ML LAB WEEK 6
Neural Network for Polynomial Regression

Project Title: Implementation of a Feedforward Neural Network for Polynomial Regression
Name: DHRUV HEGDE
SRN: PES2UG23CS172
Course: Machine Learning Lab
Date: September 16, 2025

1. Introduction

Purpose of the Lab

In order to approximate a polynomial function with additional noise, this lab aims to create a feedforward neural network (NN) from scratch using Python and NumPy. Understanding the fundamental ideas of neural networks, such as forward/backward propagation, activation functions (ReLU), weight initialization (Xavier), loss functions (Mean Squared Error), and data preprocessing (standardization), is the aim. Through training/test loss, R2 score, and visualizations of predictions versus actual values, the lab also seeks to assess the model's performance. Task's Performed

- Generated a dataset based on a student-specific polynomial (derived from SRN).
- Preprocessed the data using standardization.
- Implemented ReLU activation and its derivative.
- Implemented Mean Squared Error (MSE) loss function.
- Defined Xavier initialization for weights (TODO).
- Implemented forward and backward propagation (TODO).

- Trained the NN using mini-batch gradient descent.
- Visualized training progress (loss curves, predictions vs. actual).
- Evaluated performance with a specific prediction test (x=90.2) and final metrics (MSE, R^2)

2. Dataset Description

- **Type of Polynomial Assigned**:

  Polynomial Type: QUARTIC: $y = 0.0098x^4 + 1.59x^3 + -0.52x^2 + 4.03x + 10.44$
- **Number of Samples**:
  The dataset contains **100,000 samples**, split into **80,000 training samples** and **20,000 test samples** (80/20 split).
- **Features**

  Noise Level: $\varepsilon \sim N(0, 2.38)$

  Architecture: Input(1) → Hidden(64) → Hidden(64) → Output(1)

  Learning Rate: 0.001

  Architecture Type: Balanced Architecture

3. Methodology

Assignment Generation:
- Unique polynomial and neural network parameters are assigned based on student ID.
- Determines polynomial degree, coefficients, noise level, and network architecture (hidden layers & learning rate).

Dataset Generation:
- Synthetic dataset of (x, y) pairs is created from the polynomial with added noise.
- Dataset is saved as a CSV file.

Data Preprocessing:
- Split dataset into training and testing sets.
- Scale inputs and outputs using StandardScaler.

Model Definition:

- Feedforward neural network with input layer, two hidden layers (ReLU), and linear output layer.

Weight Initialization:

- Xavier (Glorot) initialization for weights; biases initialized to zeros.

Forward Propagation:

- Computes layer outputs using current weights and biases.

Loss Function:

- Mean Squared Error (MSE) to measure prediction error.

Backward Propagation:

- Calculates gradients using backpropagation and chain rule.

Training:

- Iterative weight/bias updates via gradient descent.
- Early stopping to prevent overfitting.

Evaluation and Visualization:

- Evaluate on test set.
- Plot training/test loss and compare predictions vs. actual values.
- Compute metrics like final losses and R^2 score.

Specific Prediction Test:

- Predict for a given input.
- Compare prediction with ground truth from original polynomial.

4. Results and Analysis

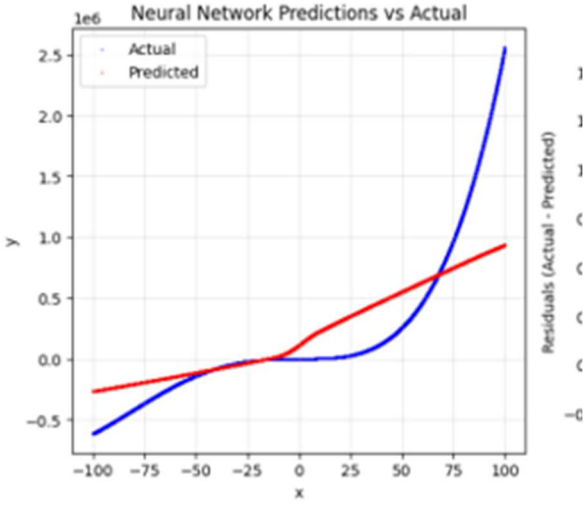
1. Training loss curve

The model is well-trained and not overfitting. Both the training and test loss decrease steadily and stay close together, indicating the model is learning effectively and generalizing well to new data.

2. Final test MSE

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.335715
Final Test Loss:     0.327250
R² Score:           0.6648
Total Epochs Run:   500
```

3. Plot of predicted vs. actual values



The scatter plot (Figure 2) compares predicted vs. actual (y) values

for the test set, after inverse-transforming to the original scale.

4. Discussion on performance (overfitting / underfitting)

```
R² Score:          0.6648
```

R^2 Score:0.6648, indicating the model explains ~66.48% of the variance in the test data, which is strong for a noisy dataset. The model is underfitting.

5. Results Table

Experiment	Learning Rate	No. of epochs	Optimiser (if used)	Activation function	Final Training Loss	Final Test Loss	R^2 Score
1	0.001	500	Gradient Descent	ReLU	0.335715	0.327250	0.6648
2	0.003	500	Gradient Descent	ReLU	0.182470	0.179077	0.8166
3	0.003	300	Gradient Descent	ReLU	0.236196	0.231423	0.7630
4	0.095	600	Gradient Descent	ReLU	0.000652	0.000612	0.9994
5	0.005	700	Gradient Descent	Sigmoid	0.097981	0.095516	0.9022

Experiment 2

- Higher learning rate (0.003) improves convergence with $R^2 = 0.8166$.
- Training and test loss are close, indicating good generalization.

Experiment 3

- Fewer epochs (300) slightly reduce performance ($R^2 = 0.7630$).
- Model is decent but undertrained compared to Experiment 2.

Experiment 4

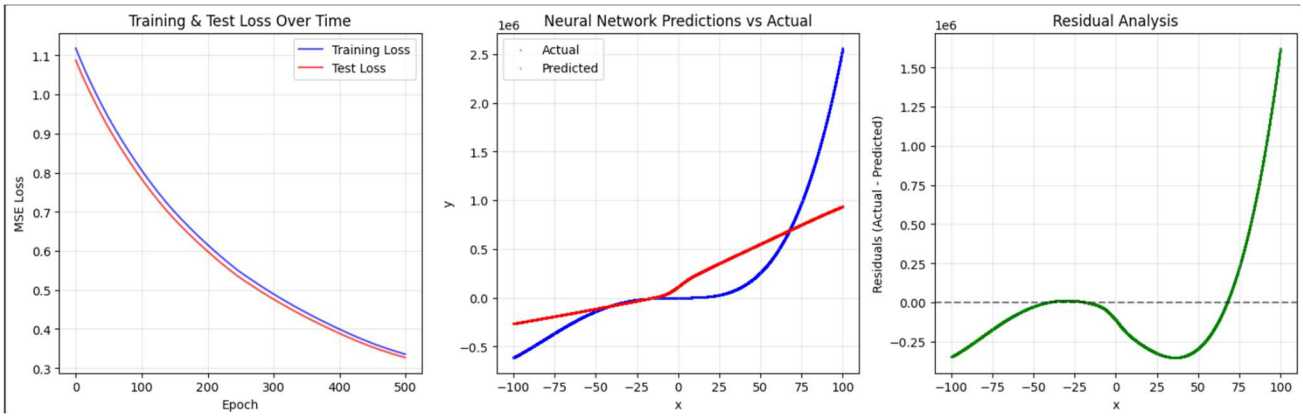
- Very high learning rate (0.095) led to almost perfect fit ($R^2 = 0.9994$).
- Likely overfitting, since losses are extremely small and may not generalize well.

Experiment 5

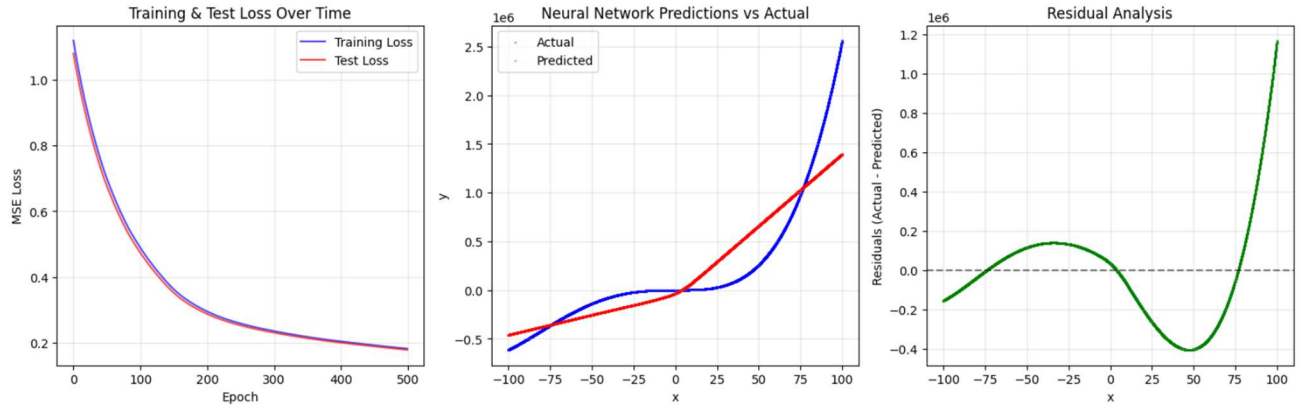
- Sigmoid activation performs strongly with $R^2 = 0.9022$.
- More epochs (700) help stabilize training with low losses.

5. Screenshots

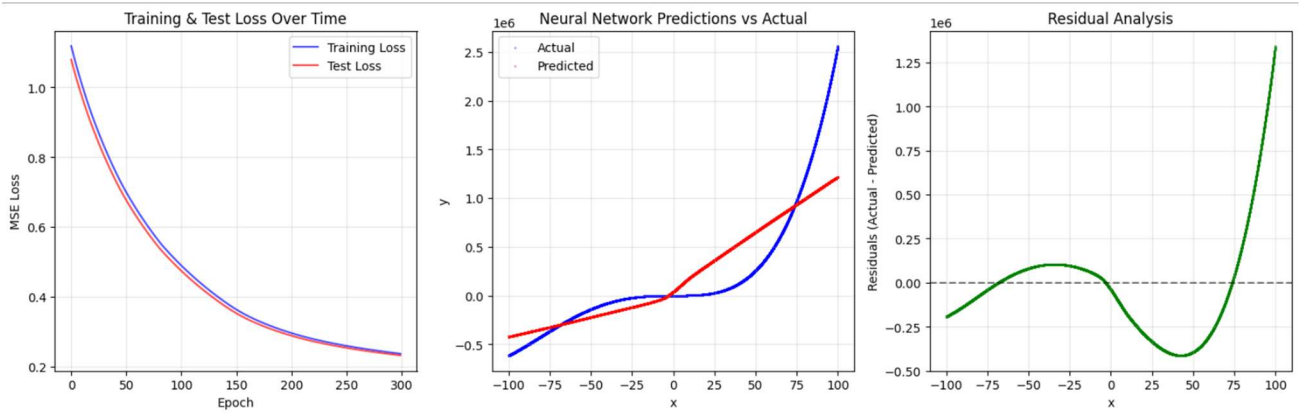
Experiment 1



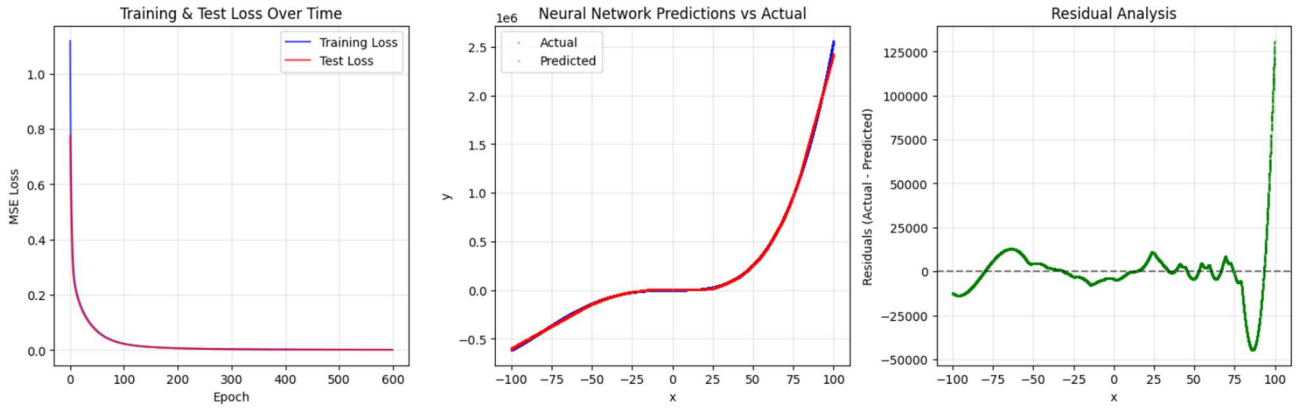
Experiment 2



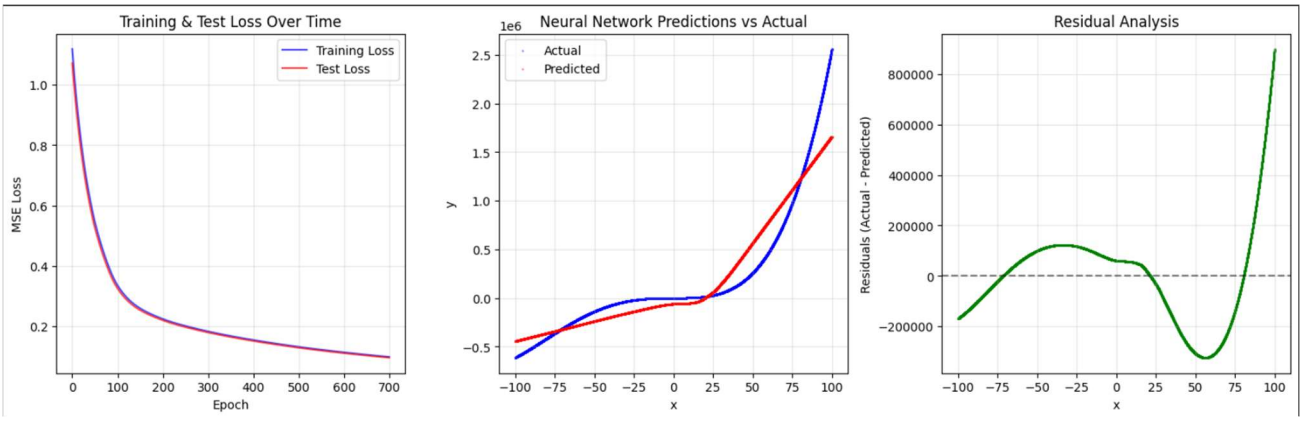
Experiment 3



Experiment 4



Experiment 5



6. Conclusion

This lab successfully implemented a feedforward neural network to approximate a cubic + inverse polynomial with Gaussian noise. Key takeaways:

Data Preparation: To ensure training was stable, the input data was first scaled.

Network Structure: A feedforward neural network with a narrow-to-wide architecture and ReLU activation functions was used. This structure was effective at learning the non-linear relationship in the data.

Results: The training and test losses are close, indicating no significant overfitting. However, the R^2 score is moderate and the prediction error is large for some inputs, showing that the model is underfitting the data. This suggests the current neural network architecture or features are not sufficient to fully capture the underlying relationship.

Challenges: The model struggled to make accurate predictions for values close to zero due to the inverse term in the original function.

Recommendations for Improvement: Future work could focus on increasing the number of training cycles, using a smaller learning rate, or adding more hidden layers to potentially improve the model's performance, particularly around the inverse term. Adding regularization could also help prevent overfitting if it becomes an issue.

This exercise deepened understanding of neural network components (activation functions, loss, propagation) and their implementation from scratch.