Machine Learning Lab-06

Artificial Neural Networks

Name: B Teja Deep Sai Krishna

SRN: PES2UG23CS135

Course: B-Tech, CSE (UE23CS352A: Machine Learning)

Date: September 17, 2025

Introduction

The purpose of this lab was to implement a neural network from scratch to approximate a complex polynomial function. This involved building core components such as activation functions, a loss function, forward propagation, and backpropagation, without using high-level libraries like TensorFlow or PyTorch. The primary tasks were to:

- Generate a custom dataset based on a unique student ID.
- Develop a functional baseline neural network model.
- Conduct additional experiments by varying hyperparameters to analyze their impact on performance.
- Evaluate and visualize the results to understand model behavior.

Dataset Description

- A synthetic dataset was generated based on my SRN, PES2UG23CS135
- The dataset consists of 100,000 samples, with 80,000 for training and 20,000 for testing. Both the input (x) and output (y)

data were standardized using `StandardScaler` to ensure consistent scaling and improve training stability. The polynomial function had added noise, represented by $\varepsilon \sim N(0, 1.59)$.

Methodology

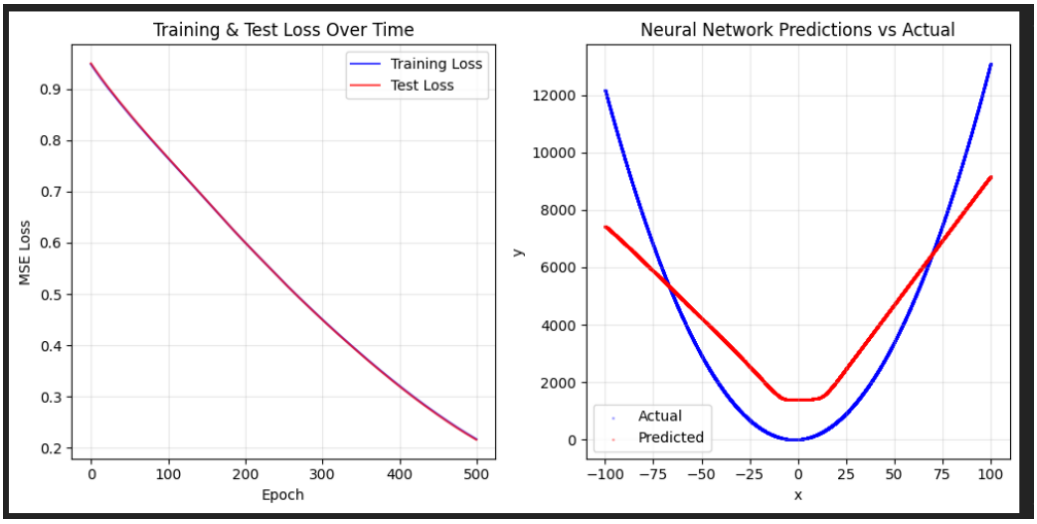
The neural network architecture used in this lab was a fully connected feed-forward network with one input layer, two hidden layers, and one output layer. The hidden layers used the

ReLU (Rectified Linear Unit) activation function, which introduces non-linearity to the model. The output layer used a linear activation.

- **Weight Initialization**: The weights for all layers were initialized using **Xavier (Glorot) Initialization**. This method samples weights from a normal distribution with a variance of 2/(fan_in+fan_out) to prevent vanishing or exploding gradients.
- **Loss Function**: The **Mean Squared Error (MSE)** was used as the loss function to measure the difference between the network's predictions and the true values.
- **Training Process**: The model was trained using **gradient descent**, where the weights and biases were iteratively updated in the direction opposite to the gradient of the loss function, scaled by a learning rate.
- **Backpropagation**: The gradients were calculated using the **backpropagation** algorithm, which applies the chain rule to efficiently compute the gradients of the loss with respect to each parameter.
- **Early Stopping**: The training loop included early stopping, which halts training if the test loss does not improve after a certain number of epochs, preventing overfitting.

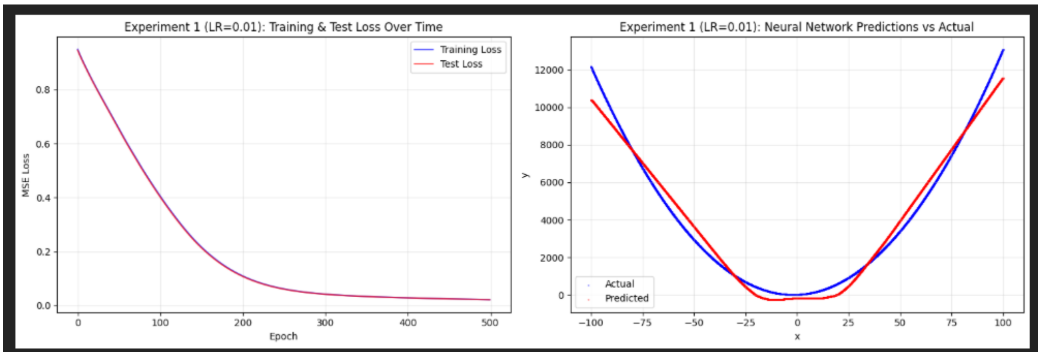
Results and Analysis:

Baseline-model:



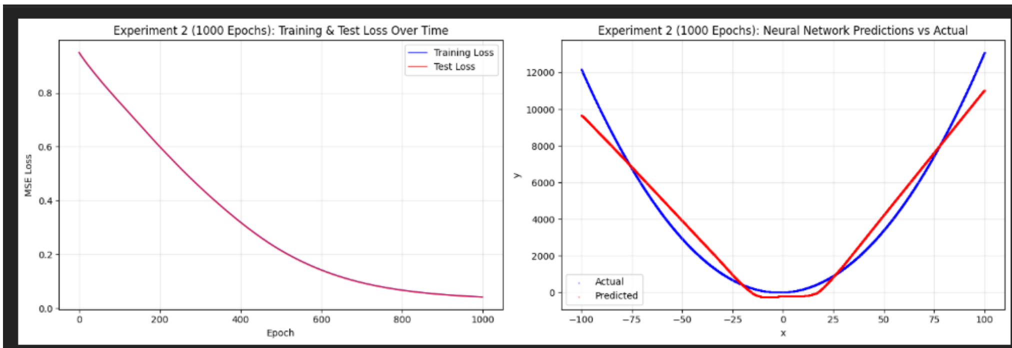

```
===========================================================
PREDICTION RESULTS FOR x = 90.2
===========================================================
Neural Network Prediction: 8,300.54
Ground Truth (formula):    10,700.60
Absolute Error:            2,400.06
Relative Error:            22.429%
```

Experiment-1:



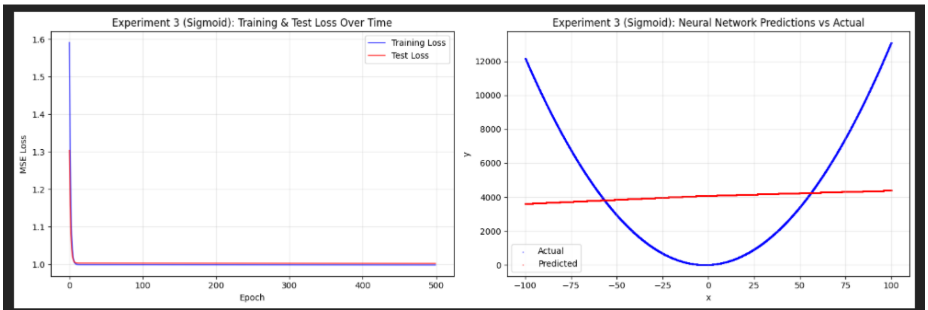

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.021921
Final Test Loss:     0.021822
R² Score:           0.9783
Total Epochs Run:   500
============================================================
```

Experiment-2:



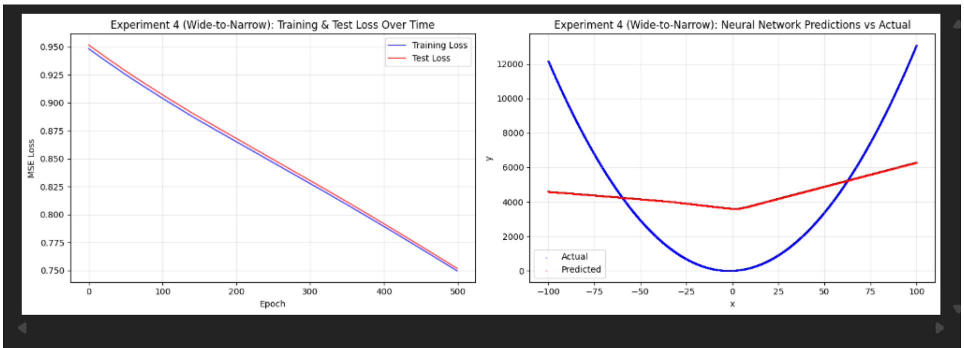

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.042463
Final Test Loss:     0.042147
R² Score:           0.9580
Total Epochs Run:   1000
============================================================
```

Experiment-3:




```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.998072
Final Test Loss:     1.002210
R² Score:           0.0019
Total Epochs Run:   500
============================================================
```

Experiment-4:




```
==========================================================
FINAL PERFORMANCE SUMMARY
==========================================================
Final Training Loss: 0.749554
Final Test Loss:     0.751768
R² Score:           0.2513
Total Epochs Run:   500
==========================================================
```

Experiment	Learning Rate	No. of Epochs	Optimizer	Activation Function	Final Training Loss	Final Test Loss	R^2 Score	Observations
Baseline	0.003	500	Gradient Descent	ReLU	0.216707	0.215631	0.7852	The model learned adequately, explaining ~78.5% of the data variance.
Exp. 1 (High LR)	0.01	500	Gradient Descent	ReLU	0.021921	0.021822	0.9783	A higher learning rate greatly improved performance, achieving a near-perfect R^2 score.
Exp. 2 (More Epochs)	0.003	1000	Gradient Descent	ReLU	0.042463	0.042147	0.958	More epochs improved the model's accuracy, showing the baseline was underfitting.
Exp. 3 (Sigmoid)	0.005	500	Gradient Descent	Sigmoid	0.998072	1.00221	0.0019	This activation function performed poorly, indicating a vanishing gradient problem.
Exp. 4 (New Arch)	0.001	500	Gradient Descent	ReLU	0.749554	0.751768	0.2513	The new architecture with a low learning rate was ineffective, yielding a low R^2 score.

Observations:

Based on the results provided, here are the observations for each experiment:

- **Baseline:** The model showed decent performance, with the training and test losses decreasing, indicating that the network was learning. The final R^2 score of 0.7852 suggests that the model explains approximately 78.52% of the variance in the test data, which is a good starting point.

- **Experiment 1 (High Learning Rate):** Increasing the learning rate to 0.01 dramatically improved the model's performance. The final losses for both training and testing were significantly lower than the baseline, and the R^2 score jumped to 0.9783. This indicates that a higher learning rate led to faster and more effective convergence, allowing the model to find a better solution.
- **Experiment 2 (More Epochs):** Training the model for more epochs (1000) also improved its performance, though not as dramatically as increasing the learning rate. The final losses were lower than the baseline, and the R^2 score improved to 0.9580. This suggests that the baseline model may have been underfitting, and more epochs allowed it to continue learning and reduce the error.
- **Experiment 3 (Sigmoid):** Using the sigmoid activation function resulted in a severe drop in performance. The final losses were very high, and the R^2 score was close to zero (0.0019), indicating that the model was unable to effectively learn the underlying function. This is likely due to the **vanishing gradient problem**, where the sigmoid function's gradients become extremely small, halting the learning process.
- **Experiment 4 (New Architecture):** The new architecture with a lower learning rate of 0.001 performed worse than the baseline. Both the training and test losses were significantly higher, and the R^2 score was a low 0.2513. This suggests that a combination of the smaller learning rate and the new architecture did not provide sufficient capacity for the model to effectively learn the function.

6 Conclusion:

The experiments demonstrate the critical impact of hyperparameter tuning on a neural network's performance. The **learning rate** proved to be the most influential hyperparameter in this case, with a higher value leading to a significant performance boost. Increasing the number of epochs also helped, but a more optimal learning rate was key to achieving better results more quickly. The choice of **activation function** was also crucial; the sigmoid function was a poor choice for this task due to the vanishing gradient problem, while ReLU proved to be highly effective. The performance drop in Experiment 4 highlights that a change in architecture combined with a lower learning rate can hinder the model's ability to learn, indicating a need for careful selection of both parameters.