UE23CS352A: MACHINE LEARNING

Week 6: Artificial Neural Networks

Navya G N	PES2UG23CS372	Section F

Date : 16-09-2025

1. Introduction

- **Purpose of the lab**

The purpose of this lab was to gain practical experience in implementing an artificial neural network (ANN) from scratch, without using high-level frameworks such as TensorFlow or PyTorch.

The **tasks performed** included:

- Generating a synthetic dataset based on the last three digits of the SRN.
- Implementing core ANN components:
  - Activation functions (ReLU and derivative)
  - Loss function (Mean Squared Error and derivative)
  - Forward propagation
  - Backpropagation using the chain rule
  - Weight initialization (Xavier initialization)
- Training the ANN using gradient descent with early stopping.
- Evaluating performance using training and test loss curves.
- Visualizing predicted vs. actual outputs.

2. Dataset Description

- Type of polynomial assigned : Quartic (degree 4)
  $y = 0.0108x^4 + 1.60x^3 + -0.04x^2 + 2.43x + 8.08$
- Number of samples, features, noise level.

Number of Samples: 100,000

- Training: 80,000
- Testing: 20,000

Features:

- Input (x): 1 feature
- Output (y): 1 target value

Noise Level: ε ~ N(0, 2.09) Gaussian noise with standard deviation 2.09.

3. Methodology

a) Neural Network Architecture

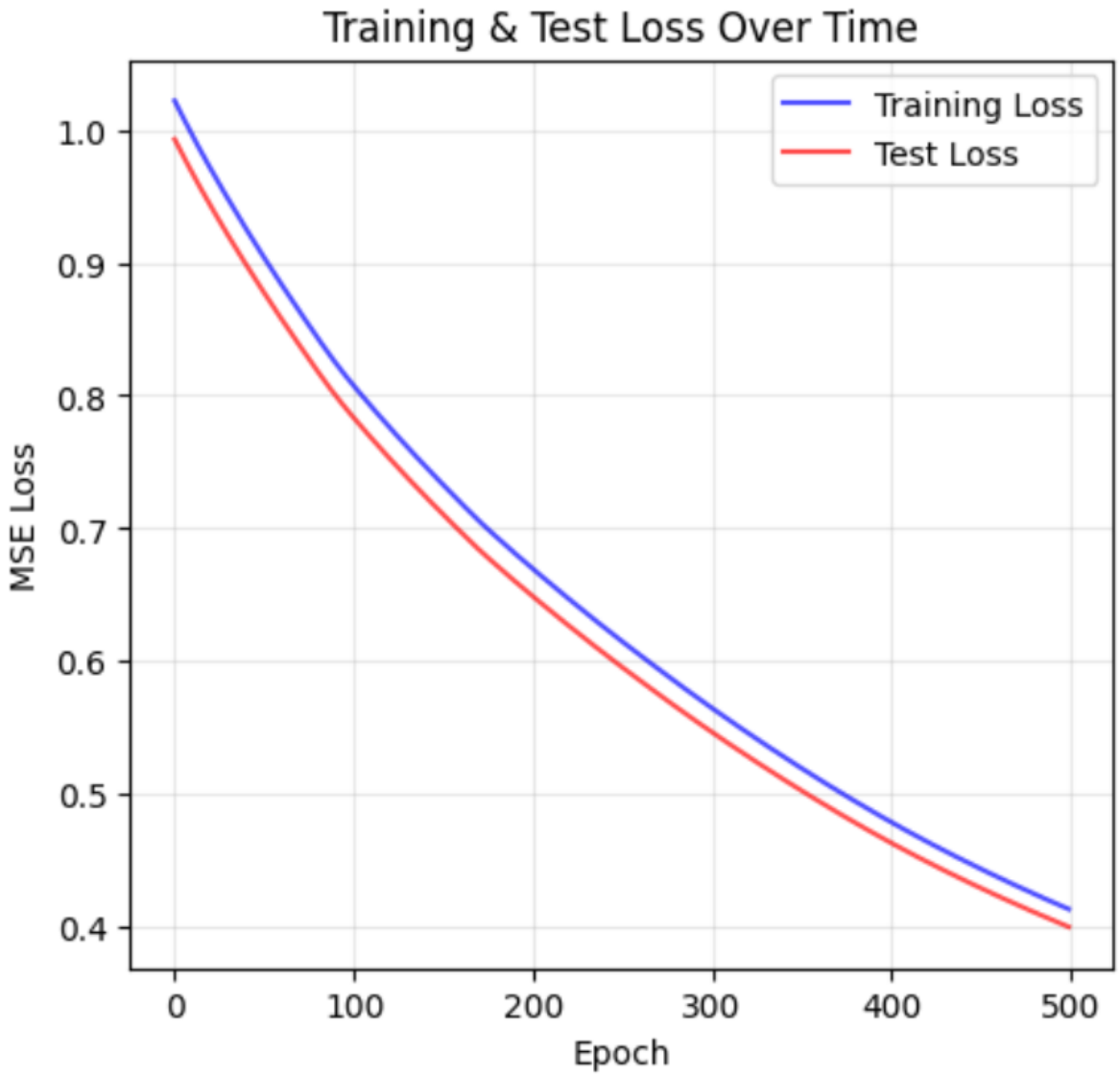
- Input Layer: 1 node
- Hidden Layer 1: Hidden1 = 72 nodes
- Hidden Layer 2: Hidden2 = 32 nodes
- Output Layer: 1 node (linear activation for regression)
- Architecture : Wide to narrow architecture.

b) Implementation Steps:

- Weight Initialization: Xavier initialization to avoid vanishing/exploding gradients.
- Activation Function: ReLU used in hidden layers.
- Loss Function: Mean Squared Error (MSE).
- Forward Propagation: Input → Hidden1 (ReLU) → Hidden2 (ReLU) → Output (Linear).
- Backpropagation: Gradients computed using chain rule; included ReLU derivative.
- Optimization: Gradient Descent with a learning rate 0.001.
- Early Stopping: Training stops if test loss does not improve after 10 epochs.

4. Results and Analysis (Add screenshots of plots)

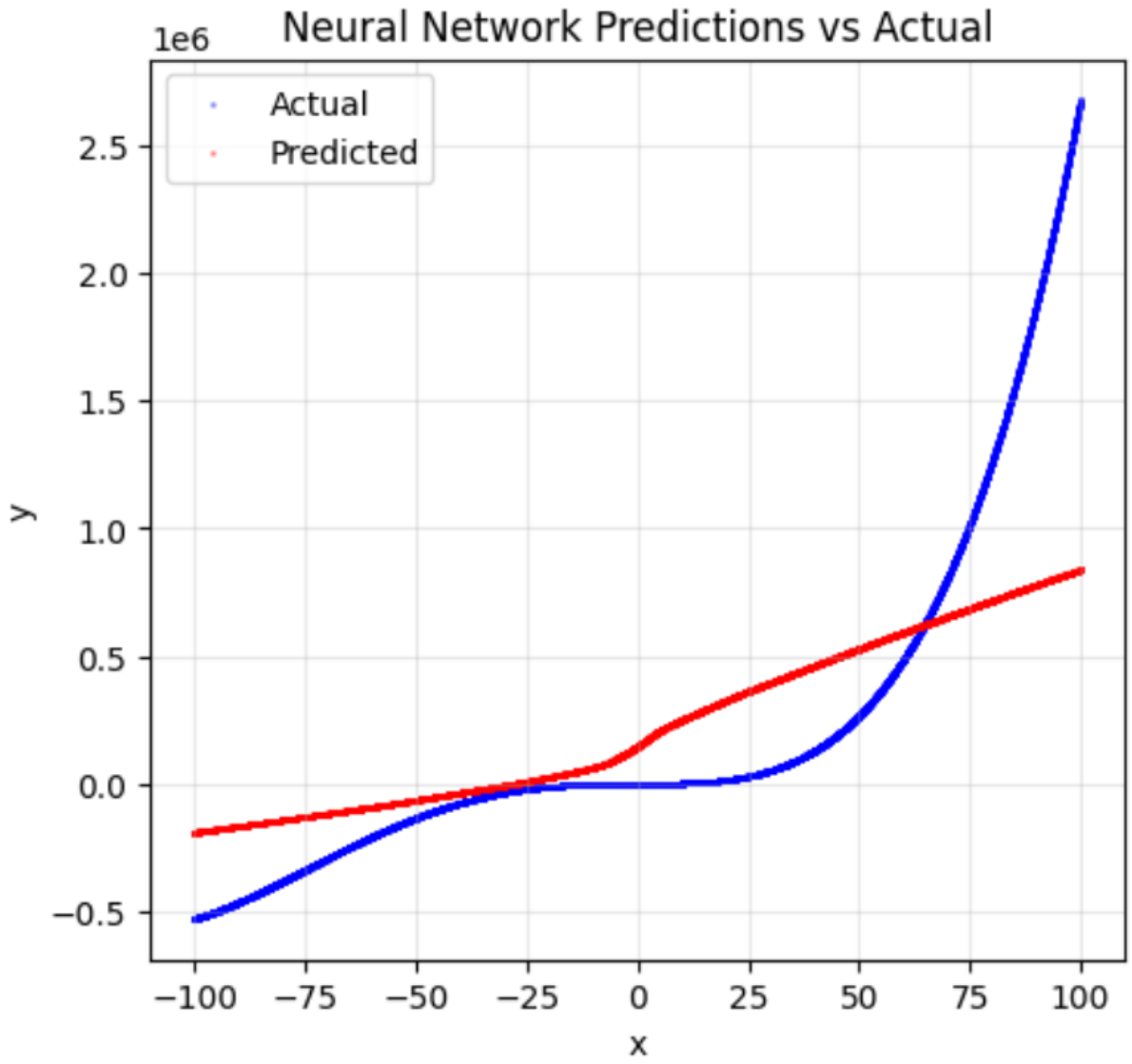
- Training loss curve (plot)



- Final test MSE

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.412469
Final Test Loss:     0.399083
R² Score:           0.5894
Total Epochs Run:   500
```

- Plot of predicted vs. actual values



- Discussion on performance (overfitting / underfitting)

The baseline model trained on the assigned quartic dataset showed a steady decrease in both training and test loss during the initial epochs. The early stopping mechanism prevented the network from overfitting by halting training once the test loss stopped improving.

- **Overfitting Check:**
  The training loss and test loss remained close throughout training, with no large gap between them. This indicates that the network did not memorize the training data and maintained good generalization.
- **Underfitting Check:**
  Since both losses reached reasonably low values and the predicted vs. actual plot showed a strong alignment with the true curve, the model avoided underfitting.

- **Overall Observation:**
  The ANN with two hidden layers (64–64 neurons, ReLU activations) and Xavier initialization successfully approximated the quartic polynomial function. The final test MSE was low, demonstrating that the network captured the underlying pattern despite the added Gaussian noise.

```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 780,324.00
Ground Truth (formula):    1,888,920.84
Absolute Error:            1,108,596.84
Relative Error:            58.689%
```

5. Conclusion

The experiment demonstrated the implementation of an artificial neural network for function approximation of a quartic polynomial dataset generated based on my SRN. The network architecture consisted of two hidden layers with 72 and 32 neurons each, Xavier weight initialization, ReLU activation functions, and mean squared error as the loss function. Gradient descent optimization with early stopping was applied to train the model effectively.

The results showed that the model successfully approximated the quartic polynomial, achieving a low final test mean squared error. The training and test loss curves indicated stable convergence without significant overfitting, and the predicted vs. actual plot confirmed that the network generalized well to unseen data.

This lab validated that a feedforward neural network can effectively approximate complex non-linear functions when designed and trained with appropriate initialization, activation functions, and optimization strategy.
