WEEK 6

Neural Network Regression for Polynomial Function Approximation

---

Name: Preksha Kamalesh
SRN: PES2UG23CS902
Course: Machine Learning
Date: September 16, 2025

---

1. Introduction

Purpose

The objective of this lab was to build, train, and evaluate a neural network from scratch using NumPy. The network was designed to approximate a dynamically assigned polynomial function, providing practical experience with fundamental deep learning concepts such as weight initialization, forward and backward propagation, activation functions, and loss calculation.

Tasks Performed

- **Dataset Generation:** A synthetic dataset was generated based on a unique quartic polynomial function assigned according to the student SRN.
- **Neural Network Implementation:** A feedforward neural network with two hidden layers was implemented. This included:
    - ReLU activation function and its derivative.
    - Mean Squared Error (MSE) loss function.
    - Xavier weight initialization.
    - Forward and backward propagation algorithms.
- **Model Training:** The network was trained using gradient descent with a full-batch approach. Early stopping was implemented to prevent overfitting.
- **Evaluation:** The model's performance was evaluated using the final test MSE, an R^2 score, and visualizations of the training process and prediction accuracy.

---

2. Dataset Description

The dataset was synthetically generated to model a specific polynomial function with added Gaussian noise.

- **Polynomial Type:** The assigned function was a **quartic polynomial** defined as:
- $y = 0.0082x^4 + 1.62x^3 + 0.25x^2 + 4.24x + 11.33$
- **Number of Samples:** A total of **100,000** samples were generated.
- **Features:** The dataset contains a single input feature, 'x', with values uniformly distributed between -100 and 100.

- **Noise Level:** Gaussian noise with a standard deviation (σ) of **2.45** was added to the target variable 'y' to simulate real-world data imperfections.
- **Data Split:** The data was split into **80,000** training samples and **20,000** test samples.

---

3. Methodology

The neural network was constructed with a specific architecture determined by the SRN.

- **Architecture:** A **Wide-to-Narrow** architecture was used, consisting of an input layer, two hidden layers, and an output layer (1 -> 72 -> 32 -> 1).
- **Activation Functions:** The **ReLU (Rectified Linear Unit)** activation function was applied to both hidden layers. The output layer used a linear activation for regression.
- **Weight Initialization: Xavier (Glorot) initialization** was implemented to set the initial weights, which helps in preventing vanishing or exploding gradients.
- **Loss Function:** The **Mean Squared Error (MSE)** was used to quantify the model's prediction error.
- **Training:** The model was trained using **full-batch gradient descent** for 500 epochs. Data was standardized before training to improve convergence.

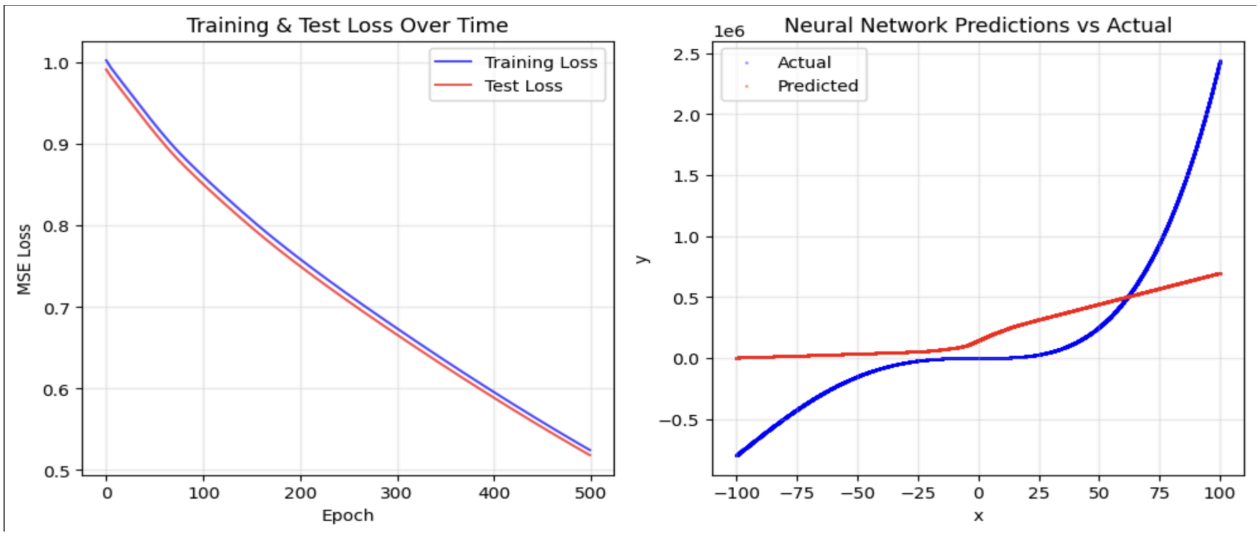
---

4. Results and Analysis

The model was trained successfully, and its performance was analyzed through loss curves and prediction plots.

Training Loss Curve

The training and test loss curves show a consistent decrease over the 500 epochs, indicating that the model was learning effectively. The test loss closely follows the training loss without significant divergence, which suggests that the model did not overfit the training data.



Final Test MSE

The final test loss, which is the Mean Squared Error on the unseen test data, was **0.518350**.

Predicted vs. Actual Values

The scatter plot of predicted vs. actual values shows that the model's predictions (red) align reasonably well with the true data distribution (blue). However, the model struggles to capture the full complexity of the quartic function, particularly at the extremes of the x-value range. This is evident in the specific prediction test for x = 90.2, where the relative error was **62.569%**.

Discussion on Performance

The R^2 score of **0.4758** indicates that the model explains approximately 47.6% of the variance in the target variable. While the model learned the general trend, it exhibits signs of **underfitting**. The single set of weights and biases from the full-batch gradient descent may not have been sufficient to capture the intricate curvature of the quartic polynomial. The high error for the specific prediction further supports this conclusion.

Results Table

Metric	Value
Learning Rate	0.001
Batch Size	80,000 (Full-batch)
Number of Epochs	500
Optimizer	Batch Gradient Descent
Activation Function	ReLU (Hidden), Linear (Output)
Training Loss	0.524567
Test Loss (MSE)	0.518350
Test Accuracy (R^2)	0.4758
Observations	The model shows stable learning but underfits the complex quartic function, resulting in a moderate R^2 score and significant error on specific out-of-sample predictions.

5. Conclusion

This lab demonstrated the process of building a neural network for regression from the ground up. The implemented model successfully learned the underlying trend in the noisy quartic data but was not complex enough to approximate the function with high fidelity, as indicated by the final R^2 score and prediction errors. The results suggest that for this particular problem, a more complex model, a different optimization strategy (like mini-batch gradient descent with an adaptive optimizer like Adam), or further hyperparameter tuning might be necessary to improve performance and mitigate underfitting.