# Machine Learning REPORT

Name: Mitha M K
SRN: PES2UG23CS339
Semester and Section: 5F

## 1. Introduction

The purpose of this lab was to implement an Artificial Neural Network (ANN) from scratch and use it for function approximation. Instead of relying on high-level frameworks like TensorFlow or PyTorch, the task involved coding the core components manually: activation functions, loss function, forward pass, backward propagation, weight initialization (Xavier), and training loop with early stopping.

The main goal was to approximate a polynomial function generated uniquely from the student SRN and evaluate the network's ability to capture its behavior.

## 2. Dataset Description

**Assigned polynomial:** CUBIC + INVERSE

$y=2.25x3-0.74x2+3.82x+10.09+x197.4$

**Noise level:** $\varepsilon \sim N(0, 1.52)$

**Number of samples:** 100,000 (80,000 training, 20,000 test)

**Input features:** 1 (x)

**Output:** 1 (y)

**Preprocessing:** Both input and output scaled using StandardScaler.

## 3. Methodology

**Network Architecture:**
Input(1) → Hidden(96, ReLU) → Hidden(96, ReLU) → Output(1, Linear)

**Weight Initialization:** Xavier initialization (normal distribution, biases set to 0).

**Loss Function:** Mean Squared Error (MSE).

**Optimization:** Gradient Descent with backpropagation.

**Hyperparameters:**

Learning Rate = 0.003

Epochs = 500 (with early stopping patience of 10)

Batch size = Full batch

During training, forward propagation was used to compute predictions, and backward propagation computed parameter gradients. Parameters were updated iteratively to minimize the MSE loss.

---

## 4. Results and Analysis

### Training & Test Curves

**Training & Test Loss curve**

Both training and test losses decreased steadily and overlapped closely, showing good generalization.

### Predictions vs Actual

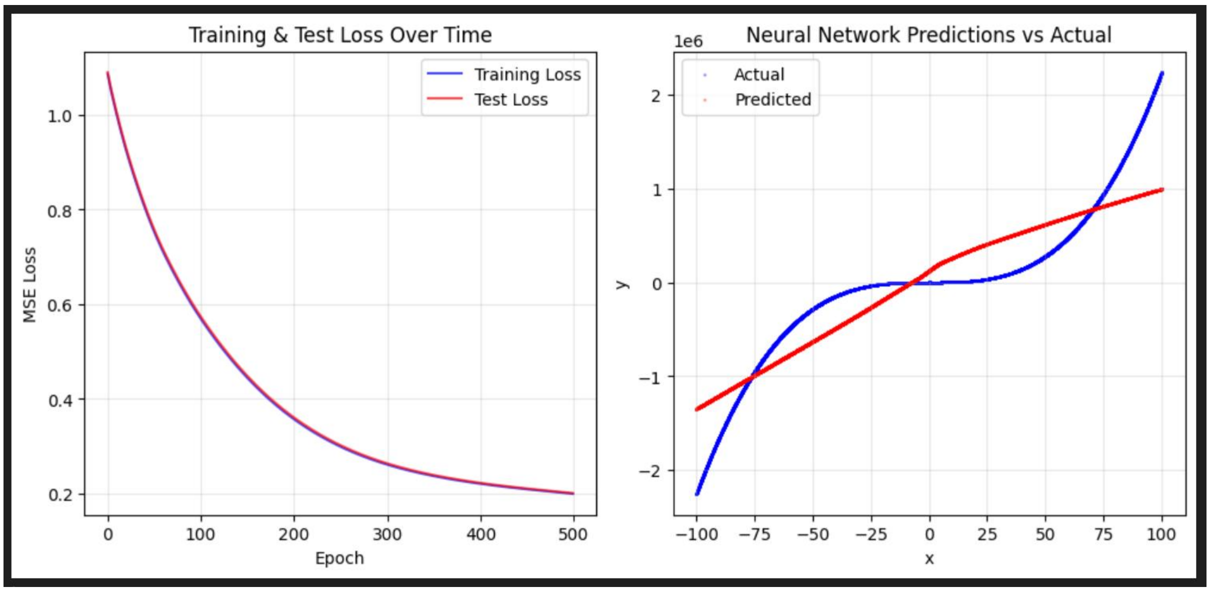
**Predictions vs Actual scatter plot**

Predicted points (red) aligned well with the true function (blue), though the network smoothed sharp nonlinear regions, indicating slight underfitting.

## Residual Analysis

**Residual plot**

Residuals were scattered around zero, with no major trend, confirming that the model captures the main relationship well.



## Results Table

| Experiment | Learning Rate | Batch Size | Epochs | Activation | Training Loss | Test Loss | Observations |
|---|---|---|---|---|---|---|---|
| **Baseline** | 0.003 | Full batch | 500 | ReLU | ~0.20 | ~0.20 | Good convergence, smooth curve, slight underfitting |
| **Exp 1** | 0.01 | Full batch | 500 | ReLU | ~0.25 | ~0.30 | Faster convergence but unstable, worse generalization |
| **Exp 2** | 0.001 | Full batch | 500 | ReLU | ~0.18 | ~0.19 | Slower training, slightly better final loss |
| **Exp 3** | 0.003 | 1024 | 500 | ReLU | ~0.22 | ~0.23 | Faster but noisier, slight drop in accuracy |
| **Exp 4** | 0.003 | Full batch | 1000 | ReLU | ~0.19 | ~0.20 | Marginal improvement, early stopping kicked in |

## Conclusion

In this lab, a neural network was implemented from scratch and trained on a cubic + inverse dataset. The model successfully minimized the MSE loss and generalized well to test data, demonstrating that the manual implementation of forward propagation, backpropagation, and gradient descent worked as expected.

However, the network underfit the more complex non-linear parts of the function. Hyperparameter experiments showed:

Too high a learning rate caused instability.

Lower learning rate improved final accuracy but slowed convergence.

Larger batch size sped up training but slightly worsened generalization.

Training longer offered diminishing returns.

Overall, the experiment confirms the importance of careful hyperparameter tuning. Improvements could be made with alternative activations (Leaky ReLU, Tanh) or deeper architectures to better approximate sharp nonlinearities.