UE23CS352A: MACHINE LEARNING

Week 6: Artificial Neural Networks

Name: Prakyath P Nayak

SRN: PES1UG23CS431

Date: 13-09-2025

Introduction

Purpose of the lab: The purpose of this lab is to gain hands-on experience in building and training an artificial neural network (ANN) from scratch to perform function approximation

Task performed: A dataset was generated using a polynomial and my SRN and I was given the task to complete the code to build a neural network which would approximate this function using linear layers and simple gradient descent

Dataset Description

For this lab, a synthetic dataset was generated based on my SRN **PES1UG23CS431**. The data follows a cubic polynomial function $y=2.45x3−0.53x2+5.94x+11.99$y = 2.45x^3 - 0.53x^2 + 5.94x + 11.99$y=2.45x3−0.53x2+5.94x+11.99$ with added Gaussian noise ($\varepsilon \sim N(0,1.63)$) to make the task more realistic. The dataset contains 100,000 samples, which were split into 80% for training and 20% for testing. Both the input and output values were standardized using “StandardScaler” to improve the neural network’s training efficiency.

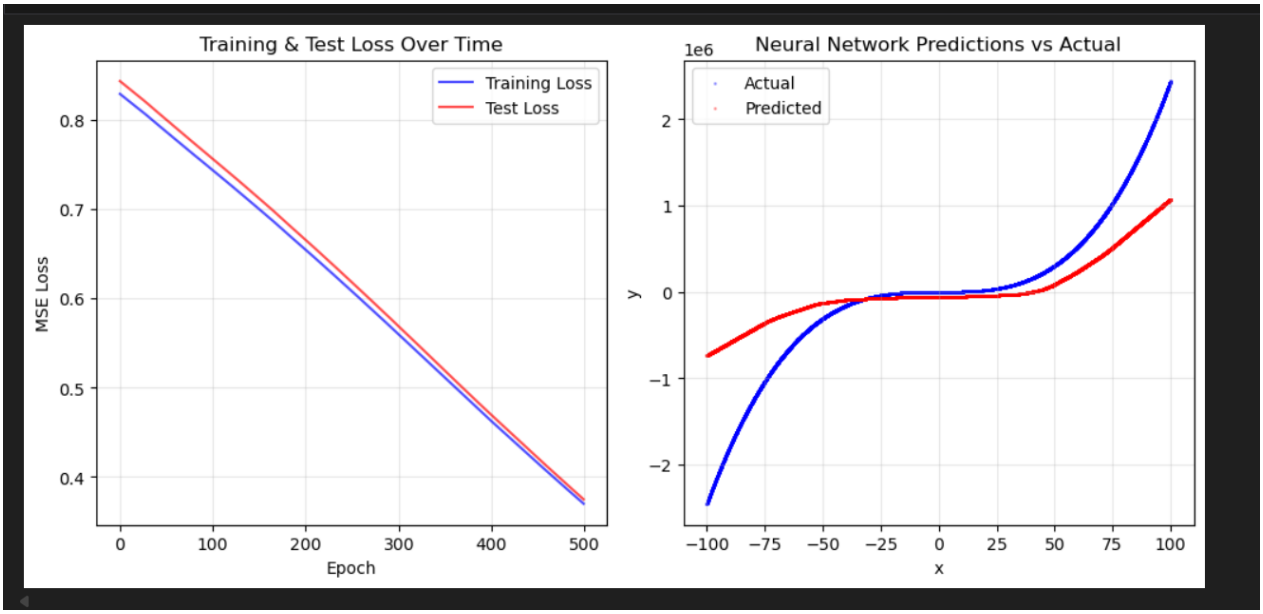
Methodology

The neural network used in this lab was implemented from scratch and followed the architecture: Input(1) → Hidden(96) → Hidden(192) → Output(1), which is a large

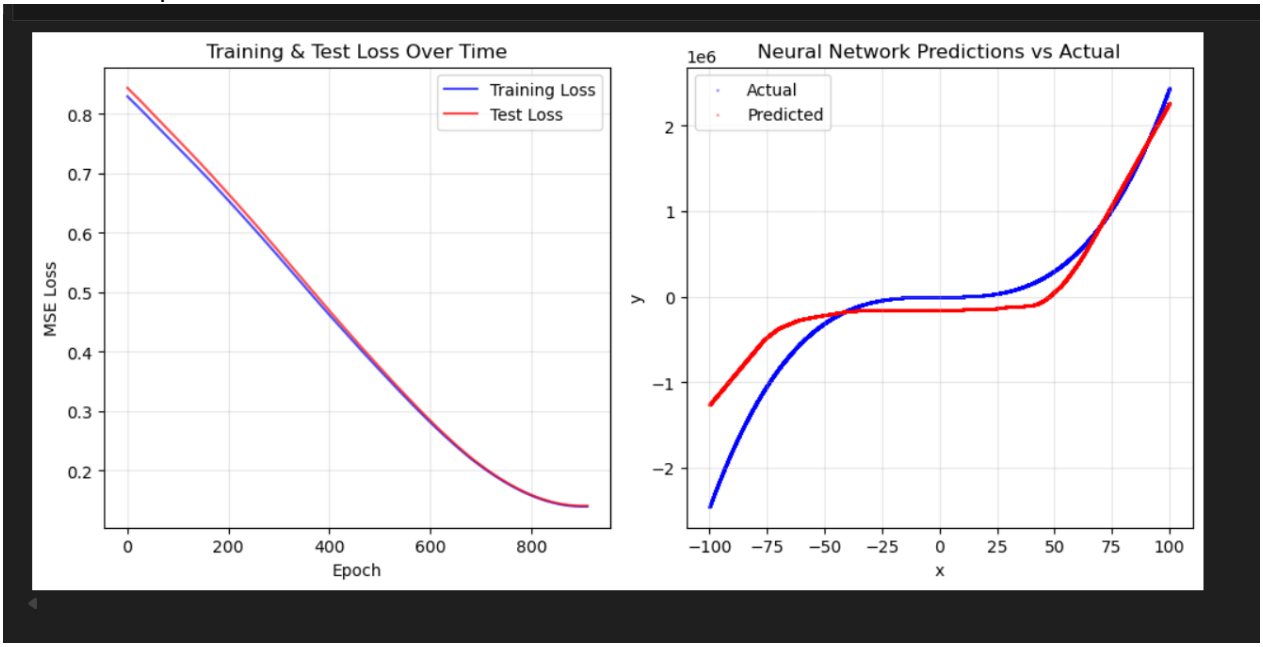
balanced architecture suitable for learning complex non-linear relationships. Initially, the model was trained using gradient descent with a learning rate of 0.003. The training loop included forward propagation, backpropagation, and weight updates, along with early stopping (patience of 10 epochs) to prevent overfitting. I wanted to experiment a bit with the learning rate. Increasing it seemed to increase the model's performance as it can be guessed that the learning rate of 0.003 can be too low for a linear model which is trying to fit to a simple polynomial function. Before getting into optimizers, I wanted to try out playing with the number of epochs. Reducing it, obviously, hurt the model performance. This is justified as reducing the number of epochs is just the same as giving an early stop. Then, I tried to bring in ADAM optimizer to see what the results would be, and as expected, I had bought a nuke to a gun fight. It gave highly efficient results and just stopped after ~150-200 epochs. But, the curve fitting, while wasn't perfect, seemed too aggressive. I wanted to see how far I could get without changing the patience level. I lowered the learning rate a bit and that seemed to do the work.

Results and Analysis

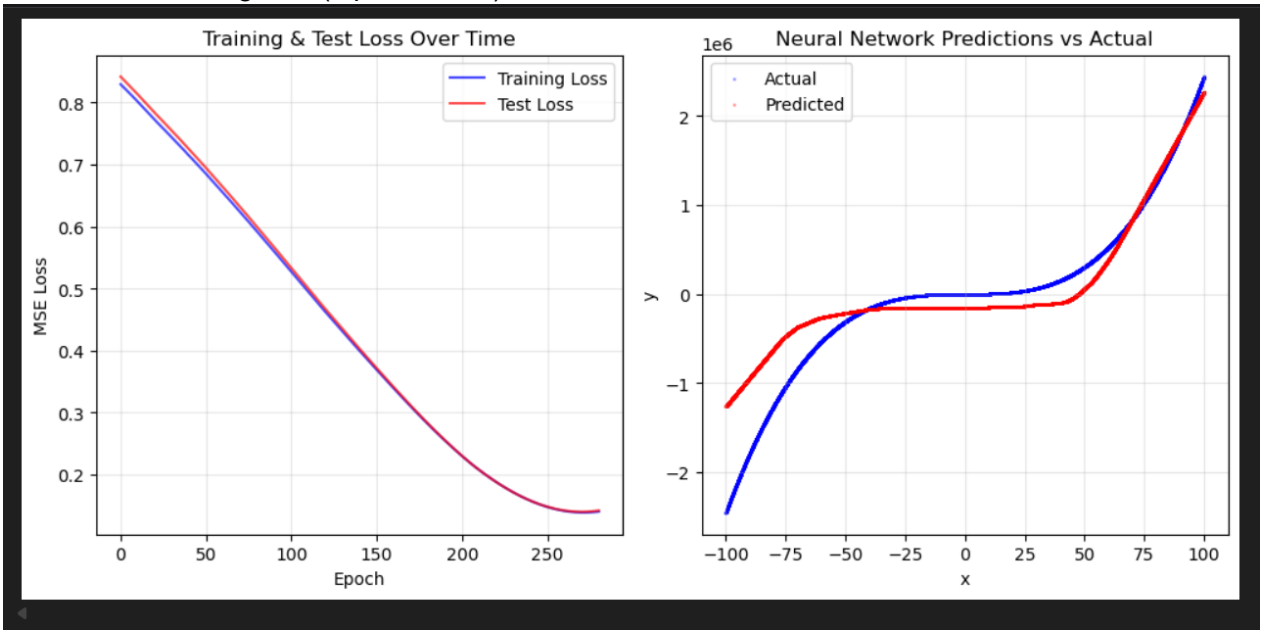
Train loss curve:
Base model



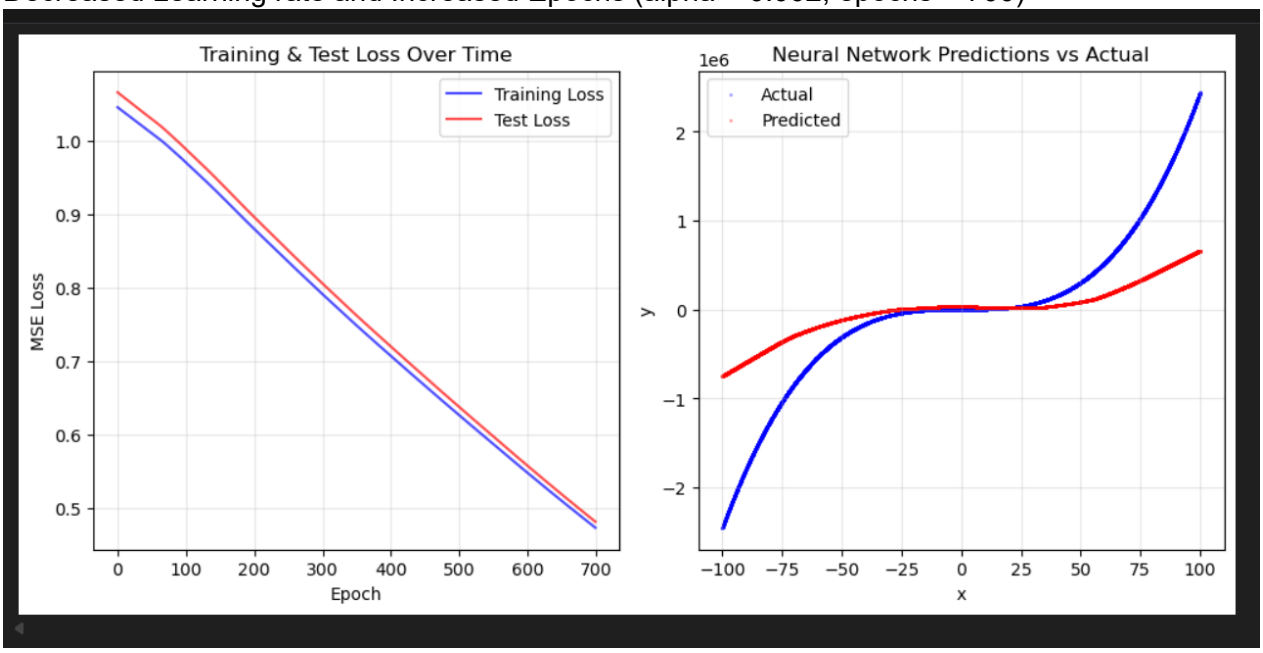
Increased Epochs



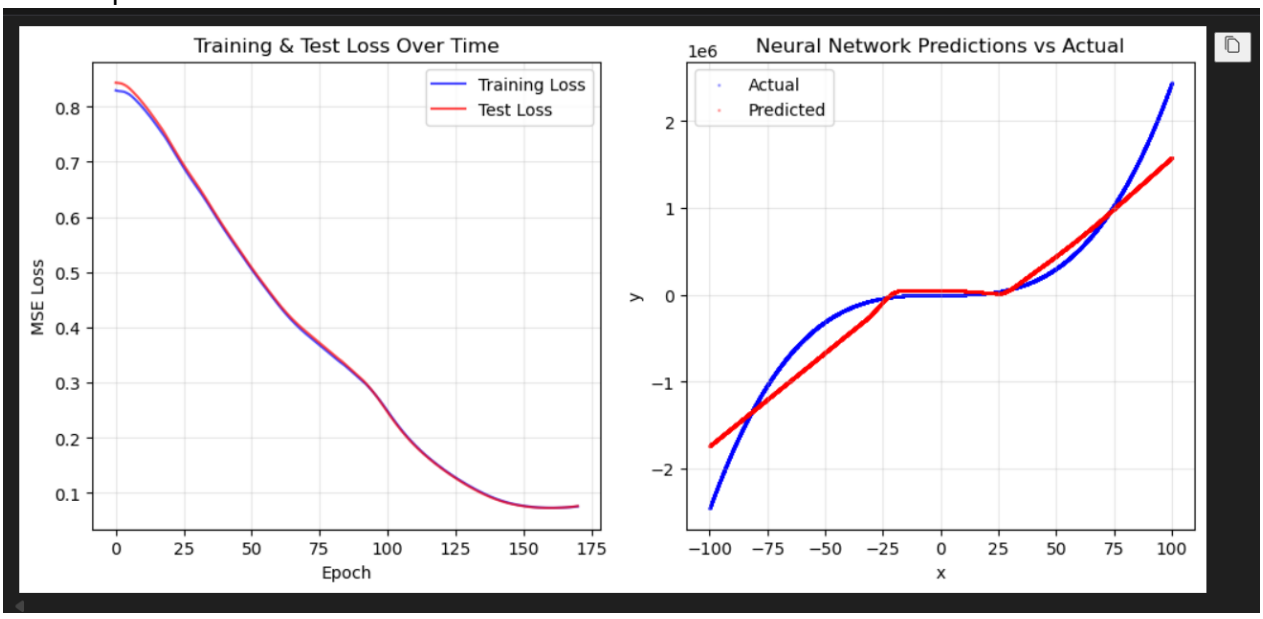
Increased Learning rate (alpha = 0.01)



Decreased Learning rate and Increased Epochs (alpha = 0.002, epochs = 700)



ADAM optimizer



As we are fitting over a simple function with some error, I saw that if I increased the number of epochs way too much the model would then try to “learn” the error, which would cause the function approximation to deviate from the actual function. ADAM optimiser did a pretty good job dealing with this. But, I may have not tuned the parameters correctly to its requirements and it hit early stopping due to the low patience level. Final loss seemed to be less than most of the other models but the final absolute and relative errors were way more than the other models.

Result table:

Experiment	Learning rate	epochs	optimizer	activation functior	final training	final test l	R^2 score
Base model	0.003	500	none	Relu	0.369646	0.374541	0.6325
High epoch	0.003	1000	none	Relu	0.139688	0.140888	0.8619
High LR	0.01	500	none	Relu	0.140595	0.142249	0.8625
Mix	0.002	700	none	Relu	0.473553	0.473553	0.5273
ADAM	0.001	500	ADAM	Relu	0.178811	0.182437	0.8224

Conclusion

From the experiments conducted, it is evident that the neural network's performance is highly influenced by the choice of hyperparameters. Increasing the number of epochs significantly improved performance, as seen in the High Epoch model ($R^2 \approx 0.86$) compared to the Base Model ($R^2 \approx 0.63$). Similarly, using a higher learning rate of 0.01 (High LR) also yielded strong results ($R^2 \approx 0.86$) with lower training and test losses, showing that the model was able to converge faster and learn the underlying pattern effectively. Introducing the ADAM optimizer further improved performance ($R^2 \approx 0.82$) while maintaining relatively low losses, indicating more stable and efficient training compared to plain gradient descent. On the other hand, the Mix configuration with 700 epochs and a lower learning rate (0.002) performed poorly ($R^2 \approx 0.53$), suggesting that the combination led to slower convergence and possible underfitting.