UE23CS352A: MACHINE LEARNING

Week 6: Artificial Neural Networks

Neural Networks for Function Approximation

Lab Report

Name: Diya Prakash

SRN: PES2UG23CS184

Course: UE23CS352A - MACHINE LEARNING

Date: 16 Sep. 25

1. Introduction

The purpose of this lab is to implement a neural network from scratch for function approximation, without using high-level frameworks. The main tasks performed include:

- Generating a synthetic dataset based on a unique polynomial assigned from the SRN.
- Implementing all core neural network components: activation functions, loss, forward and backward propagation, and weight updates.
- Training the network to fit the dataset and evaluating its performance.
- Experimenting with different hyperparameters and comparing results.

2. Dataset Description

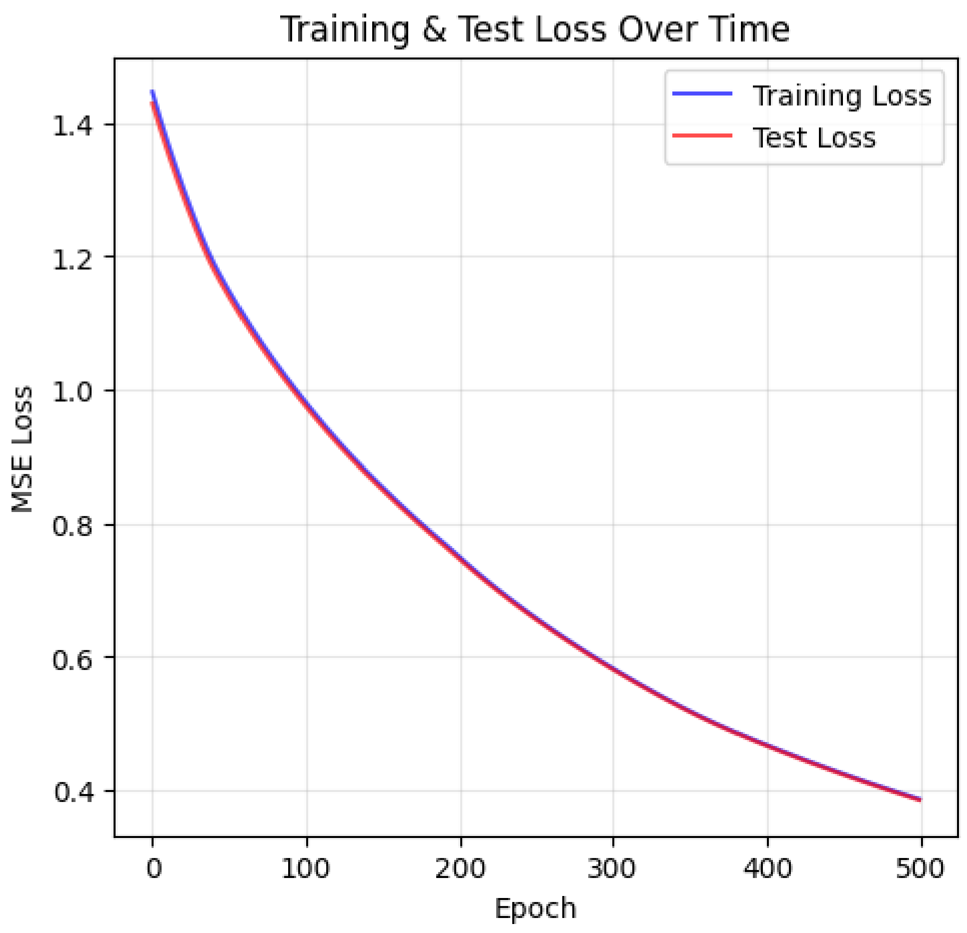
- **Type of polynomial assigned:** $y = 1.85x^3 + -0.24x^2 + 5.92x + 8.79 + 107.4/x$
- **Number of samples:** 100,000
- **Number of features:** 1 (input x)
- **Noise level:** $\varepsilon \sim N(0, 1.56)$

3. Methodology

- **Network Architecture:** Input(1) → Hidden Layer 1 → Hidden Layer 2 → Output(1)
- **Activation Function:** ReLU (and LeakyReLU for one experiment)
- **Loss Function:** Mean Squared Error (MSE)
- **Weight Initialization:** Xavier (Glorot) initialization
- **Optimizer:** Stochastic Gradient Descent (SGD)
- **Training:** Full-batch gradient descent, early stopping based on validation loss

<u>4. Results and Analysis</u>

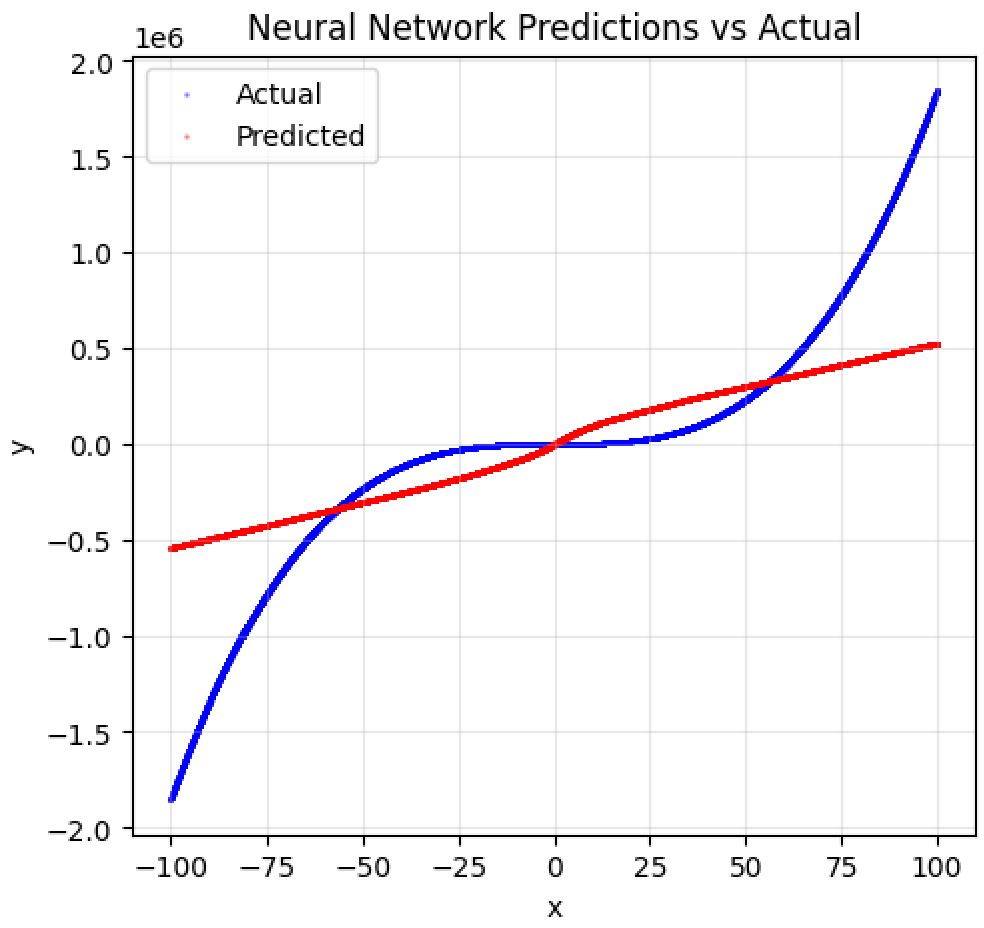
Training Loss Curve - The training and test loss curves (see plot below) show a steady decrease in Mean Squared Error (MSE) over epochs, indicating successful learning and convergence of the neural network.



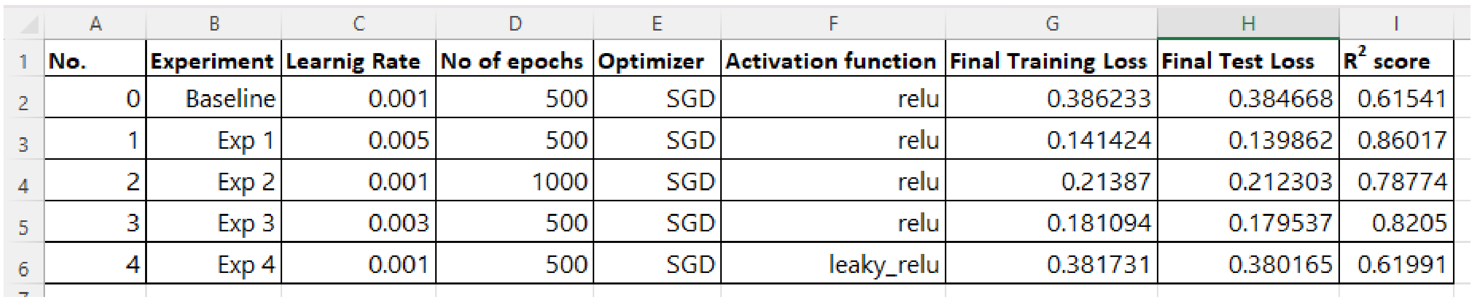
Final Test MSE:

- **Final Test Loss (MSE):** 0.384668

  Predicted vs. Actual Values - The scatter plot below compares the neural network's predictions to the true target values on the test set. The predicted values (red) follow the general trend of the actual values (blue), but there is a noticeable gap, especially at the extremes, indicating some underfitting.



Results Table:



	A	B	C	D	E	F	G	H	I
1	**No.**	**Experiment**	**Learnig Rate**	**No of epochs**	**Optimizer**	**Activation function**	**Final Training Loss**	**Final Test Loss**	**R^2 score**
2	0	Baseline	0.001	500	SGD	relu	0.386233	0.384668	0.61541
3	1	Exp 1	0.005	500	SGD	relu	0.141424	0.139862	0.86017
4	2	Exp 2	0.001	1000	SGD	relu	0.21387	0.212303	0.78774
5	3	Exp 3	0.003	500	SGD	relu	0.181094	0.179537	0.8205
6	4	Exp 4	0.001	500	SGD	leaky_relu	0.381731	0.380165	0.61991

Discussion:

Performance:
The neural network was able to fit the data with a best R^2 score of 0.86 (Exp 1) and a lowest test MSE of 0.139862. The baseline model achieved an R^2 of 0.62, indicating moderate fit. The model captures the general trend of the data, but struggles with more complex variations, as seen in the predicted vs. actual plot.

Overfitting/Underfitting:
The gap between training and test loss is small in all experiments, suggesting that the model is not overfitting. However, the relatively high MSE and the inability to capture all the non-linearities in the data (especially at the edges) indicate some underfitting. This is likely due to the network's limited capacity or the need for more training epochs or different architectures.

Hyperparameter Impact:

- **Learning Rate:** Increasing the learning rate from 0.001 (baseline) to 0.005 (Exp 1) significantly improved both training and test performance, reducing MSE and increasing R^2. However, too high a learning rate can cause instability (not observed here).
- **Epochs:** Doubling the number of epochs (Exp 2) improved performance over the baseline, but not as much as increasing the learning rate.
- **Activation Function:** Using LeakyReLU (Exp 4) did not outperform standard ReLU in this case, possibly because the data does not have many negative activations or the network is not deep enough to benefit.
- **Overall:** The best results were achieved with a higher learning rate and standard ReLU activation.

5. Conclusion:

This lab demonstrated how to implement and train a neural network from scratch for function approximation. Through systematic experimentation, it was observed that hyperparameters such as learning rate and number of epochs have a significant impact on model performance. The network was able to approximate the assigned polynomial function reasonably well, but some underfitting remained, suggesting that further tuning or architectural changes could yield better results. The exercise provided valuable insights into the mechanics of neural networks, the importance of hyperparameter selection, and the challenges of fitting complex functions.