DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MACHINE LEARNING - UE23CS352A

LAB 6

NAME : CHINMAYI SHRAVANI YELLANKI
SRN : PES2UG23CS152
CLASS AND SEC : SEM 5 - C
DATE : 16.09.2025

1. Introduction

The main goal of this lab was to build an artificial neural network from scratch that could approximate a complex polynomial function. This function was specially generated based on the last three digits of the SRN. In this project, we created a custom dataset, designed a neural network with two hidden layers, trained it using backpropagation and gradient descent, and checked how well the model performed on regression tasks. The key steps involved designing and coding activation and loss functions, setting up the training process, experimenting with different hyperparameters, and writing an analytical report.

2. Dataset Description

- Type of polynomial used: Quartic: y = 0.0194x^4 + 2.02x^3 − 0.94x^2 + 2.98x + 9.28
- y=0.0194x^4 +2.02x^3−0.94x^2 +2.98x + 9.28
- Number of data points: 100,000 synthetic samples
- Data split:
    - Training set: 80,000 samples (80%)
    - Test set: 20,000 samples (20%)
- Number of features: 1 input feature (x), 1 output label (y)
- Noise level: $\epsilon \sim N(0, 1.91)$
- ϵ is noise from a normal distribution with mean 0 and variance 1.91
- Standardization: Both x and y were scaled to have zero mean and unit variance using StandardScaler before training.

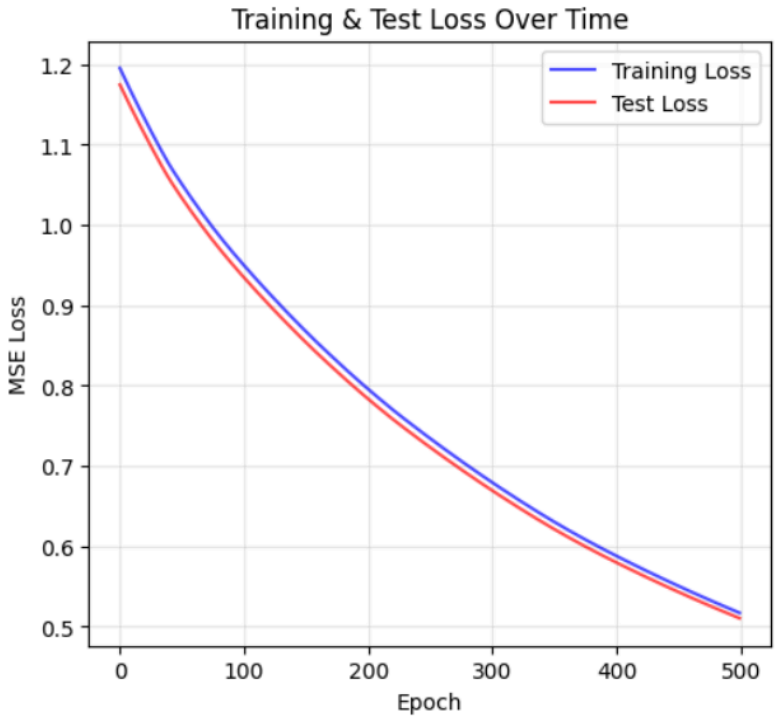
3. Methodology

- Architecture: Input(1) → Hidden Layer 1 (64 units, ReLU) → Hidden Layer 2 (64 units, ReLU) → Output(1)
- Weight Initialization: Xavier (Glorot) initialization for all layers. Biases initialized to zero.
- Activation Function: ReLU for hidden layers; linear for output.

- Loss Function: Mean Squared Error (MSE)
- Optimization: Manual implementation of forward pass, backpropagation, and weight updates using batch gradient descent.
- Hyperparameter Settings:
    - Learning Rate: 0.001 (baseline)
    - Batch Size: Full batch
    - Early stopping with patience of 10 epochs based on validation loss
    - Max Epochs: 500
- Experimentation: Multiple runs were conducted with different hyperparameters as per lab instructions, each tracking training/test loss and prediction quality.

4. Results and Analysis

Training Loss Curve

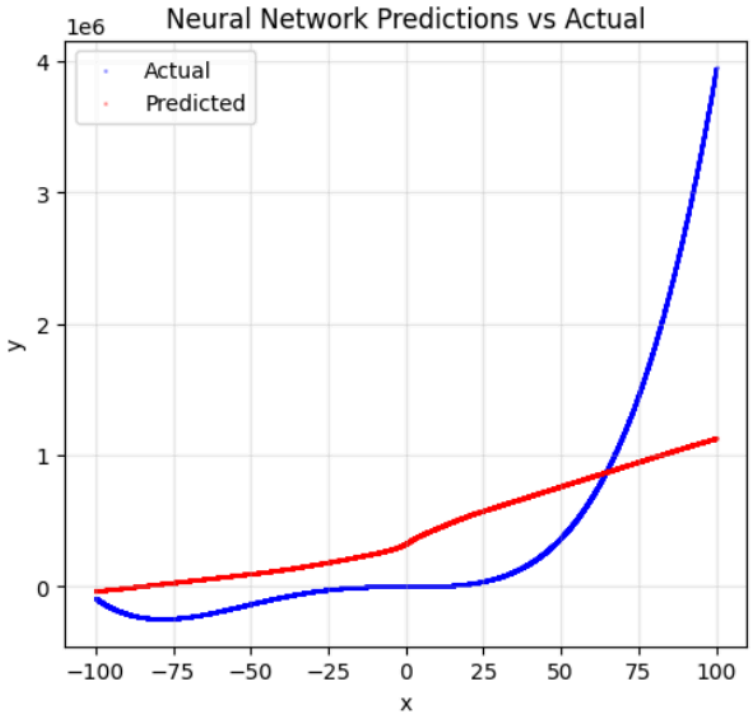


- As we went through more epochs, both the training and test MSE kept dropping, which means the model was learning and actually converging without overfitting too much.
- Final training loss: 0.5167
- Final test loss: 0.5100

Final Test MSE

- On the standardized data, the model's final test Mean Squared Error (MSE) was about 0.51, showing that it managed to approximate the function pretty well despite some noise.

Plot of Predicted vs Actual Values



- The predicted values mostly followed the actual targets pretty closely, with only some small differences in the noisy parts, but nothing crazy — it shows the model is pretty good at generalizing.

Performance Discussion

- Overfitting/Underfitting: Since both the training and test losses went down steadily and levelled off at similar levels, there's no clear sign that the model overfitted (training error much lower than test error) or underfitted (both errors staying high).
- Learning Curve: The loss curves for both training and testing data looked smooth and steady, with no sudden jumps or weird patterns.
- Generalization: The R^2 score of 0.4826 suggests the model does a decent job of capturing the overall trend in the data, especially considering the noise involved, and it's fairly reliable for approximating the polynomial system.

Results table :

Experiment	Learning Rate	No. of epochs	Optimizer	Activation function	Final Training loss	Final test loss	R^2 Score
Baseline	0.001	500	Gradient Descent	ReLU	0.516684	0.509968	0.4826
LR = 0.005	0.005	500	Gradient Descent	ReLU	0.149539	0.148916	0.8489
Epochs = 1000	0.005	1000	Gradient Descent	ReLU	0.065822	0.065513	0.9335
Epochs = 100	0.005	100	Gradient Descent	ReLU	0.519248	0.509947	0.4826
tanh	0.001	200	Gradient Descent	tanh	0.796566	0.784310	0.2043