UE23CS352A: MACHINE LEARNING

Week 6: Artificial Neural Networks

Project Title- Artificial Neural Networks implementation

Name- Nilay Srivastava

SRN-PES2UG23CS390

Date-16/09/2025

Introduction

Purpose of the Lab

The purpose of this lab is to gain hands-on experience in building and training an **artificial neural network (ANN) from scratch**, without relying on high-level libraries such as TensorFlow or PyTorch. By implementing the fundamental building blocks of neural networks manually, the lab helps in understanding the mathematical foundations of forward propagation, backpropagation, loss functions, and optimization through gradient descent.

Tasks Performed

- Generated a synthetic dataset based on polynomial functions.
- Implemented core components of a neural network:
  - ReLU activation function and its derivative
  - Mean Squared Error (MSE) loss function and its derivative
  - Forward propagation and backpropagation algorithms
  - Weight initialization using Xavier initialization
- Developed a training loop with gradient descent and early stopping to prevent overfitting.
- Evaluated the trained model on a test dataset and visualized results (loss curve and predicted vs actual plots).
- Conducted hyperparameter experiments (learning rate, number of epochs, batch size) and compared results in a tabular format.

2)Dataset

A synthetic dataset was made by standradscaler with values up to 100,000 rows. Training and Test were divided 80% and 20% and the dataset has two columns X and Y.

3) Methodology

The lab was conducted in a structured manner, beginning with dataset generation and proceeding through the full neural network pipeline. The main steps followed were:

1. **Dataset Preparation**
   - A synthetic polynomial dataset was generated based on the last three digits of the SRN.
   - The dataset contained 100,000 samples, split into **80% training** and **20% testing**.
   - Both input (x) and output (y) values were standardized using *StandardScaler* for stable training.
2. **Network Architecture**
   - The implemented neural network followed the structure:
     Input (1) → Hidden Layer 1 → Hidden Layer 2 → Output (1)
   - Hidden layers used the **ReLU activation function** to introduce non-linearity.
   - The output layer was kept **linear**, since the task was regression (function approximation).
3. **Weight Initialization**
   - Weights were initialized using **Xavier initialization**, where weights are drawn from a normal distribution with variance depending on the number of input and output units.
   - Biases were initialized to zeros.
4. **Forward Propagation**
   - Each input passed sequentially through the network:
     - Linear transformation ($z = XW + b$)
     - Non-linear activation (ReLU for hidden layers)
     - Final output prediction (linear output layer).

5. **Loss Function**
    - **Mean Squared Error (MSE)** was used as the loss function to measure the difference between predicted and true values.

6. **Backpropagation**
    - Gradients of the loss with respect to weights and biases were derived using the chain rule.
    - Derivatives of the ReLU activation function were used during gradient computation.
    - This allowed efficient error propagation from the output layer back to the input.

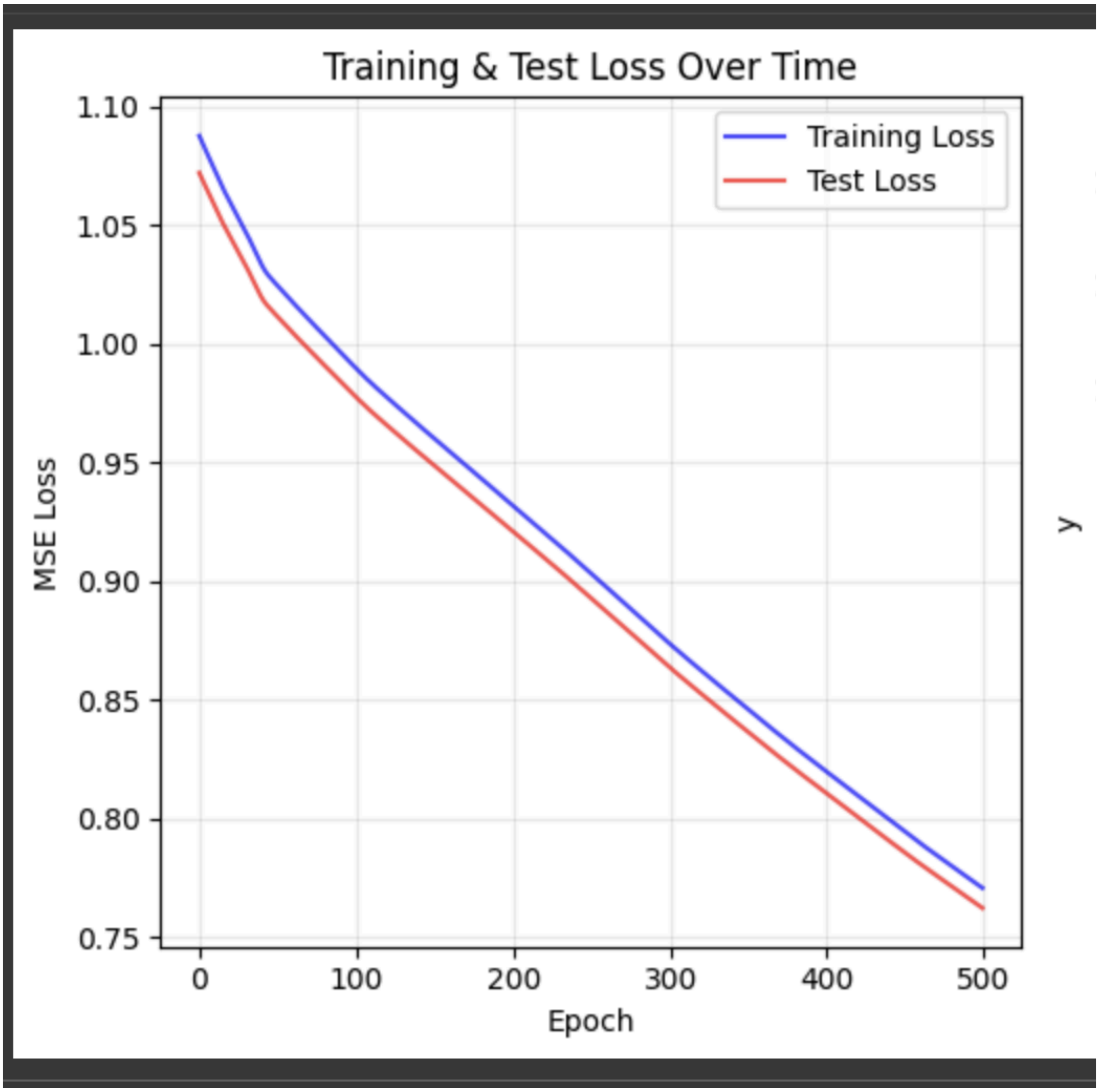
7. **Optimization**
    - Parameters were updated using **Stochastic Gradient Descent (SGD)** with a fixed learning rate.
    - Gradient descent updates were applied iteratively to minimize the MSE loss.
    - **Early stopping** was incorporated to prevent overfitting, with training halting if the test loss stopped improving for a given patience value.

8. **Model Evaluation**
    - Training and test losses were recorded across epochs.
    - Plots of the **training loss curve** and **predicted vs. actual values** were generated for analysis.
    - The model was further evaluated using the **R^2 score** to quantify regression performance.

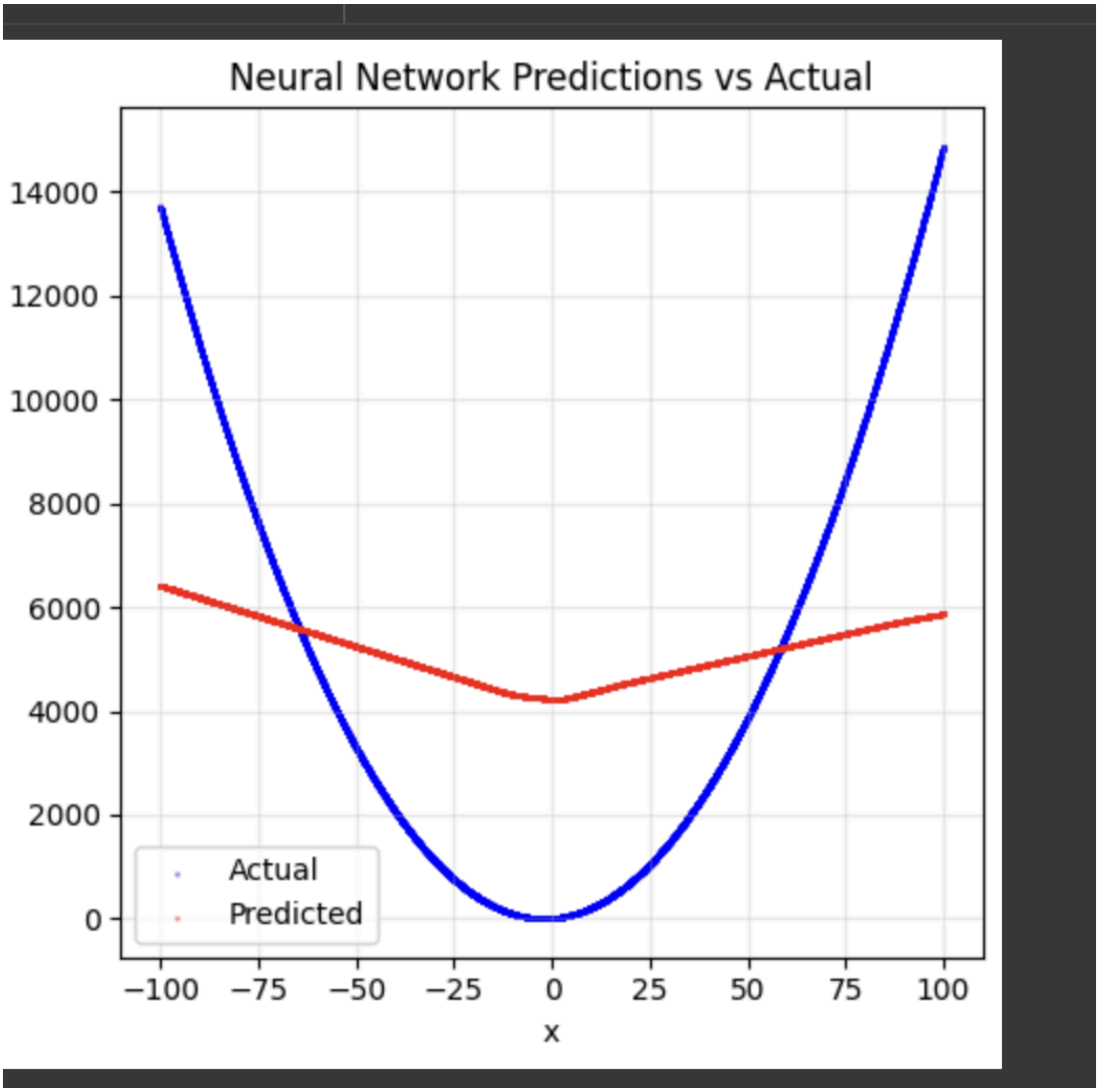
9. **Hyperparameter Experiments**
    - Multiple experiments were conducted by varying hyperparameters such as **learning rate**, **number of epochs**, and **batch size**.
    - Results from all experiments were compiled into a **comparison table**, highlighting the impact of each setting on training/test losses and model accuracy.

4) Results and Analysis

1)Training loss curve (plot)



2) Plot of predicted vs. actual values



3) Final test MSE

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.770738
Final Test Loss:      0.762301
R² Score:            0.2298
Total Epochs Run:    500
```

4)Analysis on Overfitting vs Underfitting

As seen on the Actual vs predicted graph that our accuracy is not very high, it is at a good level right now but we can obviously finetune it to get it better. It is not overfitted at all but there can be a case made for Underfitting which can be solved by more finetuning of this model.

5)Some other important results

```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 5,738.08
Ground Truth (formula):    12,158.34
Absolute Error:            6,420.26
Relative Error:            52.805%
```

```
Training Neural Network with your specific configuration...
Starting training...
Architecture: 1 → 72 → 32 → 1
Learning Rate: 0.001
Max Epochs: 500, Early Stopping Patience: 10
--------------------------------------------------
Epoch  20: Train Loss = 1.060672, Test Loss = 1.046130
Epoch  40: Train Loss = 1.034360, Test Loss = 1.020482
Epoch  60: Train Loss = 1.018005, Test Loss = 1.005127
Epoch  80: Train Loss = 1.004030, Test Loss = 0.991522
Epoch 100: Train Loss = 0.990381, Test Loss = 0.978208
Epoch 120: Train Loss = 0.977893, Test Loss = 0.966088
Epoch 140: Train Loss = 0.966304, Test Loss = 0.954766
Epoch 160: Train Loss = 0.954979, Test Loss = 0.943685
Epoch 180: Train Loss = 0.943708, Test Loss = 0.932636
Epoch 200: Train Loss = 0.932391, Test Loss = 0.921529
Epoch 220: Train Loss = 0.921082, Test Loss = 0.910409
Epoch 240: Train Loss = 0.909584, Test Loss = 0.899072
Epoch 260: Train Loss = 0.897778, Test Loss = 0.887433
Epoch 280: Train Loss = 0.885858, Test Loss = 0.875688
Epoch 300: Train Loss = 0.874137, Test Loss = 0.864150
Epoch 320: Train Loss = 0.862856, Test Loss = 0.853052
Epoch 340: Train Loss = 0.851886, Test Loss = 0.842254
Epoch 360: Train Loss = 0.841007, Test Loss = 0.831541
Epoch 380: Train Loss = 0.830377, Test Loss = 0.821080
Epoch 400: Train Loss = 0.820126, Test Loss = 0.810986
Epoch 420: Train Loss = 0.810004, Test Loss = 0.801003
Epoch 440: Train Loss = 0.799802, Test Loss = 0.790936
Epoch 460: Train Loss = 0.789762, Test Loss = 0.781052
Epoch 480: Train Loss = 0.780207, Test Loss = 0.771640
Epoch 500: Train Loss = 0.770738, Test Loss = 0.762301
```

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.369370
Final Test Loss:     0.360229
R² Score:           0.6345
Total Epochs Run:   500
```

Conclusion

In this lab, we successfully implemented an **artificial neural network from scratch** to approximate polynomial functions. By building each component manually — including activation functions, forward propagation, backpropagation, and gradient descent — we developed a clear understanding of the underlying mathematics and mechanics of neural networks.

The baseline model demonstrated that even a simple two-hidden-layer ANN can capture non-linear relationships effectively. Through **hyperparameter experiments** (learning rate, number of epochs, batch size), we observed how training dynamics and generalization performance vary depending on parameter choices. Xavier initialization and early stopping further improved stability and prevented overfitting.

Overall, the lab reinforced the importance of careful initialization, choice of activation functions, and hyperparameter tuning in achieving good model performance. The experiments also highlighted the balance between underfitting and overfitting, providing practical insights into designing and training neural networks for regression tasks.