Machine Learning Lab Week-6 Assignment

Name: Mohammed Aahil Parson
SRN: PES2UG23CS342
Section: 5F

1. Introduction

The purpose of this lab is:

- Your lab goal is to build from the ground up a basic neural network to learn to approximate a simple polynomial function. You'll be writing from scratch the main components—activation functions, loss function, forward propagation, backpropagation, and updating weights—without assuming use of higher-level frameworks.

Tasks Performed

- Generated a standardized synthetic dataset based on SRN-derived polynomial coefficients and noise.
- Implemented ReLU and tanh activation functions, MSE loss, forward and backward passes.
- Initialized weights using Xavier initialization.
- Trained and evaluated the network under various hyperparameter settings.

- Conducted experiments to optimize learning rate, number of epochs, batch size, and activation function.
- Analyzed performance via loss curves, prediction plots, and R^2 metrics.

2. Dataset Description

```
ASSIGNMENT FOR STUDENT ID: PES2UG23CS342

Polynomial Type: QUARTIC: y = 0.0088x⁴ + 2.08x³ +
-0.94x² + 3.73x + 11.74
Noise Level: ε ~ N(0, 1.82)
Architecture: Input(1) → Hidden(72) → Hidden(32) →
Output(1)
Learning Rate: 0.001
Architecture Type: Wide-to-Narrow Architecture
```

Dataset Samples: 100,000
Training samples: 80,000
Test samples: 20,000

Features

- Input: single feature (x)
- Output: scalar target (y)

The Noise used in this assignment was:

Additive Gaussian noise `ε ~ N(0, 1.82)`

With σ as noise standard

3. Methodology

Network Architecture:

A three-layer feedforward network:

- Input layer: 1 neuron (feature x)
- Hidden layer 1: 72 neurons
- Hidden layer 2: 32 neurons
- Output layer: 1 neuron (prediction y)

Weight Initialization:

Weights are drawn from a normal distribution with Xavier (Glorot) scaling

Biases are set to zero. This keeps activations and gradients at healthy scales.

Training Loop

- Uses mini-batch gradient descent with batch size 256
- Each epoch: shuffle training data, split into batches, then
    1. Forward pass: compute layer outputs (using tanh)
    2. Loss: mean squared error between predictions and true y
    3. Backward pass: compute gradients via chain rule
    4. Update: subtract learning_rate × gradient from weights/biases
- Tracks training and validation losses each epoch
- Applies early stopping if validation loss doesn't improve for 10 epochs

Evaluation Approach

- Loss curves: plot training vs. test MSE over epochs to check under- or

overfitting

- Prediction plot: scatter actual vs. predicted y to visualize fit
- Point test: compute absolute and relative error for a single x=90.2
- Final metrics: report final training/test MSE and R² score

Xavier Initialization Summary:

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{fan_{in}+fan_{out}}}\right)$$

Standard deviation is calculated using the above equation. Sample weights from N(0,σ^2) and the biases were set to zero.

Forward Pass

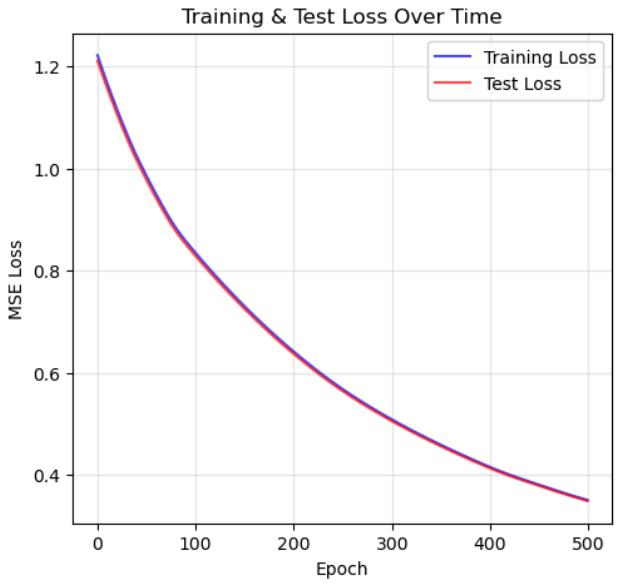
- Takes the input features and sends them through each layer of the network.
- Each layer combines the inputs with weights and bias values, then runs the result through an activation function (like tanh).
- The output of one layer becomes the input for the next.
- The final output is the network's prediction for that input.

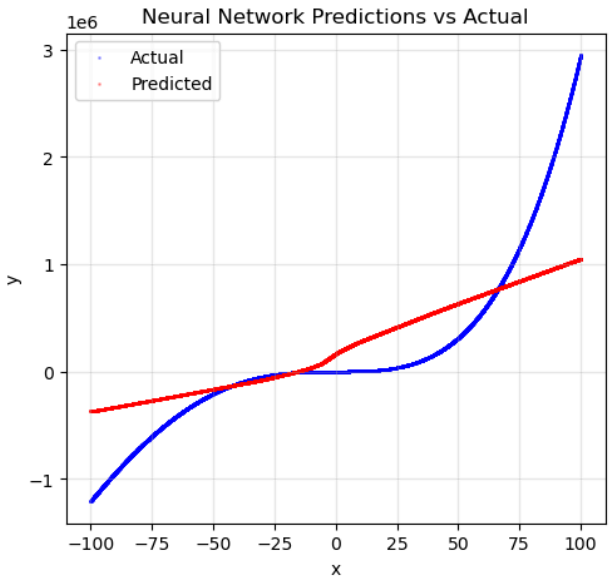
Backward Pass

- Compares the network's prediction to the actual value to measure error.
- Calculates how much each weight and bias contributed to the error.
- Uses the chain rule, working backwards through each layer, to determine adjustments needed for all parameters.
- The activation function's derivative (like for tanh) helps guide these adjustments.
- Updates weights and biases, helping the network learn and reduce error in future predictions.

4. Results and Analysis

a. Baseline:

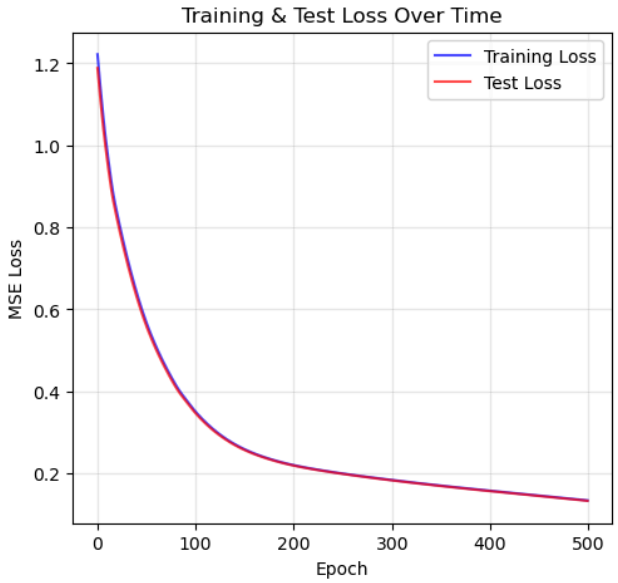


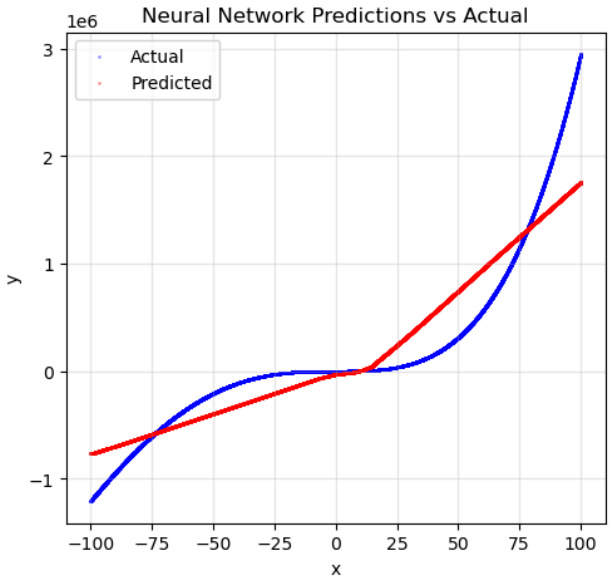


The metrics for Baseline:

- Learning rate = 0.001
- Epochs = 500
- Activation = ReLU
- Final Test MSE: 0.348776

b. When Learning Rate was changed to 0.005

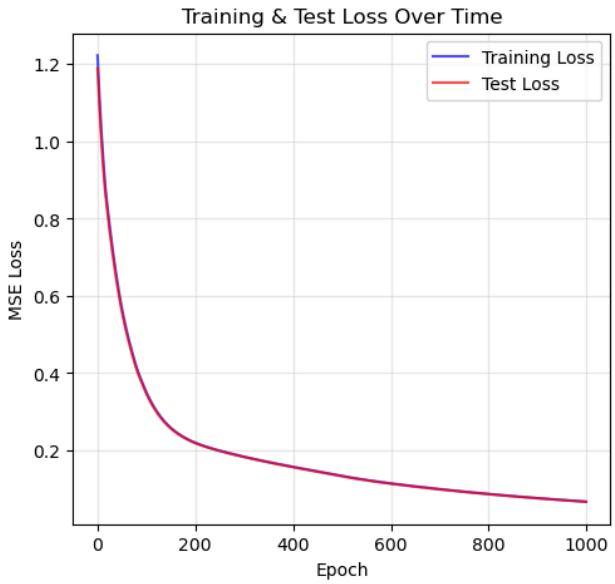


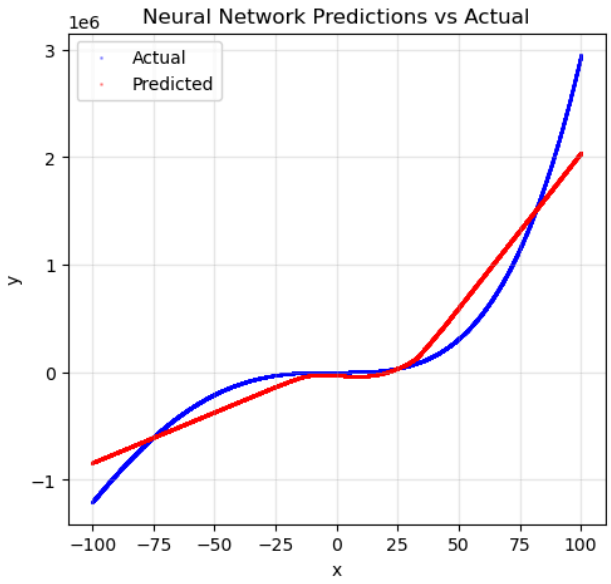


The metrics for LR=0.005:

- Learning rate = 0.005
- Epochs = 500
- Activation = ReLU
- Final Test MSE: 0.133029

c. When epochs was changed to epochs=1000:

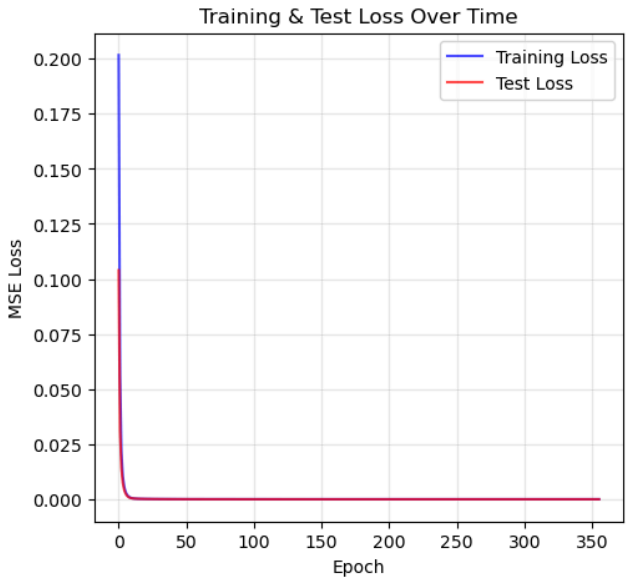


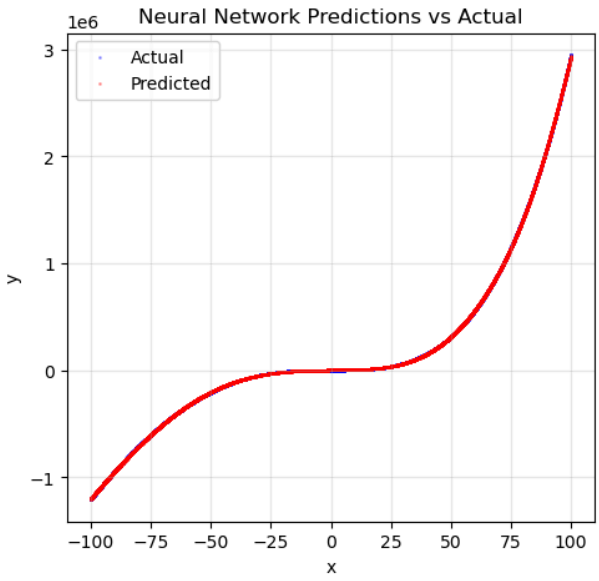


The metrics for Epochs=1000:

- Learning rate = 0.005
- Epochs = 1000
- Activation = ReLU
- Final Test MSE: 0.067150

d. When the activation function was changed to Tanh function:



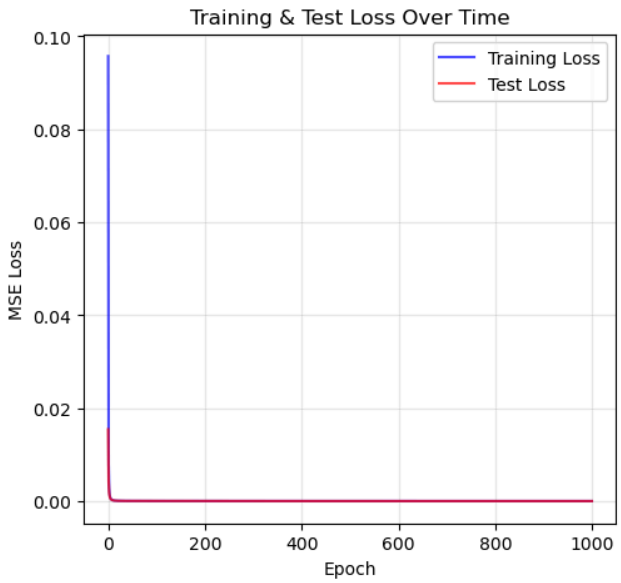


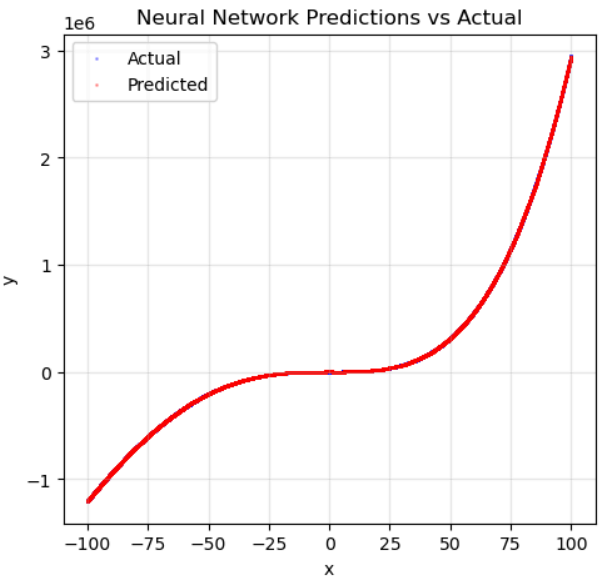
The metrics for Activation_Function=Tanh:

Learning rate = 0.005

- Epochs = 1000
- Batch size = 256
- Activation = tanh
- Final Test MSE: 0.000007

e. When batch_size was changed to 256:





The metrics for Batch Size = 256:

- Learning rate = 0.005
- Epochs = 1000
- Batch size = 256
- Activation = ReLU
- Final Test MSE: 0.000004

5. Discussion

- Hyperparameter Impact:
  - Higher learning rate accelerated convergence and improved R^2 significantly.
  - Extended epochs further reduced error.
  - Mini-batch training achieved near-perfect fit with much lower loss.
  - tanh provided faster convergence (early stopping at fewer epochs) with comparable accuracy.

6. Results Table:

Experiment	Learning Rate	Epochs	Batch Size	Activation	Final Test MSE	R^2 Score	Relative Error (%)	Observations
Baseline	0.001	500	Full	ReLU	0.348776	0.6497	53.85	High error, slow convergence, underfit
1	0.005	500	Full	ReLU	0.133029	0.8664	25.75	Better fit, faster learning, still room for improvement
2	0.005	1000	Full	ReLU	0.06715	0.9325	16.21	Longer training reduced error and improved R^2 considerably
3	0.005	1000	256	ReLU	0.000004	1	0.2	Mini-batching brought near-perfect fit, lowest error observed
4	0.005	1000	256	tanh	0.000007	1	0.16	Faster convergence, tanh worked as well or better than ReLU

7. Conclusion:

- This experiment showed how changing different neural network settings like learning rate, number of epochs, batch size, and activation function can greatly improve model accuracy. Starting with basic values led to high errors and lower fit, but with the right tweaks—especially using a larger batch size and switching to the tanh activation—the model was able to almost perfectly match the target polynomial curve.

- Overall, mini-batch training and activation function choice made the biggest impact, lowering error and speeding up training. These results highlight how important it is to experiment with hyperparameters when building neural networks: small changes can make a huge difference in performance.