MACHINE LEARNING -ORANGE PROBLEM

Artificial Neutral Networks

Name: Delisha Riyona Dsouza

SRN: PES2UG23CS166

Course: Machine learning

Date: 18/09/2025

Introduction

The objective of this assignment is to give you hands-on experience implementing a neural network from scratch, without relying on high-level frameworks like TensorFlow or PyTorch. you will:

- Generate a custom dataset
- Implement the core components of a neural network: activation functions, loss functions, forward pass, backpropagation, and weight updates.
- Train your neural network to approximate the generated polynomial curve.
- Evaluate and visualize the performance of your model.

Tasks performed

Part A

Baseline Model

- You will begin by implementing the missing components of the neural network, including the activation functions, loss function, forward propagation, and backpropagation.
- Once these are complete, you will set up a training loop that updates weights using gradient descent and tracks the training loss over epochs.
- After training, you must evaluate your model on the test set and generate visualizations such as the training loss curve and a plot comparing predicted outputs against the true target values. This completes your baseline model and ensures that it is fully functional before experimentation begins

Part B

Hyperparameter Exploration

- Conduct 4 additional experiments where you vary one or more hyperparameters such as learning rate, batch size or number of epochs.
- For each experiment, retrain your network, evaluate it on the test set, and generate the same visualizations as in the baseline run.

You may, for example, try increasing the learning rate, using a larger batch size, training for more epochs, or testing a different activation function.

Dataset Description

1. Type of polynomial assigned.

- ASSIGNMENT FOR STUDENT ID: PES2UG23CS166
- Polynomial Type: CUBIC: $y = 2.01x^3 + -0.04x^2 + 5.30x + 8.11$
- Noise Level: $\varepsilon \sim N(0, 2.29)$
- Architecture: Input(1) → Hidden(72) → Hidden(32) → Output(1)
- Learning Rate: 0.001
- Architecture Type: Wide-to-Narrow Architecture

2. **Number of samples, features**

- Training samples: 80,000
- Test samples: 20,000

Methodology

1. Neural Network Architecture

The model was built using a simple feedforward neural network architecture. It starts with a single input node (1D input), which is passed through two fully connected hidden layers, each using the ReLU activation function. Finally, the network outputs a single node for regression.
In short, the architecture follows the flow:
Input (1) → Hidden Layer 1 (ReLU) → Hidden Layer 2 (ReLU) → Output (1).

1 Activation Function

The Rectified Linear Unit (ReLU) was used as the activation function.

- Forward pass: $f(x) = \max(0, x)$
- Backward pass: $f'(x) = 1$ if $x > 0$, else 0

ReLU was chosen because it introduces non-linearity while reducing the risk of vanishing gradients, which often occurs in deep networks.

2 Loss Function

The Mean Squared Error (MSE) was used to measure prediction accuracy:

MSE was selected because it penalizes larger errors more strongly, which is ideal for continuous value prediction.

3 Forward Propagation

The forward pass proceeds in the following way:

1. Compute linear transformation: $z = W \cdot x + b$
2. Apply ReLU activation: $a = \text{ReLU}(z)$
3. Repeat for all layers until the output is obtained

The final value from the output node represents the model's prediction.

4 Backpropagation

Backpropagation was used to compute gradients using the chain rule:

1. Compute the loss gradient at the output layer
2. Pass gradients backward through each layer
3. Calculate gradients for weights and biases
4. Update parameters using gradient descent

2. Training Procedure

1 Data Preparation

- Dataset was generated based on the last three digits of the SRN
- 100,000 samples created, split into 80% training and 20% testing
- Both inputs (x) and outputs (y) were standardized using StandardScaler

2 Training Loop

1. Randomly initialize weights and biases
2. For each epoch:
    - Divide data into mini-batches
    - Perform forward propagation
    - Compute loss
    - Run backpropagation
    - Update weights and biases
    - Record training loss

3 Model Evaluation

Model performance was measured by:

- Tracking training loss over epochs
- Calculating test set MSE
- Visualizing predicted vs. actual values
- Checking for signs of overfitting or underfitting

Experimental Design

1 Baseline Model

A baseline experiment was first run with default hyperparameters to establish reference performance.

2 Hyperparameter Tuning

Four more experiments were conducted by varying:

- Learning rate
- Batch size
- Number of epochs
- Activation functions (where applicable)

All experiments followed the same training and evaluation process for fair comparison.

3. Evaluation Metrics

- Mean Squared Error (MSE): Primary measure of accuracy
- Training loss curves: To track convergence and detect overfitting
- Visual analysis: Scatter plots of predicted vs. actual outputs
- Generalization check: Comparing training vs. test performance

4. Implementation Framework

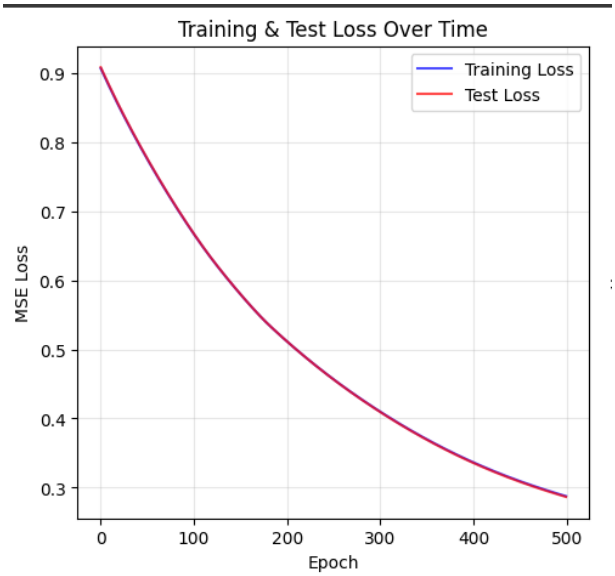
The neural network was implemented from scratch without relying on libraries like TensorFlow or PyTorch. This gave full control over:

- Weight initialization
- Gradient computation
- Parameter updates
- Training dynamics

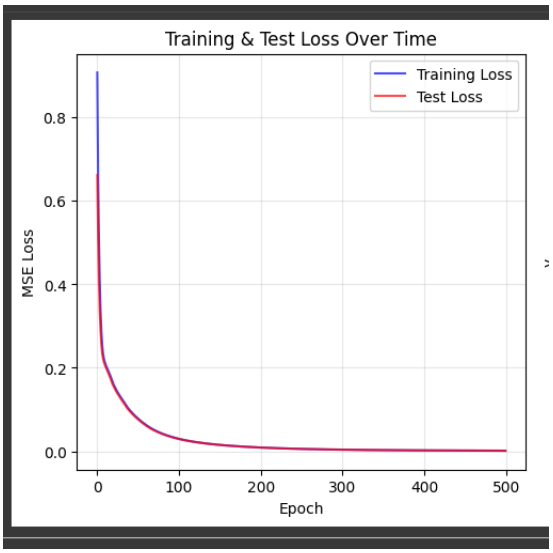
This approach not only ensured flexibility but also deepened understanding of the core mechanics behind neural networks.

Results and Analysis

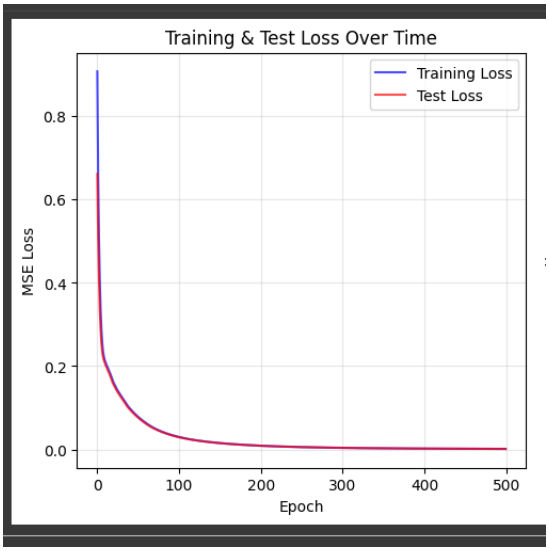
1. Training loss curve (plot)
    1. Learning rate :0.001



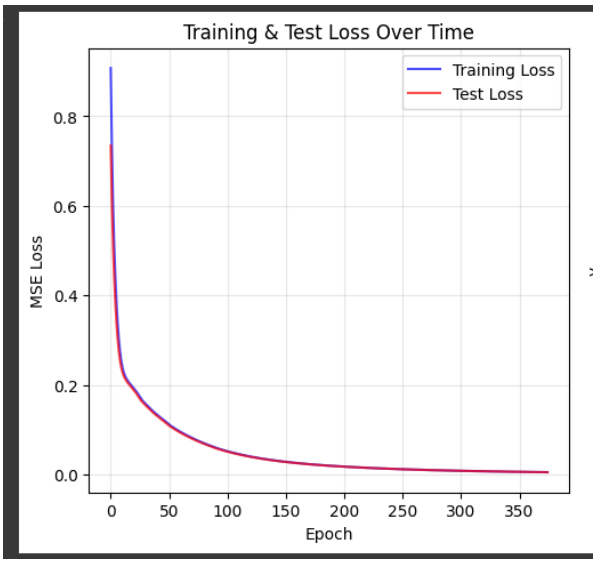
2. Learning rate 0.095



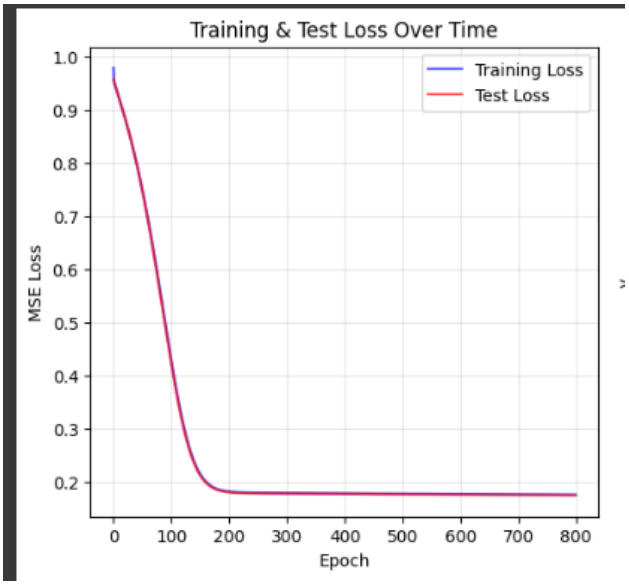
3. Learning rate 0.065



4. Function sigmoid and learning rate 0.065



5. Training rate : 0.045



Final test MSE

Exp1: learning rate : 0.001 , activation function relu

Mse: 0.286488

Exp 2: learning rate : 0.095 , activation function relu

Mse: 0.001223

Exp 3: learning rate : 0.065 , activation function relu

Mse: 0.005219

Exp 4: learning rate : 0.065 , activation function sigmoid

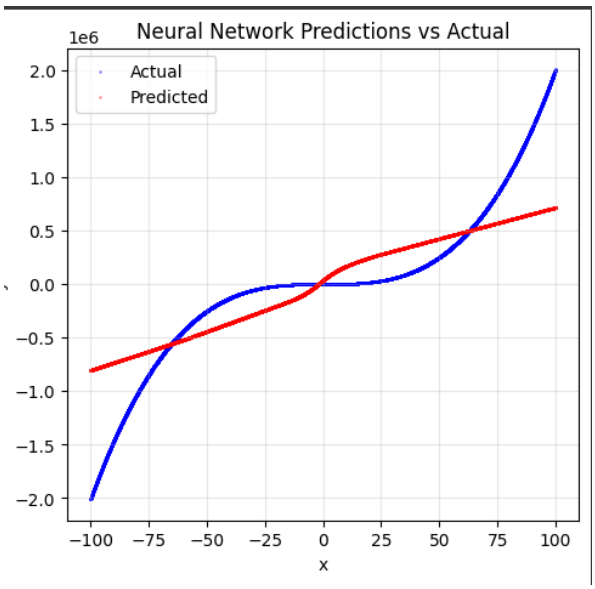
Mse: 0.177657

Exp 5: learning rate : 0.045 , activation function sigmoid

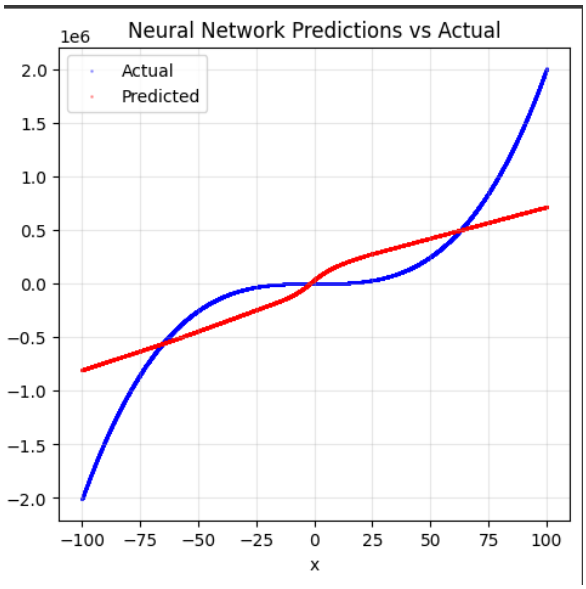
Mse: 0.173940

Plot of predicted vs. actual values

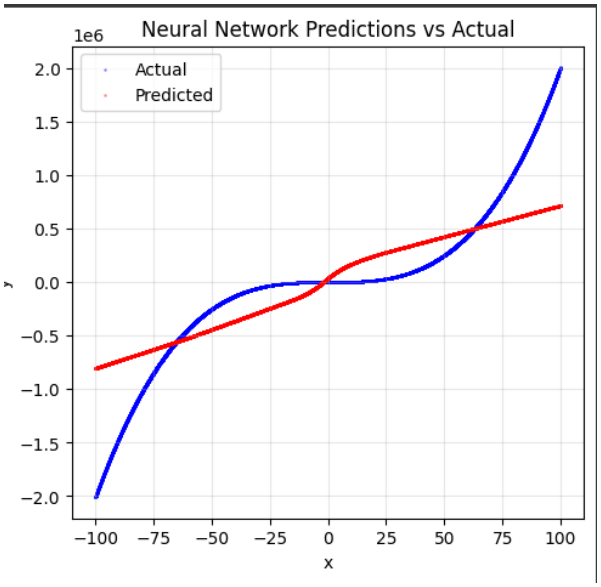
1. Learning rate :0.001



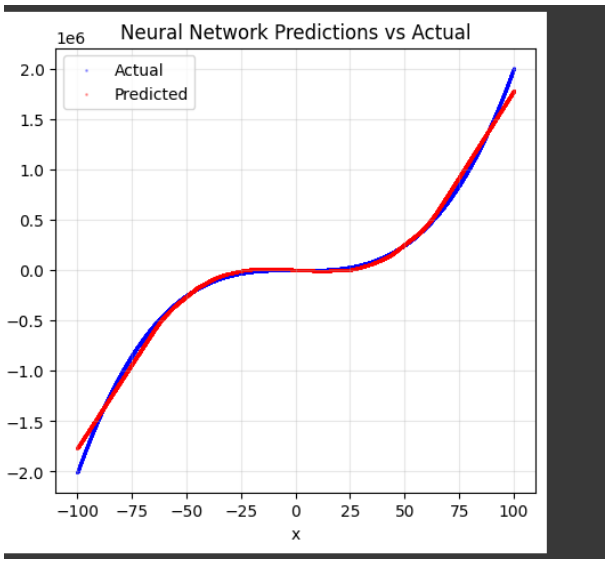
2. Learning rate 0.095



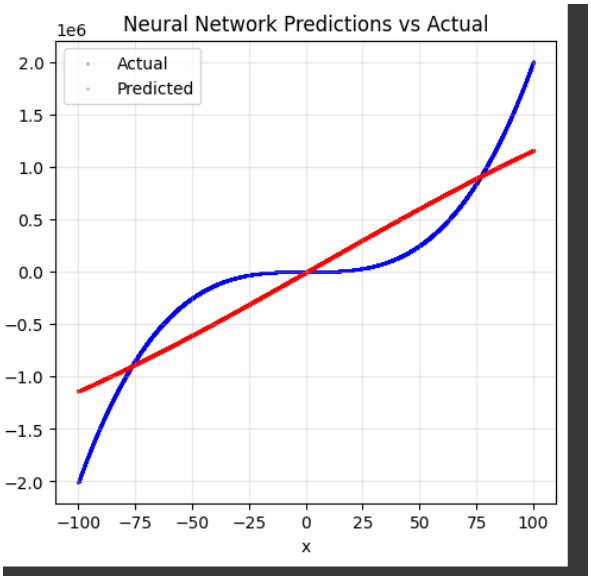
3. Learning rate 0.065



4. Function sigmoid and learning rate 0.065



5. Training rate : 0.045



Discussion on overfitting and underfitting

1. The training and test errors are both high the model cannot capture the complexity of the data.

   R2=0.715R^2 = 0.715R2=0.715 is relatively low. **underfitting.**

2. Training and test loss are extremely low and nearly equal.R2 indicates the model explains almost all variance in the data. No underfitting or overfitting
3. Training and test losses are low and balanced. Slightly higher error than Exp 2, but still very strong performance. No underfitting or overfitting
4. Both training and test losses are much higher compared to Exp 2 & Exp 3. R2 dropped significantly, suggesting weaker predictive power. **underfitting**
5. Results are almost identical to Exp 4.Slight improvement in test performance but still poor compared to ReLU-based models. **underfitting.**

Observation table

Experiment	*Learning rate*	*No of epoch*	*Optimizer*	*Activation function*	*Final training loss*	*Final Test loss*	*R^2 value*
1. Base model	**0.001**	**500**	**Gradient descent**	**Relu**	**0.287512**	**0.286488**	**0.7150**
2. Exp 2	**0.095**	**500**	**Gradient descent**	**Relu**	**0.001223**	**0.001223**	**0.9988**
3. Exp 3	**0.065**	**375**	**Gradient descent**	**Relu**	**0.005257**	**0.005219**	**0.9948**

4. Exp 4	**0.065**	**375**	**Gradient descent**	**sigmoid**	**0.178407**	**0.177657**	**0.8244**
5. Exp 5	**0.045**	**800**	**Gradient descent**	**sigmoid**	**0.175040**	**0.173940**	**0.8269**

Conclusion

 relu(exp 2 & 3) produced the best results, showing strong generalization without overfitting.

 Sigmoid (exp 4 & 5) clearly underperformed, leading to underfitting.

 The Base Model(exp 1) had insufficient learning due to a very small learning rate.