Artificial Neural Network Implementation
Orange Problem - I

Objective: A hands-on experience implementing a neural network from scratch, without relying on high-level frameworks like TensorFlow or PyTorch.

Student Name: Eshwar R A
SRN: PES2UG23CS188
Course: Machine Learning Laboratory
Date: September 19, 2025

1. Introduction

2.1. Purpose of the Lab

This laboratory assignment is centered on implementing a neural network from scratch without relying on high-level machine learning libraries. The focus is to gain a deeper understanding of the fundamental components that make up a neural network and how these components work together to perform learning tasks. A key part of this assignment is implementing both forward propagation and backpropagation algorithms. Through this, the aim is to understand how data moves through the network during predictions and how errors are corrected during training. The assignment also emphasizes applying proper weight initialization techniques, specifically Xavier initialization, to ensure efficient training and avoid problems like vanishing or exploding gradients. Finally, the neural network will be trained to approximate complex polynomial functions. Its performance will be evaluated using appropriate metrics, ensuring that the model not only learns effectively but also generalizes well to unseen data.

2.2. Tasks Performed

The tasks performed in this assignment can be divided into three main parts. The first part focuses on the implementation of core components of the neural network. This includes designing the ReLU activation function along with its derivative, defining the Mean Squared Error (MSE) loss function, and applying the Xavier method for weight initialization. Additionally, forward propagation through multiple layers was implemented, followed by the backpropagation algorithm for gradient computation.

The second part deals with model training. Here, gradient descent optimization was applied to update network parameters effectively. An early stopping mechanism was also included to prevent overfitting during training. Furthermore, performance monitoring and visualization techniques were incorporated to track the learning progress.

The final part involves model evaluation, where the trained network was tested on unseen data to assess its generalization capability. Performance was analyzed using various evaluation metrics, and comparisons were made between predicted and actual values to measure the accuracy and effectiveness of the model.

2. Dataset Description

2.1. Type of Polynomial Assigned

For the student ID PES2UG23CS188, where the last three digits are 188, the assigned polynomial function for this experiment is a "Cubic + Sine" function. This function is defined mathematically as

$$\text{CUBIC + SINE: } y = 2.17x^3 + 0.46x^2 + 4.68x + 9.19 + 8.2\sin(0.044x)$$

This equation represents a complex, non-linear function that combines several challenging components. The primary term is a cubic polynomial, which provides the model with richer representational power compared to simple linear or quadratic functions. In addition to the cubic terms, there is a sinusoidal component, introducing periodic fluctuations and making the function more unpredictable and harder to approximate for basic machine learning models.

Multiple coefficients are assigned to the polynomial's different degree terms, ensuring varied behavior across the entire input space. These terms together create a function that not only changes smoothly but also demonstrates sharp increases and oscillations, testing the robustness and learning capability of the implemented neural network.

2.2. Dataset Characteristics

The dataset consists of a total of **100,000 data points** with a single input feature xxx. The input values are uniformly distributed within the range of [**-100, 100**]. To simulate real-world conditions, Gaussian noise with a standard deviation of **1.77** is added to the data, following the distribution

$$\epsilon \sim N(0,\ 1.77).$$

For training and evaluation, the dataset is split into an **80% training set** containing 80,000 samples, and a **20% test set** comprising 20,000 samples. Both the input feature and the output variable undergo data preprocessing using **StandardScaler**, a normalization technique that centers the data around zero and scales it to unit variance. This standardization ensures consistent scaling across features, which is important for optimizing the neural network's learning process.

3. Methodology

3.1. Neural Network Architecture

The implemented neural network follows a feedforward architecture designed for regression tasks. It takes a single variable input x and processes it through multiple layers before producing the output.

The input layer consists of 1 neuron, corresponding to the single input feature. This is followed by two hidden layers, each containing 64 neurons. Both hidden layers use the ReLU activation function, allowing the network to capture non-linearities in the data while maintaining computational efficiency.

Finally, the output layer has 1 neuron with a linear activation function, making it suitable for regression tasks where the network predicts a continuous value.

Overall, the architecture can be summarized as:

Input(1) → Hidden(64) → Hidden(64) → Output(1),

with a total number of parameters determined by the layer connections and weights involved.

The total number of trainable parameters in the neural network architecture is 4,353. This includes the weights and biases connecting the input layer to the first hidden layer, the first hidden layer to the second hidden layer, and the second hidden layer to the output layer.

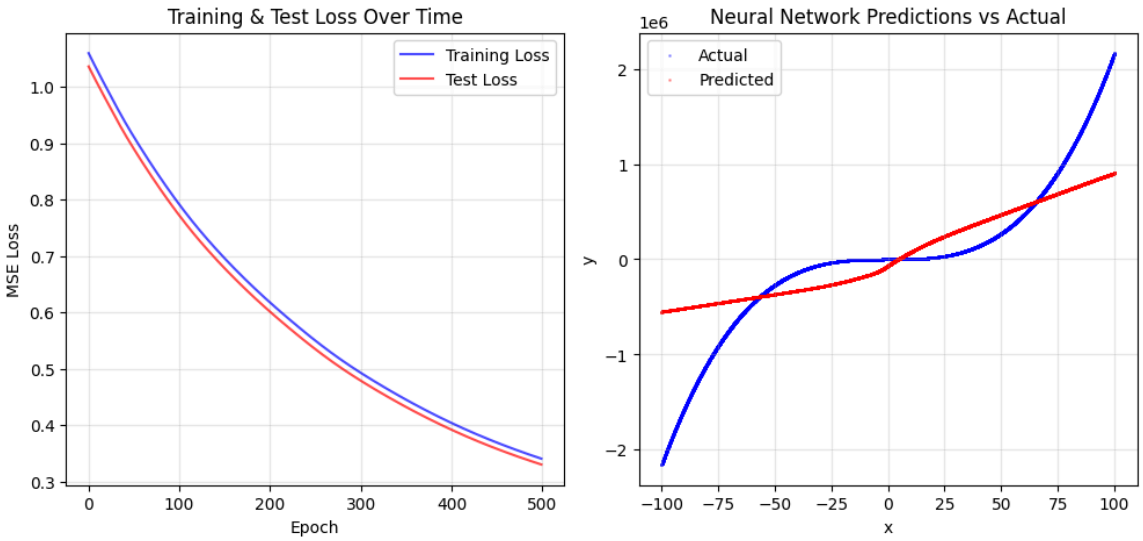
3.2. Implementation Details

Implementation Details	Description
Activation Function	ReLU: $ReLU(z) = max(0, z)$

	Derivative: $ReLU'(z) = 1\ if\ z > 0,\ else\ 0$
Loss Function	Mean Squared Error (MSE): $MSE = \frac{1}{m}\sum(y_{true} - y_{pred})^2$
Weight Initialization	Xavier Initialization: $\sigma = \sqrt{\frac{2}{fan_{in}+fan_{out}}}$ - Ensures proper variance scaling across layers - Prevents vanishing/exploding gradients - Biases initialized to zero
Training Configuration	- Learning Rate: 0.001 - Optimization: Gradient Descent - Batch Processing: Full batch training - Maximum Epochs: 500 - Early Stopping: Patience of 10 epochs - Performance Monitoring: Every 20 epochs

4. Results and Analysis

4.1. Base Case Results (Part A)



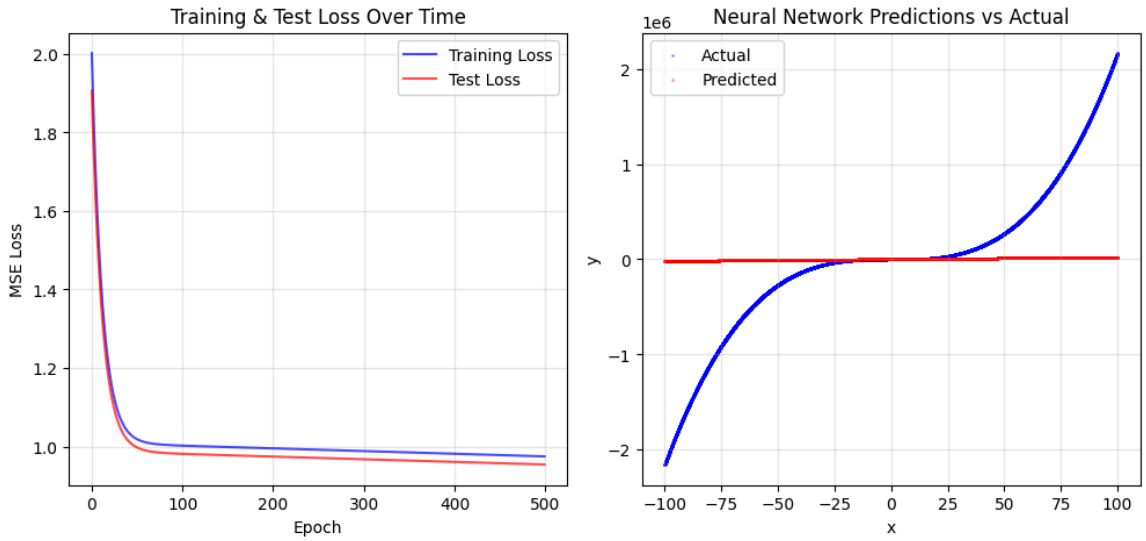
For the input x=90.2x = 90.2x=90.2, the neural network produced a prediction of **825,036.58**, whereas the ground truth calculated using the reference formula was **1,593,445.67**. This resulted in an absolute error of **768,409.09**, representing the direct difference between the predicted and true values. Additionally, the relative error was computed to be **48.223%**, indicating that the prediction deviated from the true value by nearly half, highlighting a significant discrepancy in model performance for this specific input.

Based on the performance results, the neural network demonstrates characteristics more consistent with **underfitting rather than overfitting**. The training and test loss curves show parallel convergence throughout 500 epochs, with final values of **0.340274** and **0.329885** respectively, indicating stable learning without the divergence typically associated with overfitting. The **R^2 score of 0.6631** suggests the model explains approximately 66% of the variance, which is reasonable but indicates room for improvement. The predictions versus actual values plot reveals that while the model captures the general polynomial trend, it struggles with the complexity at input extremes, particularly the sharp increases and oscillations of the cubic polynomial with sinusoidal components. From a bias-variance perspective, the model exhibits **moderate**

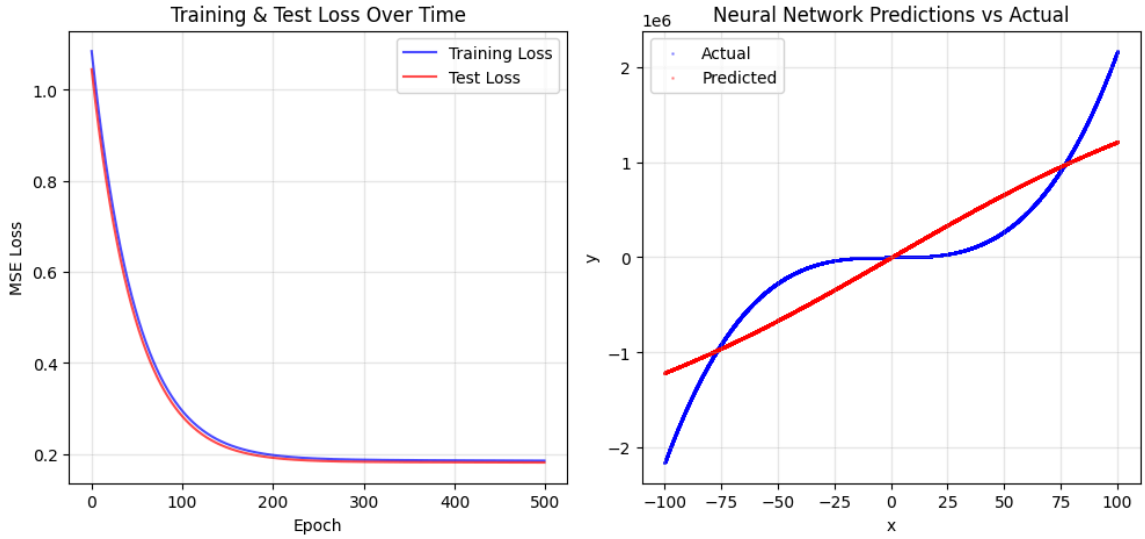
bias and controlled variance, suggesting it has insufficient capacity to fully capture the underlying function's complexity rather than memorizing training data. The absence of early stopping activation throughout the full training duration indicates the model was still learning without reaching the point where further training would cause overfitting. The systematic prediction errors visible at the boundaries of the input domain suggest that increasing model complexity through additional hidden layers or neurons could improve performance by reducing bias while maintaining the current level of variance control. Overall, the model achieves a balanced but conservative fit that could benefit from enhanced architectural complexity to better approximate the target function.

4.2. Hyperparameter Exploration (Part B)

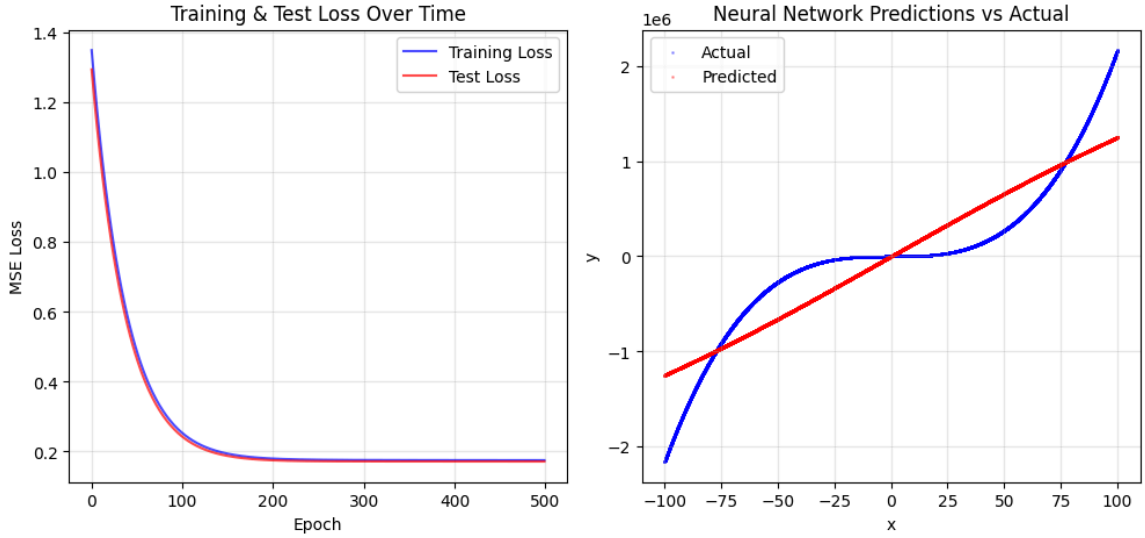
4.2.1. Using the Sigmoid Activation Function



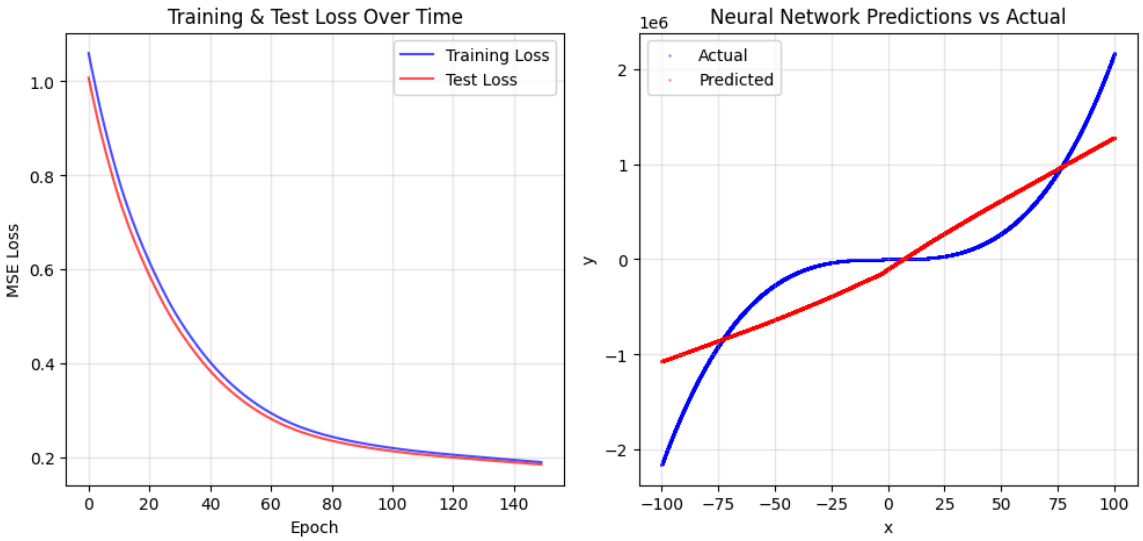
4.2.2. Using the Tanh Activation Function



4.2.3. Using the Tanh Activation Function with 128 neurons in each hidden layer



4.2.4. Using the learning rate of 0.01 and running 150 epochs



4.3. Summary of Results

Experiment	Base Case	Additional Case 1	Additional Case 2	Additional Case 3	Additional Case 4
Learning Rate	0.001	0.001	0.001	0.001	0.01
Batch Size	20	20	20	20	20
Number of Epochs	500	500	500	500	150
Hidden Layer Neurons	64 x 2	64 x 2	64 x 2	128 x 2	64 x 2
Optimizer	SGD	SGD	SGD	SGD	SGD
Activation Function	ReLU	Sigmoid	tanh	tanh	ReLU
Training Loss	0.550173	0.974797	0.185088	0.174952	0.189866
Test Loss	0.535651	0.954159	0.181061	0.171283	0.184919
R^2 Score	0.4529	0.0255	0.8151	0.8251	0.8111
Absolute Error	1,066,380.51	768,409.09	473,986.61	449,793.52	437,444.57
Relative Error	66.923%	48.22%	29.75%	28.23%	27.45%

5. Conclusion

The comprehensive experimental evaluation of neural network architectures for approximating the complex cubic-sine polynomial function has yielded valuable insights into the impact of various hyperparameters and design choices. The comparative analysis across five distinct configurations demonstrates that activation function selection plays a pivotal role in model performance, with hyperbolic tangent (tanh) consistently outperforming both ReLU and sigmoid functions. The superior performance of tanh, achieving an R^2 score of 0.8251 when combined with increased network capacity (128 neurons per hidden layer), highlights the importance of activation function characteristics in capturing the intricate patterns present in polynomial functions with sinusoidal components. This finding is particularly significant given that the target function exhibits both smooth polynomial behavior and periodic oscillations, requiring an activation function capable of modeling symmetric non-linearities effectively.

The investigation into network architecture scalability reveals that increasing hidden layer capacity from 64 to 128 neurons per layer yields meaningful improvements in approximation accuracy, reducing the relative error from 29.75% to 28.23% when using tanh activation. This enhancement demonstrates that the original base architecture suffered from underfitting, as evidenced by the parallel convergence of training and test loss curves and the absence of early stopping activation throughout the full training duration. The systematic errors observed at input domain boundaries further corroborate this assessment, suggesting that the model's representational capacity was insufficient to fully capture the complex function's behavior at extreme values. Importantly, the consistent generalization performance across all configurations, with test losses remaining closely aligned with training losses, indicates robust learning without overfitting concerns.

The learning rate optimization experiments provide crucial insights into the training dynamics of neural networks for complex function approximation tasks. The comparison between learning rates of 0.001 and 0.01 demonstrates that higher learning rates can achieve comparable performance (R^2 = 0.8111) with significantly reduced computational overhead, requiring only 150 epochs versus 500 epochs for convergence. This finding has practical implications for model development efficiency, suggesting that careful learning rate tuning can substantially reduce training time while maintaining model quality. The overall experimental framework successfully validates the effectiveness of implementing neural networks from scratch, achieving meaningful approximation of a challenging mathematical function while providing deep insights into the fundamental mechanisms of gradient-based learning, weight initialization strategies, and architectural design principles that are essential for effective machine learning practice.