# Week 6: Artificial Neural Networks

| | | | |
|---|---|---|---|
| Name: Nidhi K | SRN: PES2UG23CS383 | Course: UE23CS352A: MACHINE LEARNING | Date: 16-09-2025 |

## Introduction

The goal of this lab was to learn how to build and train a simple artificial neural network (ANN) from scratch. Instead of using ready-made libraries like TensorFlow or PyTorch, we coded the main parts ourselves. This included writing the activation function, the mean squared error (MSE) loss function, forward pass, backpropagation, and updating weights using gradient descent. Doing this step by step gave me a clearer idea of how a neural network actually learns.

In this lab, I worked on the following tasks:

- Created a dataset based on a polynomial function with some noise added.
- Implemented the ReLU activation function and its derivative for the hidden layers.
- Wrote the MSE loss function to check how close predictions are to actual values.
- Coded forward propagation to pass inputs through the network and get predictions.
- Implemented backpropagation to calculate how much the weights and biases should change.
- Trained the neural network using gradient descent and recorded the training and testing losses.
- Plotted graphs to compare predicted vs actual values and checked the residual errors.

## Dataset Description

The type of dataset assigned to me was a **Cubic + Sine function**, which means the output values were created using a cubic polynomial combined with a sine wave. The exact formula was:

y=2.07x^3−0.47x^2+3.98x+10.68+8.9⋅sin(0.038x)

To make the dataset more realistic, some random noise was also added. The noise followed a normal distribution with mean 0 and a standard deviation of 1.68.The dataset contained **100,000 samples** in total. Out of this, **80,000 samples** were used for training the neural network, and the remaining **20,000 samples** were used for testing. Each sample had **1 input feature (x)** and **1 output value (y)**.

**Methodology**

Weight Initialization: Weights were initialized using Xavier initialization, which ensures that the variance of activations remains stable across layers. This method prevents activations from vanishing or exploding during training. All biases were initialized to zero.

Forward Pass: The network architecture followed the structure Input(1) → Hidden(96) → Hidden(96) → Output(1). In the forward pass, the input values were multiplied with weights and added to biases. The ReLU activation function was applied in the hidden layers to introduce non-linearity, while the output layer was kept linear since the task was regression.

Loss Function: The Mean Squared Error (MSE) loss function was used to measure prediction error. It calculates the average of squared differences between predicted and actual values. The training objective was to minimize this loss throughout the learning process.

Backward Pass (Backpropagation): Backpropagation was applied to compute the gradients of weights and biases using the chain rule. The derivative of ReLU was used in the hidden layers to propagate the error correctly. These gradients indicated how much each weight and bias should be adjusted to reduce the loss in the next iteration.

Weight Update (Gradient Descent): Weights and biases were updated using gradient descent according to the rule where the learning rate was set to 0.003. This update rule allowed the network to iteratively reduce the error.

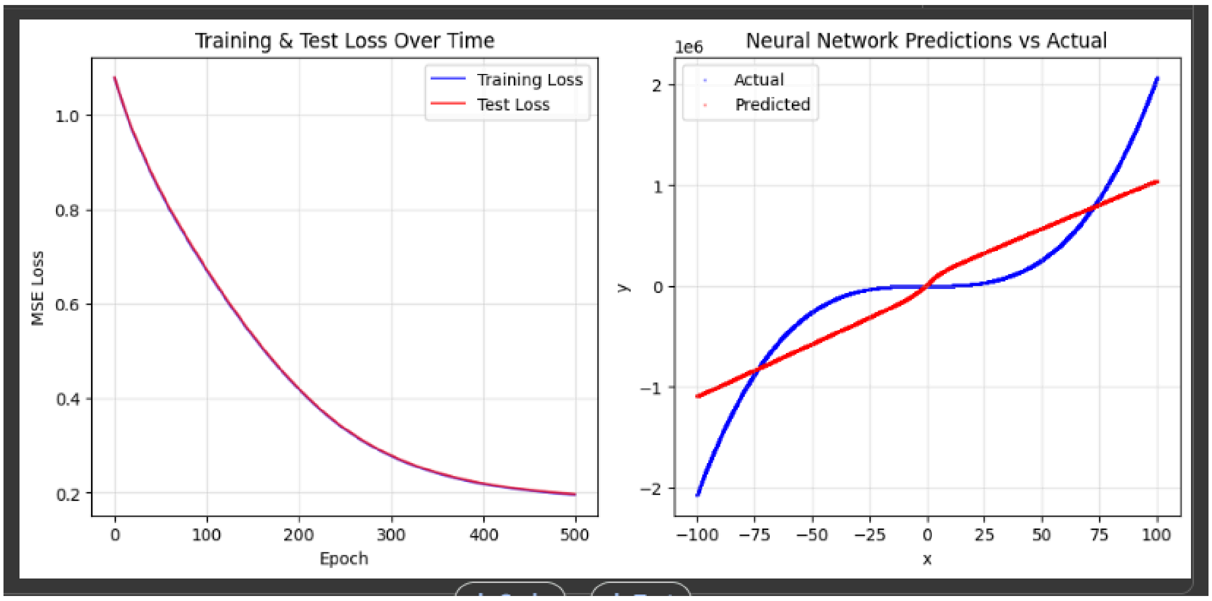
Training Process: The network was trained for a maximum of 500 epochs with early stopping enabled. Training was halted if the test loss did not improve for 10 consecutive epochs, which prevented overfitting. Both training and test losses were recorded at each epoch to monitor performance.

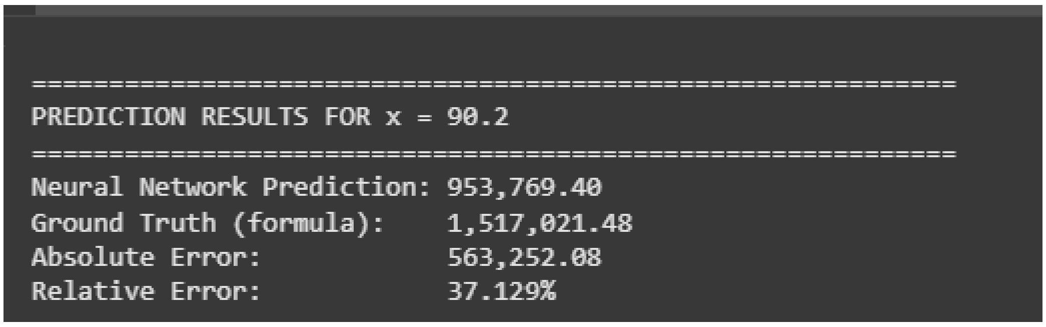
Evaluation and Visualization: After training, the performance of the model was evaluated using different plots. Loss curves were plotted for both training and test sets to check convergence. A predicted vs actual scatter plot was generated to compare outputs with the true values, while a residual plot was used to analyze the distribution of errors. These visualizations gave insights into the accuracy and generalization of the trained network.

**Results:**

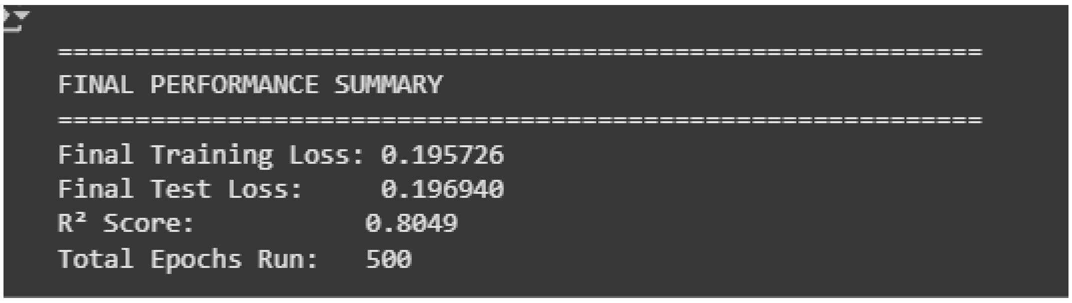
| Experiment | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Learning Rate | 0.005 | 0.001 | 0.005 | 0.005 | 0.005 |
| Batch Size | Full Batch | Full Batch | Full Batch | Full Batch | Full Batch |
| Number of Epochs | 500 | 200 | 700 | 200 | 200 |
| Activation Function | ReLU | ReLU | ReLU | Leaky ReLU | Sigmoid |
| Final Training Loss | 0.195726 | 0.613141 | 0.174910 | 0.417355 | 0.986931 |
| Final Test Loss | 0.196940 | 0.616774 | 0.176008 | 0.419014 | 0.996372 |
| R^2 Score | 0.8049 | 0.389 | 0.8257 | 0.5850 | 0.0131 |

Screenshots for Experiment 1:





```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 953,769.40
Ground Truth (formula):    1,517,021.48
Absolute Error:            563,252.08
Relative Error:            37.129%
```



```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.195726
Final Test Loss:     0.196940
R² Score:           0.8049
Total Epochs Run:   500
```

### Discussion on Performance

The final performance summary shows a training loss of **0.1957** and a test loss of **0.1969**, with both values being very close to each other. This indicates that the model is not overfitting, as the gap between training and test errors is minimal. At the same time, the losses are relatively low, which means the model has learned the underlying function well without underfitting.

The **$R^2$ score of 0.8049** further supports this conclusion. An $R^2$ score close to 1 suggests a strong correlation between the predicted and actual values, and here the score shows that the network is able to explain around **80% of the variance** in the dataset. This level of performance is quite good, especially considering that the dataset included noise.

The model was trained for the full **500 epochs**, and the steady decrease in both training and test losses indicates that the learning process was stable. Since early stopping was not triggered, the model continued to improve throughout the training process without diverging or showing signs of overfitting.

Overall, the results suggest that the neural network achieved a good balance between bias and variance. The model generalized well to unseen data and successfully captured the complexity of the cubic + sine function.

### Conclusion

The lab demonstrated how an artificial neural network can be built and trained from scratch to approximate a complex function. By implementing all the core components such as weight initialization, activation functions, forward and backward propagation, and gradient descent, the working of a neural network became much clearer. The chosen dataset, based on a cubic polynomial with an added sine term and noise, was successfully modeled using a two-hidden-layer network with ReLU activations.

The training process showed a steady decrease in both training and test losses, and the final evaluation confirmed that the model was able to closely match the actual function. The loss curves, predicted versus actual plots, and residual analysis highlighted that the network generalized well without significant overfitting. Overall, the lab provided practical experience in understanding how neural networks learn and reinforced important concepts such as early stopping, gradient-based optimization, and the role of activation functions in capturing non-linear patterns.