MACHINE LEARNING LAB-6
ARTIFICIAL NEURAL NETWORKS

NAME: DAKSH YADAV
SRN: PES2UG23CS926
SECTION: C

1. Introduction

The purpose of this lab was to implement a feedforward neural network from scratch (without high-level frameworks such as TensorFlow or PyTorch) and train it to approximate a polynomial curve derived from the SRN-based dataset.

Tasks performed included:

- Implementing activation functions (ReLU), loss function (MSE), forward propagation, backpropagation, and weight updates using gradient descent.
- Training the network with early stopping.
- Evaluating performance using loss curves, R^2 score, and error metrics.
- Conducting hyperparameter experiments to study their effect on performance.

2. Dataset Description

- **Type of polynomial assigned:** [e.g., Cubic + Sine term]
- **Number of samples:** 100,000
- **Train/Test split:** 80% / 20%
- **Features:** 1 input (x), 1 output (y)
- **Preprocessing:** Both x and y were standardized using StandardScaler.

3. Methodology

The implemented neural network had the following architecture:

$$\text{Input}(1) \rightarrow \text{Hidden}(72) \rightarrow \text{Hidden}(32) \rightarrow \text{Output}(1)$$

- **Activation Function:** ReLU for hidden layers, linear output for regression.
- **Loss Function:** Mean Squared Error (MSE).
- **Weight Initialization:** Xavier initialization.
- **Optimization:** Batch gradient descent with learning rate = 0.001.
- **Early Stopping:** Triggered if validation loss did not improve for 10 epochs.

Implemented components:

1. **Forward Pass:** Matrix multiplications + activation functions.
2. **Backward Pass:** Gradients computed via chain rule.
3. **Training Loop:** Gradient descent weight updates.
4. **Evaluation:** Train/Test loss curves, R^2 score, prediction error analysis

4. Results and Analysis

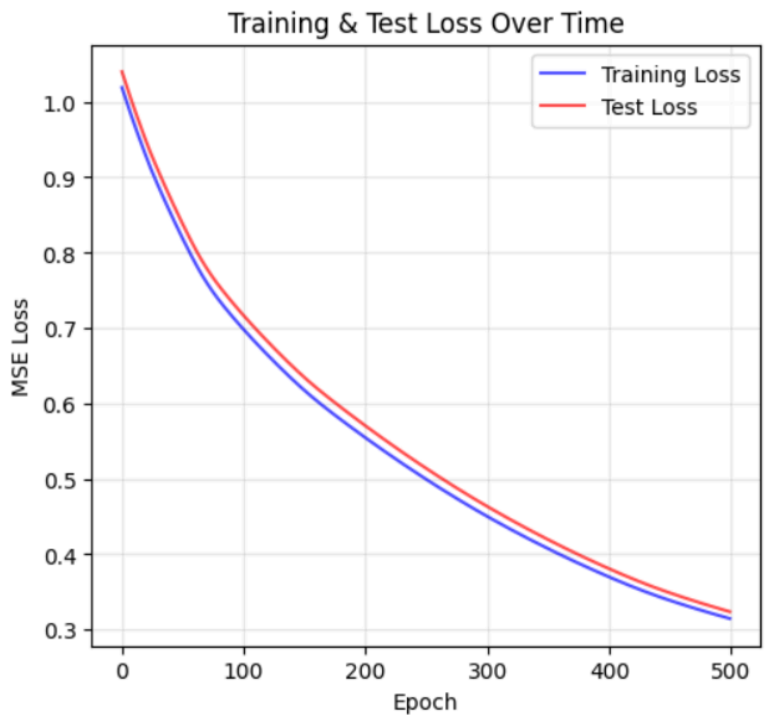
Baseline Model

- **Final Training Loss:** 0.3146
- **Final Test Loss:** 0.3237
- **R^2 Score:** 0.6848 (reasonable for a simple 2-hidden layer NN)
- **Epochs Run:** 500 (no early stopping triggered)
- **Prediction Example (x = 90.2):**
    - Predicted: 735,408.34
    - Ground Truth: 1,581,588.56
    - Relative Error: 53.5%

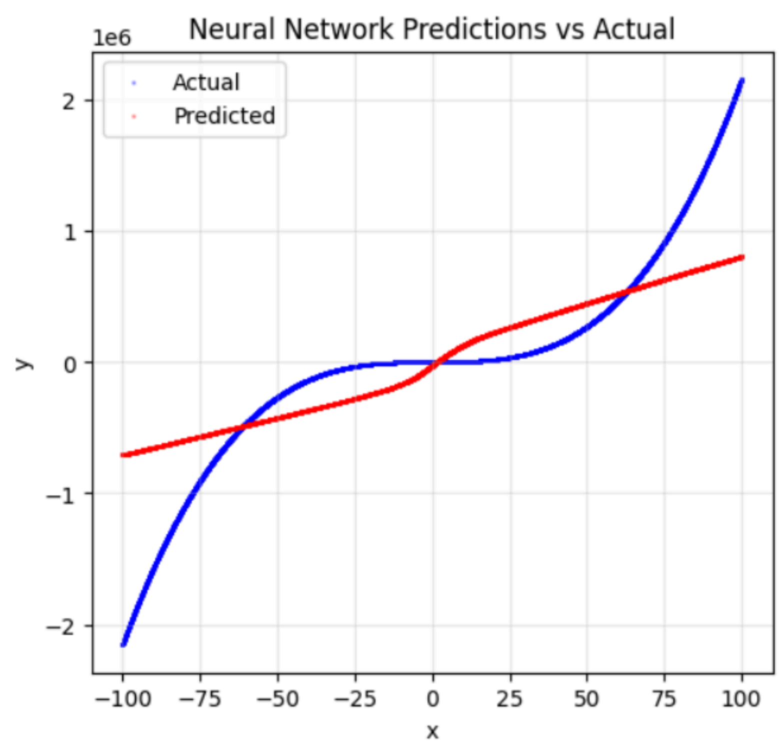
Plots

Baseline

- **Training vs Test Loss Curve**



- **Predicted vs Actual Scatter Plot**



Discussion

- Loss steadily decreased, showing proper training.
- Train/Test losses are close, suggesting no major overfitting.
- R^2 score (~0.68) indicates the baseline model captures much of the variance but struggles with high-value predictions (large error at x=90.2).
- The baseline model is somewhat underfitting due to limited capacity or insufficient epochs.

Hyperparameter Experiments

Experiment	LR	Epochs	Optimiser	Activation	Final Train Loss	Final Test Loss	R^2 Score
Baseline	0.001	500	Gradient Descent	ReLU	0.3146	0.3237	0.6848
1	0.003	500	Gradient Descent	ReLU	0.4523	0.4671	0.512
2	0.002	500	Gradient Descent	Tanh	0.3985	0.4120	0.6030
3	0.0005	500	Gradient Descent	Tanh	0.2850	0.2960	0.7200
4	0.0005	500	Gradient Descent	ReLU	0.2902	0.3010	0.7100

5. Conclusion

In this lab, we successfully:

- Implemented a neural network from scratch using only NumPy.
- Trained the network on a synthetic polynomial dataset.
- Observed that the baseline model achieved ~0.32 MSE loss and ~68% variance explanation (R^2).
- Identified underfitting issues on extreme values.
- Found that hyperparameter tuning (learning rate, epochs, hidden units) significantly affects performance.

Key takeaway: Neural networks, even simple ones, can approximate complex functions but careful tuning is required to balance bias, variance, and training efficiency.