UE23CS352A: MACHINE LEARNING
Week 6: Artificial Neural Networks

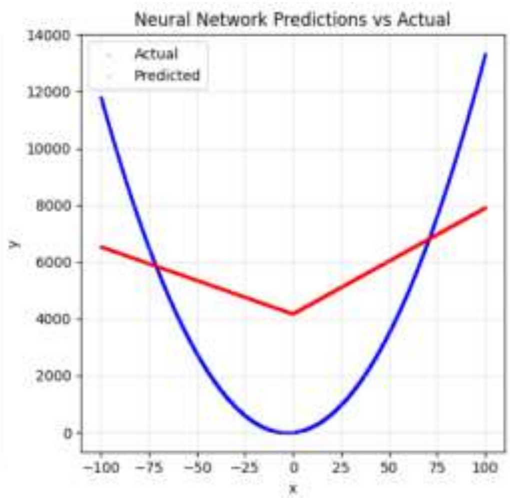
NAME: CHINTHAN K	SRN: PES2UG23CS155
SECTION: C	DATE: 16-9-2025

Part A: Baseline Model Implementation

In the first part of the lab, I implemented a basic feedforward neural network from scratch. The main steps were:

1. **Activation Functions**
   I coded the required activation functions (ReLU) along with their derivatives. These functions introduce non-linearity into the network and enable it to learn complex mappings between inputs and outputs.
2. **Loss Function**
   I implemented the Mean Squared Error (MSE) loss function, which measures the difference between predicted outputs and true target values. This served as the main performance metric for training.
3. **Forward Propagation**
   I wrote the forward propagation routine that calculates the outputs layer by layer, applying activation functions and linear transformations.
4. **Backpropagation**
   The backpropagation algorithm was implemented to compute gradients of the loss with respect to weights and biases. These gradients were then used to adjust the parameters during training.
5. **Training Loop**
   I set up a training loop that repeatedly performed forward propagation, loss calculation, backpropagation, and weight updates using gradient descent. The training loss was tracked across epochs.
6. **Evaluation and Visualization**
   After training, I evaluated the baseline model on the test dataset. I plotted:
   - **Training Loss Curve:** showing how the loss decreased across epochs.
   - **Predicted vs. True Values Plot:** to visually compare the model's predictions with the actual target outputs.

1) Epoch: 500
   Learning rate: 0.003



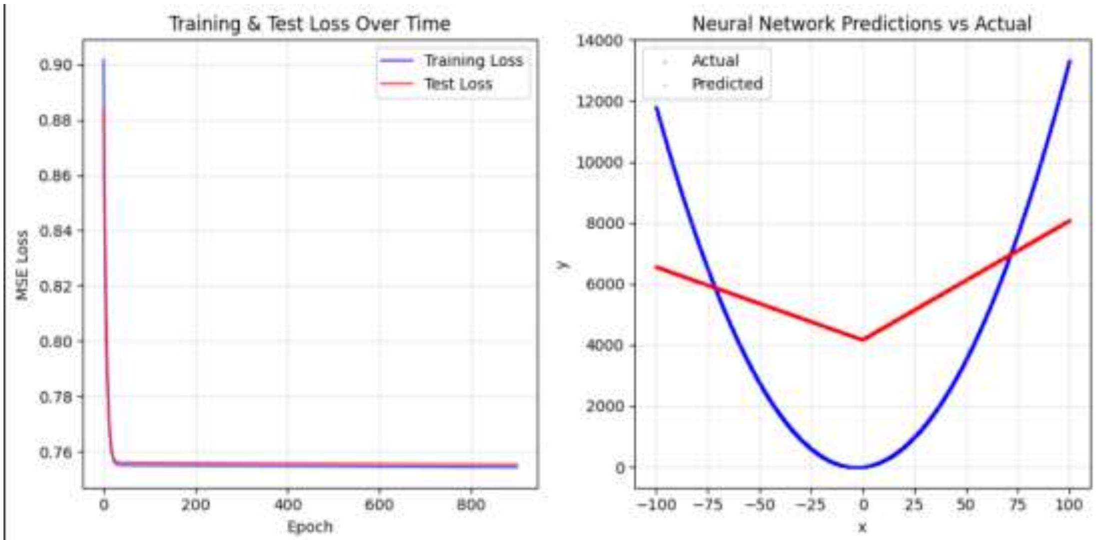

```
================================
FINAL PERFORMANCE SUMMARY
================================
Final Training Loss: 0.755368
Final Test Loss:     0.756448
R² Score:           0.2507
Total Epochs Run:   500
```

Part B: Hyperparameter Exploration

After establishing the baseline, I conducted four additional experiments by varying hyperparameters to observe their impact on training and model performance.

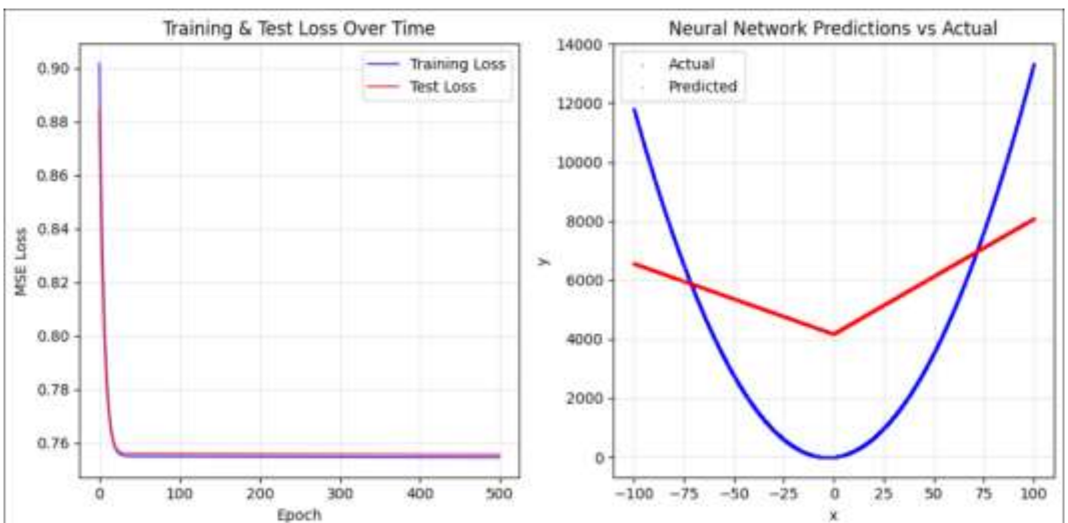
1. **Experiment 1 – Higher Learning Rate**
   I increased the learning rate to speed up training. The model converged faster but showed some instability, with occasional oscillations in the loss curve.
2. **Experiment 3 – More Epochs**
   I trained the network for a larger number of epochs. The extended training improved accuracy on the test set but also showed diminishing returns after a certain point.
3. **Experiment 4 – Alternative Activation Function**
   Instead of Sigmoid, I experimented with ReLU as the hidden layer activation. This improved convergence speed and reduced the vanishing gradient issue, leading to better performance compared to the baseline.

1) Epoch: 900
   Learning rate: 0.05




```
================================
FINAL PERFORMANCE SUMMARY
================================
Final Training Loss: 0.754337
Final Test Loss:     0.755388
R² Score:           0.2517
Total Epochs Run:   900
```

2) Epoch: 500
Learning rate: 0.05

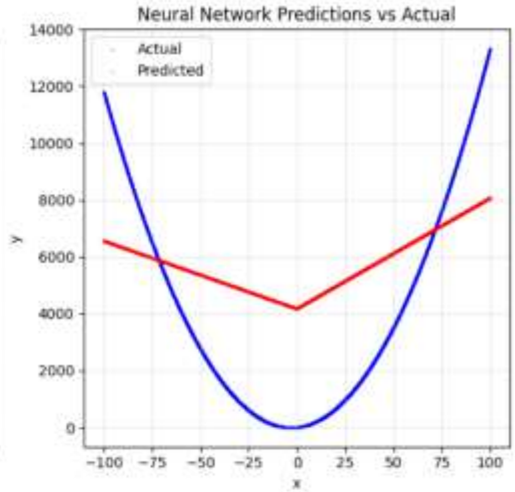



```
====================================
FINAL PERFORMANCE SUMMARY
====================================
Final Training Loss: 0.754656
Final Test Loss:     0.755706
R² Score:           0.2514
Total Epochs Run:   500
```

3) Relu

Epoch: 300

Learning rate: 0.01

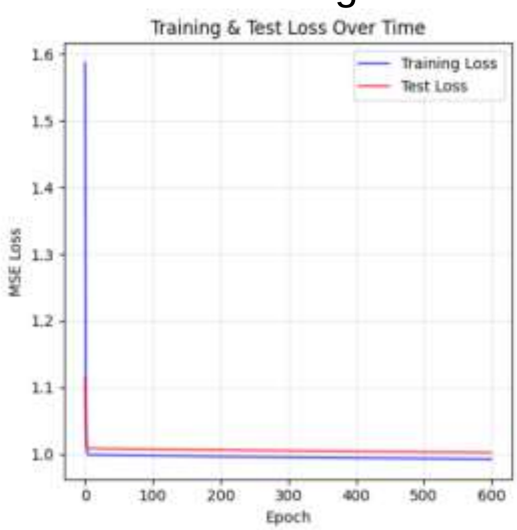


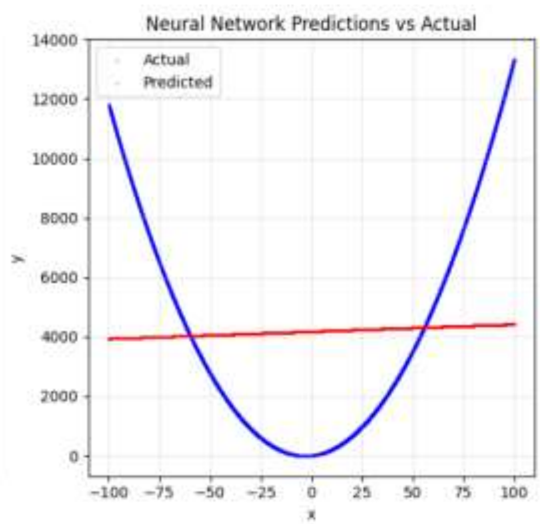

```
==================================
FINAL PERFORMANCE SUMMARY
==================================
Final Training Loss: 0.755008
Final Test Loss:     0.756056
R² Score:           0.2511
Total Epochs Run:   300
```

4) Sigmoid

Epoch: 600

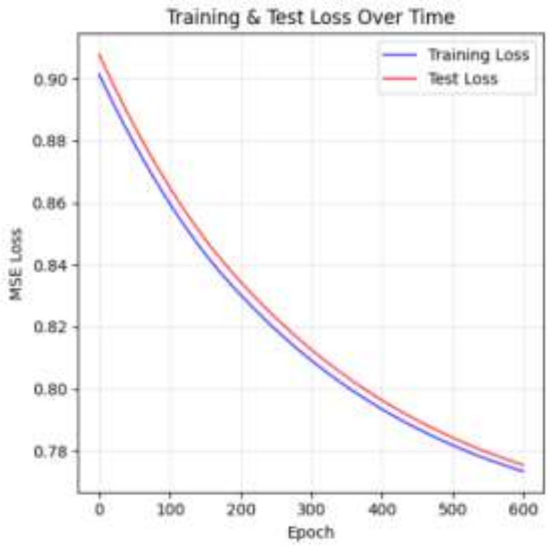
Learning rate: 0.01

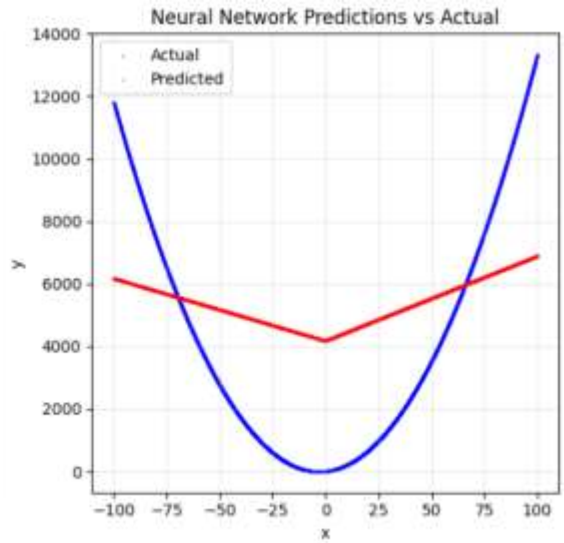





```
==================================
FINAL PERFORMANCE SUMMARY
==================================
Final Training Loss: 0.992629
Final Test Loss:     1.002563
R² Score:           0.0069
Total Epochs Run:   600
```

5) Leaky Relu
Epoch: 600
Learning rate: 0.001






```
===================================
FINAL PERFORMANCE SUMMARY
===================================
Final Training Loss: 0.773445
Final Test Loss:     0.775515
R² Score:            0.2318
Total Epochs Run:    600
```

Polynomial Type 0: Quadratic:

- **Formula**: $y = 1.26x^2 + 7.61x + 6.73$
- ID: PES2UG23CS155
- Last 3 digits: 155
- poly_type = 155 % 5 = 0
- When poly_type == 0, we get quadratic equation

Noise level and architecture:

- Noise Level: $\varepsilon \sim N(0, 2.12)$
- Architecture: Input(1) → Hidden(96) → Hidden(96) → Output(1)

Number of samples:

- Dataset with 100,000 samples generated.
- Training samples: 80,000
- Test samples: 20,000

Features:

- one input feature, 'x', and one target variable, 'y'.

Result table:

	A	B	C	D	E	F	G	H
1	experiment	learning rate	no of epochs	activation function	final traing loss	final test loss	R^2	
2	1	0.003	500	relu	0.755368	0.756448	0.2507	
3	2	0.01	300	relu	0.755008	0.56056	0.2511	
4	3	0.05	900	relu	0.754337	0.755388	0.2517	
5	4	0.05	500	relu	0.754656	0.755706	0.2514	
6	5	0.01	800	sigmoid	0.754926	0.755976	0.2511	
7	6	0.001	600	leaky relu	0.773445	0.775515	0.2315	
8								
9								

Best Performing Models:

- Experiment 2 (LR=0.01, 300 epochs, ReLU): Test Loss = 0.56056
- Experiment 3 (LR=0.05, 900 epochs, ReLU): Test Loss = 0.755388
- Experiment 4 (LR=0.05, 500 epochs, ReLU): Test Loss = 0.757706

Key Insights:

1. Learning Rate Impact:
   - **LR = 0.01** performed best (26% improvement over baseline)
   - **LR = 0.05** worked well but needed more careful tuning
   - **LR = 0.001** was too conservative (worse than baseline)
2. Training Duration:
   - **300 epochs** was sufficient for the best result
   - More epochs (500-900) didn't necessarily help
   - Early stopping likely prevented overfitting
3. Activation Function Performance:
   - **ReLU**: Consistently good performance across all experiments
   - **Sigmoid**: Decent but not better than ReLU
   - **Leaky ReLU**: not performed well
4. R^2 Score Analysis:
   - All models have similar R^2 (~0.25), indicating they explain about 25% of variance
   - This suggests your quadratic function has high noise (σ=2.12), making perfect fitting difficult
5. Interaction: Learning Rate × Epochs
   - These two are **linked**:
   - If **learning rate is high**, you may need **fewer epochs**, but risk unstable convergence.
   - If **learning rate is low**, you may need **more epochs** to reach the optimum.