Week 6: Artificial Neural Networks

Name	Diya D Bhat
SRN	PES2UG23CS183
Section	C
Date	16/09/25
Course name	Machine Learning

1. Introduction

1.1 Purpose

The goal of this lab is to offer hands-on experience in building a fundamental Artificial Neural Network (ANN) from scratch, without depending on advanced machine learning frameworks such as TensorFlow or PyTorch. The lab focuses on understanding the core mechanisms of neural networks, including activation functions, loss computation, forward propagation, backpropagation, weight updates, and how the network learns to approximate complex functions through iterative optimization techniques.

1.2 Tasks Performed

- Generated a synthetic dataset based on polynomial functions.
- Implemented ReLU activation, MSE loss, forward and backward propagation, and Xavier weight initialization.
- Trained the network using gradient descent with early stopping.
- Evaluated performance on the test set and visualized loss curves and predictions.
- Conducted hyperparameter experiments to study their impact.

2. Dataset Description

The dataset used was synthetically generated from a cubic polynomial with an added sinusoidal component and small gaussian noise. It contains 500 samples with 1 input feature and 1 continuous output. The data was split into training and test sets (80:20). Small gaussian noise was added to the target values, making the task slightly noisy but still learnable. This ensures the model can’t just memorize a perfect function and has to generalize.

3. Methodology

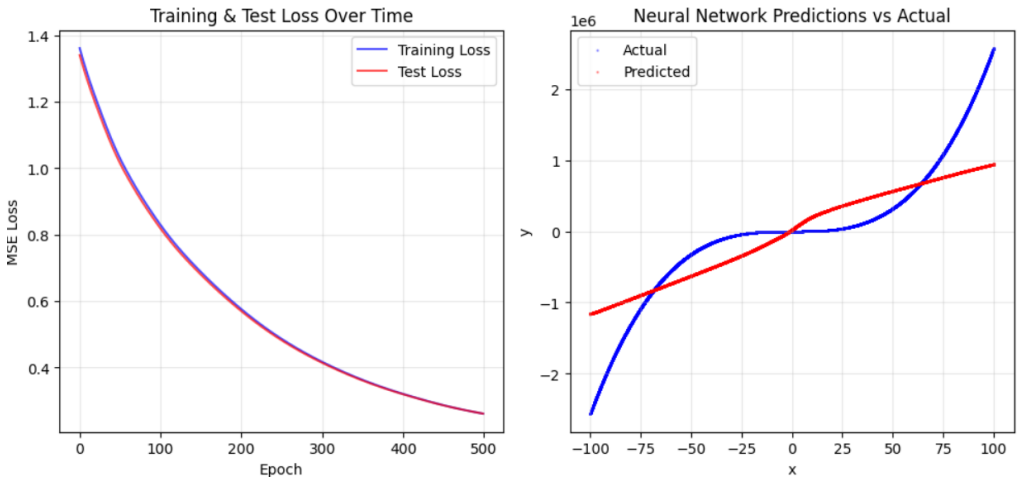
The goal was to train a fully connected neural network to predict a noisy cubic-plus-sinusoidal function. The dataset had 500 samples, each with one input and one continuous output. The data was split into 80% for training and 20% for testing. Before training, the data was standardized to help the model learn faster. The network had two hidden layers, each with 96 neurons, and used ReLU as the activation function for the first 3 experiments and the baseline model. It was trained using mini-batch gradient descent, and early stopping was applied based on validation loss to stop training when the model wasn't improving. The loss function used was Mean Squared Error, and the R^2 score was also tracked to measure how well the model performed. After the baseline model was trained, four more experiments were run to see how different settings affected the model's performance (hyperparameter tuning).

In Experiment 1, the learning rate was increased to make training faster.
In Experiment 2, the learning rate was lowered and the number of training steps was increased to allow for more stable learning.
In Experiment 3, the patience for early stopping was reduced to end training earlier.
In Experiment 4, the activation function was changed from ReLU to Tanh to see if smoother nonlinearities helped.

For each experiment, the model's training and test losses were recorded, along with the R^2 score. These results were compared to the baseline to understand how well the model was learning, whether it was overfitting or underfitting, and how the different settings affected overall performance.

4. Results and Analysis

4.1 PART A: Baseline Training

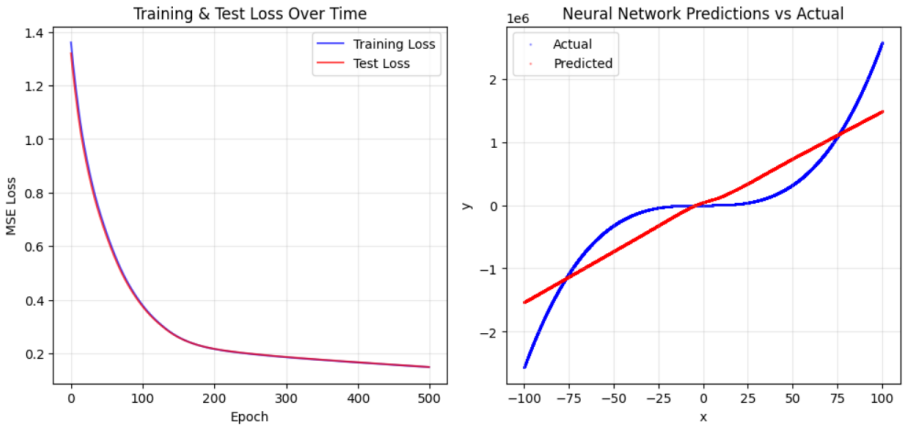


```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.260496
Final Test Loss:     0.260519
R² Score:           0.7356
Total Epochs Run:   500
```

With a learning rate of 0.003, its **overfitting** because validation loss stops improving and eventually diverges a bit from training loss and the model starts to memorize training noise rather than improving generalization.

4.2 PART B: After Hyperparameter Training

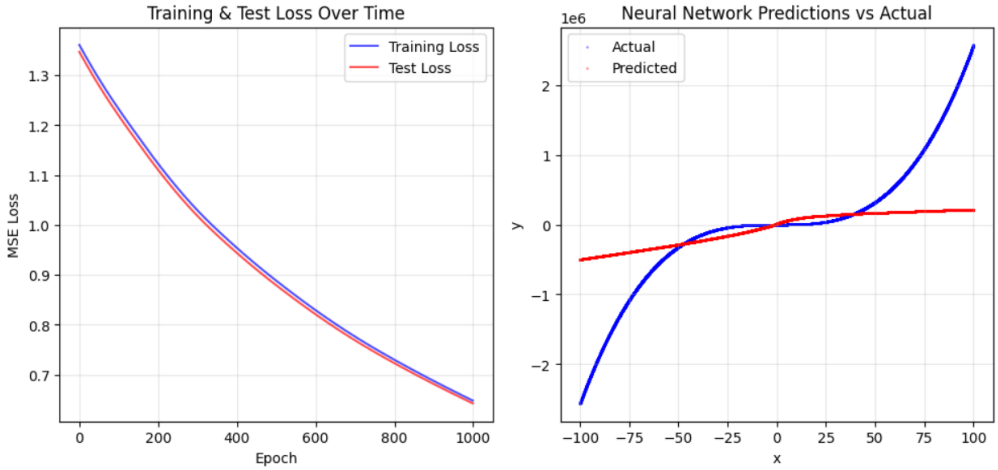
4.2.1 Experiment 1:




```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.147980
Final Test Loss:     0.148451
R² Score:           0.8493
Total Epochs Run:   500
```

With a higher learning rate of 0.01, the model converged faster and reached a final training loss of 0.148 and test loss of 0.148 with almost no gap, indicating no significant overfitting. The validation/test performance improved over the baseline model from part A, and both curves kept decreasing steadily, suggesting the model is well-fit and could potentially improve slightly with even more training (**not underfitting/overfitting**).

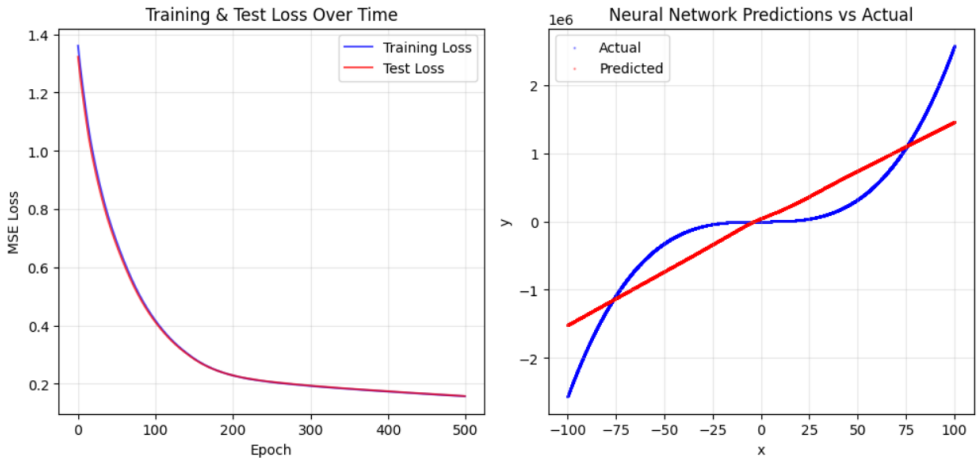
4.2.2 Experiment 2:




```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.647676
Final Test Loss:     0.642112
R² Score:           0.3482
Total Epochs Run:   1000
```

Using a very small learning rate of 0.0005 with 1000 epochs led to very slow convergence. Both training and test losses remained relatively high and R^2 dropped to 0.35, indicating the model is **underfitting** the data. Although the training and test curves remain very close (no overfitting), the network failed to learn the underlying function adequately within the given epochs.

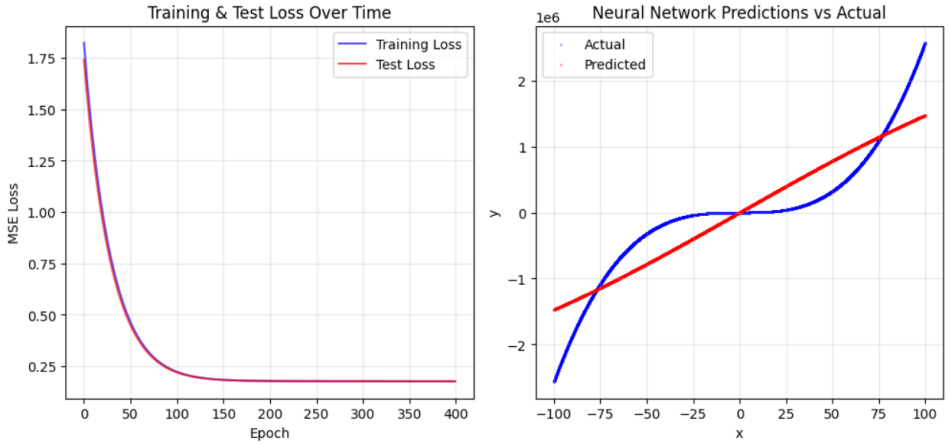
4.2.3 Experiment 3:




```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.156633
Final Test Loss:     0.157261
R² Score:           0.8404
Total Epochs Run:   500
```

Reducing early stopping patience to 5 did not significantly affect training because validation loss kept improving, so training ran for all 500 epochs. The final training/test losses were very close, confirming no overfitting. R^2 score is slightly below experiment 1 (0.85), so performance was similar but not better.

4.2.4 Experiment 4:




```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.175474
Final Test Loss:     0.176229
R² Score:           0.8211
Total Epochs Run:   400
```

Replacing ReLU with Tanh led to stable convergence and nearly similar training/test loss (0.175 and 0.176), confirming **no overfitting**. However, final loss and R^2 score were slightly worse than the best ReLU configuration, indicating ReLU was more effective for this dataset and architecture.

4.3 Results Table

Experiment	Learning Rate	No. of Epochs	Patience	Optimizer	Activation Function	Final Training Loss	Final Test Loss	R^2 Score
1.	0.01	500	10	SGD	Relu	0.147980	0.148451	0.8493
2.	0.0005	1000	15	SGD	Relu	0.647676	0.642112	0.3482
3.	0.009	500	5	SGD	Relu	0.156633	0.157261	0.8404
4.	0.003	400	10	SGD	Tanh	0.175474	0.176229	0.8211

5. Conclusion

The experiments demonstrated how key hyperparameters affect neural network performance on a smooth regression task. The baseline model using ReLU activation and a learning rate of 0.003 had good results, with a good balance between training and test loss.

When the learning rate was increased (Experiment 1), the model learned faster and performed slightly better without becoming unstable, making it the best setup overall. When the learning rate was lowered but the number of training steps was increased (Experiment 2), the model did not learn well enough, leading to higher losses and lower R^2 scores. Lowering the early stopping patience (Experiment 3) didn't hurt the performance much, since the loss kept improving, and the results were similar to Experiment 1. Changing the activation function to Tanh (Experiment 4) made the learning process smoother but resulted in slightly worse final performance compared to ReLU.

Overall, the study found that using a moderately higher learning rate with ReLU activation leads to the most effective training for this dataset, while using a learning rate that's too small slows down learning even with more training steps.