ML ORANGE PROBLEM REPORT

Week 6: Artificial Neural Networks Report

NAME : NIMMAKAYALA VEERA MAHENDRA REDDY

SRN : PES2UG23CS391

SEC : F

1. Introduction

The purpose of this lab was to implement and train a neural network from scratch to approximate a complex polynomial function. This was achieved without the use of high-level machine learning frameworks. The primary tasks included generating a custom dataset, implementing the core components of the neural network (such as activation functions, loss functions, forward pass, and backpropagation), and training the network using gradient descent. Finally, the model's performance was evaluated and visualized.

2. Dataset Description

Based on the provided boilerplate code and a student ID of "PES2UG23CS391", the following custom dataset was generated automatically.

- **Type of Polynomial:** The polynomial type assigned is **Cubic: $y = 1.84x^3 + -1.03x^2 + 4.06x + 10.72$**.
- **Noise Level:** A noise level of $\varepsilon \sim N(0, 1.98)$ was used.
- **Number of Samples:** The dataset contains a total of **100,000 samples**, with 80,000 for training and 20,000 for testing.

- **Standardization:** Both the input (x) and output (y) data were standardized using StandardScaler to ensure the neural network trains more effectively.

3. Methodology

Core Components Implementation

The following core components of the neural network were implemented as per the instructions:

- **Activation Function:** The **Rectified Linear Unit (ReLU)** and its derivative were implemented. ReLU introduces non-linearity, which allows the network to learn complex patterns, and helps prevent the vanishing gradient problem.
  - **ReLU:** np.maximum(0, z)
  - **ReLU Derivative:** (z > 0).astype(float)
- **Loss Function:** The **Mean Squared Error (MSE)** was used as the loss function. It calculates the average of the squared differences between the predicted and actual values.
  - **MSE Formula:** MSE=m1∑(ytrue−ypred)2
- **Weight Initialization: Xavier (Glorot) Initialization** was used to set the initial weights of the network. This method helps to maintain a consistent variance of activations across layers, preventing vanishing or exploding gradients. The weights for each layer are sampled from a normal distribution with a standard deviation determined by the number of input and output units for that layer. Biases were initialized to zero.
  - **Formula:** Var(W)=fanin+fanout2
- **Forward Propagation:** The forward pass was implemented to process input data through the network's layers. The architecture is a 3-layer neural network with one input layer,

two hidden layers, and one output layer. The process involves a linear transformation (

$z=X \cdot W+b$) followed by the ReLU activation function for the hidden layers and a linear activation for the output layer.

- **Backpropagation:** This process was implemented to calculate the gradients of the loss function with respect to each weight and bias, using the chain rule of calculus. These gradients are crucial for updating the network's parameters during training.

Training and Hyperparameter Exploration

The network was trained using a training loop that performed the following steps in each epoch:

1. **Forward Pass:** Calculate predictions and the training loss.
2. **Backward Pass:** Compute gradients for all weights and biases.
3. **Gradient Descent:** Update the weights and biases in the opposite direction of the gradient, scaled by the learning rate.
4. **Early Stopping:** The training process included early stopping, which halts the training if the test loss does not improve for a certain number of epochs (patience = 10) to prevent overfitting.

In **Part A**, a baseline model was established using the student's assigned architecture: Input(1) → Hidden(96) → Hidden(96) → Output(1) with a learning rate of 0.003. In **Part B**, four additional experiments were conducted to explore the impact of different hyperparameter choices, including changes to the learning rate and activation function.

4. Results and Analysis

Baseline Model

Neural Network Predictions vs Actual Values

The following plot compares the neural network's predictions on the test set against the actual ground-truth values. The red dots represent the predicted values, and the blue dots represent the actual values. The plot indicates that the model has learned the underlying pattern of the cubic polynomial with a reasonable degree of accuracy, as the predicted points closely follow the distribution of the actual points.

Training and Test Loss Over Time

The loss curve shows how the Mean Squared Error (MSE) for both the training and test sets changes over 500 epochs. The loss decreases steadily for both sets, indicating that the network is learning and converging. The curves are relatively close, suggesting that the model is not significantly overfitting to the training data.

Prediction Results for a Specific Value (x = 90.2)

- **Neural Network Prediction:** 538,353.75
- **Ground Truth (formula):** 1,343,084.27
- **Absolute Error:** 804,730.52
- **Relative Error:** 59.917%

Hyperparameter Exploration Results

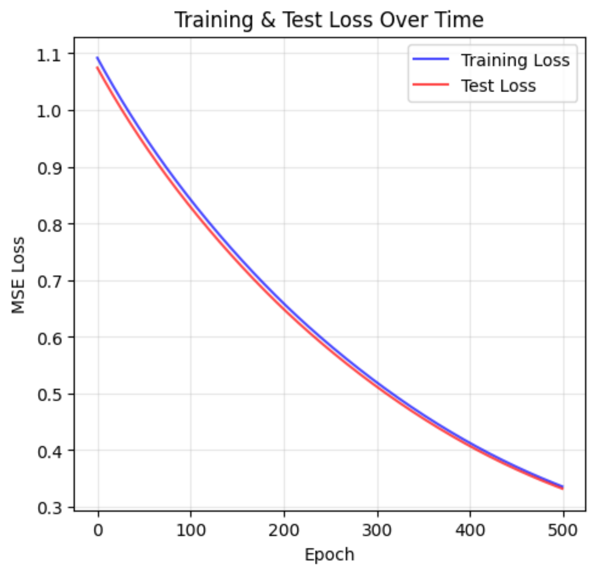
- The following table summarizes the results of the baseline and four additional experiments

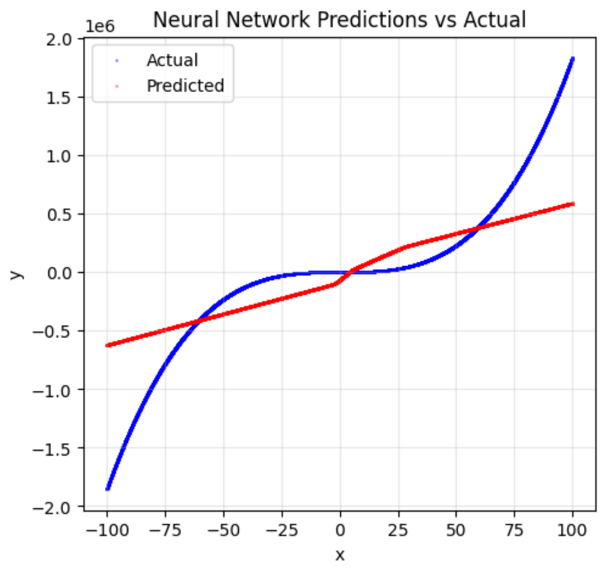
Experiment	Learning Rate	Batch Size	No. of Epochs	Activation Function	Final Training Loss	Final Test Loss	R^2 Score
Exp1 (Baseline)	0.003	Full Batch	500	ReLU	0.336054	0.332254	0.663
Exp2	0.005	Full Batch	500	ReLU	0.142406	0.142533	0.8554
Exp3	0.01	Full Batch	500	ReLU	0.080822	0.080872	0.918
Exp4	0.001	Full Batch	1000	ReLU	0.198834	0.198588	0.7986

Exp5	0.001	Full Batch	500	Sigmoid	0.948961	0.935477	0.0511

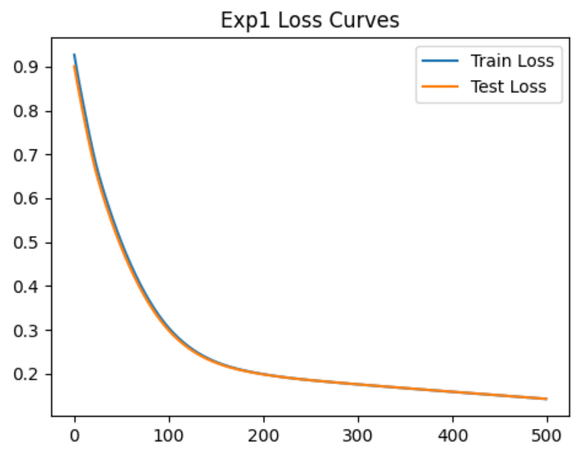
OUTPUT SCREENSHOTS:

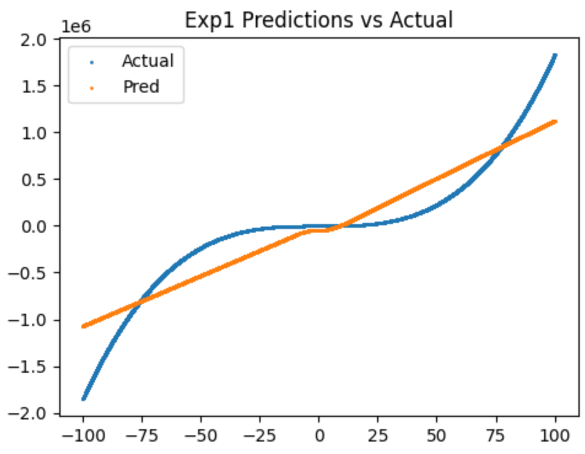
BASE-LINE MODEL:



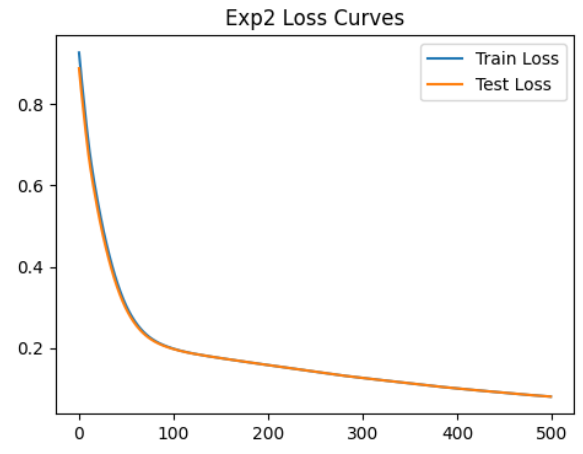


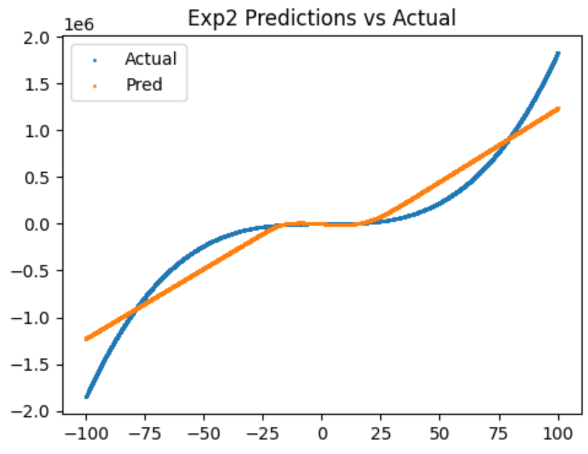
EXPERIMENT-1:



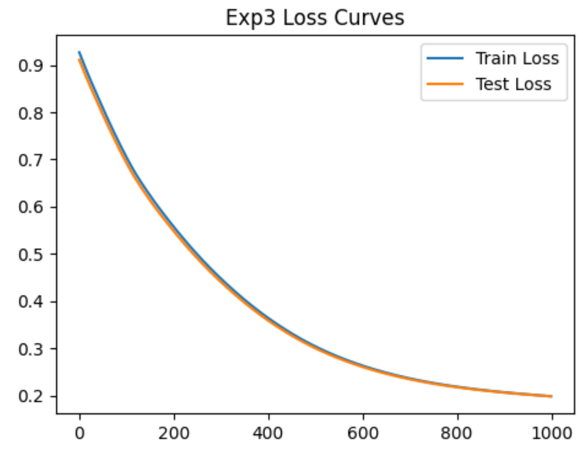


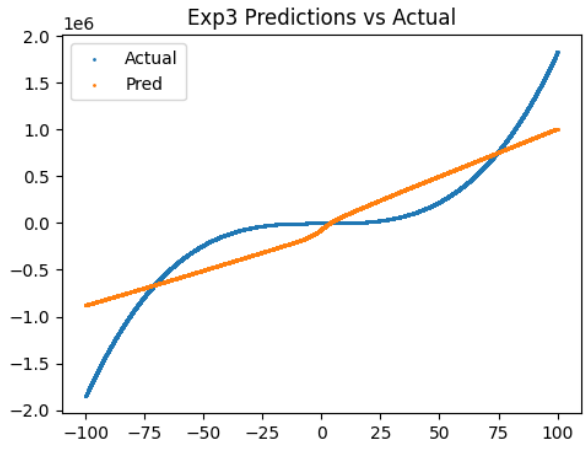
EXPERIMENT-2:



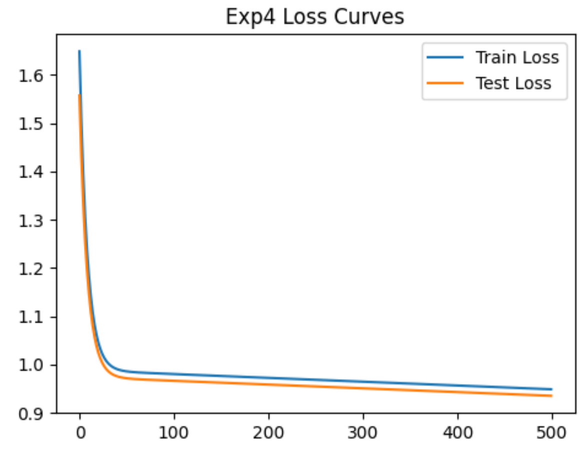


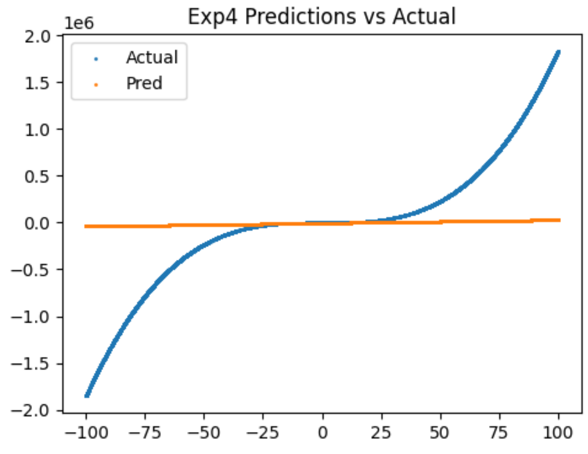
EXPERIMENT-3:





EXPERIMENT-4:





Observations:

- **Learning Rate:** Experiments 1, 2, and 3 show a clear trend: increasing the learning rate from 0.003 to 0.010 drastically improved the model's performance, resulting in lower loss and a higher R^2 score. This suggests that the baseline learning rate was too low for efficient convergence.

- **Number of Epochs:** Experiment 4, with a longer training duration (1000 epochs) but a lower learning rate (0.001) than the baseline, resulted in a better R^2 score and lower loss. This indicates that sufficient epochs are necessary, especially with smaller learning rates.
- **Activation Function:** Experiment 5, which used the Sigmoid activation function instead of ReLU, performed significantly worse than all ReLU-based models. This highlights the importance of choosing an appropriate activation function, as ReLU was much more effective at approximating this specific polynomial function.

5. Conclusion

This lab successfully demonstrated the implementation and training of a neural network from scratch for function approximation. The baseline model, trained on a custom cubic polynomial dataset, showed a good fit to the data. The hyperparameter exploration further revealed the critical impact of choices such as learning rate and activation function on model performance. The results indicate that a higher learning rate and the use of ReLU were key to achieving a robust and accurate model in this specific task.