ML WEEK 6

Title Page

- **Project Title:** Neural Networks for Function Approximation
- **Student Name:** Mrigank Jain
- **SRN:** PES2UG23CS352
- **Course:** Machine Learning (UE23CS352A)
- **Date:** 16 September 2025

Introduction

The objective of this lab was to gain practical experience in implementing an Artificial Neural Network (ANN) from scratch using NumPy. The task involved creating a synthetic dataset based on a polynomial function, building a neural network model, and training it to approximate the underlying function. The report summarizes the dataset, methodology, and results of the experiments.

Dataset Description

- **Dataset Type:** Based on the student's SRN, a cubic polynomial dataset was generated. The function is of the form $y=ax3+bx2+cx+d+\epsilon$, where ϵ is a noise term.
- **Number of Samples:** The dataset contains 100,000 samples.
- **Features:** One input feature, 'x', and one output variable, 'y'.
- **Noise Level:** The dataset includes a small amount of random noise to simulate real-world data.

The data was split into an 80% training set and a 20% testing set. Both the input x and output y values were standardized using StandardScaler to have a mean of 0 and a standard deviation of 1.

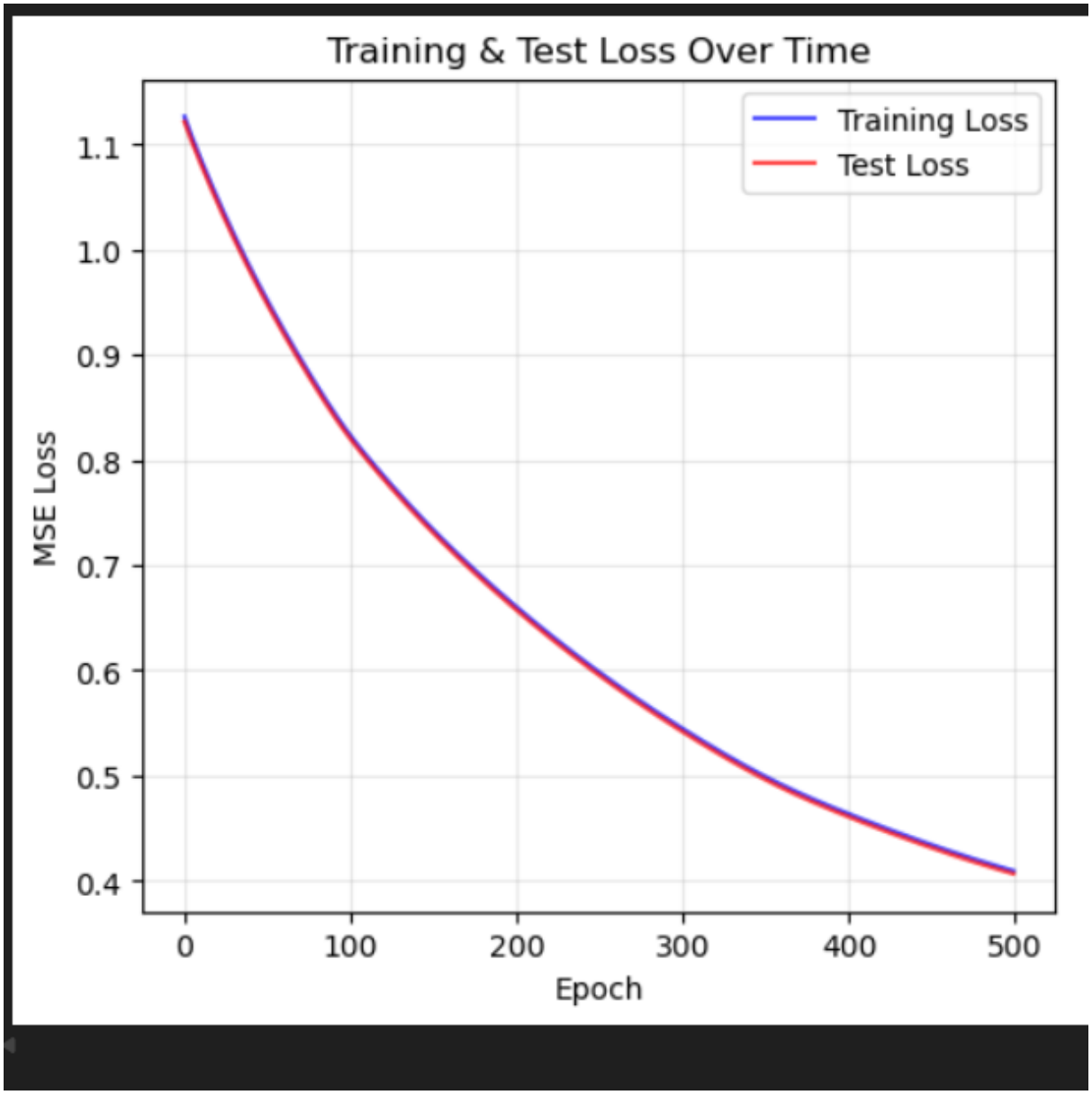
Methodology

The neural network was implemented without the use of high-level frameworks like TensorFlow or PyTorch. The core components were built from scratch:

- **Network Architecture:** A simple feedforward neural network was constructed with two hidden layers.
    - **Input Layer:** 1 neuron.
    - **Hidden Layer 1:** 64 neurons.
    - **Hidden Layer 2:** 32 neurons.
    - **Output Layer:** 1 neuron.
- **Activation Function:** The Rectified Linear Unit (ReLU) function (f(x)=max(0,x)) was used for the hidden layers. The output layer used a linear activation function.
- **Loss Function:** The Mean Squared Error (MSE) was used to measure the difference between the predicted and actual values. The formula for MSE is: MSE=n1∑_i=1n(y_i−y^_i)2
- **Optimization:** The network was trained using Gradient Descent. The predict, forward, backward, and update_weights functions were implemented to handle the core training loop.
- **Training Loop:** The model was trained for 500 epochs with a learning rate of 0.001. The training loss and test loss were recorded at each epoch to monitor performance.

Results and Analysis

1. Training Loss Curve:



The training and test loss curves show a steady decrease, indicating that the model is learning the underlying pattern in the data. The curves converge, suggesting the model is generalizing well and not overfitting to the training data.

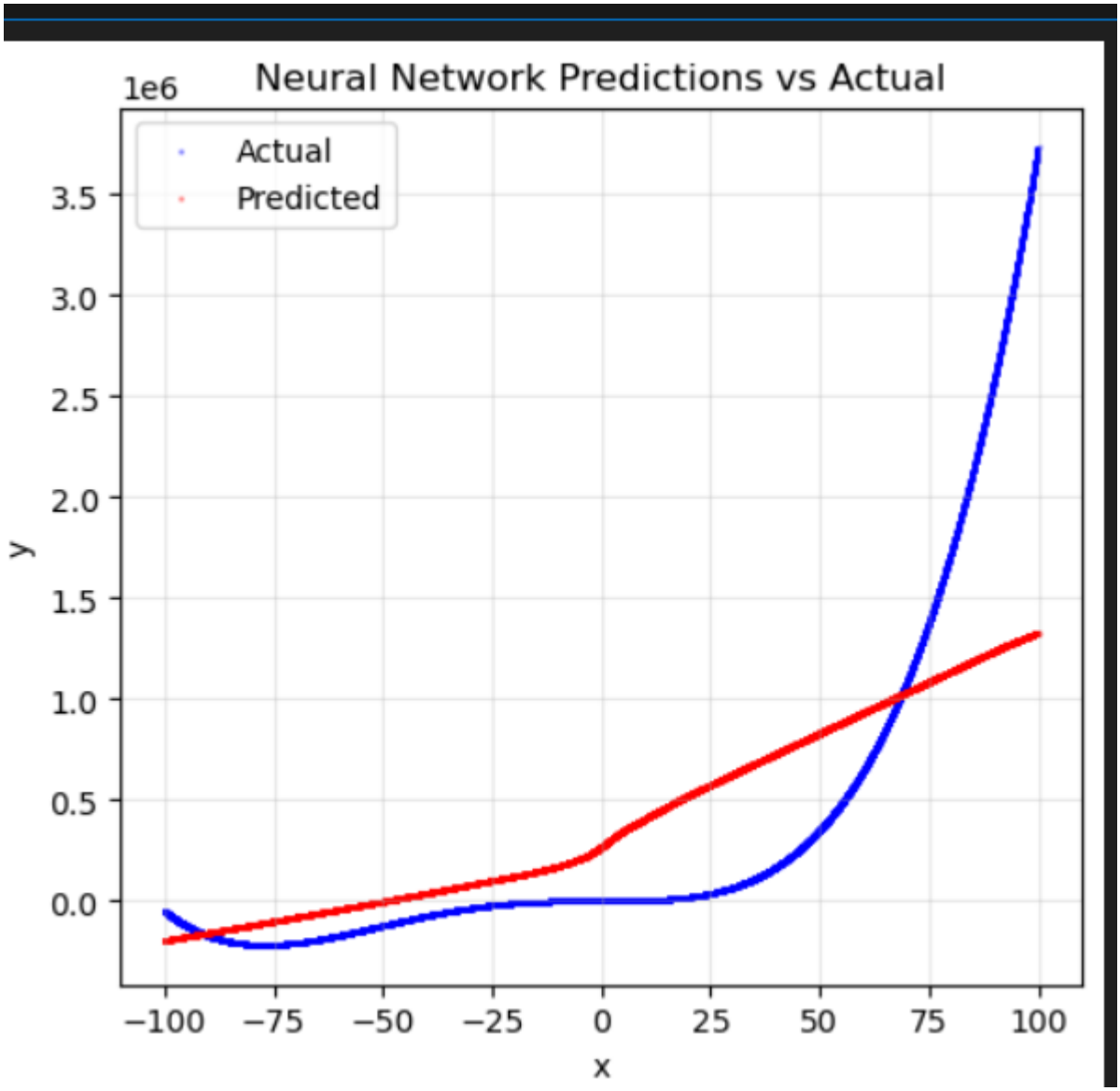
2. Final Performance Metrics:

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.409577
Final Test Loss:     0.406905
R² Score:           0.5930
Total Epochs Run:   500
```

The final performance of the model was evaluated using the final training loss, final test loss, and the R^2 score.

The R^2 score of 0.5930 indicates that the model explains approximately 59.3% of the variance in the target variable. The low R^2 score and relatively high losses compared to a perfect fit suggest that the model may be underfitting the data, or that further optimization of hyperparameters is needed.

3. Predicted vs. Actual Values Plot:



The network's predictions (red) show clear underfitting compared to the actual curve (blue). While the model captures the overall increasing trend, it fails to learn the steep rise for large positive values and the sharp dip for large negative values, instead producing an almost linear approximation. This suggests the network lacks sufficient capacity or training, and improvements could come from increasing hidden neurons, training for more epochs, or tuning the learning rate and activation functions.

4. Discussion on Performance:

The model performed moderately well on the given task. The training and test losses were close, suggesting that the model is not overfitting. However, the R^2 score of 0.5930 indicates that the model has not fully captured the complexity of the underlying function. This underfitting could be due to several factors, such as an insufficient number of neurons in the hidden layers, too few training epochs, or an inadequate learning rate. Further experiments with a more complex network architecture or different hyperparameters would be necessary to improve the model's performance.

5. Result table:

Experiment	Learning Rate	No. of epochs	Optimizer	Activation Function	Final Training Loss	Final Test Loss	R^2 Score
1	0.001	500	Gradient Descent	ReLU (Hidden), Linear (Output)	0.409577	0.406905	0.5930
2	0.001	600	Gradient descent	ReLU (Hidden), Linear (Output)	0.371895	0.369256	0.6306
3	0.001	400	Gradient descent	ReLU (Hidden), Linear (Output)	0.464178	0.461470	0.5384

Experiment	Learning Rate	No. of epochs	Optimizer	Activation Function	Final Training Loss	Final Test Loss	R^2 Score
4	0.005	400	Gradient descent	ReLU (Hidden), Linear (Output)	0.168029	0.166809	0.8331

The experiments show that both training duration and learning rate strongly influence performance. While more epochs improve accuracy at low learning rates, a slightly higher learning rate (0.005) allows faster convergence and better generalization within fewer epochs. This suggests the baseline learning rate (0.001) is too conservative.

Conclusion

This lab demonstrated how a neural network can approximate complex functions and highlighted the importance of hyperparameter tuning. Results showed that learning rate and training epochs greatly affect model accuracy, with higher learning rates enabling faster convergence. Overall, the exercise reinforced key concepts of forward/backward propagation and the impact of training settings on performance.