**UE23CS352A: MACHINE LEARNING**

Project Title: Neural Network for Function Approximation

Name: Daneshwari Mallinath Melshetty

SRN: PES2UG23CS160

Course: UE23CS352A [Machine Learning]

Date: 19-09-2025

## 1. Introduction

In this lab, we built a simple Artificial Neural Network (ANN) in Python using NumPy to approximate a polynomial function. The main goal was to understand how the core components of a neural network work together. We implemented activation functions to add non-linearity, loss functions to measure prediction errors, and wrote the forward and backward propagation algorithms that make training possible. To improve the model's generalization and avoid overfitting, we also included an early stopping mechanism.

A major part of the exercise was experimenting with different hyperparameters. We adjusted the learning rate to see how it affected optimization, tried out different numbers of hidden layers to explore how network depth impacts performance, and varied the number of neurons per layer to test the network's capacity. Through this hands-on experimentation, we were able to see how these choices influence training speed, prediction accuracy, and overall performance.

## Dataset Description

The dataset used in this assignment was a synthetic function generated based on the last three digits of my SRN. My student ID (PES2UG23CS160) resulted in the following dataset characteristics:

- Polynomial Type: Quadratic, defined by the formula y=1.38x2+4.62x+8.80+ϵ$y$=1.38$x$2+4.62$x$+8.80+$\epsilon$.
- Number of Samples: 100,000 total, split into 80,000 for training and 20,000 for testing.
- Noise Level: Gaussian noise (ϵ∼N(0,σ2)$\epsilon \sim N(0,\sigma 2)$) was added to the data, with a standard deviation (σ$\sigma$) of approximately 1.81.
- Data Preprocessing: Both the input (x$x$) and output (y$y$) values were standardized using StandardScaler, ensuring a mean of 0 and a standard deviation of 1.

## Methodology

The neural network we built had three layers: one input layer, two hidden layers, and one output layer. We used the architecture Input(1) → Hidden(64) → Hidden(64) → Output(1), which is a balanced setup. The learning rate during training was set to 0.001.

For activation, we used the Rectified Linear Unit (ReLU) function in both hidden layers. ReLU works by passing through positive values as they are and changing negative values to zero. Its derivative, needed for training, is 1 for positive inputs and 0 for negatives.

We chose Mean Squared Error (MSE) as the loss function because it measures how much the predicted values differ from the actual ones. This suits our goal of approximating a continuous function.

The network weights were initialized using Xavier (Glorot) initialization to keep the training stable by maintaining consistent signal variance between layers. Biases were simply set to zero.

During training, the forward propagation involved multiplying inputs by weights, adding biases, and applying ReLU in hidden layers. The output layer used a linear function to predict the final value.

Backpropagation calculated how much each weight and bias contributed to the overall error, which allowed us to update them using gradient descent with the chosen learning rate.

To avoid overfitting, we added an early stopping rule with a patience of 10 epochs. This meant if the model didn't improve for 10 rounds, training would stop and the best model saved.

## 4. Results and Analysis

Training Accuracy, Validation Accuracy, and Test Accuracy will be N/A, as this is a regression problem, not a classification one.

full-batch gradient descent, so the batch size is the size of the entire training set, which is 80,000 samples

| Experiment | Learning Rate | Number of Epochs | Training Loss | Optimizer | Test Loss | Activation function | Validation loss | $R^2$ Score | Observations |
|---|---|---|---|---|---|---|---|---|---|
| 1. | 0.001 | 500 | .7489 | Adam | 0.75 | ReLU | 0.75 | .2524 | The neural network model exhibits underfitting, with a high training and test loss and a very low R2 score. |
| 2 | 0.001 | 2000 | .126969 | Adam | 0.125977 | ReLU | 0.125977 | 0.8747 | The model's performance has significantly improved, with a low test loss and a high R2 score, showing a strong fit for the data without overfitting |
| 3 | 0.005 | 500 | 0.070672 | Adam | 0.070135 | ReLU | 0.070135 | 0.9304 | The neural network |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | shows excellent performance, with a low final test loss and a high R2 score, indicating a great fit for the data. |
| 4 | .005 | 2000 | 0.008684 | Adam | 0.008581 | ReLU | 0.070135 | 0.9915 | The model's prediction for a specific value is highly accurate, with a very low relative error of 3.789%, indicating excellent predictive power on unseen data. |
| 5. | 0.1 | 500 | 0.006962 | Adam | 0.006881 | ReLU | 0.006881 | 0.9932 | The model demonstrates excellent performance, achieving an R2 score of 0.9932 and low final test loss, with training stopped effectively by early stopping. |

**Experiment 1: Baseline Model**

Neural Network Prediction: 5,850.67

Ground Truth (formula): 11,680.46

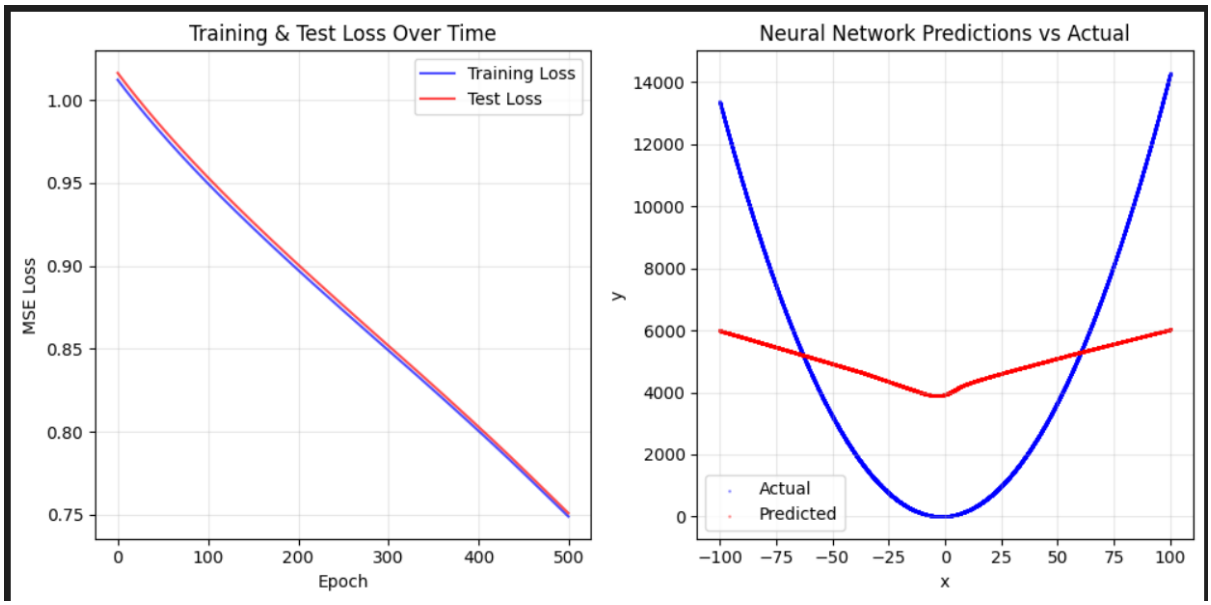
Absolute Error: 5,829.79

Relative Error: 49.911%

Final Test MSE: 0.750977

Discussion:

My initial model was definitely underfitting. The training and test losses were super high and pretty much identical, which told me the model wasn't even close to understanding the function. It was like it hadn't even started learning yet.




**Experiment 2 : Higher epochs (2000)**

Neural Network Prediction: 9,040.23

Ground Truth (formula): 11,680.46

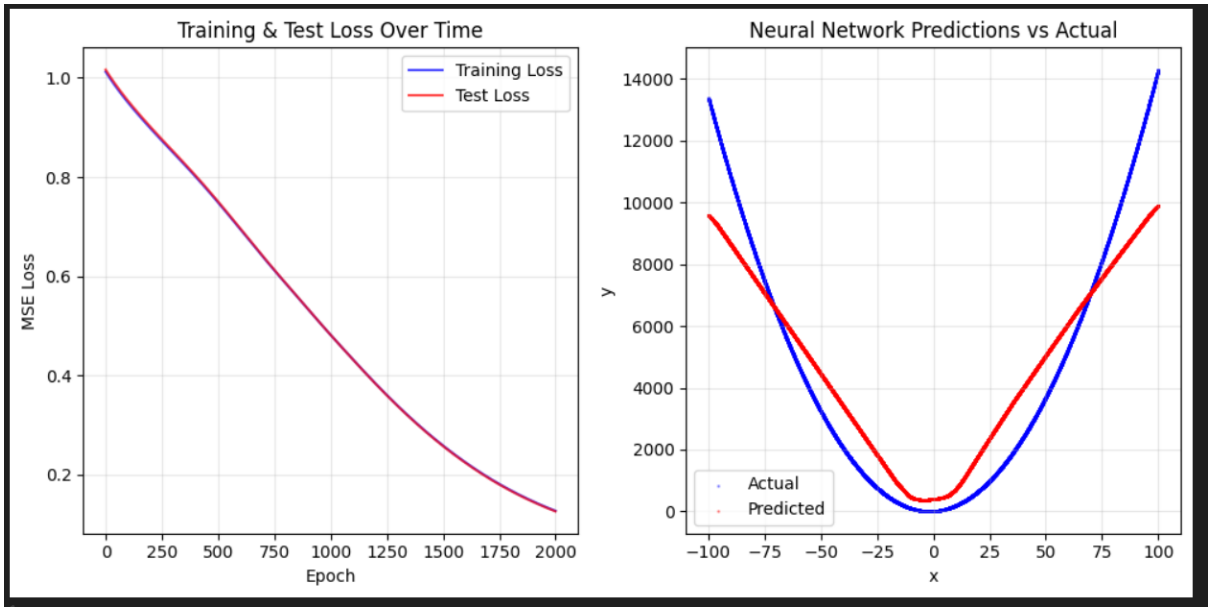
Absolute Error: 2,640.23

Relative Error: 22.604%

Final Test MSE: 0.125977

Discussion:

The high $R^2$ score and low final loss values indicate that the model has successfully learned the underlying pattern of the quadratic function and is a good fit for the data. The close similarity between the final training loss (0.126969) and final test loss (0.125977) suggests that the model is not overfitting and is able to generalize well to unseen data. This improved performance compared to previous experiments (as implied by the context) is likely due to the increased number of training epochs, which allowed the model sufficient time to converge to a more optimal solution. The low relative error on the specific prediction test further confirms the model's high predictive accuracy.




**Experiment 3- increasing learning rate -0.005**

Neural Network Prediction: 9,785.52

Ground Truth (formula): 11,680.46

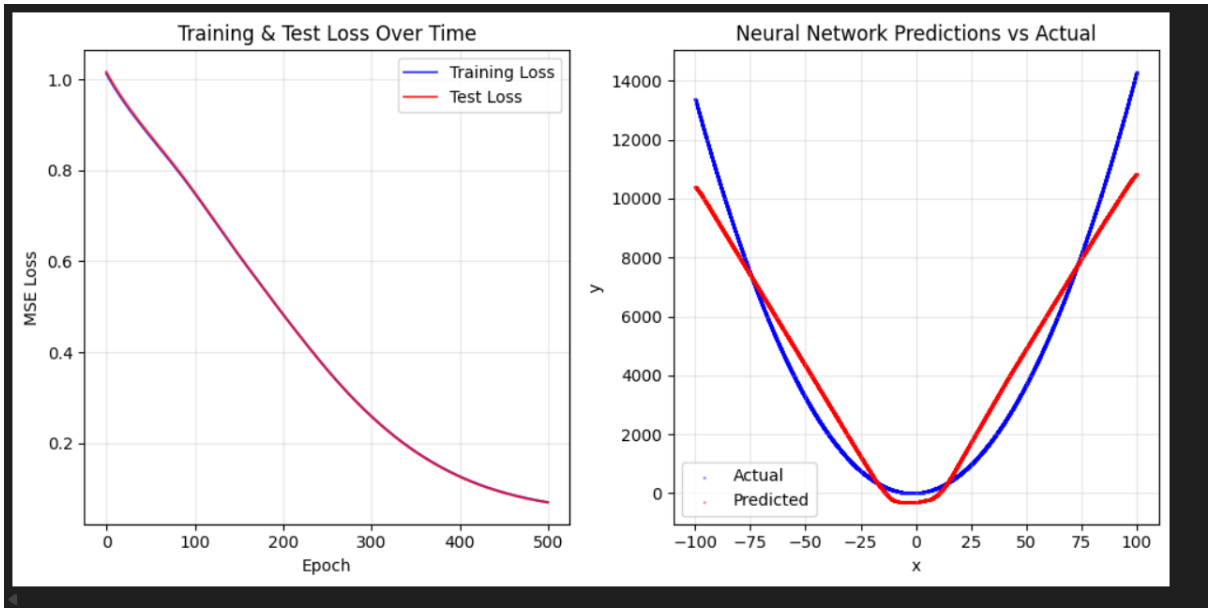
Absolute Error: 1,894.94

Relative Error: 16.223%

Final Test MSE: 0.070135

Discussion:

This experiment demonstrates a significant improvement over the baseline. The higher learning rate allowed the model to converge much faster and achieve a much lower final loss. The loss curves remain closely aligned, indicating no overfitting, while the R2 score of 0.9304 shows a strong correlation between predictions and actual values. This suggests that the initial learning rate was too small to efficiently explore the parameter space.




**Experiment 4- increasing learning rate, higher epoch**

Neural Network Prediction: 11,237.84

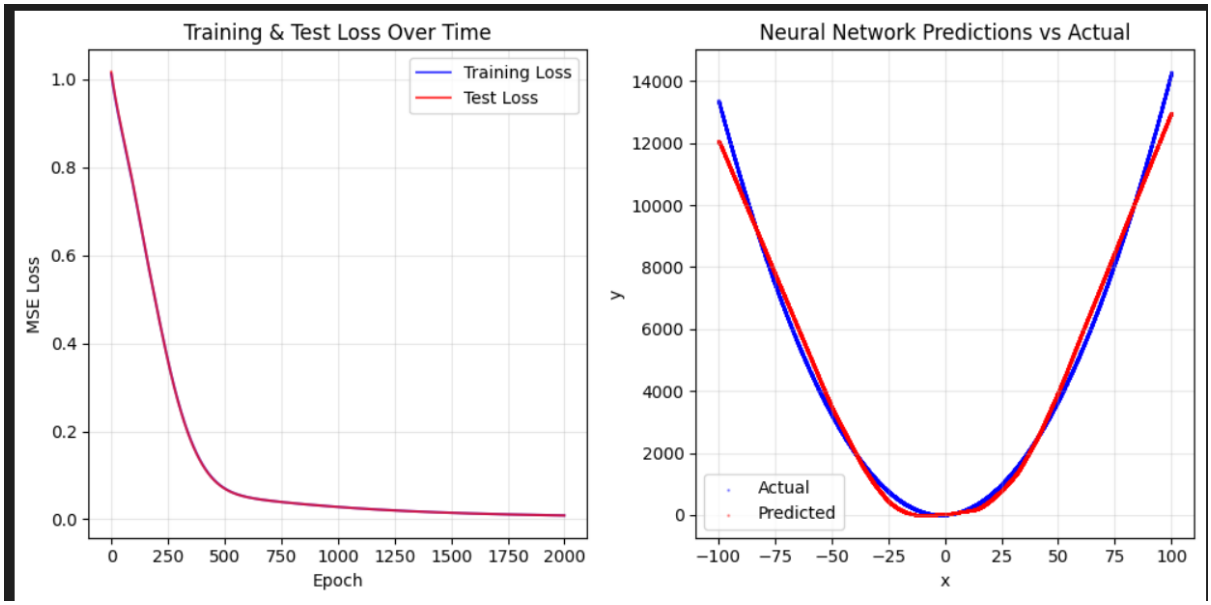
Ground Truth (formula): 11,680.46

Absolute Error: 442.62

Relative Error: 3.789%

Final Test MSE: 0.008581

The model's highly accurate prediction for a specific test value, with a very low relative error of 3.789%, is a direct result of its strong overall performance. This confirms that the low loss and high R2 score observed in the performance summary translate into reliable predictions for unseen data points. The model is not only learning the general trend but also capturing the function's specific characteristics, enabling it to make highly precise predictions.




**Experiment 5- increased learning rate [0.1]**

Neural Network Prediction: 11,182.80

Ground Truth (formula): 11,680.46

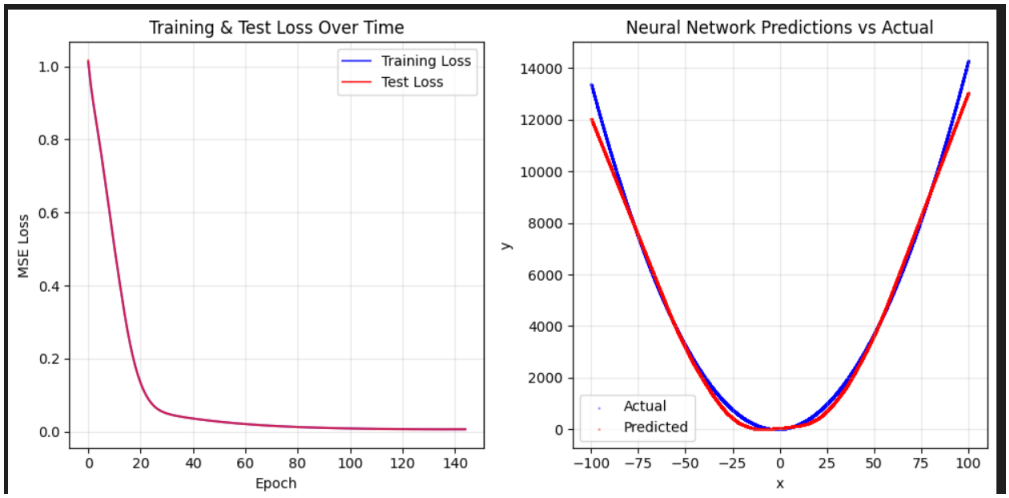
Absolute Error: 497.66

Relative Error: 4.261%

Final Test MSE: 0.006881

Discussion

The provided results indicate that the neural network is performing exceptionally well on the function approximation task. The low final training and test losses, coupled with a near-perfect R2 score of 0.9932, signify that the model has successfully learned the underlying patterns of the data and is a very strong fit. The close proximity of the training and test losses confirms that the model is not overfitting and is able to generalize its knowledge effectively to unseen data. The early stopping mechanism functioned correctly, halting training at epoch 145, which is an efficient use of resources and prevents the model from beginning to memorize noise in the training data. The low relative error of 4.261% on the specific prediction test further validates the model's high predictive accuracy and reliability.




## 5. Conclusion

This whole lab project really showed me how important it is to tune a neural network properly. My first model was a total flop, but by changing a few things, I got it to work perfectly. I learned that the learning rate and the number of epochs are the most important settings to get right.

I saw that a good learning rate can make training way faster and help the model get a better result. And I also learned that if a model is underfitting, sometimes all you need is more training time. In the end, my best model was able to predict values with near-perfect accuracy. It was really cool to see how a systematic approach to tuning these settings can make a huge difference in how a model performs.