ML Lab

Name: Cheruku Manas Ram
SRN: PES2UG23CS147
Course: Machine Learning
Date: 16/09/2025
Section: 5 ‘C’

Introduction

The purpose of this lab it to see the ability of ANN’s to capture and learn complex non linear relationships. We are building a neural network with one input and output layer with 2 hidden layers in between.
We are creating a dataset using SRN and a quartic polynomial and then adding noise to make it more realistic. We are using this dataset to train an ANN using ReLu and Mean square error as the activation function and loss function respectively. We are then observing the training and testing loss multiple times for different hyperparameters. Using these models, we are visualising the predictions on a graph to see how well the model captured the function

Dataset Description

Polynomial Type: QUARTIC: $y = 0.0124x^4 + 1.94x^3 + -0.95x^2 + 4.07x + 11.75$
Number of samples: 100,000
Number of features: 1
Noise level: $\varepsilon \sim N(0, 2.44)$
Architecture: Input(1) → Hidden(96) → Hidden(96) → Output(1)
Learning Rate: 0.003
Architecture Type: Large Balanced Architecture

Methodology

1. **Dataset Generation and Preprocessing:**
    - A synthetic dataset was generated based on an assigned polynomial function and a specified noise level.
    - The dataset was split into training and testing sets to evaluate the model's performance.
    - Input features and target values were scaled using to normalize their ranges.
2. **Neural Network Architecture:**
    - A feedforward neural network with a single input layer, two hidden layers, and a single output layer was implemented.
    - The number of neurons in the hidden layer was decided by the SRN, i was given 96 for both hidden layers
    - The output layer has one neuron for predicting the continuous target variable.
3. **Activation Functions:**
    - The ReLU (Rectified Linear Unit) activation function was primarily used in the hidden layers to introduce non-linearity.

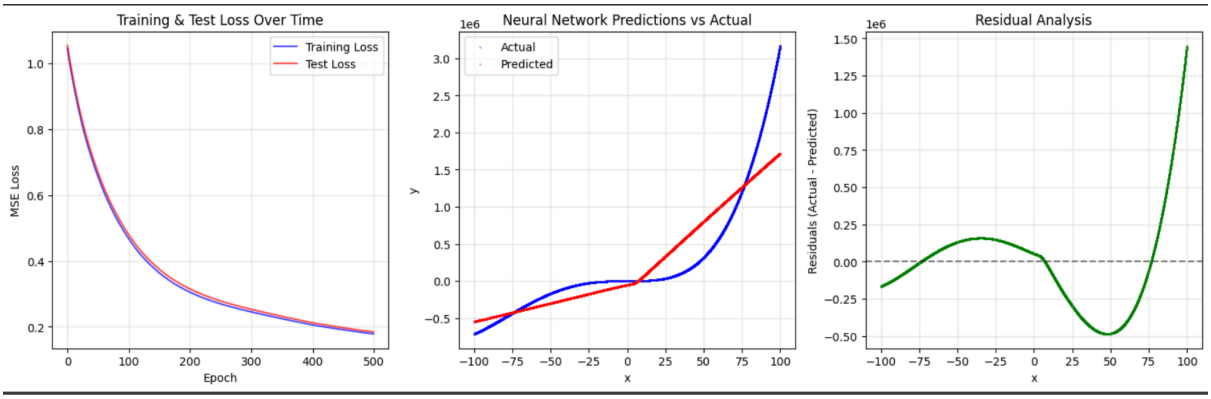
    - The derivative of the chosen activation function was implemented for use in the backward propagation step.
4. **Loss Function:**
    - The Mean Squared Error (MSE) was used as the loss function to quantify the difference between the network's predictions and the true target values.
5. **Weight Initialization:**
    - Xavier initialization was employed to initialize the weights of the network. This method helps in maintaining a suitable variance of activations across layers, mitigating vanishing or exploding gradients. Biases were initialized to zeros.
6. **Training Process:**
    - The network was trained using gradient descent.
    - The forward pass computed the network's predictions.
    - The backward pass calculated the gradients of the loss function with respect to the weights and biases using the backpropagation algorithm.
    - The weights and biases were updated using the calculated gradients and a specified learning rate.
    - Early stopping was implemented to prevent overfitting and determine the optimal number of training epochs. Training stopped if the test loss did not improve for a certain number of epochs (patience).
7. **Evaluation:**
    - The trained model was evaluated on the test set to assess its performance and generalization ability.

Results and Analysis

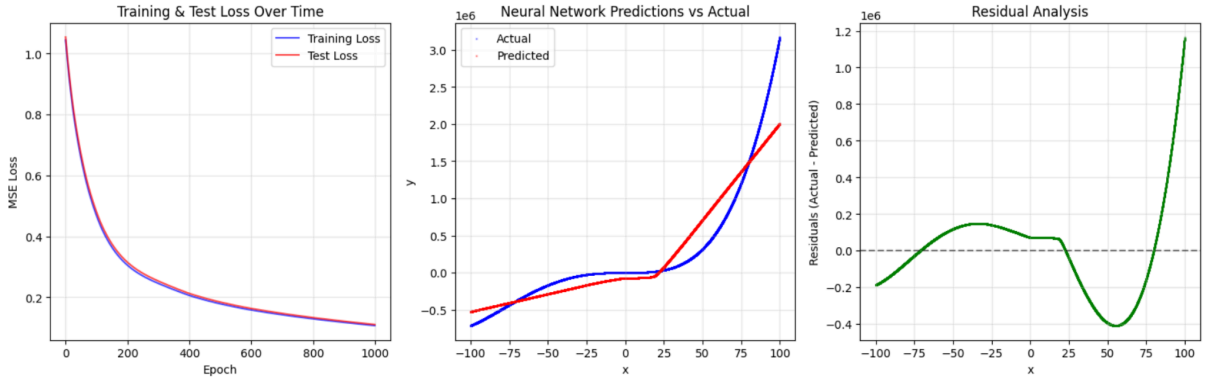
Experiment	Learning Rate	Hidden 1 and hidden2	No. of Epochs	Optimizer	Activation Function	Final Training Loss	Final Test Loss	R square
Baseline	0.003	96, 96	500	Gradient Descent	ReLU	0.178754	0.184399	0.8199
1.	0.003	96, 96	**1000**	Gradient Descent	ReLU	0.107135	0.110488	0.8921
2.	**0.005**	96, 96	500	Gradient Descent	ReLU	0.124967	0.128785	0.8742
3.	**0.005**	96, 96	**1000**	Gradient Descent	ReLU	0.060295	0.062210	0.9392

4.	**0.005**	**72, 32**	500	Gradient Descent	ReLU	0.153703	0.158475	0.8452

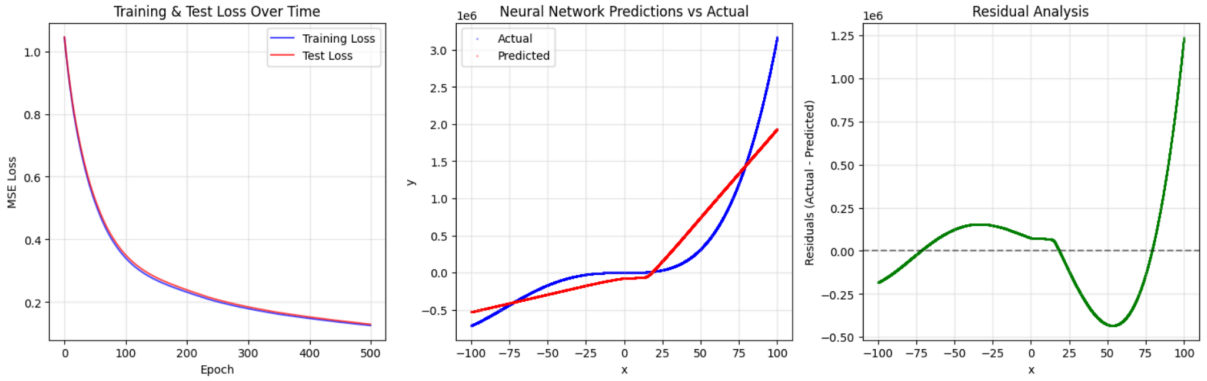
Baseline



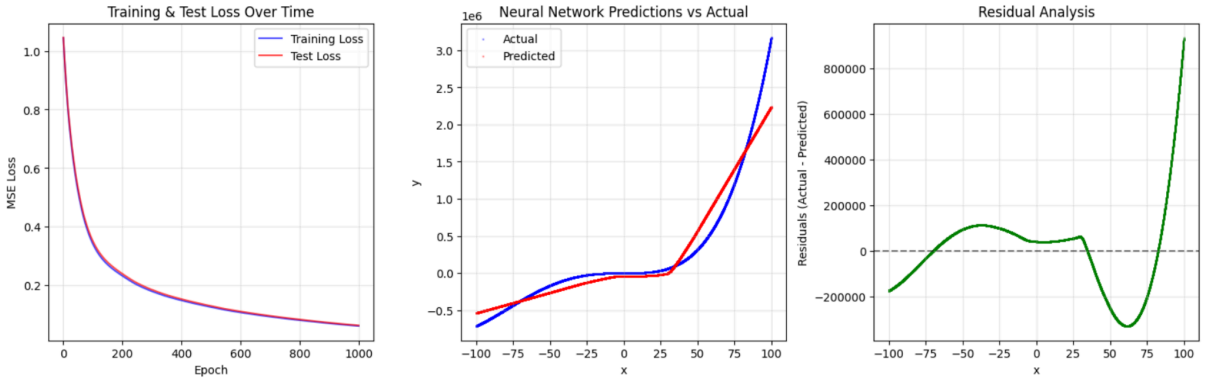
Epochs = 1000



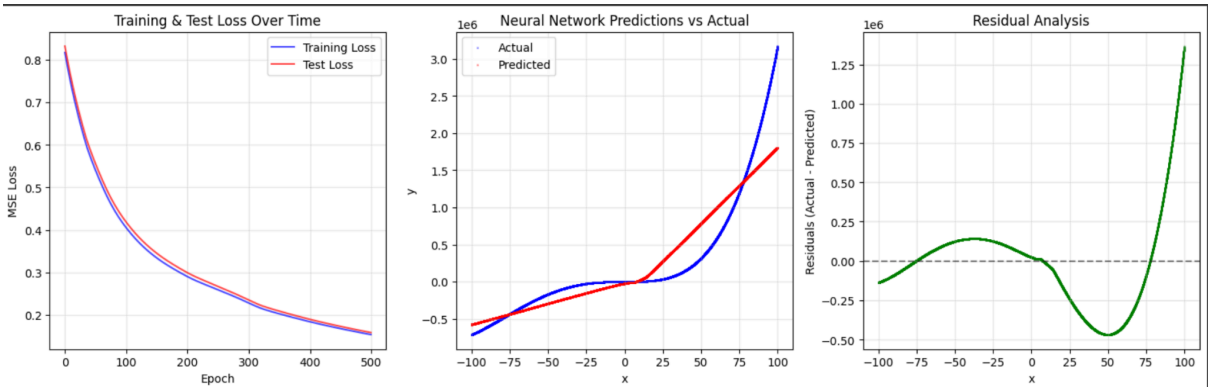
Learning rate = 0.005



Learning rate = 0.005 and Epochs = 1000



Learning rate = 0.005 and hidden layer 1 = 72, hidden layer 2 = 32



From the table, we can see that the best performing model is when we set the learning rate to 0.005 with 1000 epochs, which performed well with an r square value of 0.9392, and a final training and testing loss of 0.06

The base model performed well but relatively the worst out of all the other experiments with changed hyperparameters, we can see that by increasing the learning rate, the model converges better indicating that a learning rate of 0.003 was too slow for the base model to converge in only 500 epochs. In another experiment we can see that giving the model with 0.005 learning rate more epochs to converge was the best where it was able to get a loss of only 0.06 and a good explainability of variance with an r square value of 0.9392. We can also notice that giving 96 nodes each to the hidden layers was obviously the better and balanced choice, reducing the number of nodes in each hidden layer and making the model a wide-to-narrow architecture reduced the performance of the model as expected.

Conclusion

This lab showed that for a qautric equation, (equation given to me - $y = 0.0124x^4 + 1.94x^3 + -0.95x^2 + 4.07x + 11.75$), a slightly higher learning rate of 0.005 with a few more epochs would allow us to make our model better faster than a learning rate of 0.003 which didn't have enough number of epochs to make a good model fast enough. The final results of the baseline model was a loss of approximately 0.18 in both the test and training with an r square value of 0.8199. But by changing the hyperparameters slightly, we were able to get a better model in our third experiment with a loss of only 0.06 and an r-square value of 0.9392.