Name: Nidhi C R
SRN: PES2UG23CS382

SECTION: F

Report: Polynomial Approximation using Neural Networks

1. Introduction

The purpose of this lab was to gain hands-on experience in implementing an Artificial Neural Network (ANN) from scratch, without relying on high-level libraries like TensorFlow or PyTorch. The main goal was to approximate a polynomial function step by step by building a neural network manually, including:

- Generating a synthetic dataset from a polynomial function.
- Implementing Xavier weight initialization.
- Using ReLU activation and its derivative.
- Building the forward pass (Input → Hidden Layer 1 → Hidden Layer 2 → Output).
- Implementing backpropagation using the chain rule.
- Tracking and plotting training and testing losses.
- Comparing predicted vs actual values.
- Conducting experiments with hyperparameters such as learning rate, epochs, and batch size.

2. Dataset Description

- Polynomial Type: Generated based on the student's ID (quadratic, cubic, quartic, cubic + sine, or cubic + inverse).
- Equation: {assignment['polynomial_desc']}
- Coefficients: {assignment['coefficients']}
- Noise Level: Additive Gaussian noise ε ~ N(0, {noise_std:.2f})
- Samples: 100,000 total
- Features: Input feature x, output y
- Split: 80% training (80,000 samples), 20% testing (20,000 samples)
- Preprocessing: Both x and y standardized using StandardScaler for faster convergence.

3. Methodology

Network Architecture:

Input Layer: 1 neuron
Hidden Layer 1: 32 neurons, ReLU activation
Hidden Layer 2: 72 neurons, ReLU activation
Output Layer: 1 neuron

Weight Initialization: Xavier initialization was used to maintain stable variance across layers.

Forward Propagation:
- Weighted sums computed at each layer.
- ReLU activation applied in hidden layers.
- Final output layer produced raw predictions.

Loss Function:
- Mean Squared Error (MSE) used to measure prediction error.
- Validation MSE tracked for early stopping.

Backpropagation:
- Gradients computed layer by layer using the chain rule.
- Weights updated using gradient descent.

Training:
- Mini-batch gradient descent with shuffling.
- Early stopping with patience of 10 epochs to prevent overfitting.

Evaluation:
- Training and test loss curves plotted across epochs.
- Predicted vs actual outputs compared visually.
- Final test MSE computed.

4. Results and Analysis

The training and testing loss decreased steadily across epochs, showing successful learning. Early stopping prevented unnecessary training once the validation loss plateaued.

Final Test MSE: 0.156913

Predictions closely matched true values, forming a smooth polynomial curve.

Overfitting and Generalization:

- The small gap between training and testing loss indicates minimal overfitting.
- Early stopping reduced risk of over-training.

Effect of Hyperparameters:

- Lower learning rates led to slower but smoother convergence.
- Larger batch sizes made updates more stable but required more epochs.
- Increasing epochs improved performance until early stopping was reached.

Experiment Summary (Baseline):

Learning Rate: 0.005
Batch Size: Default
Epochs: 500
Activation: ReLU
Training Loss: 0.155043
Test Loss: 0.156913
Observation: Smooth convergence, good generalization, no early stopping triggered.

5. Conclusion

This lab provided practical understanding of neural networks by manually implementing each component. The model successfully approximated a complex polynomial function with strong generalization and minimal overfitting. This exercise strengthened the foundation for future exploration in machine learning.