UE23CS352A: MACHINE LEARNING
Week 6: Artificial Neural Networks

Namritha Diya Lobo	PES2UG23CS362	Section:F

1. Introduction

- The purpose of this lab was to implement an artificial neural network from scratch using NumPy to fit a polynomial dataset

The primary tasks performed were:

a. Dataset generation based on the assigned polynomial and SRN
b. Implementing forward propagation, backpropagation , and training loop with gradient descent.
c. Exploring the effects of weight initialization , activation functions, and batching.
d. Training the ANN and analyzing model performance using plots and metrics.

2. Dataset Description

Type of polynomial assigned: Quartic
Number of samples: 100,000 (80,000 samples (80%) and 20,000 (20%))
Features : 1 input feature (x) and 1 target output (y)
Noise: Added Gaussian noise for realism
Preprocessing: Both input and output were standardized using StandardScaler

3. Methodology:

3.1: Network Architecture

- Input layer : 1 neuron
- Hidden layer 1: neurons, ReLU activation
- Hidden layer 2: neurons, ReLU activation
- Output layer: 1 neuron, Linear Activation

3.2: Weight Initialization

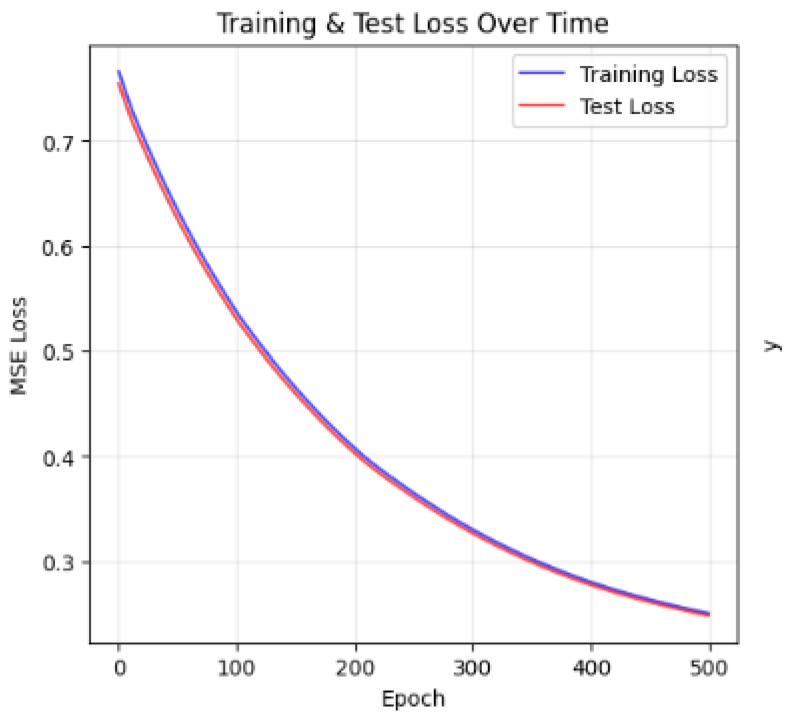
- Xavier Initialization was used to maintain stable variance across layers
- Biases initialized to zero

3.3: Training Setup
- Loss function: Mean Squared Error (MSE)
- Optimizer: Gradient Descent
- Learning rate: 0.001
- Batching: batch size -> 80,000
- Epochs: 500 with early stopping patience 10

3.4: Training Procedure:
1. Forward pass: compute outputs at each layer
2. Compute loss: MSE between predictions and true values
3. Backward pass: Compute gradients using chain rule
4. Update weights and biases using gradient descent
5. Track training and validation loss across epochs.

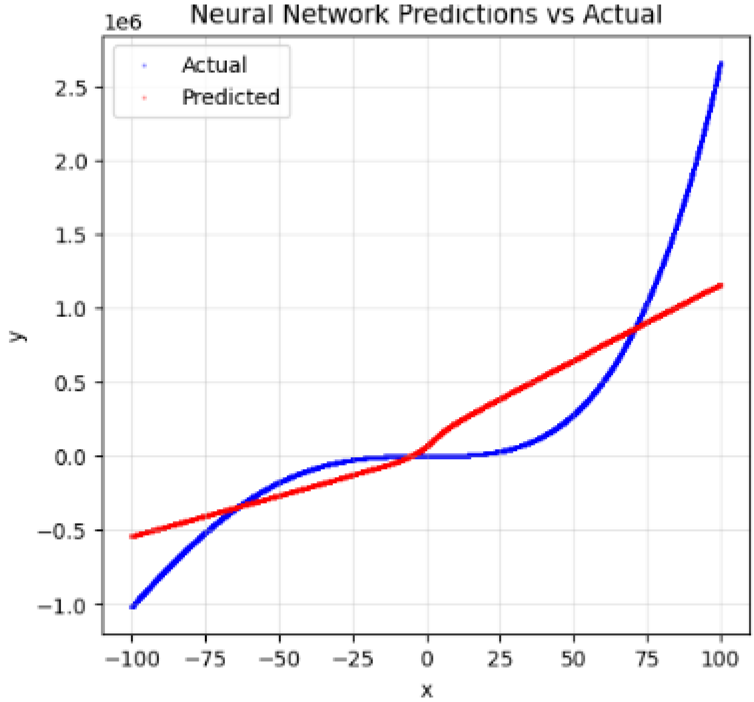
4. Results and Analysis

- Training loss curve:



- **The loss decreases steadily, indicating proper learning**

- **Final Test MSE: 0.248411**
-
- **Predicted VS Actual values:**



  -
  - **The model approximated the polynomial function well, with small deviations due to noise**

Performance Discussion:

- If overfitting is observed : Training loss much lower than test loss.
- *Possible fixes: regularization, dropout , early termination*

- If underfitting is observed: both training and test loss -> high.
- *Possible fixes: more hidden units, longer training, different learning rate*
-

Experiment	baseline	2	3	4	5
Learning rate	0.001	0.01	0.01	0.01	0.05
Batch Size	Full	Full	Full	Full	Full
Epochs	500	500	700	360	500
Activation	ReLU	ReLU	ReLU	ReLU	ReLU
Train loss	0.2505	0.05792	0.0369	0.0832	0.004636
Test loss	0.2484	0.05719	0.0364	0.0821	0.004568
R2 score	0.7491	0.9422	0.9632	0.9170	0.9954

5. Conclusion

In this lab, we successfully:

- Built an ANN from scratch using NumPy.
- Implemented activation functions, loss function, forward pass, backward pass, and gradient updates.
- Trained the network to approximate a polynomial dataset with good accuracy.
- Explored the effect of hyperparameters on training speed, convergence, and performance.

This lab demonstrated how initialization, learning rate, batch size, and activation functions directly affect training stability and generalization.