Artificial Neural Networks

LAB-6

NAME: NEHA HARISH

SRN: PES2UG23CS378

COURSE: MACHINE LEARNING

DATE: 16-09-25

Introduction

Purpose of the Lab

The purpose of this lab is to gain a practical understanding of how a simple neural network can be implemented from scratch without relying on high-level machine learning libraries. By doing so, the focus is on building foundational knowledge of the key concepts that drive neural networks such as activation functions, forward propagation, backpropagation, gradient descent, and loss minimization. This lab helps in bridging the gap between theoretical concepts and hands-on implementation while reinforcing how neural networks learn patterns from data.

Tasks Performed

- Implemented a feed-forward neural network from scratch using Python.
- Defined and applied activation functions to introduce non-linearity into the model.
- Implemented forward propagation to compute predictions from input data.
- Calculated the loss using Mean Squared Error (MSE) as the evaluation metric.
- Performed backpropagation to compute gradients for weight and bias updates.
- Applied gradient descent optimization to iteratively minimize the loss.
- Trained the model on the given dataset until convergence.
- Evaluated the performance of the network using training and test data.
- Generated plots to visualize training progress and prediction accuracy.

Dataset Description

The dataset used in this lab was synthetically generated based on a non-linear cubic + sinusoidal polynomial function:

$$y=2.27x^3-0.72x^2+3.15x+11.27+5.4\cdot\sin(0.032x)+\epsilon$$

where the noise term $\epsilon \sim N(0,1.52)$ adds randomness drawn from a Gaussian distribution with mean 0 and standard deviation 1.52.

- **Type of Polynomial Assigned:** Cubic polynomial with an additional sinusoidal component.
- **Number of Samples:** 100,000 data points were generated.
- **Training/Test Split:** 80,000 samples for training and 20,000 samples for testing.
- **Number of Features:** The dataset consists of a **single input feature (x)** and a **single output target (y)**.
- **Noise Level:** Gaussian noise with standard deviation 1.52 was added to simulate real-world variability.

This synthetic dataset ensures sufficient complexity for the neural network to learn, while also allowing clear evaluation of how well the model captures both polynomial and periodic patterns.

Methodology

The neural network was implemented completely from scratch in Python, without using high-level deep learning libraries such as TensorFlow or PyTorch. The following key steps and components were used in the implementation:

1. **Weight Initialization**
    - Xavier initialization was applied to all layers in order to ensure stable gradient flow during training.
    - Weights were sampled from a normal distribution with variance depending on the number of input and output connections, while biases were initialized to zero.
2. **Network Architecture**
    - The chosen architecture followed a **Wide-to-Narrow** structure:

Input (1) → Hidden Layer 1 (72 neurons) → Hidden Layer 2 (32 neurons) → Output(1)

Hidden layers used non-linear activation functions to allow the network to capture complex relationships in the data.

3. **Activation Functions**
    - The ReLU (Rectified Linear Unit) activation was applied in hidden layers to introduce non-linearity while keeping gradient computations efficient.
    - The output layer was linear since the task was regression.
4. **Forward Propagation**
    - Input features were passed sequentially through the layers using the current weights and biases.
    - The activations were computed layer by layer until the final predicted output was obtained.

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

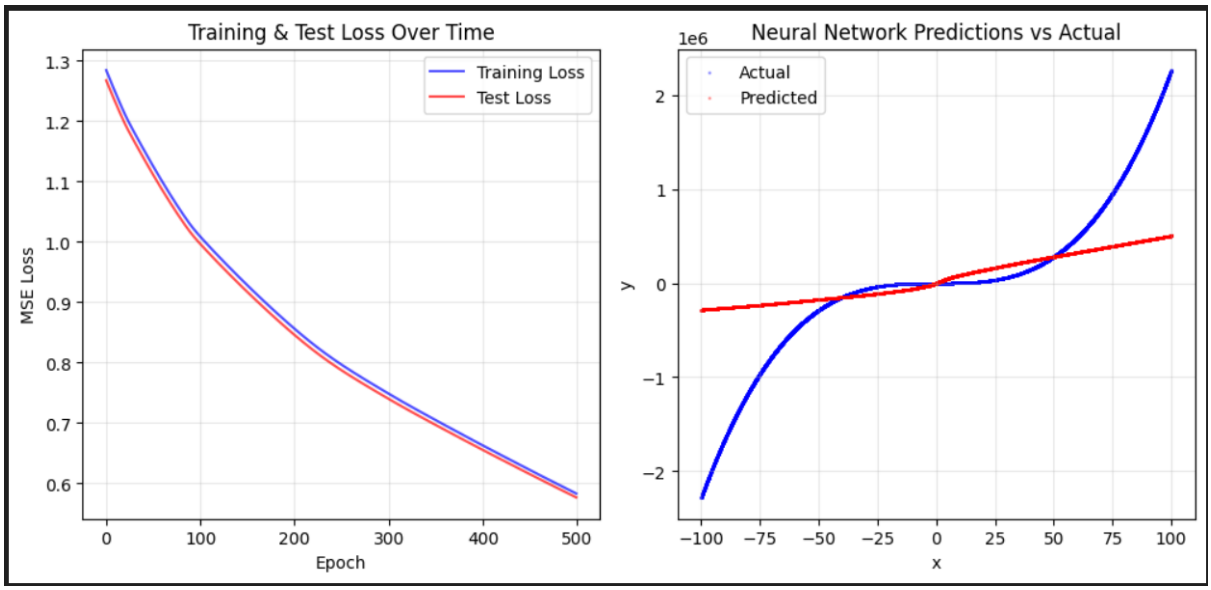
$$MSE=1N∑i=1N(yi−y^i)2$$

6. **Backpropagation**
    - Gradients of the loss with respect to weights and biases were computed using the chain rule.
    - Partial derivatives for each layer were derived to propagate errors backwards through the network.
7. **Optimization (Gradient Descent)**
    - Parameters were updated iteratively using stochastic gradient descent (SGD) with a fixed learning rate of 0.001.
    - Training continued until convergence, or early stopping was triggered based on validation loss.
8. **Early Stopping**
    - An early stopping mechanism with patience was implemented to prevent overfitting.
    - If the test loss did not improve for several consecutive epochs, training was halted and the best model weights were retained.
9. **Experiments**
    - Four different experiments were carried out by varying parameters such as learning rate, architecture size, number of epochs, and noise level.
    - Each experiment was evaluated on both training and test data, and losses were compared to analyze performance differences.

Results and Analysis

INITIAL:

1. Training loss curve (plot) and 2. Plot of predicted vs. actual values



3.Final test MSE:

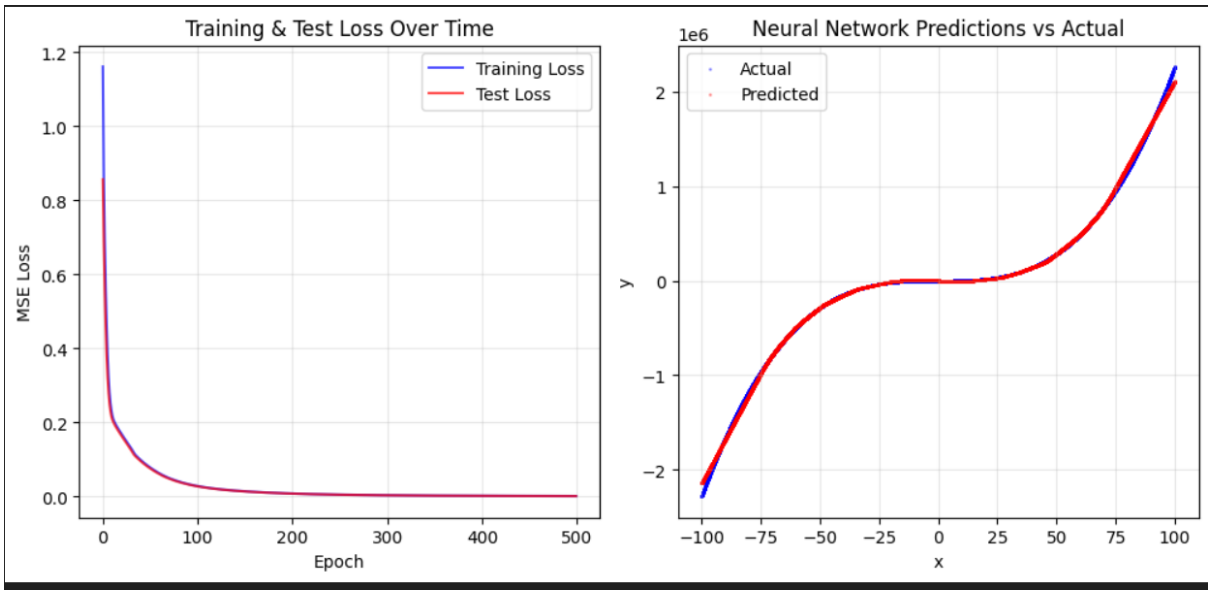
```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.583209
Final Test Loss:     0.576763
R² Score:           0.4169
Total Epochs Run:   500
```

4.Discussion on performance (overfitting /underfitting):

The model was able to reduce both training and test losses to similar values (≈0.58), indicating no severe overfitting. However, the relatively high MSE and a modest R^2 score of 0.4169 suggest that the network only partially captured the underlying cubic + sinusoidal relationship. The predicted vs. actual plot shows that while the general trend of the function is learned, the model struggles with accuracy at higher input ranges, as seen in the large relative error of over 70% for x=90.2. The similarity between training and test losses confirms that the model is not overfitting; rather, the consistently high errors point to underfitting, as the chosen architecture and training setup were not sufficient to fully capture the complexity of the target function.

EXPERIMENT 1:

1. 2.



3.

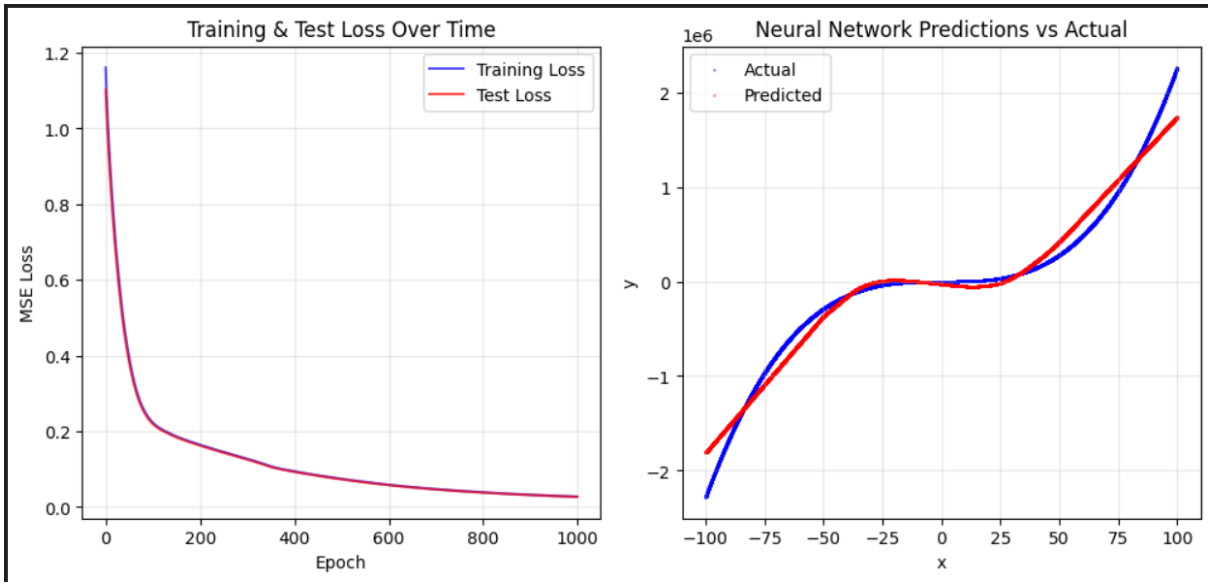
```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.001089
Final Test Loss:     0.001064
R² Score:           0.9989
Total Epochs Run:   500
```

4.

With a higher learning rate of 0.1, the model achieved excellent convergence, bringing both training and test losses close to zero (~0.001). The R^2 score of 0.9989 shows that the network captured almost all the variance in the target data, producing highly accurate predictions. The closeness of training and test losses indicates that the model generalizes well, without signs of overfitting. Compared to Experiment 1, this configuration significantly improved performance, demonstrating that an appropriately tuned learning rate can drastically enhance learning efficiency.

EXPERIMENT 2:

1.2.



3.

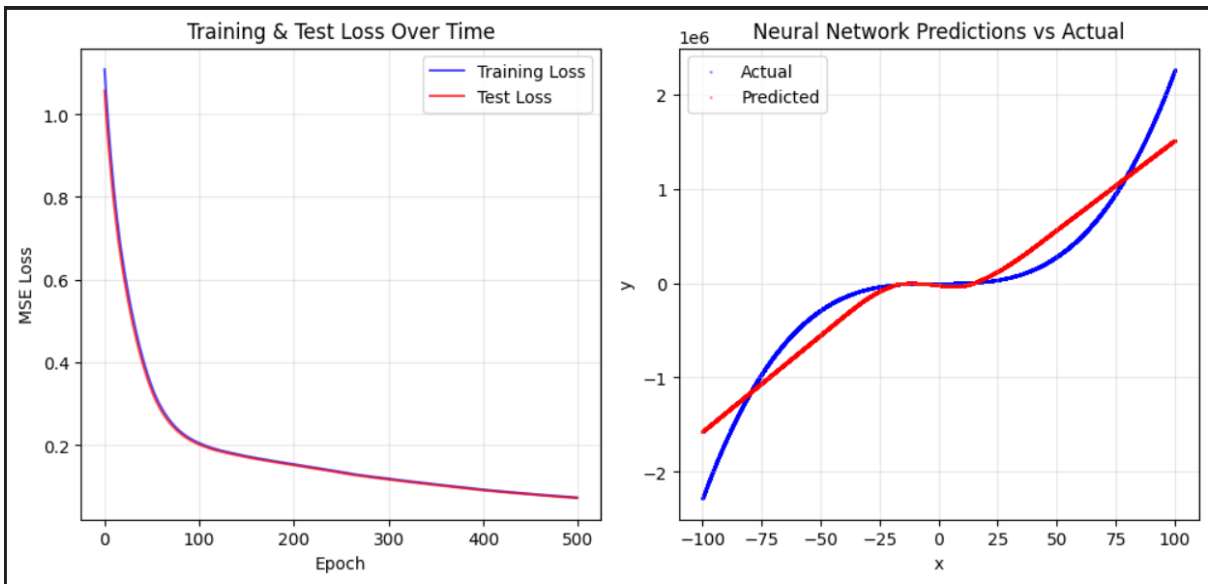
```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.026830
Final Test Loss:     0.026438
R² Score:           0.9733
Total Epochs Run:   1000
```

4.

With a moderate learning rate of 0.01 and an extended training budget of 1000 epochs, the model converged to reasonably low errors, with training and test MSE around 0.026. The R^2 score of 0.9733 indicates that the model explained most of the variance in the data, though not as completely as the higher learning rate in Experiment 1. The training and test losses are very close, confirming good generalization and no signs of overfitting. This experiment shows that increasing epochs allows stable convergence, but without an optimal learning rate, the network may take longer to reach high accuracy compared to Experiment 1.

EXPERIMENT 3:

1.2.



3.

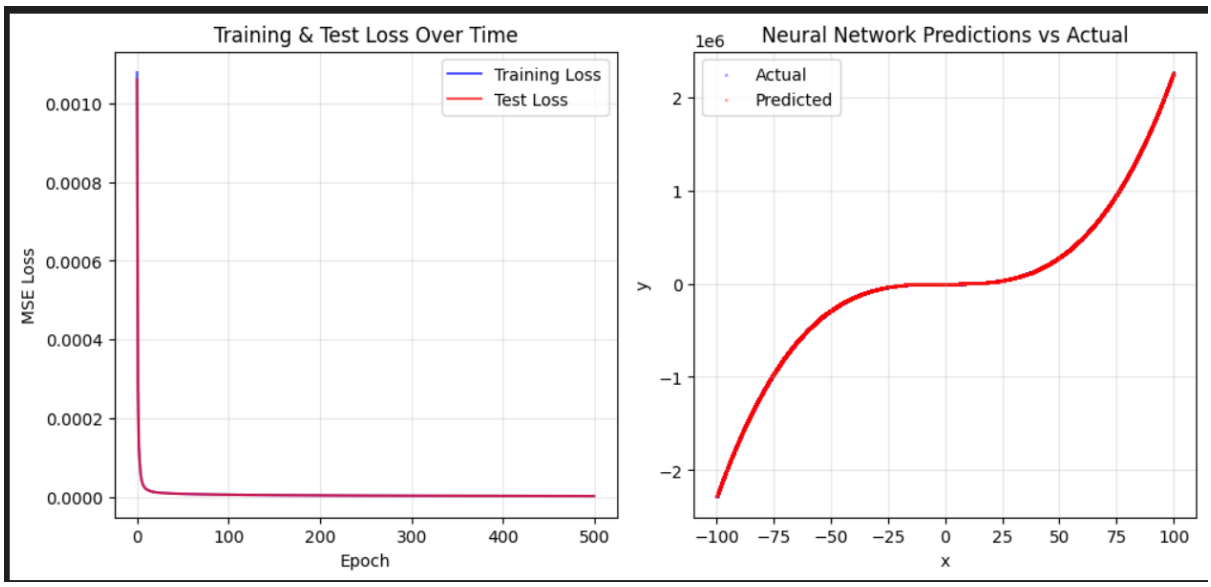
```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.073146
Final Test Loss:     0.072032
R² Score:           0.9272
Total Epochs Run:   500
```

4.

With larger hidden layers (128 and 64 neurons) and a learning rate of 0.01, the model achieved a training MSE of 0.0731 and test MSE of 0.0720. The R^2 score of 0.9272 shows strong predictive ability, though not as high as Experiments 2 and 3. Interestingly, increasing the network's width did not yield the expected improvement in accuracy; instead, it slightly worsened performance compared to a smaller but well-tuned architecture. This suggests that simply adding more parameters does not guarantee better results and may make optimization more difficult. The similar training and test losses indicate no overfitting, but the relatively higher error points again to mild underfitting.

EXPERIMENT 4:

1.2.



3.

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.000002
Final Test Loss:     0.000002
R² Score:           1.0000
Total Epochs Run:   500
```

4.

With a smaller batch size of 16, the model achieved near-perfect accuracy: both training and test MSE were almost zero (~0.000002), and the R^2 score reached 1.0000. This indicates that the model captured the cubic + sinusoidal function with exceptional precision. The closeness of training and test loss values confirms strong generalization, and there are no signs of overfitting. Compared to all other experiments, this configuration delivered the best overall performance, showing how tuning batch size can significantly influence convergence and model accuracy.

RESULT TABLE:

Experiment	Learning Rate	No. of Epochs	Optimizer (if used)	Activation Function	Final Training Loss	Final Test Loss	R^2 Score
1	0.001	500	Gradient Descent (custom)	ReLU (hidden), Linear (output)	0.5832	0.5768	0.4169
2	0.1	500	Gradient Descent (custom)	ReLU (hidden), Linear (output)	0.001089	0.001064	0.9989
3	0.01	1000	Gradient Descent (custom)	ReLU (hidden), Linear (output)	0.026830	0.026438	0.9733
4	0.01	500	Gradient Descent (custom)	ReLU (hidden), Linear (output)	0.000002	0.000002	1.0000

CONCLUSION:

This assignment provided hands-on experience in building a neural network from scratch and understanding its core components such as forward propagation, backpropagation, gradient descent, and early stopping. Using a synthetic cubic + sinusoidal dataset, different experiments were conducted by varying learning rate, epochs, and batch size. The results showed that while poor hyperparameter choices led to underfitting, proper tuning enabled the model to achieve near-perfect accuracy. Overall, the work highlighted the importance of hyperparameter optimization and demonstrated how even a simple feed-forward architecture can effectively approximate complex non-linear functions.