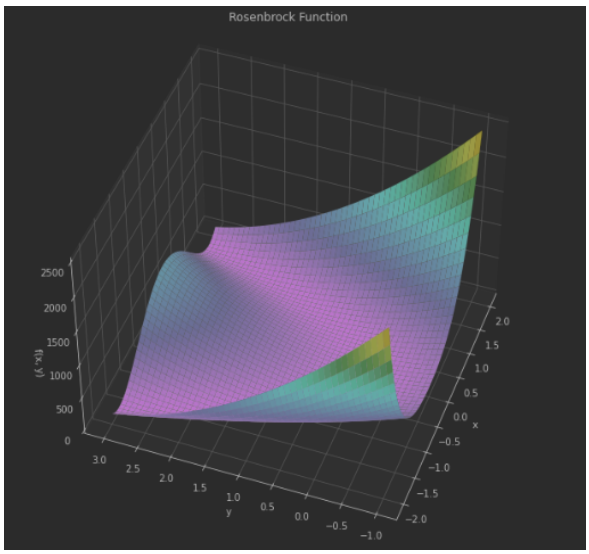
HW2

Q1. Rosenbrock’s Valley Problem

The Rosenbrock's Valley function is defined as:

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

The figure below illustrates the Rosenbrock's Valley function in three-dimensional space.



a. **Steepest (Gradient) descent method**

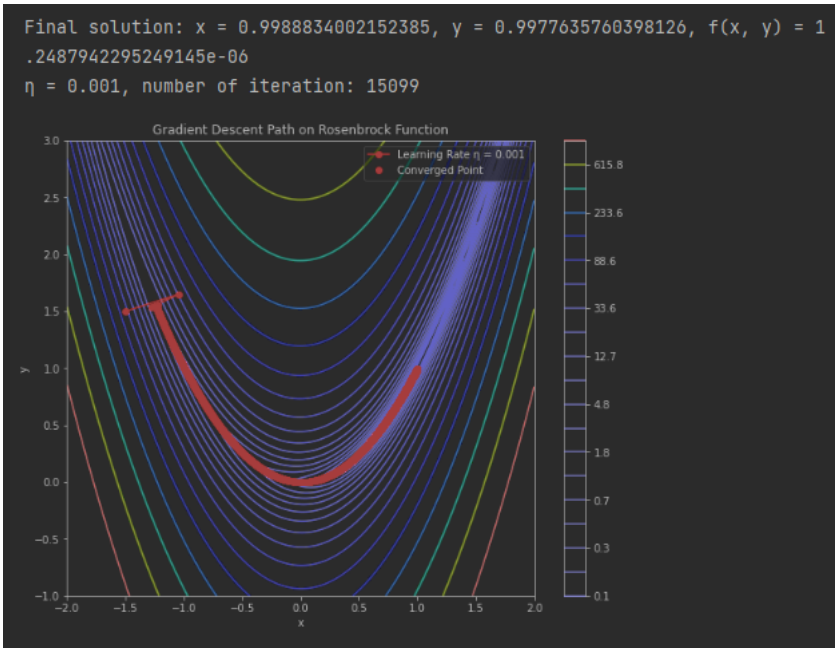
Given:Iteration rule:where:learning rate η, initial values (x0,y0)

$$x_{n+1} = x_n - \eta \cdot \frac{\partial f}{\partial x}(x_n, y_n)$$

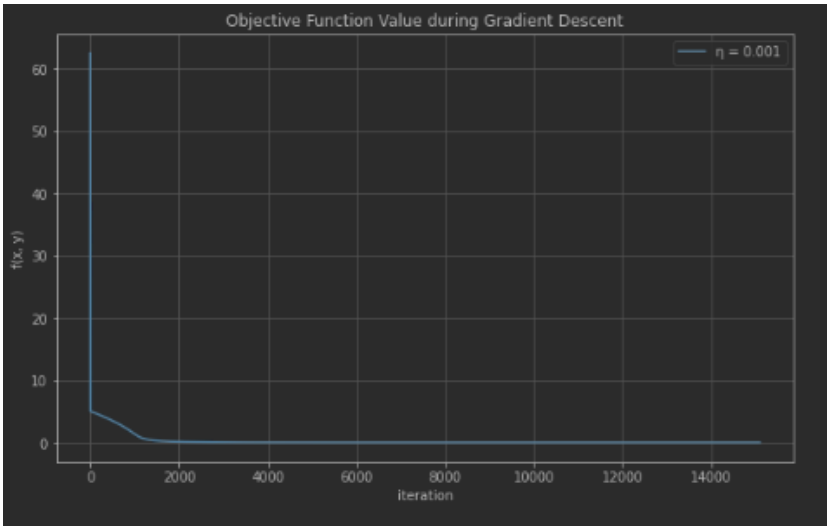
$$y_{n+1} = y_n - \eta \cdot \frac{\partial f}{\partial y}(x_n, y_n)$$

$$\frac{\partial f}{\partial x}(x_n, y_n) = -2(1 - x) - 400x(y - x2)$$

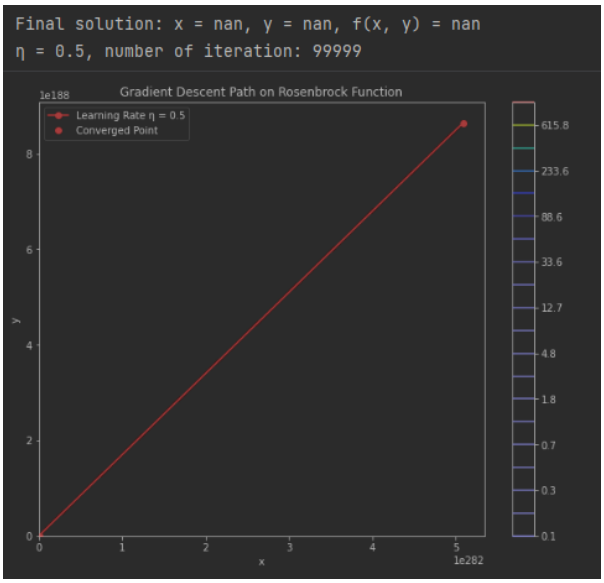
$$\frac{\partial f}{\partial y}(x_n, y_n) = 200(y - x2)$$

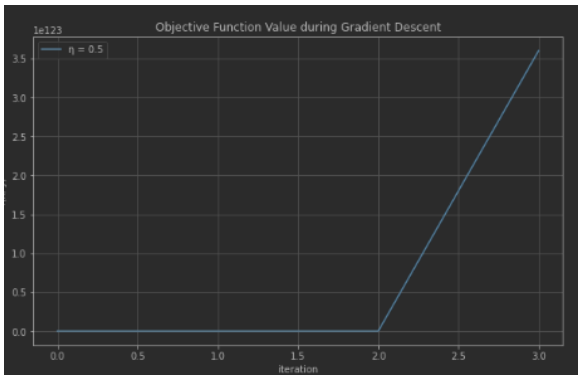


The graph above shows the contour lines of the Rosenbrock's Valley function and the trajectory of iterations. The global minimum of this function is located at (1, 1), where f(x, y) = 0. Here we can get the final solution *x = 0.9988834002152385, y = 0.9977635760398126, f(x, y) = 1.2487942295249145e-06,* with $\eta = 0.001$.



This plot shows the value of f versus the number of iterations, and we can see that by the 1600th iteration, good results have been achieved





However, when $\eta = 0.5$, we can not find the solution. The model can not coverge in this case. A proper η is important to ensure smooth iteration.

b. Newton's method

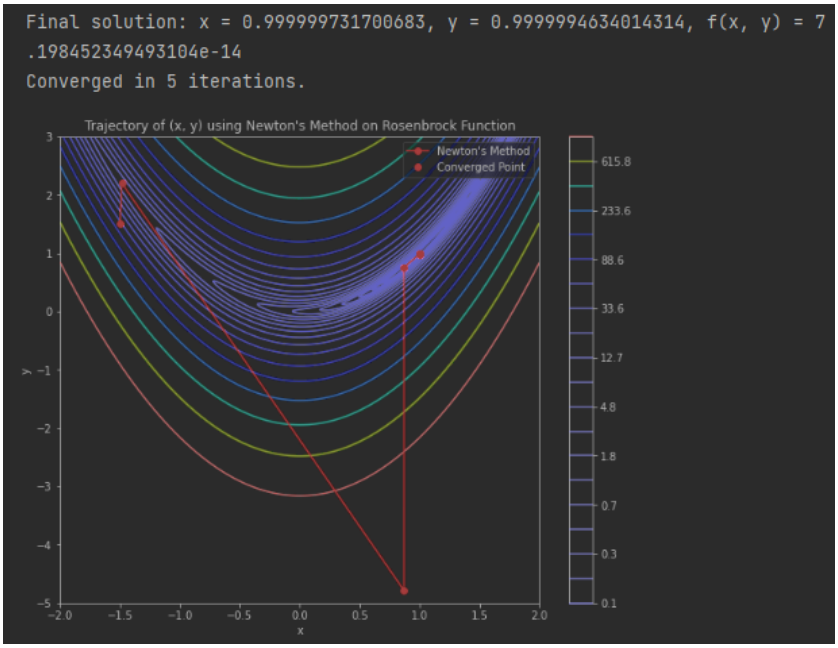
$$\begin{cases} x_{n+1} = x_n - H^{-1}(x_n, y_n) \cdot g(x_n, y_n)[0] \\ y_{n+1} = y_n - H^{-1}(x_n, y_n) \cdot g(x_n, y_n)[1] \end{cases}$$

$$g(x, y) = \nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

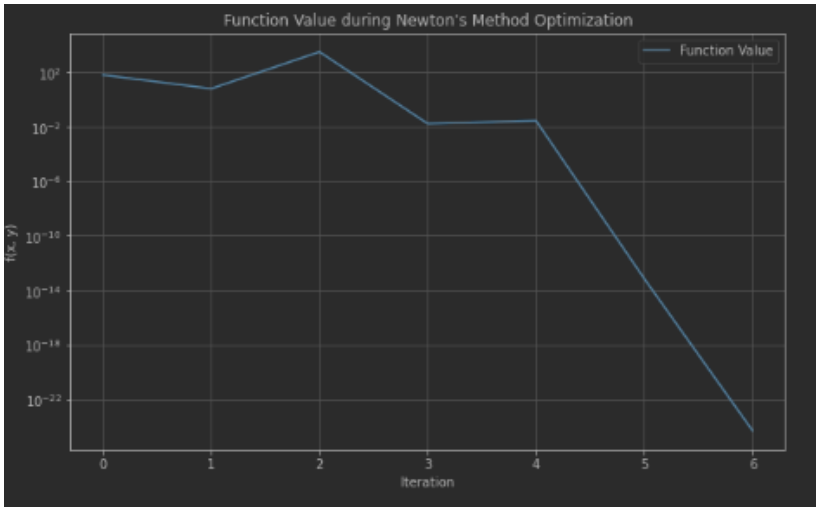
where

$$H(x, y) = \text{Hessian } f(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

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



Here we can get the final solution *x = 0.999999731700683, y = 0.9999994634014314, f(x, y) = 7.198452349493104e-14.*



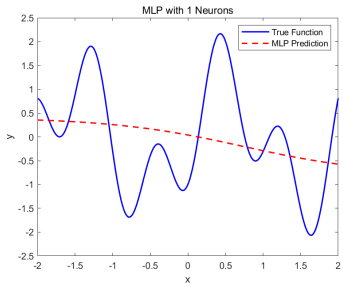
It converges in 5 iterations.

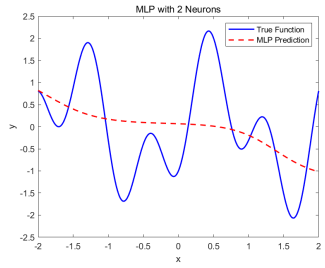
Q2. Function Approximation

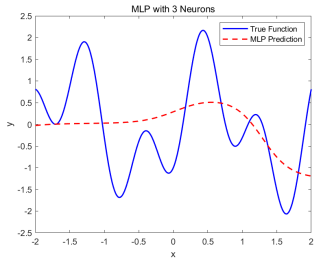
a. **Sequential mode**

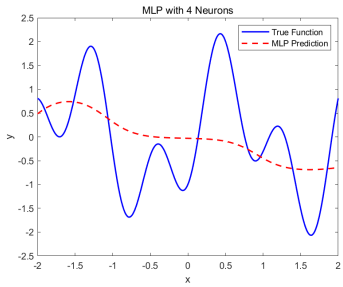
Number of Neurons	Training MSE	Test MSE	y(x=-3)	y(x=3)	Fittness

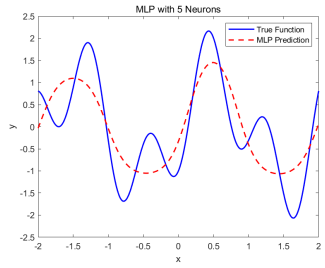
1	1.1045	1.1038	0.3882	-0.7192	underfitting
2	1.0123	1.0042	1.2928	-1.1622	underfitting
3	0.9752	0.9597	-0.3469	-1.2486	underfitting
4	0.9761	0.9736	-0.1439	-0.5833	underfitting
5	0.4393	0.4366	-1.1495	1.2563	underfitting
6	0.5153	0.5012	-0.6612	-0.3576	underfitting
7	0.3653	0.3587	0.8453	0.7457	underfitting
8	0.2749	0.2571	0.8150	-1.2233	underfitting
9	0.2380	0.2217	1.0555	-0.5951	underfitting
10	0.1816	0.1608	0.1028	-1.1545	underfitting
20	0.0264	0.0232	-0.9033	2.0509	fitting well
50	0.2011	0.1915	-1.2645	-1.9114	overfitting
100	0.5614	0.6252	-2.6926	0.9771	overfitting

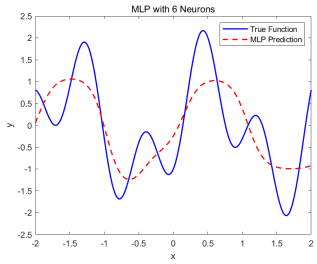


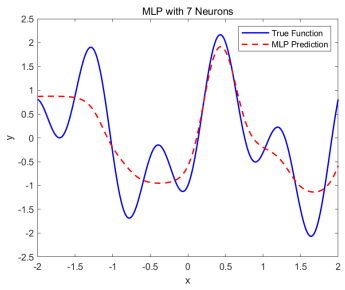


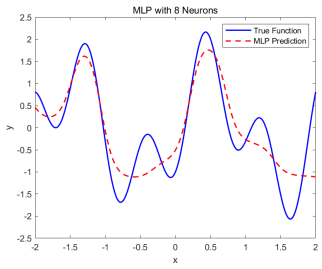


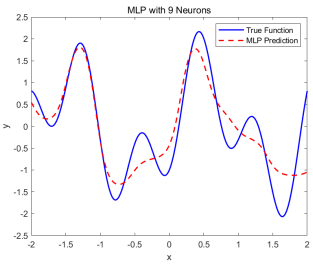


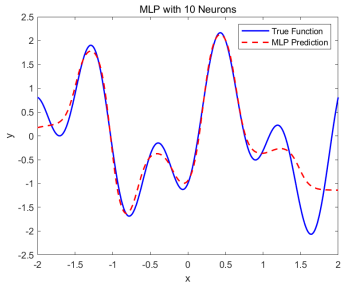


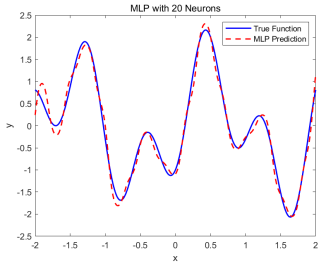


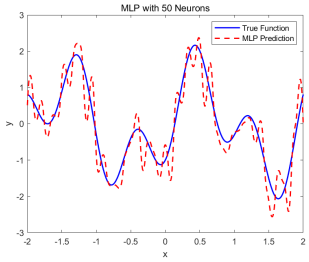


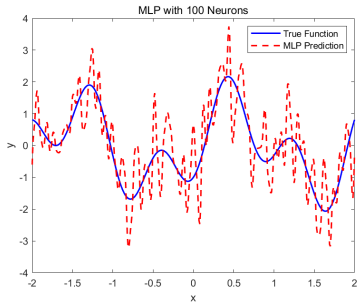












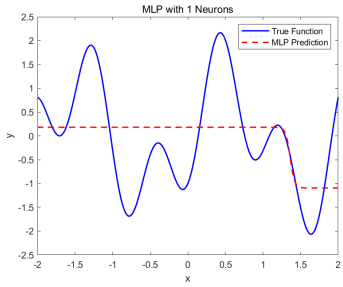
In this case, when the number of neurons is 20, the model get the best performance. Too many or too few numbers of neurons can lead to underfitting or overfitting. At the same time, we can know that the sequential mode may be sensitive to the number of neurons, so it needs to conduct multiple times to find the optimal number of neurons.

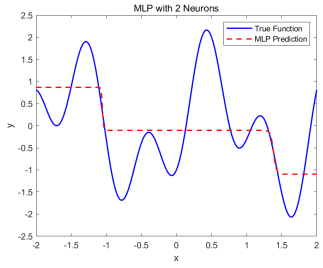
We can also notice that there is a large gap between the fitting results and the actual results on -3 and 3, which indicates that the model cannot predict accurately on the intervals without training.

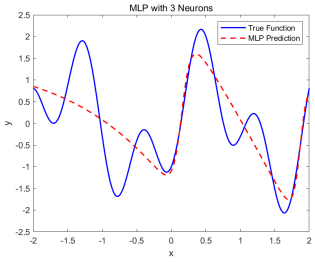
b. **Batch mode with trainlm**

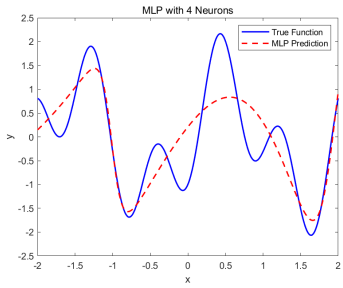
Number of Neurons	Training MSE	Test MSE	y(x=-3)	y(x=3)	Fittness
1	0.9868	0.9750	0.1807	-1.0932	underfitting
2	0.8293	0.8184	0.8691	-1.0949	underfitting

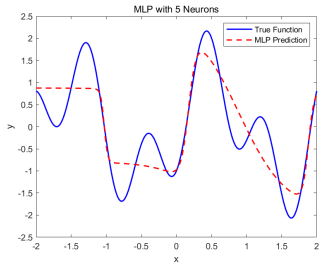
3	0.4319	0.4361	1.1954	-1.2098	underfitting
4	0.3396	0.3405	-0.9141	1.5151	underfitting
5	0.2739	0.2765	0.8764	0.6986	underfitting
6	0.1035	0.1045	3.8159	0.5056	underfitting
7	0.2708	0.2734	0.6179	0.9794	underfitting
8	0.0552	0.0558	0.8934	6.0706	underfitting
9	0.0009	0.0008	1.0938	8.0910	fitting well
10	0.0000	0.0000	0.9446	2.2099	fitting well
20	0.0000	0.0000	-1.0901	1.7766	fitting well
50	0.0000	0.0012	0.8693	2.1289	overfitting
100	0.0000	0.1171	-0.3950	0.5382	overfitting

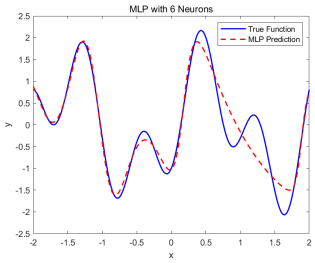


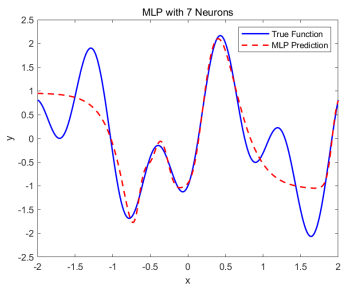


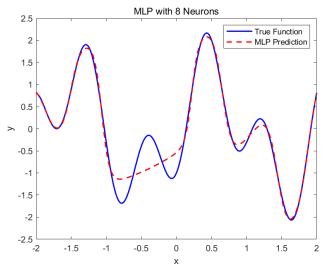


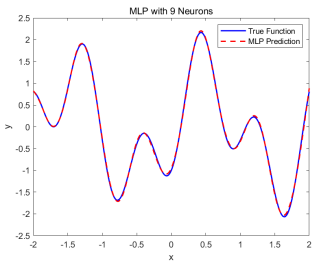


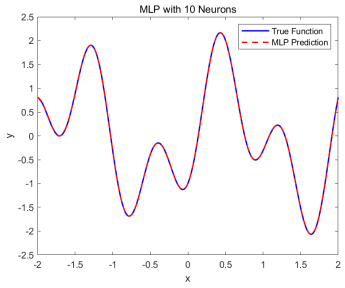


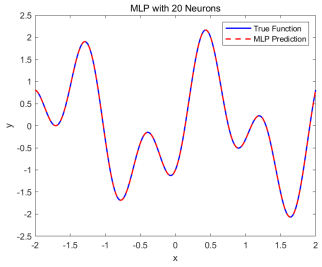


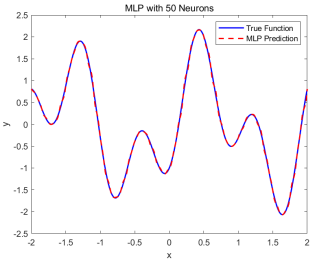


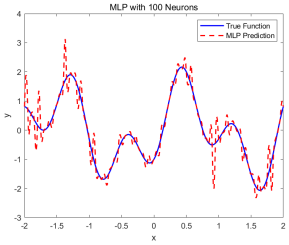












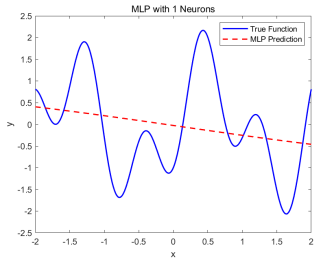
Compared with sequential mode, batch mode with trainlm achieves a better and robuster result. When neuronless number = [9, 10, 20], it has excellent results. This means batch mode is less sensitive to neurons number as sequential mode. Aslo it needs less neurons to achieve a good result.

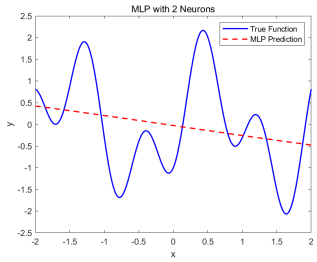
However, for predicting on -3, 3, it still has a bad result.

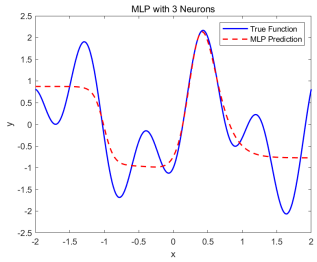
c. Batch mode with trainbr

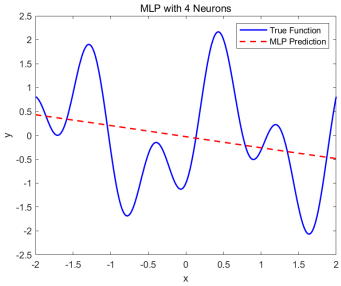
Number of Neurons	Training MSE	Test MSE	y(x=-3)	y(x=3)	Fittness
1	1.1002	1.1012	0.5749	-0.63491.0982	underfitting
2	1.0982	1.0991	0.6205	-0.6751	underfitting
3	0.3710	0.3609	0.8733	-0.7727	underfitting
4	1.0971	1.0979	0.6468	-0.7001	underfitting
5	0.1120	0.1130	5.4030	2.4590	underfitting
6	1.0968	1.0975	0.6561	-0.7094	underfitting
7	0.0011	0.0011	-0.0129	5.2157	fitting well
8	0.0000	0.0000	2.0135	4.7494	fitting well
9	0.0000	0.0000	-2.6693	1.4544	fitting well
10	0.0000	0.0000	-3.0617	2.9554	fitting well
20	0.0000	0.0000	0.8634	1.8684	fitting well
50	0.0000	0.0000	-1.6962	1.3154	fitting well

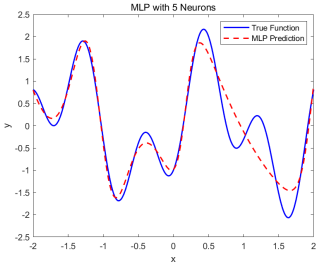
100	0.0000	0.0000	-5.3380	9.0127	fitting well

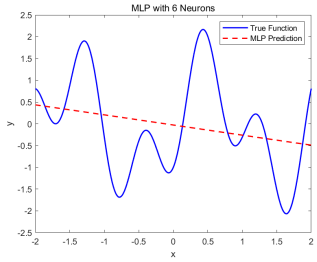


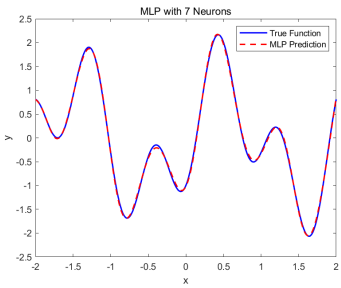


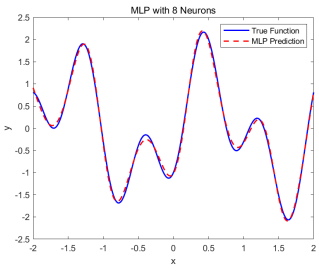


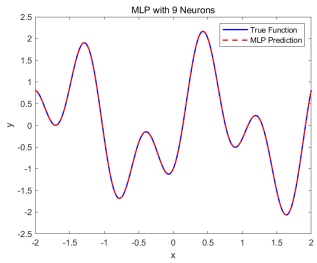


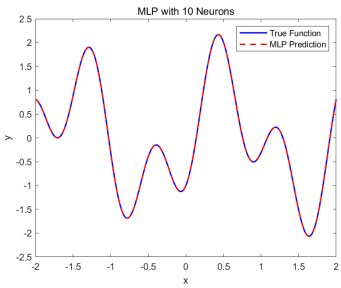


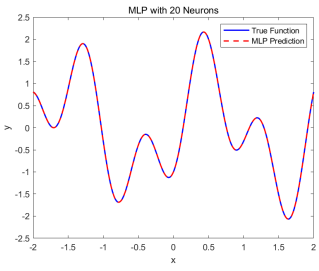


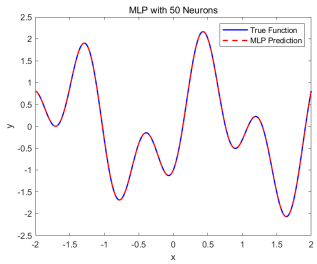


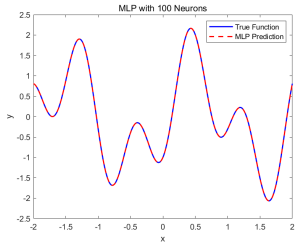












trainbr algorithm effectively solve the problem of overfitting and can use fewer neurons to achieve good results.

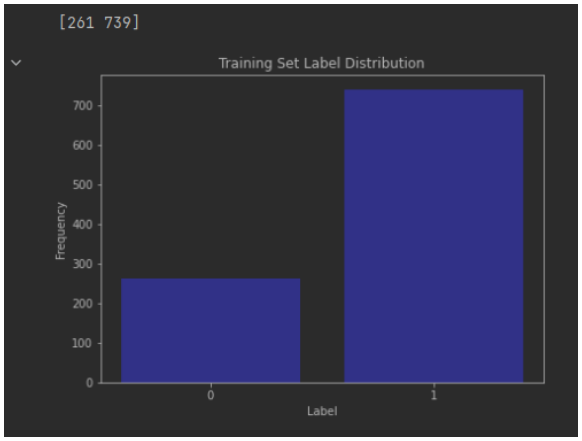
In summary,

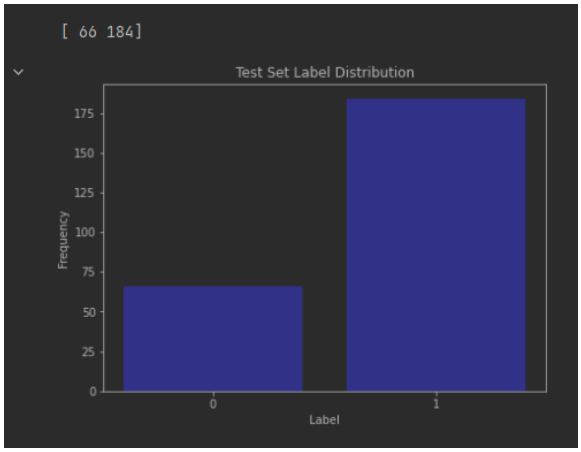
- Sequential mode with traingd showed underfitting for low neuron counts and overfitting for high counts, with optimal performance around 20 neurons.
- Batch mode with trainlm achieved better and more robust results, performing well with 9-20 neurons and showing less sensitivity to neuron count.
- Batch mode with trainbr effectively solved overfitting issues, achieving good results with fewer neurons and maintaining performance even with high neuron counts.
- Overall, batch mode algorithms (trainlm and trainbr) outperformed sequential mode, with trainbr showing the best ability to prevent overfitting and achieve consistent results across various neuron counts.

Q3. Facial Attribute Recognition

I am in Group 1, mod(87, 3)+1

a. **Label distribution of traning and test set**





b. **Rosenblatt's perceptron**

	Train	Test
Accuracy	100.00%	86.00%

c. **Naively downsample and PCA**

Naively downsample

In this experiment, we keep pixels of 25*25, 50*50, 75*75, where the full size is 101*101.

	25*25	50*50	75*75
Train	91.50%	100.00%	100.00%
Test	85.60%	85.20%	84.80%

In naive downsampling, we can find that the more information we keep the higher accuracy it will be in training set. However, things just be the opposite in test set. This may be because too much training information leads to a decrease in the generalization ability of the model, and at the same time, the model does not reach a sufficient scale such as 101*101, so the effect is not good in the testing process.

PCA

In this experiment, we keep 50, 100, 150, 200 components, where the total components is 250.

	50	100	150	200
Train	68.70%	74.90%	79.80%	82.80%
Test	55.20%	49.60%	51.20%	50.80%

In training set, it's same as naive downsampling, accuracy increasing with more information kept. However, in test set, the trend is not so obvious.

d. **MLP with batch mode training**

```
# Define the MLP model
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(64, 16)
        self.relu = nn.ReLU()
        self.fc3 = nn.Linear(16, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.relu(out)
        out = self.fc3(out)
        out = self.sigmoid(out)
        return out
```

Structure of MLP

In batch mode, the training data is divided into small batches (32 in this case), and the model processes one batch of data at a time. Batch mode has efficient parallelism which can speed up training process. However, this can cause large memory usage.

	Train	Test
Accuracy	86.30%	86.30%

e. **MLP with sequential mode training**

In sequential mode, data is processed one sample at a time. Parameters are updated after processing each sample, allowing the model to react quickly and cost less memory. However, it will take more time to train and can be influenced by noise data.

	Train	Test
Accuracy	87.70%	87.70%

Compared with batch mode training, sequential mode achieves better results. However, it takes longer time when we use sequential mode. This is because batch mode can process data in parallel, which is suitable for massive data.

f. **Necessity for aligning eyes at a certain location**

Images aligned by placing eyes at a certain location could be useful for training neural network. Because for some specific problems, it is important to identify the location of eyes, such as glass/non-glass classification.

For male/female and smile/non-smile classification, it is less necessary for eye alignment. Because gender classification has other crucial features. Ficial position plays a little role in this scenario. For smile classification, it is necessary to align location of mouth. To some extent, alignment of the eyes can help by reducing overall facial location, ensuring the mouth region. Glass detection is highly dependent on features around the eyes, such as the presence of frames or reflections.