Multi-Layer Perceptron for Non-Linear Regression

Bryan Lee
922649673

March 8, 2025

1 Introduction

This project focuses on implementing a Multi-Layer Perceptron (MLP) for non-linear regression, with the goal of predicting continuous values based on input data. Unlike classification models that assign discrete class labels, regression models generate numerical predictions that capture complex, non-linear relationships within the data. The objective is to train a basic MLP to learn a function that effectively maps x to y, identifying the underlying pattern in the dataset. Since this is a regression task, Softmax and Argmax activation functions, commonly used for classification, are unnecessary. Instead, the model outputs a single continuous value, representing the result of the learned non-linear function.

2 Data

The dataset used for this project consists of 1,000 data points, where x represents the independent (predictor) variable and y is the dependent (response) variable. Using a linear regression would yield a straight line-of-best fit; however, it fails to capture the relevant discrepancy and patterns in the data. To achieve our objective, an MLP model with hidden layers and non-linear activation functions can effectively capture the complex patterns and provide a non-linear fit to the data.

	x	y
0	3.40	9.30
1	0.36	8.92
2	7.84	6.84
3	5.61	9.06
4	1.33	15.46
...	...	...
995	5.17	8.35
996	8.37	7.07
997	7.93	6.82
998	2.89	10.29
999	5.90	8.69

1000 rows × 2 columns

Figure 1: The provided dataset.

3 MLP Class

The MLP model that fits the data the best consists of four connected layers: three hidden layers and one output layer. The input to the first layer is x, which is normalized and reshaped to a 2D tensor during pre-processing. The hidden layers use ReLU activation functions, while the output layer produces a single continuous value without activation. The best architecture (creating a smooth curve) for the MLP model is as follows:

- Hidden Layer: $1 \rightarrow 400$ neurons, ReLU activation
- Hidden Layer 2: 400 neurons $\rightarrow$ 250 neurons, ReLU activation
- Hidden Layer 3: 250 neurons $\rightarrow$ 80 neurons, ReLU activation
- Output Layer: 80 neurons $\rightarrow$ 1 neuron

4 Data Preprocessing

As the data is not centered around zero, it is normalized using z-score standardization to origin (0, 0). Normalization ensures that all features have similar magnitudes. The data normalization process is as follows:

- Compute the mean and standard deviation for both x and y.
- Normalize x and y using the z-score standardization formula:

$$x_{norm} = \frac{x - \mu_x}{\sigma_x}, \quad y_{norm} = \frac{y - \mu_y}{\sigma_y}$$

After training, the model's prediction will be denormalized for plotting via:

$$Y_{\text{denorm}} = f\left(\frac{X - \mu_X}{\sigma_X}\right) \cdot \sigma_Y + \mu_Y$$

5 Training Process

The MLP is trained using the Mean Squared Error (MSE) loss and the Adam optimizer. The training process takes in the provided data loader, calculates the loss, and adjusts the weights through backpropagation. The training loop is as follows:

1. For each epoch:
    - Reshape feature and label to 2D tensors.
    - Perform forward propagation to compute the predicted values.
    - Compute the MSE loss between the predicted and true values. The MSE loss function is defined as:
    $$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y^{[i]} - \hat{y}^{[i]})^2$$
    - Perform backward propagation to compute gradients.
    - Update the weights using the Adam optimizer.
2. Track the loss at each epoch.

6 Hyper-Parameters Search

The hyperparameters explored for the MLP model are as follows:

1. Batch Size: [32, 64, 128]
2. Learning Rate: [0.01, 0.001, 0.0001]
3. Number of Epochs: [25, 50, 100]

Each combination of hyperparameters is evaluated via the training process and the loss is recorded. Before the training process, the normalized data is placed into a data loader with the specified batch size and shuffled. The combination with the less loss is selected as the best hyperparameters.

7 Results

After training the MLP, there were varying results; however, two hyperparameter combinations gave a fairly smooth and closely approximated regression curve.

The 2nd best hyperparameters:

- Batch Size: 64
- Learning Rate: 0.001
- Number of Epochs: 100
- Activation Function: ReLU
- Optimizer: Adam
- Number of Layers: 4
- Number of Neurons per Layer: 400, 250, 80, 1

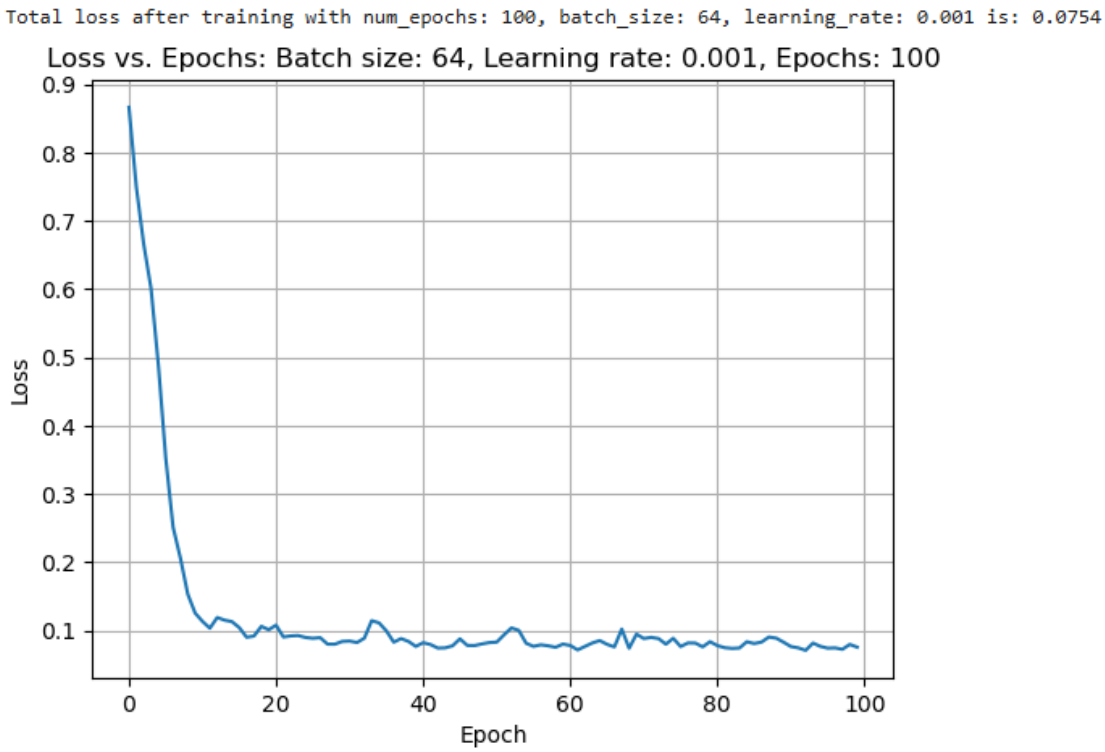

Figure 2: Training loss for the 2nd best hyperparameters.

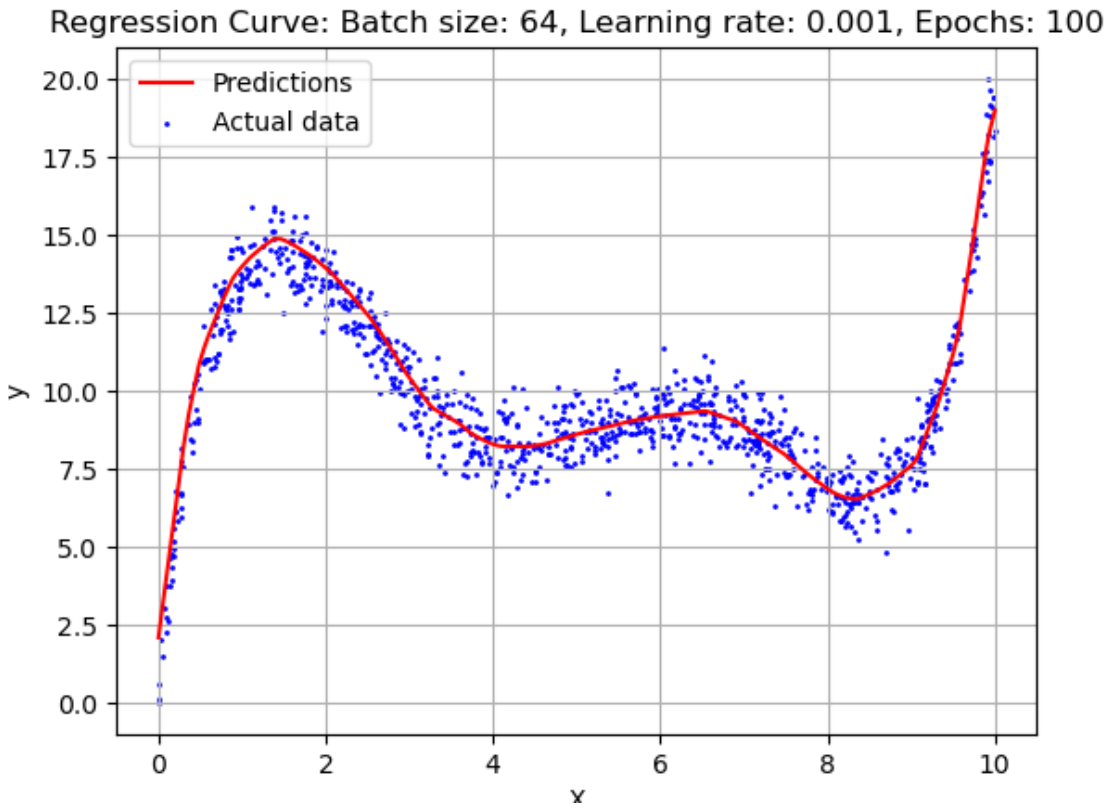

Figure 3: Regression curve for the 2nd best hyperparameters.

The best hyperparameters:

- Batch Size: 128
- Learning Rate: 0.001
- Number of Epochs: 100
- Activation Function: ReLU
- Optimizer: Adam
- Number of Layers: 4
- Number of Neurons per Layer: 400, 250, 80, 1

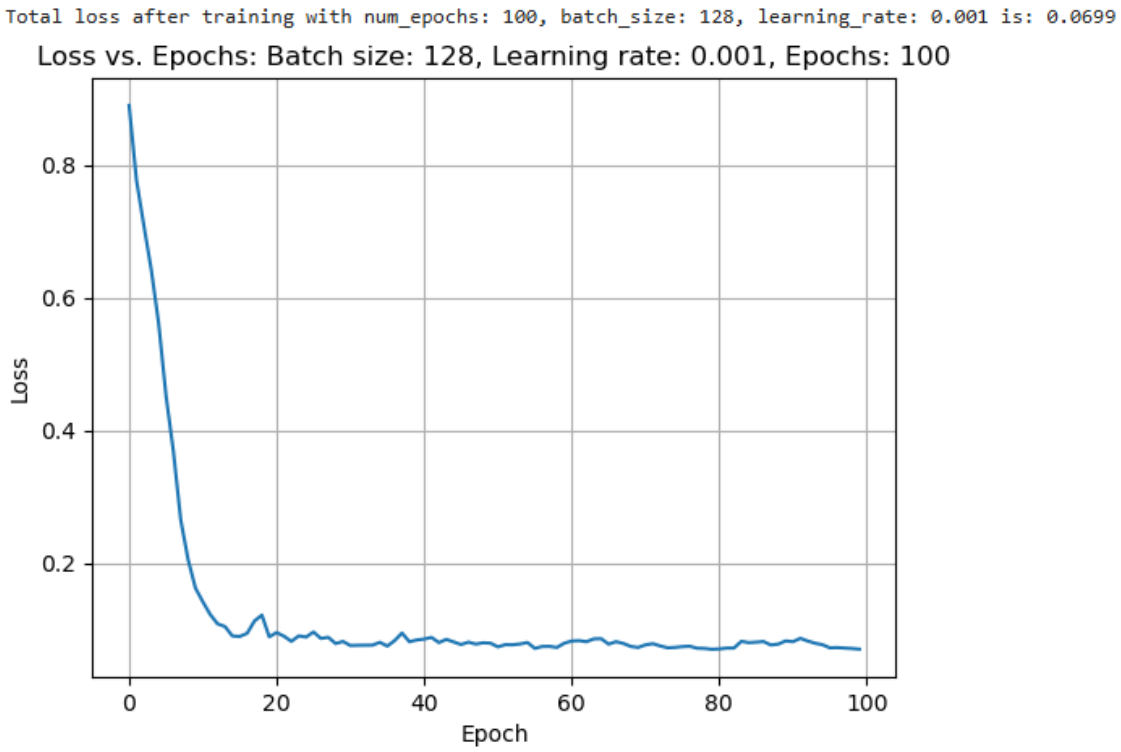

Figure 4: Training loss for the best hyperparameters.

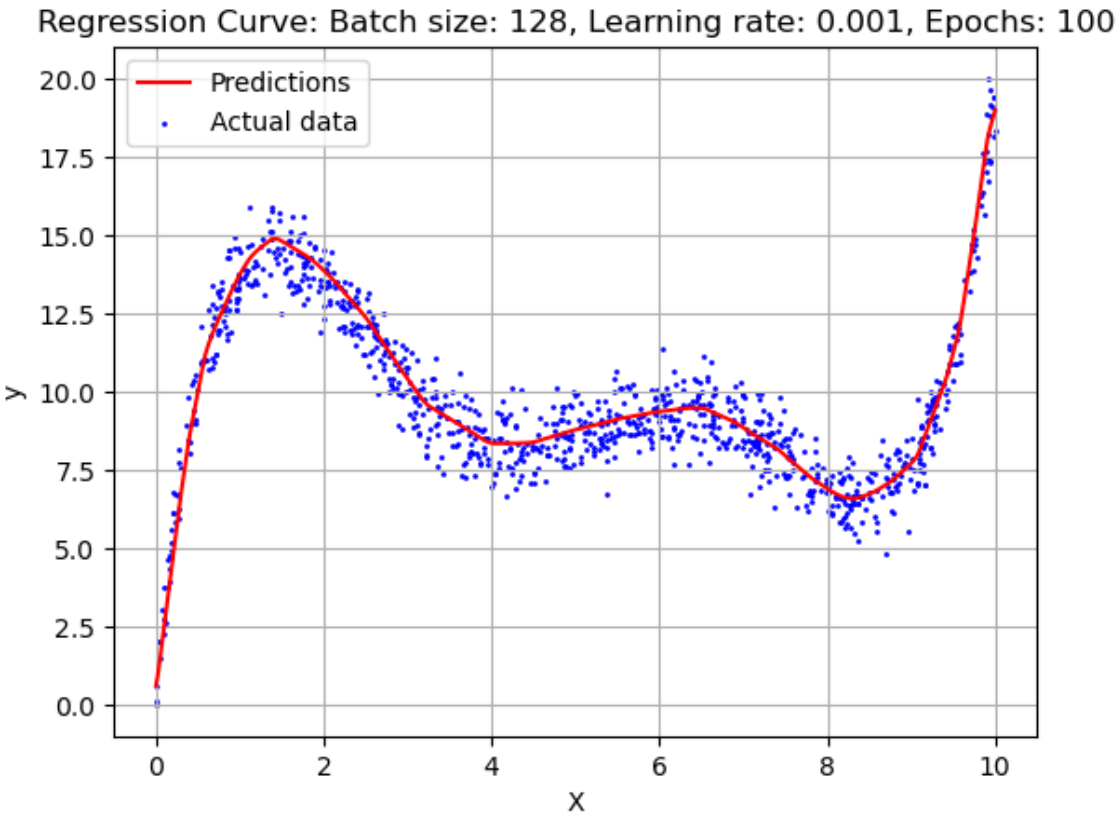

Figure 5: Regression curve for the best hyperparameters.

8 Conclusion

While a larger data size and higher learning rate might seem to offer the best approximation, the results do not necessarily support this assumption. The difference between the second-best hyperparameters and the best ones is only about 0.55 percent. Several factors influence the choice of the best model, such as prioritizing performance over accuracy or considering different data sizes. In conclusion, the model could be further optimized with a more complex architecture, additional data, and more epochs; however, the improvements would likely be marginal compared to the significant increase in training time.