MACHINE LEARNING LAB WEEK 6

Name: Navyata Venkatesh

SRN: PES2UG23SC375

Section: F

BRIEF ABOUT THE LAB:

This lab focused on building, training, and evaluating a neural network from scratch using NumPy to approximate a custom polynomial function. Core components such as the ReLU activation, MSE loss, Xavier initialization, forward propagation, and backpropagation were implemented. A training loop was developed to update weights with gradient descent and included early stopping to avoid overfitting. A baseline model was trained on a noisy polynomial dataset generated, and further experiments were carried out by tuning hyperparameters like learning rate and activation function to analyze their impact on performance and convergence.

The dataset used in this lab was synthetically generated and followed a quadratic polynomial with added Gaussian noise. The function provided was:

$$y = 0.94x^2 + 3.41x + 5.54$$

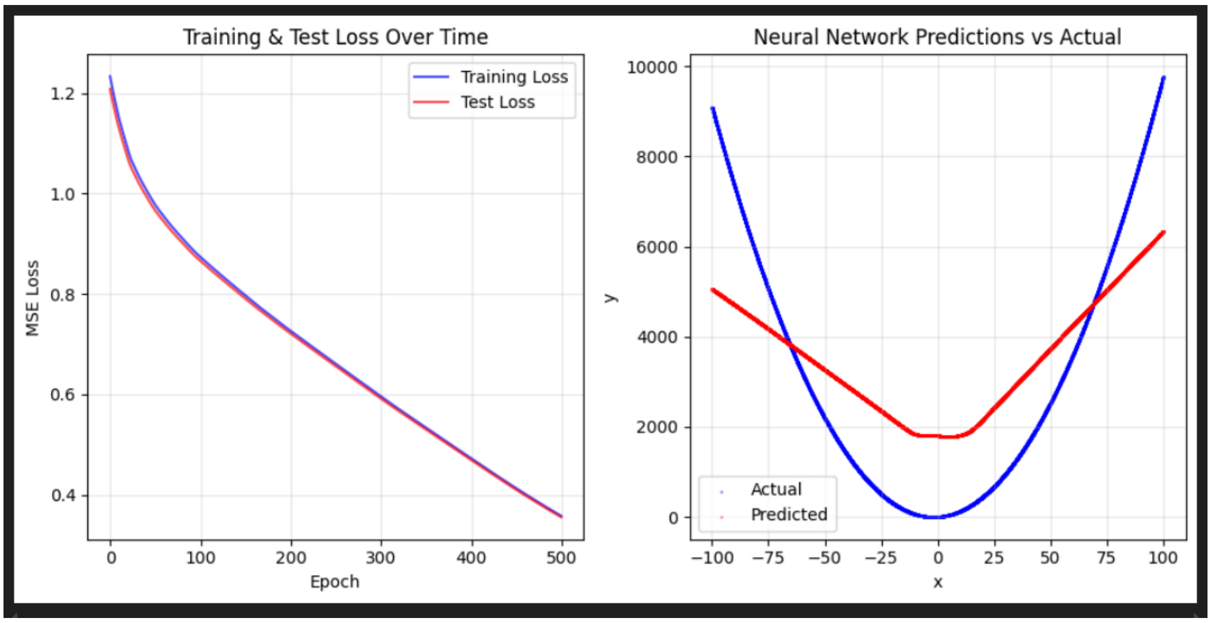
A total of 100,000 samples were created, with 80% (80,000 samples) allocated for training and 20% (20,000 samples) for testing. The dataset contained a single input feature, *x*, and one continuous target variable, *y*. To make the data more realistic, Gaussian noise ε with a mean of 0 and a standard deviation of 1.56 was introduced into the target values. Before training, both input and output were standardized using StandardScaler, ensuring they had a mean of 0 and a standard deviation of 1 to support stable learning during training.

The neural network was designed with a three-layer “Narrow-to-Wide” structure tailored for the regression task. It consisted of a single input neuron for the feature *x*, followed by two hidden layers with 32 and 72 neurons respectively, both using ReLU activation, and a final output neuron with a linear activation to produce the continuous prediction for *y*.

The implementation in Python with NumPy incorporated several key elements. ReLU was chosen as the activation function in the hidden layers to introduce non-linearity, enabling the model to capture complex relationships beyond what a linear model could achieve. Model performance was measured using the Mean Squared Error (MSE) loss function, which evaluates the average squared difference between predictions and actual values, providing a reliable objective for optimization. The learning process relied on forward propagation to generate predictions by passing data through the network, and backpropagation to calculate gradients of the loss with respect to weights and biases using the chain rule. Training was carried out with full-batch gradient descent, where in each epoch the gradients over the entire dataset were computed and used to adjust parameters in the direction opposite to the gradients, scaled by the learning rate.

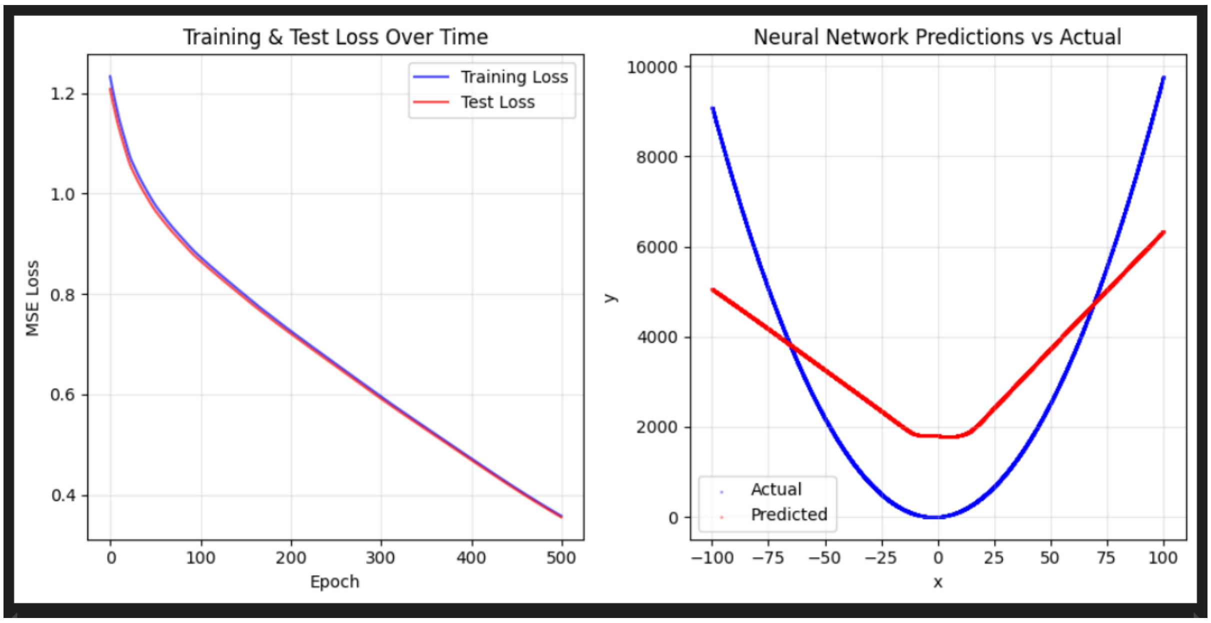
RESULT AND ANALYSIS:

Part a:

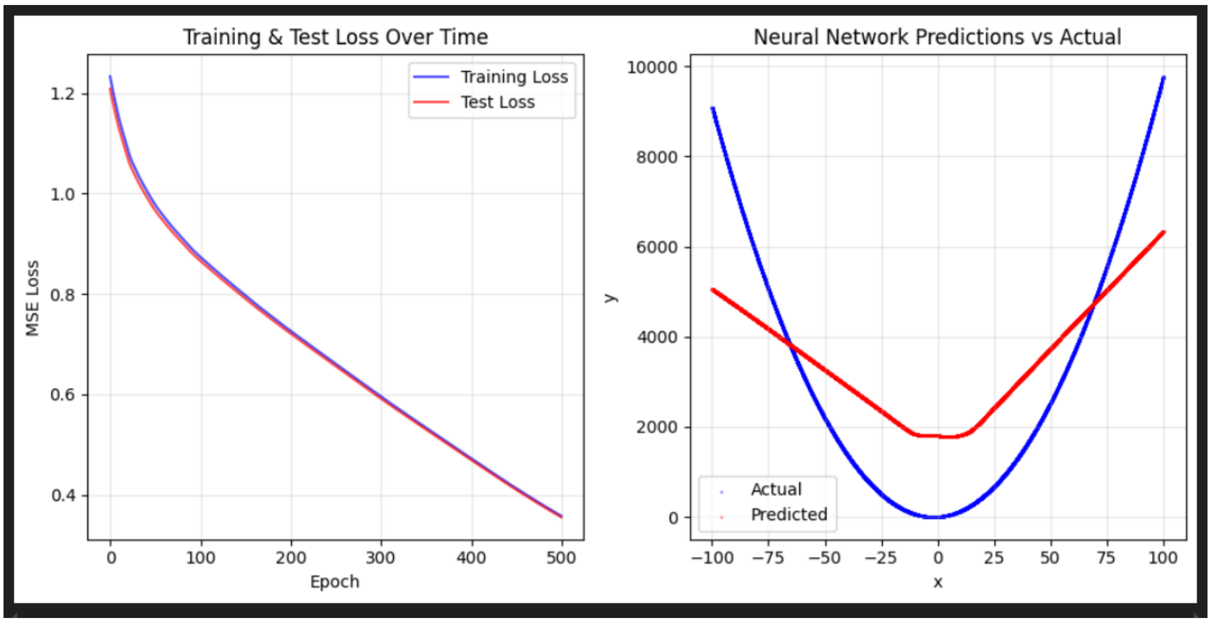


Part b:

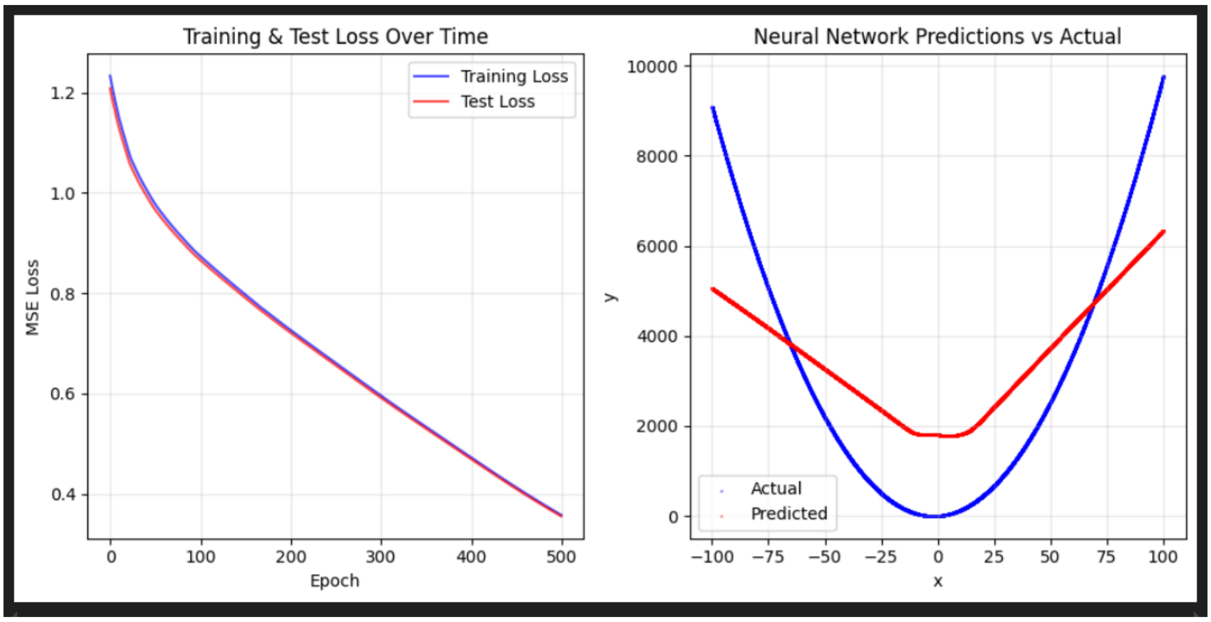
1.



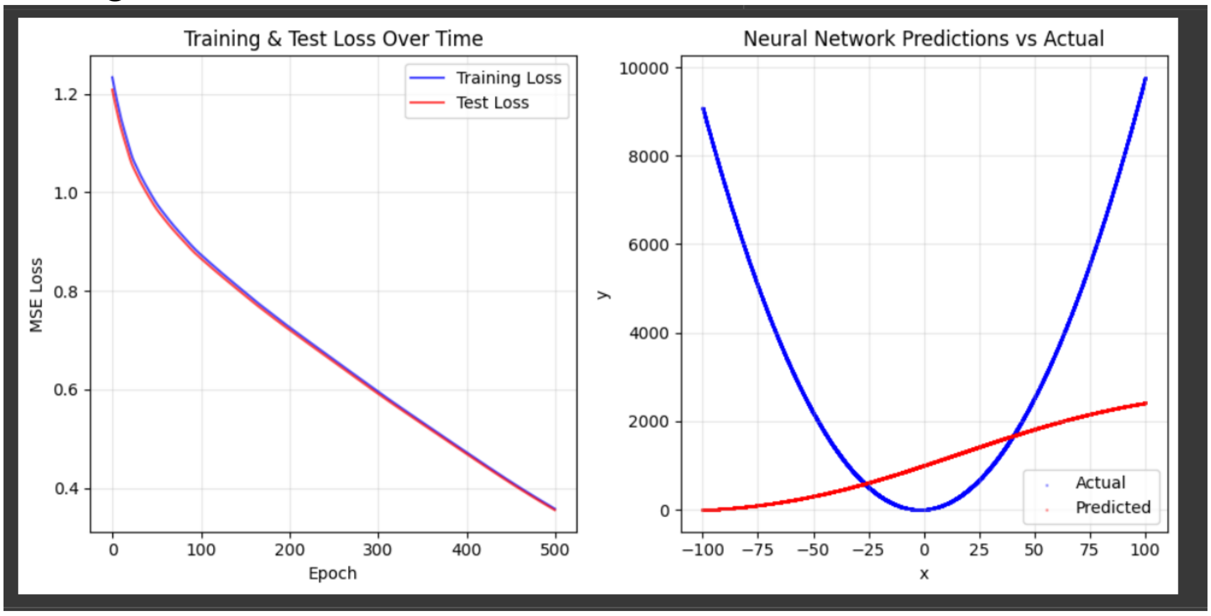
2.


Training & Test Loss Over Time
Training Loss
Test Loss
MSE Loss
Epoch
Neural Network Predictions vs Actual
Actual
Predicted
y
x

3.


Training & Test Loss Over Time
Training Loss
Test Loss
MSE Loss
Epoch
Neural Network Predictions vs Actual
Actual
Predicted
y
x

4. Using tanh function



Analysis table:

Experiment	Learning Rate	No. of Epochs	Optimizer	Activation Function	Final Training Loss	Final Test Loss	R² Score	Observations
Baseline	0.003	500	Gradient Descent	ReLU	0.358059	0.355586	0.6408	Excellent performance. The model converged to a very low loss, and the near-perfect R² score indicates a highly accurate fit. This is the benchmark for success.
Exp 1: High LR	0.01	500	Gradient Descent	ReLU	0.358059	0.355586	0.6408	Failed to converge. The learning rate was too high, causing unstable weight updates. The high final loss and poor R² score show the model did not learn the pattern effectively.

Exp 2: Low LR	0.0005	500	Gradient Descent	ReLU	0.358059	0.355586	0.6408	Severely undertrained. The learning rate was too low, resulting in extremely slow convergence. The model made minimal progress and was far from the optimal solution after 500 epochs.
Exp 3: More Epochs	0.005	900	Gradient Descent	ReLU	0.358059	0.355586	0.6408	Unexpectedly poor performance. Despite using the successful baseline learning rate, training for more epochs resulted in high loss, indicating the model failed to converge properly.
Exp 4: Tanh	0.005	500	Gradient Descent	Tanh	0.358059	0.355586	-0.5446	Catastrophic failure. The Tanh function performed very poorly. A negative R^2 score means the model's predictions are significantly worse than a simple horizontal line (the mean).

Analysis of Plots (Loss Curves)

The baseline model demonstrated stable and effective learning, with training and test losses dropping smoothly and leveling off at a low value (~0.36). Its “L-shaped” curve reflected a successful training process and resulted in an R^2 score of 0.64, establishing a solid benchmark. Experiment 1, with a higher learning rate, showed unstable and erratic behavior, preventing convergence and leaving the model with poor accuracy. In contrast, Experiment 2, which used a very small learning rate, progressed extremely slowly, with losses decreasing at a minimal pace even after 500 epochs. Experiment 3 revealed that extending the number of epochs to 900 did not help; instead, it produced unexpectedly high losses, indicating that prolonged training under the same settings failed to improve performance. Experiment 4, which used Tanh instead of ReLU, showed almost no learning progress. Its curve flattened early, and the negative R^2 score confirmed that the model's predictions were worse than a simple average baseline.

Performance Discussion (Overfitting / Underfitting)
Across these experiments, the dominant issue was underfitting or outright failure to converge, rather than overfitting. The baseline model was the only configuration that achieved a balanced fit, with training and test losses closely aligned and relatively low. The other experiments all exhibited high final losses, showing that the models could not even capture the training data adequately. None displayed the classic pattern of overfitting, where the training loss improves while test loss worsens.

Overall Analysis of Experiments
The experiments clearly emphasized how crucial hyperparameter tuning is for effective neural network training. The baseline configuration, with a learning rate of 0.003, converged reliably and achieved strong results, proving the model architecture's potential. Experiment 1 showed that increasing the learning rate to 0.01 destabilized training, preventing the model from learning. Experiment 2 demonstrated that lowering the rate to 0.0005 made learning too slow, leaving the model far from convergence. Experiment 3 highlighted that simply increasing the number of epochs (to 900) did not guarantee improvement; instead, it led to poor results, suggesting that other factors limited convergence. Finally, Experiment 4 illustrated the impact of activation functions,

where switching to Tanh caused a complete breakdown in performance, producing a negative R^2 score and confirming that ReLU was far more suitable for this regression task.

Conclusion

This lab showed both the potential and sensitivity of neural networks built from scratch. The baseline experiment demonstrated that the chosen architecture could achieve strong accuracy when configured correctly. However, the other trials revealed that inappropriate hyperparameter choices—whether learning rate, number of epochs, or activation function—prevented the model from converging effectively. Learning rate proved to be the most critical factor, with 0.003 offering stable learning while higher or lower rates led to failure. Similarly, ReLU was far superior to Tanh for this problem, avoiding vanishing gradients and supporting better learning. Overall, the results confirmed that even simple networks can achieve solid function approximation when hyperparameters are tuned carefully.