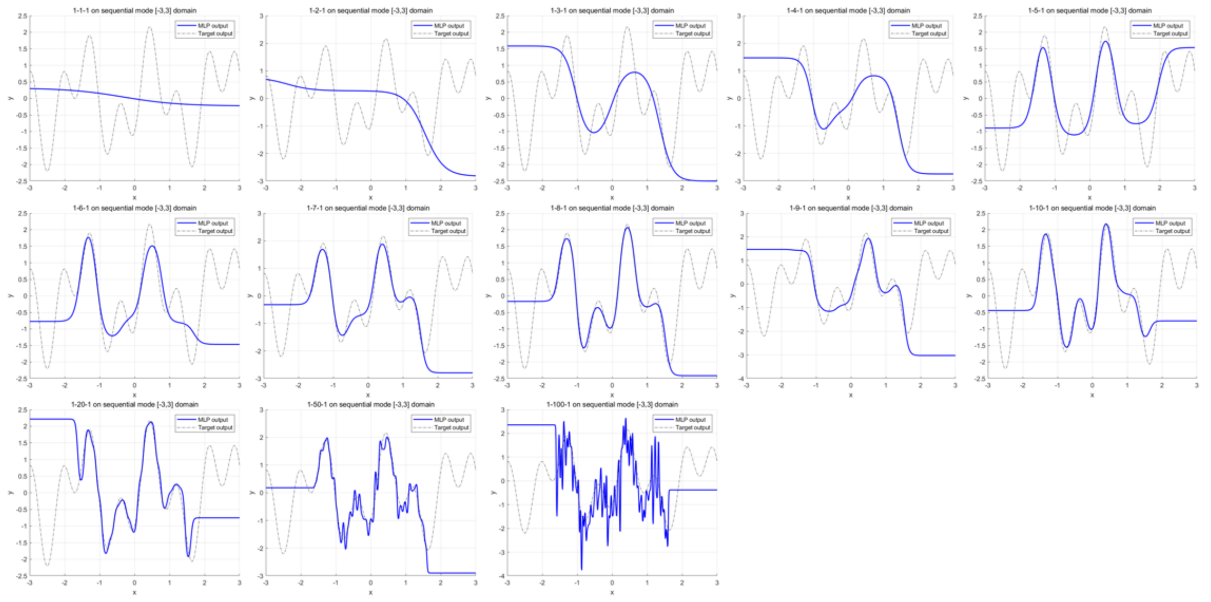


Figure 5. Regression plots with respect to the number of hidden neurons for domain [-3,3]

As can be observed, since the model was trained with the input data ranging from -1.6 to 1.6, it completely failed to make reasonable predictions outside of the domain of the input limited by the training set.

b) Use the batch mode with trainlm algorithm to repeat the above procedure

Fig. 6 below shows the plots generated by the trainlm algorithm. The red line in each graph indicates the predicted outputs in the domain range [-1, 1]. As can be observed, the predicted outputs are much more accurate than those previously obtained in **Q2-a**. As the backpropagation function of the algorithm uses Jacobian derivatives, the time taken to complete the same task as the one in **Q2-a** was much faster. However, the trainlm algorithm could not make reasonable predictions outside of the domain of the input limited by the training set as well.

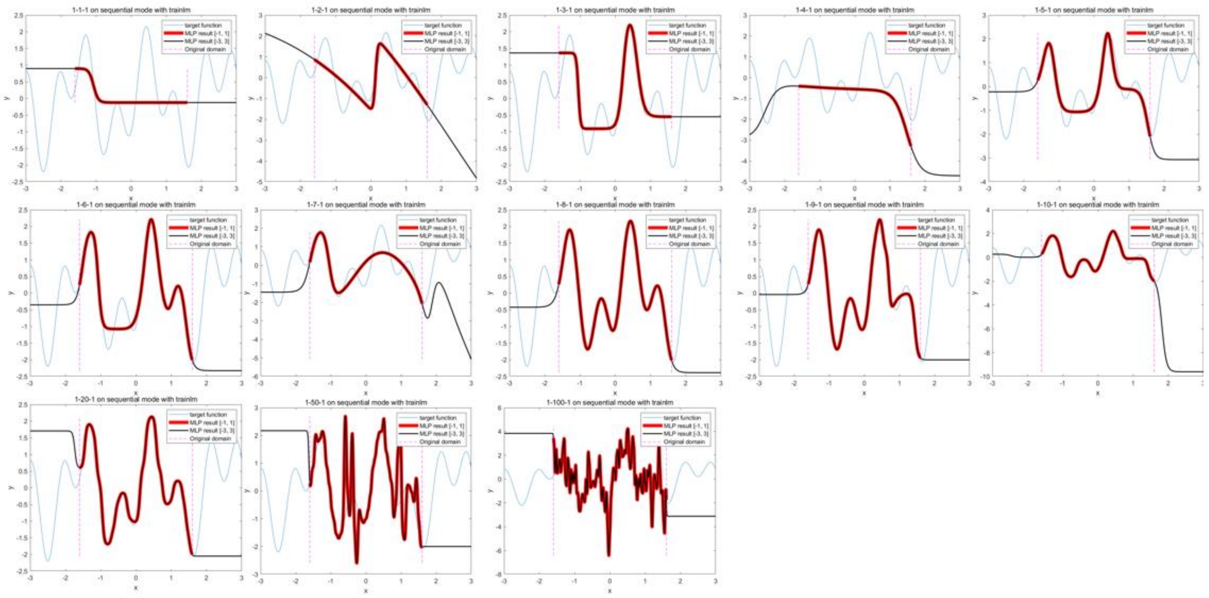


Figure 6. Regression plots obtained by the trainlm algorithm

Table 2 illustrates the results obtained by the trainlm algorithm to examine the fitting types. As shown in the table, the results are the same as those in **Table 1**. However, one thing to note is that the new ANN structures (1-10-1 and 1-20-1) displayed a much better prediction accuracy as compared to the previous old ANN structures in Q2-a. Further, the minimal number of hidden neurons for the function was observed to be (1-8-1), which followed the guideline given in the lecture note.

Table 2. Fitting type summarization for the given hidden neuron groups generated by trainlm

Fitting type	ANN structure
Under-Fitting	(1-1-1), (1-2-1), (1-3-1), (1-4-1), (1-5-1), (1-6-1), (1-7-1)
Proper-Fitting	(1-8-1), (1-9-1), (1-10-1), (1-20-1)
Over-Ftting	(1-50-1), (1-100-1)

c) Use the batch mode with trainbr algorithm to repeat the above procedure

In terms of the prediction performance of the inputs outside of the domain [-1, 1], the trainbr algorithm also failed to predict them as can be seen in **Fig. 7**. However, as for the prediction accuracy in the input domain [-1, 1], the algorithm was able to predict accurately regardless of the number of hidden neurons starting from 8 hidden neurons. Hence, no over-fitting behavior was observed in (1-50-1) and (1-100-1) ANN structures. **Table 3** summarizes the results obtained by the trainbr algorithm to examine the fitting types.

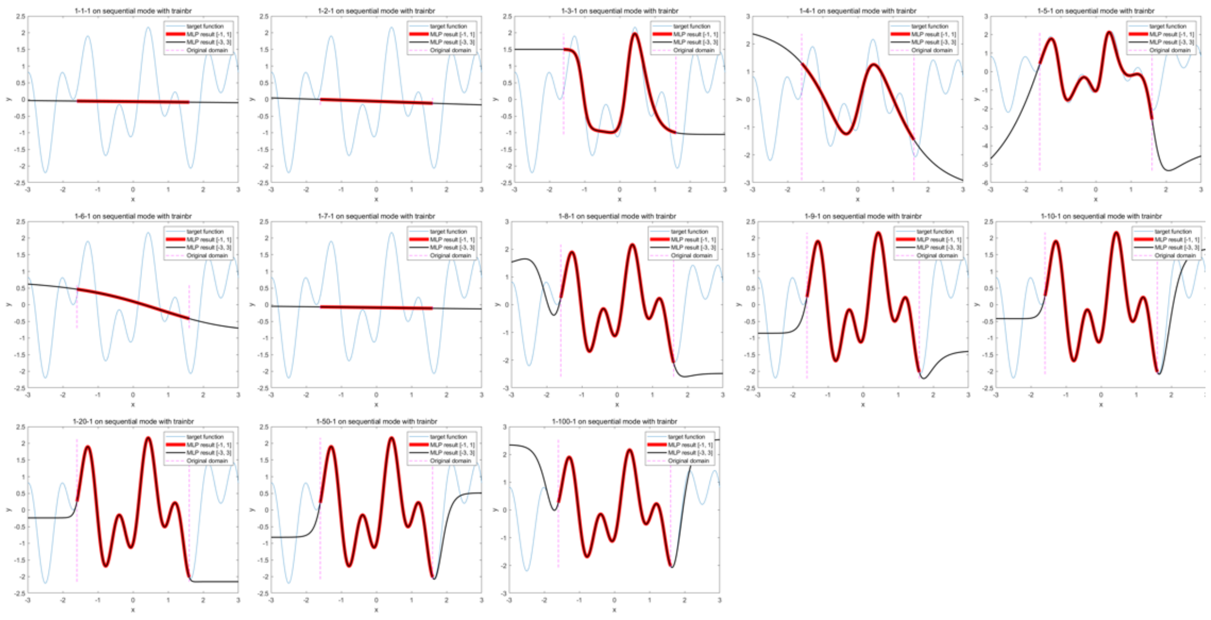


Figure 6. Regression plots obtained by the trainbr algorithm

Table 3. Fitting type summarization for the given hidden neuron groups generated by trainbr

Fitting type	ANN structure
Under-Fitting	(1-1-1), (1-2-1), (1-3-1), (1-4-1), (1-5-1), (1-6-1), (1-7-1)
Proper-Fitting	(1-8-1), (1-9-1), (1-10-1), (1-20-1), (1-50-1), (1-100-1)
Over-Ftting	-

Question 3

a) Apply Rosenblatt's perceptron (single layer perceptron) to the dataset of your assigned group. After the training procedure, calculate the *classification accuracy* for *both* the training set and validation set, and evaluate the performance of the network.

The built-in perceptron function in Matlab was used to implement *Rosenblatt's perceptron (single layer perceptron)* to investigate the change in the classification accuracy for both the training set and validation set with respect to the number of different iterations. In total, 10 different iterations from 10 to 100 with an interval of 10 were initially tested to examine the change in accuracy depending on iterations. **Fig. 7 (a)** shows how the training and validation accuracy values change depending on the number of iterations. As can be observed in the figure, accuracy values of both sets remain between 50% and 54% throughout all the iterations. However, as the trend was observed where the classification accuracy seemed to improve according to the increasing iterations, more iteration numbers were attempted. Hence, much higher iteration numbers (500, 1000, 1500, 2000) were fed into the code to examine if there would be a dramatic change or improvement for the accuracy.

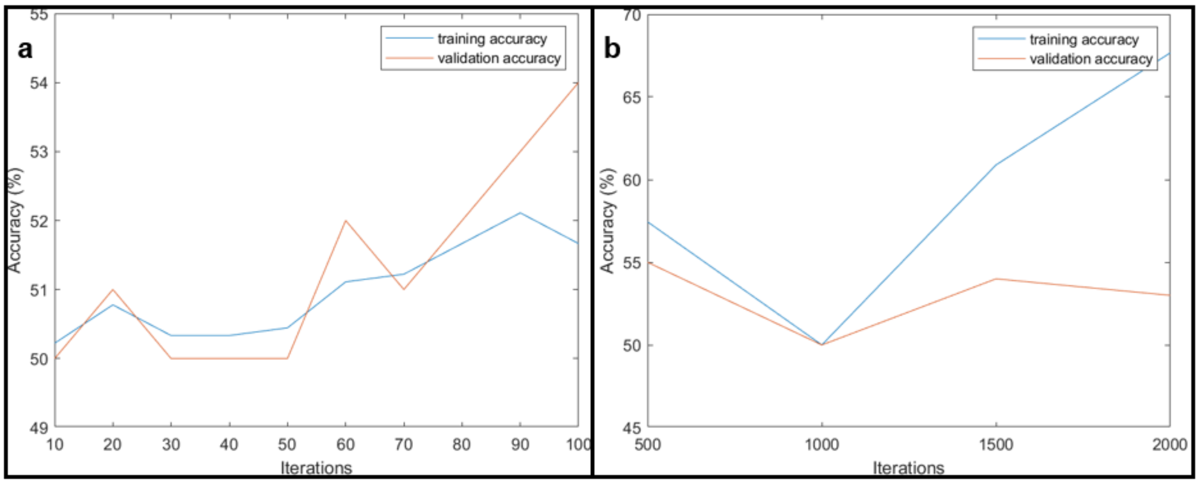


Figure 7. Accuracy change for training and validation sets with respect to iterations

As can be seen in **Fig. 7 (b)**, the training accuracy generally increased as more iterations were attempted. However, the validation accurate did not improve as much. **Table 4** summarizes the accuracy results with respect to the iteration numbers. The highest training accuracy obtained was 67.67% at 2000 iterations, and the highest validation accuracy was 55 at 500 iterations. Even though the accuracy difference between the training and validation sets is not too big until 1500 iterations, at 2000 iterations, the difference reaches 14.67% (67.67% -53.00%), which is relatively big. As the increase in training accuracy does not help increase the validation accuracy significantly, it is not desirable to select too high iteration numbers.

Table 4. Accuracy results for the training and validation sets

No. of iterations	Training accuracy (%)	Validation accuracy (%)
10	50.22	50.00
20	50.78	51.00
30	50.33	50.00
40	50.33	50.00
50	50.44	50.00
60	51.11	52.00
70	51.22	51.00
80	51.67	52.00
90	52.11	53.00

100	51.67	54.00
500	57.44	55.00
1000	50.00	50.00
1500	60.89	54.00
2000	67.67	53.00

b) The global mean and variance of a dataset may influence the stability of training and the final performance of the model obtained. You can try to calculate the global mean and variance of the whole dataset, then *subtract* the mean value from each image and *divide* each image by the variance. *Compare* the result with that in a) and *explain* it.

As instructed in the question, each image was subtracted by the mean value and divided by the variance, thereby normalizing all the data in the range [-1, 1]. The new dataset containing different values, hence, was created. **Fig. 8** shows accuracy change with respect to iterations after considering the global mean and variance of the dataset.

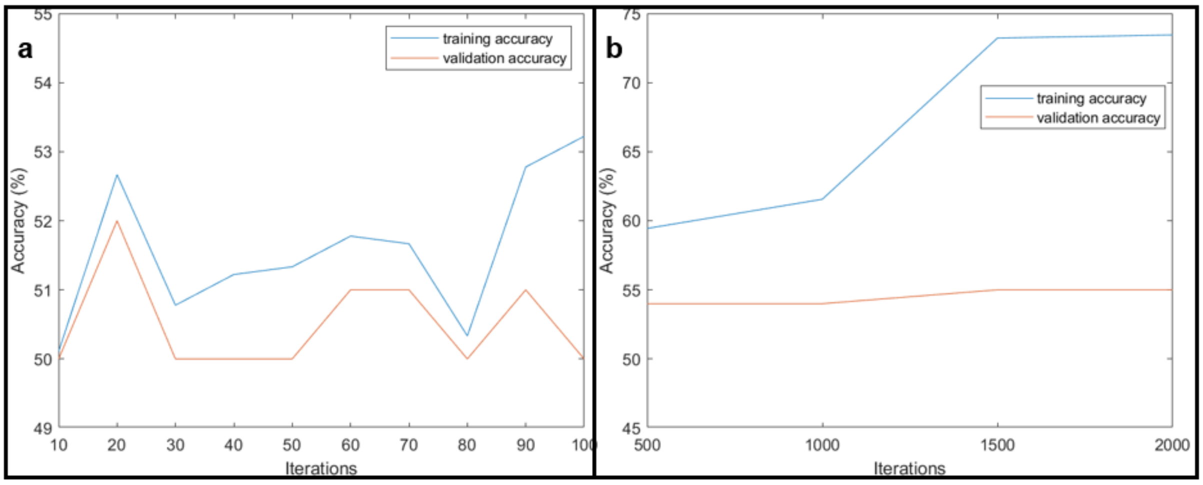


Figure 8. Accuracy change for training and validation sets with respect to iterations by new approach

Table 5 compares the accuracy results between **Q 3-a** and **Q 3-b**. In the table, it can be found that the accuracy does not change significantly until 500 iterations. However, starting from 1000 iterations, the latter approach produces better accuracy. Hence, it can be concluded that the latter approach helps improve the training accuracy, although the validation accuracy does not improve.

Table 5. Accuracy comparison between **Q 3-a** and **Q 3-b**

No. of iterations	Question 3-b		Question 3-a	
	Training accuracy (%)	Validation accuracy (%)	Training accuracy (%)	Validation accuracy (%)
10	50.22	50.00	50.11	50.00
20	50.78	51.00	52.67	52.00
30	50.33	50.00	50.78	50.00
40	50.33	50.00	51.22	50.00
50	50.44	50.00	51.33	50.00
60	51.11	52.00	51.78	51.00
70	51.22	51.00	51.67	51.00
80	51.67	52.00	50.33	50.00
90	52.11	53.00	52.78	51.00
100	51.67	54.00	53.22	50.00
500	57.44	55.00	59.44	54.00

1000	50.00	50.00	61.56	54.00
1500	60.89	54.00	73.22	55.00
2000	67.67	53.00	73.44	55.00

c) Apply MLP to the dataset of your assigned group using *batch mode* training. After the training procedure, calculate the classification accuracy for both the training set and validation set, and evaluate the performance of the network.

For the batch mode training, 20 different hidden neurons from 10 to 200 with an interval of 10 were tested. The other parameters were set constant as follows: Learning rate: 0.01, training function: traingdx, maximum number of iterations: 1000, and the error goal: 1e-8. The algorithm was designed to stop when the training accuracy reached 100% specifying the number of iterations taken. **Fig. 9** shows the change in accuracy for the validation set with respect to the number of hidden neurons. As can be observed in the figure, the accuracy increases significantly from approximately 66% to 78% (that is equivalent of 10 to 60 of hidden neurons). However, after that, the accuracy starts to fluctuate and is not able to raise anymore. Hence, of the hidden neurons tested, 60 would be the minimum and best number of the neurons that compute good results.

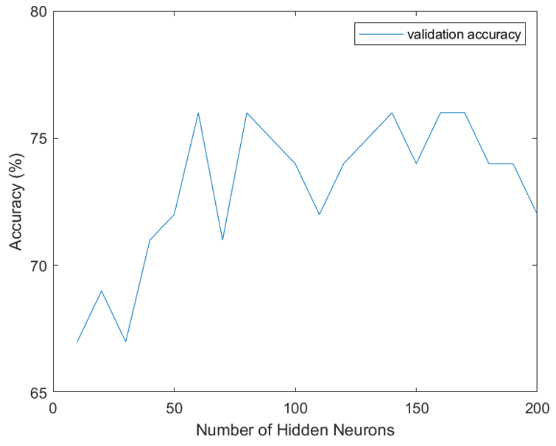


Figure 9. The accuracy change for the validation set with respect to hidden neurons

Table 6 summarizes all the results obtained depending on the number of hidden neurons. From the table, it can be observed that the training accuracy reaches 100% within 382 iterations. Lower iterations could mean that the gradient goal, set in advance, was achieved earlier. The highest accuracy value obtained was 78% with 60 hidden neurons at 260 iterations.

Table 6. Summarization of the results obtained by the batch mode (MLP)

No. of hidden neurons	Training accuracy (%)	Validation accuracy (%)	Iterations
10	100	66.00	382
20	100	70.00	263
30	100	72.00	262
40	100	76.00	262
50	100	76.00	261
60	100	78.00	260
70	100	70.00	260
80	100	75.00	259
90	100	73.00	259
100	100	71.00	259
110	100	72.00	259
120	100	74.00	258
130	100	73.00	259

140	100	77.00	258
150	100	75.00	258
160	100	75.00	262
170	100	77.00	262
180	100	76.00	258
190	100	77.00	267
200	100	73.00	267

d) Please determine whether your trained MLP in c) is overfitting. If so, please specify when (i.e. after which training epoch) it becomes overfitting. Try weights regularization and observe if it helps. (you may set the regularization strength by 'performParam.regularization')

Only one number of hidden neurons (n=60) was chosen to investigate the overfitting behavior. The same values of the parameters were used as in the previous experiment. **Fig. 10** shows the plots in which the overfitting occurs after certain training epochs. **Fig. 10 (a)** is the graph without the regularization setup. Without the help of weights regularization, the overfitting occurs after 93 epochs. Because, as can be seen in the figure, the training cross-entropy keeps decreasing significantly whereas the validation value increases little by little, thereby resulting in a greater accuracy difference. As this indicates that the accuracy for the training and validation sets begins to diverge, it can be considered as overfitting. **Fig. 10 (c-f)** shows how the regularization helps delay the overfitting phenomenon. 0.1, 0.2, 0.3, 0.4 and 0.5 were used as the regularization value for each plot (c-f), respectively. For the graph (b and e), at epoch 84 and 91, the best classification accuracy was achieved. Although the divergence gap between the training and validation sets was smaller than of the one without the regularization, the overfitting was not delayed. For the other graphs (c, d and f), the best accuracy was achieved at 104, 96 and 111 epochs, respectively. Hence, this implies that the overfitting was delayed when the regularization values were 0.2, 0.3 and 0.5. The conclusion drawn from the results obtained is that the regularization can delay the overfitting if a proper value is used.

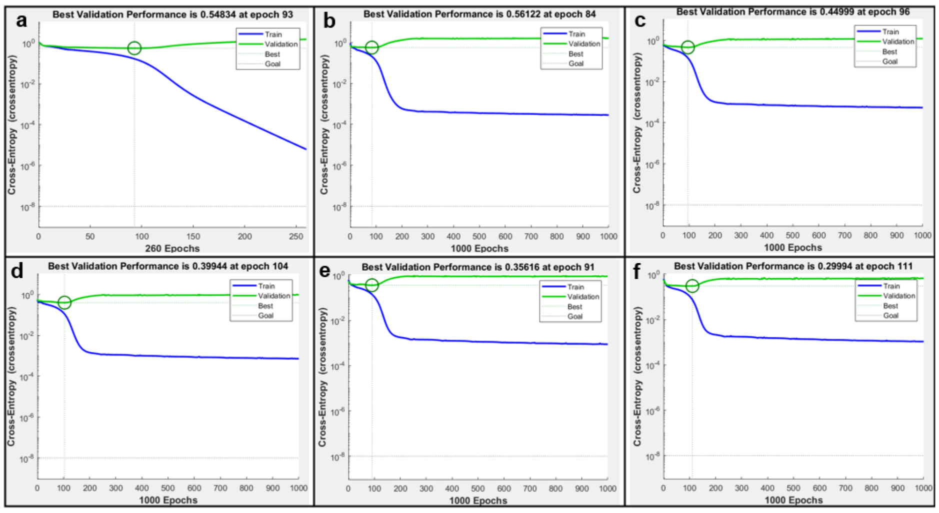


Figure 10. Cross-entropy plots with respect to epochs with/without weights regularization (a: without it, b: 0.1, c: 0.2, d: 0.3, e: 0.4, f: 0.5)

e) Apply MLP to the dataset of your assigned group using *sequential mode training*. After the training procedure, calculate the classification accuracy for both training set and validation set, and evaluate the performance of the network. Compare the result to part c), and make your recommendation on the two approaches.

The sequential mode training method was used with the max epoch of 260. The reason for choosing 260 as the max epoch was because, in **Q3-c**, epochs around 260 were obtained for the most of hidden neuron numbers. Hence, by inputting the same value, it was believed that a distinct comparison could be made later.

Due to the extremely slow computation process unlike the batch mode, only 3 different hidden neurons were tested: 60, 70 and 80. **Table 7** shows the accuracy comparison between sequential and batch modes. As can be seen in the table, except when the number of hidden neurons is 70, the batch mode computes better accuracy values. Hence, batch mode would be much more preferred because the computation process is unexceptionally faster, which gave us adequate time to test other hyperparameters/functions to optimize the MLP model.

Table 7. Accuracy comparison between sequential mode and batch mode

No. of hidden neurons	Sequential Mode		Batch Mode	
	Training accuracy (%)	Validation accuracy (%)	Training accuracy (%)	Validation accuracy (%)
60	100	73.00	100	78.00
70	100	76.00	100	70.00
80	100	73.00	100	75.00

f) Try to propose a scheme that you believe could help to improve the performance of your MLP and please explain the reason briefly.

Based on the results obtain thus far, the following conclusions were made to improve the performance of the MLP model.

1) Single perceptron or Multi-layer perceptron? ***Multi-layer perceptron***

As relatively low accuracy values (not higher than 60%) were obtained regardless of the iteration numbers in the single perceptron, whereas the MLP was able to compute as high as 78%, the MLP would be in much more favor. This could be due to the existence of the hidden layer and hidden neurons, which realizes its advanced parallelization ability that can solve extremely complicated non-linear relationships in high dimensions.

2) Batch mode or Sequential mode? ***Batch mode***

In terms of the processing time, the batch mode outperformed the sequential mode in this experiment. Even though the sequential mode is known to result better solutions as it can jump out of the local minima and move into a deeper (preferred) local minimum due to the noisy estimate of the true gradient that it computes, it did not compute significantly better prediction values as compared to the batch mode in this experiment (but, rather, their values were close). This might be because the amount of data given was not big enough for the sequence learning. Hence, it was determined that the batch mode would be a better option to choose as 1) it processed a lot faster, and 2) outputs as good as those obtained in the sequential mode were obtained in the experiment.

3) The ratio of the training data and validation data

An appropriate amount of the dataset needs to be used to train the MLP model. For example, if we have the dataset of 300 samples, and only use 10% of the dataset (30 samples) for the training and the remaining (270 samples) for the validation, it would not compute good results for the validation as the model has too little information to train itself. Hence, determining a good ratio is a critical factor in achieving the improvement of the MLP performance.

4) Hyperparameter optimization (such as learning rate, number of hidden neurons/layers, etc.)

As neural network models can also produce significantly different results depending on the hyperparameter values due to their sensitivity, it would be desirable to find the best combination after many trials. Perhaps, the optimization can be achieved by performing the k-fold cross validation method.

5) Possible solution: Singular Value Decomposition (SVD)

As SVD can be used to eliminate redundant data while not damaging original images, the image compression can be achieved by properly choosing the effective rank. Hence, faster processing and computation can be realized.

5) Normalization

Normalization can be used to inspect and delete all duplicate data by effectively grouping data redundancies together.