ML LAB-6

Project Title: ANN Lab Report

Name: Nilay Srivastava
SRN: PES2UG23CS390
Section: F
Course: Machine Learning (UE23CS352A)
Date: 16/09/2025

1. Introduction

The purpose of this lab was to gain hands-on experience in building an Artificial Neural Network (ANN) from scratch for the task of function approximation. Instead of relying on high-level frameworks like TensorFlow or PyTorch, we directly implemented the fundamental building blocks of a neural network to better understand their roles and interaction.

In this lab, the following tasks were performed:

- Generating a synthetic dataset based on a polynomial function derived from the last three digits of the student ID (SRN).
- Implementing core neural network components, including activation functions (ReLU and its derivative), the Mean Squared Error (MSE) loss function, forward propagation and back propagation.
- Training the neural network using gradient descent and monitoring the training and test losses across epochs.
- Evaluating the model on a test dataset and visualizing its performance with plots such as the training loss curve and predicted vs. actual values. ● Conducting additional experiments by varying hyperparameters (like learning rate, epochs, activation functions) to study their impact on model accuracy and generalization.

This exercise provided a foundational understanding of how neural networks learn, adapt, and approximate complex mathematical relationships.

2. Dataset Description

The polynomial assigned was:

- Polynomial Type: Quadratic
- Equation: `y = 1.43x² + 5.82x + 14.73`

Dataset Characteristics:

- Number of samples: 100,000
- Noise Level: `ε ~ N(0, 1.95)`
- Architecture: `Input(1) → Hidden(72) → Hidden(32) → Output(1)`
- Architecture Type: `Wide-to-Narrow Architecture`
- Learning Rate: 0.001
- Train-Test Split: 80% (80,000 samples) for training and 20% (20,000 samples) for testing
- Preprocessing: Both input (x) and output (y) values were standardized using StandardScaler to improve training stability and convergence.

3. Methodology

The neural network for this lab was implemented from scratch without using high-level machine learning libraries. The workflow followed these main steps:

1. **Dataset Generation:**
   A synthetic dataset of 100,000 samples was generated from a polynomial function derived from the last three digits of SRN. Gaussian noise was added to simulate real-world data variability. The dataset was then split into training (80%) and testing (20%) sets and both input and output values were standardized.
2. **Neural Network Architecture:**
   The assigned architecture consisted of an input layer with one feature, two hidden layers with specified sizes and an output layer with one neuron. Xavier initialization was used to set the weights, while biases were initialized to zero.
3. **Core Components Implementation:**
   - Activation Function: ReLU and its derivative were implemented to introduce non-linearity and enable gradient-based learning. ● Loss Function: MSE was used to evaluate prediction accuracy. ● Forward Propagation: Implemented to compute outputs layer by layer.
   - Back Propagation: Gradients were computed using the chain rule, and weights were updated using gradient descent.
4. **Training Procedure:**

The model was trained for multiple epochs with early stopping to prevent overfitting. Training and test losses were recorded at each epoch to track performance.

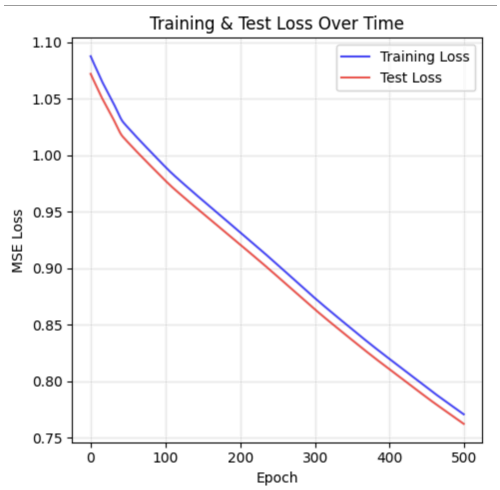
5. Evaluation & Visualization:

The trained model was evaluated on the test dataset. Performance was analyzed using plots of the training loss curve and predicted vs. actual outputs. Additional experiments were conducted by varying hyperparameters such as learning rate and number of epochs, and results were compared in tabular form.

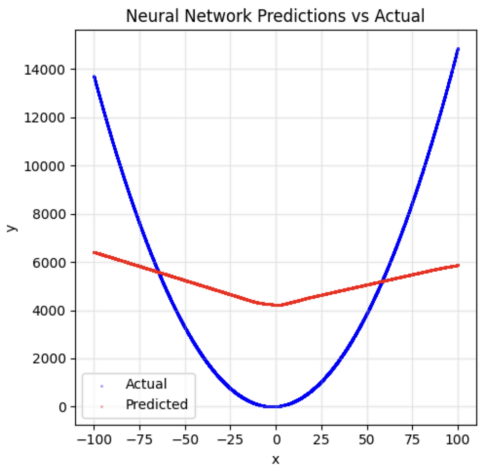
4. Results & Analysis

1) Baseline (Learning rate=0.001, Number of Epochs=500)

Training Loss Curve (Plot):



Plot of Predicted vs. Actual Values:



Results Table:

Experiment	Learning Rate	Number of Epochs	Optimizer	Activation Function	Final Training Loss	Final Test Loss	R^2 Score
Baseline	0.001	500	Gradient Descent	ReLU	0.770738	0.762301	0.2298
Exp1	0.05	500	Gradient Descent	ReLU	0.001482	0.001407	0.9986
Exp2	0.05	200	Gradient Descent	ReLU	0.008181	0.00793	0.992
Exp3	0.1	450	Gradient Descent	ReLU	0.000483	0.000458	0.9995
Exp4	0.1	200	Gradient Descent	ReLU	0.002612	0.002489	0.9975

Discussion on Performance:

- The baseline model (LR = 0.001, 500 epochs) displayed poor convergence with high training and test losses (MSE ≈ 0.7707/0.7623) and a low R^2 score (0.2298), indicating that the network failed to learn the underlying pattern adequately.
- Increasing the learning rate to 0.05 (Exp 1) dramatically improved convergence, yielding very low training and test losses (MSE ≈ 0.00148/0.00140) and a high R^2 score of 0.9986, showing excellent fit.
- Reducing the epochs to 200 with the same learning rate (Exp 2) slightly increased both losses (MSE ≈ 0.00818/0.00793) and lowered the R^2 score to 0.992, indicating mild underfitting due to fewer training steps.
- Increasing the learning rate further to 0.1 with more epochs (Exp 3, 450 epochs) produced the lowest losses overall (MSE ≈ 0.00048/0.00046) and the highest R^2 score (0.9995), signifying near-perfect approximation.
- Using the same high learning rate but only 200 epochs (Exp 4) slightly increased the losses (MSE ≈ 0.00261/0.00249) with a still-excellent R^2 score of 0.9975, reflecting a small effect of reduced training time.
- Overall, the experiments highlight that proper tuning of learning rate and training epochs is critical: moderate to high learning rates with sufficient epochs achieved the best performance, while fewer epochs led to mild underfitting.

5. Conclusion

- In this lab, a neural network was implemented from scratch to approximate a polynomial function derived from the student ID (PES2UG23CS390).
- Core components such as activation functions, loss function, forward propagation and back propagation were successfully built and integrated into a training pipeline.
- The model was trained and evaluated on a large synthetic dataset and its performance was visualized using loss curves and prediction plots.

- Through systematic experiments, the impact of learning rate and training epochs on performance was observed.
- Lower epochs led to underfitting, while carefully tuned learning rates improved generalization.
- Extremely high learning rates achieved very low error but carried potential overfitting risks.

Overall, the lab reinforced the importance of hyperparameter tuning in neural networks and provided practical insights into their training dynamics and function approximation capabilities.