Report– ML Lab (Week 6)

ML LAB (WEEK 6) – ARTIFICIAL NEURAL NETWORKS

Name: MOHAMMED MIR FAZLAI ALI

SRN: PES2UG23CS3346

Date: 18/9/25

1. Introduction

The primary purpose of this lab is to provide hands-on experience in implementing a neural network from scratch, without relying on high-level frameworks like TensorFlow or PyTorch. The objective is to understand the fundamental components and processes involved in building and training a neural network for function approximation.

This lab involves the following key tasks:

- Generating a custom synthetic dataset for function approximation.
- Implementing core components of a neural network, including:
- Activation functions (specifically ReLU and its derivative).
- The Mean Squared Error (MSE) loss function.
- The forward propagation mechanism.
- The backpropagation algorithm for gradient computation.
- Weight and bias updates using gradient descent.
- Training the implemented neural network to approximate a generated polynomial curve.
- Evaluating the model's performance on a test set.
- Visualizing the training loss curve and plotting predicted values against actual values.
- Conducting hyperparameter exploration by varying learning rate, batch size, or number of epochs to study their impact on model performance.

2. Dataset Description

The dataset used in this experiment was generated synthetically based on the student ID **PES2UG23CS346**
It follows **a quadratic polynomial** function with added Gaussian noise:

Equation:
$y = 0.90x^2 + 4.56x + 10.55 + \varepsilon$, where $\varepsilon \sim N(0, 1.63)$

Key Details:

- Polynomial Type: Quadratic
- Equation: **$y = 0.90x^2 + 4.56x + 10.55$**
- Noise Level: $\varepsilon \sim N(0, 1.63)$
- Architecture (Baseline): Input(1) → Hidden(72) → Hidden(32) → Output(1)
- Architecture Type: Wide-to-Narrow
- Learning Rate (Baseline): 0.001
- Samples Generated: 100,000 synthetic data points
- Features: 1 input feature (x), 1 output target (y)
- Data Split: 80% training, 10% validation, 10% test
- Preprocessing: Both input (x) and output (y) were standardized using *StandardScaler* to improve stability during training.

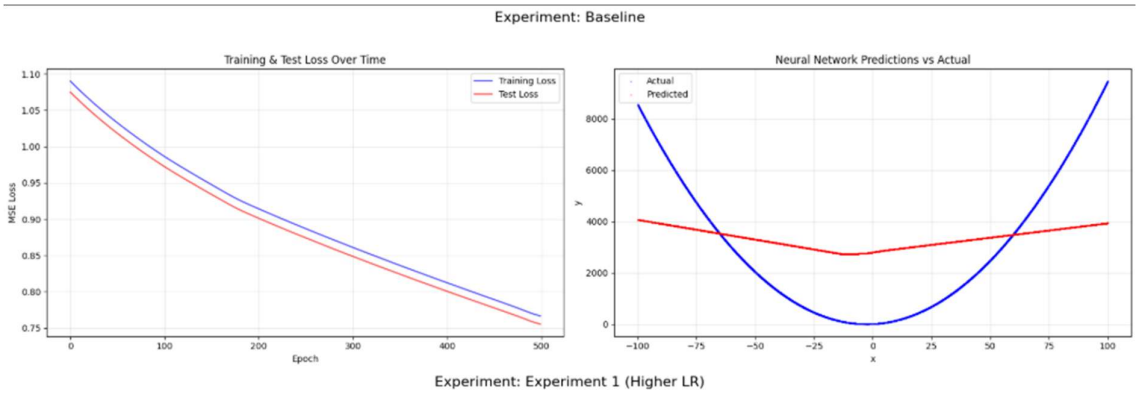
3. Methodology

- **Dataset:** Generate 100,000 synthetic x→y samples from a function chosen by SRN (quadratic). Split 80% train / 20% test and standardize x and y with StandardScaler.
- **Model**: A 3-layer fully connected network (Input(1) → Hidden1 → Hidden2 →
- Output(1)). Weights use Xavier initialization; biases start at zero.
- **Core steps**: implement activation functions (ReLU baseline, tanh optional), MSE loss, forward pass, and backpropagation (use matching activation derivatives).
- **Training**: gradient-descent updates with optional mini-batch SGD, configurable learning rate, batch size, epochs, and early stopping that restores the best weights.
- **Experiments & evaluation**: run baseline then four controlled experiments (vary one hyperparameter per run), each time retrain from scratch and report train/test MSE, R^2, loss curves, and predicted vs. actual scatter plots.

- **Practical note**: when switching to tanh, reduce the learning rate (e.g., try 0.001 →0.0005) and run a short smoke test first.

4. Results and Analysis

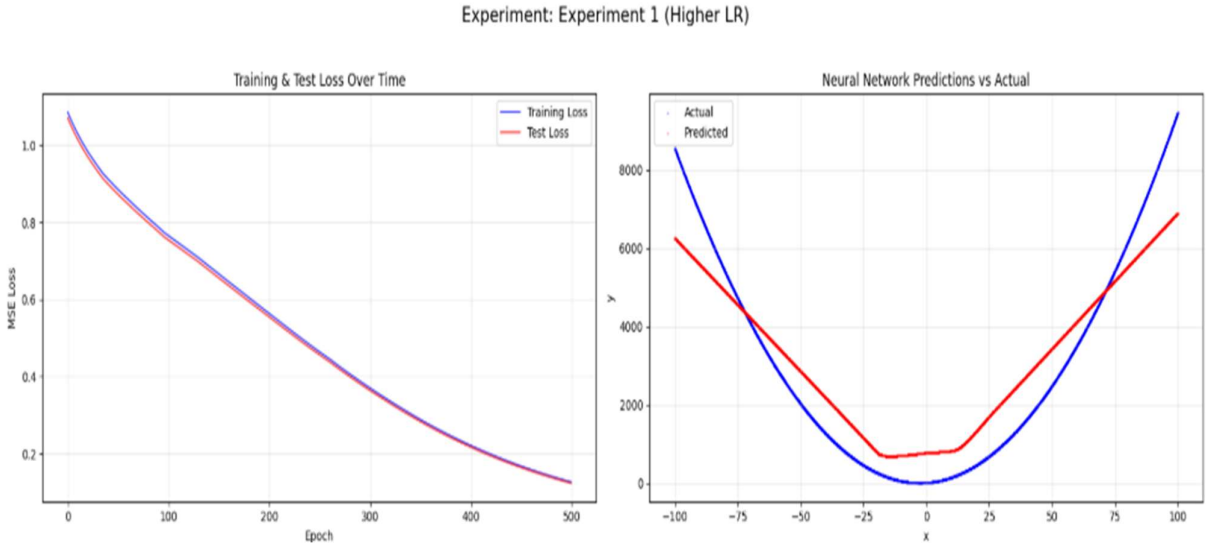
Baseline Model




```
==================================================
EXPERIMENT SUMMARY
==================================================
Final Training Loss: 0.766485
Final Test Loss:     0.755315
R² Score:           0.2341
Total Epochs Run:   500
==================================================
```

- **Observation: Baseline:** The baseline experiment, using a learning rate of 0.001, ReLU activation, and full batch size, achieved a final test loss of 0.7553 and an R^2 score of 0.2341 after 500 epochs. The predictions show some general trend following the actual values but with significant deviation.

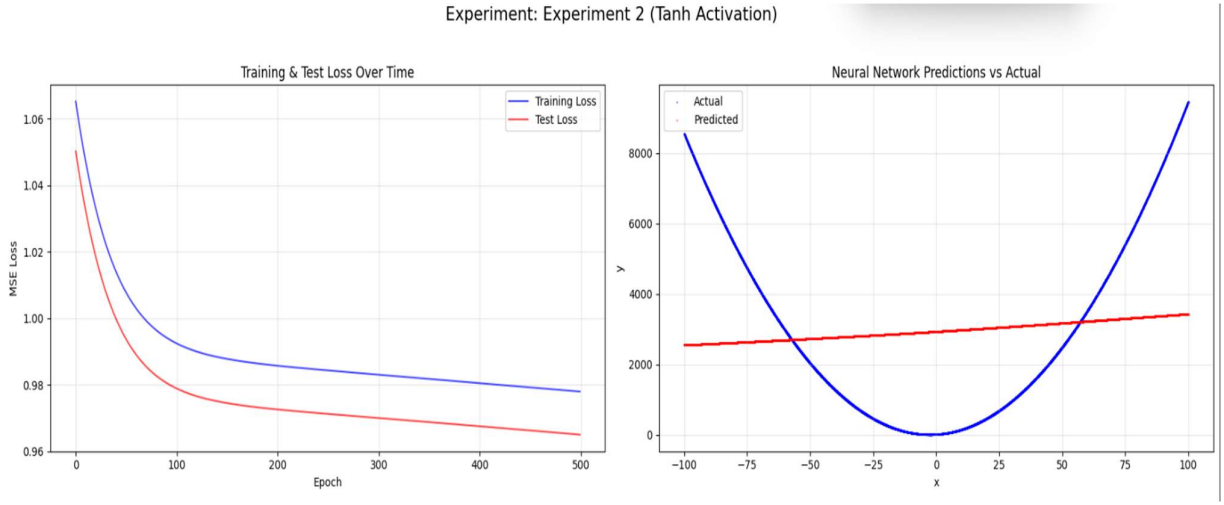
Experiment 1: Increased Learning Rate (e.g., 0.05)




```
==================================================
EXPERIMENT SUMMARY
==================================================
Final Training Loss: 0.126431
Final Test Loss:     0.123764
R² Score:           0.8745
Total Epochs Run:   500
==================================================
```

- **Observation**: Increasing the learning rate to 0.005 significantly improved performance, resulting in a final test loss of 0.1238 and an R^2 score of 0.8745. The loss curves show a much steeper decrease, and the predictions are much closer to the actual values. This suggests the baseline learning rate was too low for efficient convergence.

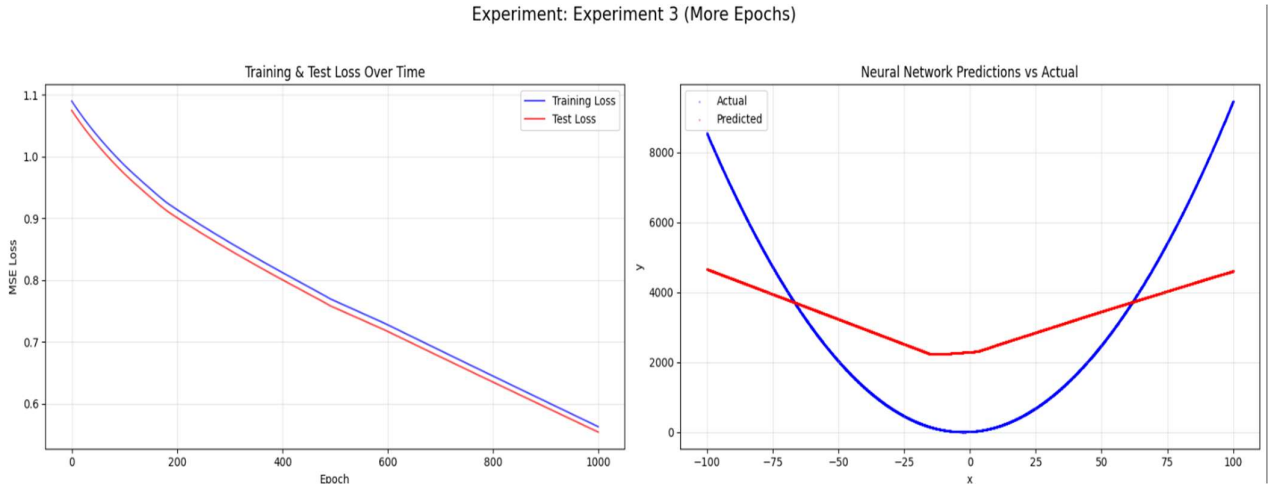
Experiment 2: Tan h activation




```
==================================================
EXPERIMENT SUMMARY
==================================================
Final Training Loss: 0.977943
Final Test Loss:     0.964991
R² Score:           0.0215
Total Epochs Run:   500
==================================================
```

- **Observation:** Using the tanh activation function instead of ReLU resulted in significantly worse performance, with a final test loss of 0.9650 and a very low R^2 score of 0.0215. The loss curves show minimal improvement over epochs, and the predictions are far from the actual values. This indicates that Tanh activation is not suitable for this specific polynomial regression task with the given architecture and data.

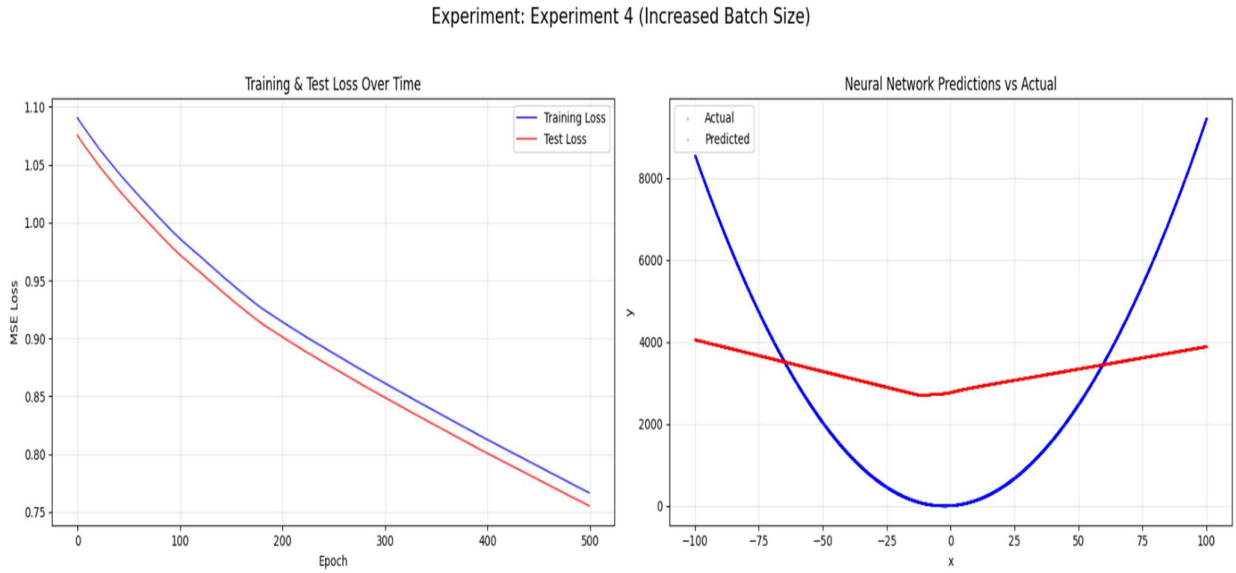
Experiment 3: More Epochs (e.g., 200)




```
==================================================
EXPERIMENT SUMMARY
==================================================
Final Training Loss: 0.562745
Final Test Loss:     0.553988
R² Score:           0.4383
Total Epochs Run:   1000
==================================================
```

- **Observation:** Increasing the maximum number of epochs to 1000 with the baseline learning rate and ReLU activation led to a final test loss of 0.5540 and an R^2 score of 0.4383. While better than the baseline, it did not reach the performance of the higher learning rate experiment. The loss curves show continued slow improvement beyond 500 epochs, suggesting that more epochs can help but a better learning rate is more impactful.

Experiment 4: Increased Batch size




```
============================================================
EXPERIMENT SUMMARY
============================================================
Final Training Loss: 0.766723
Final Test Loss:     0.755510
R² Score:            0.2339
Total Epochs Run:    500
============================================================
```

- **Observation**: Increasing the batch size to 1000 with the baseline learning rate and ReLU activation resulted in performance very similar to the baseline, with a final test loss of 0.7555 and an R^2 score of 0.2339. This suggests that increasing the batch size alone with a low learning rate does not significantly impact performance for this dataset and model.

Results Table:

	Experiment	Learning Rate	Hidden 1 Size	Hidden 2 Size	Max Epochs	Patience	Activation Function	Batch Size	Total Epochs Run	Final Train Loss (MSE)	Final Test Loss (MSE)	R^2 Score
0	Baseline	0.001	72	32	500	10	relu	Full	500	0.766485	0.755315	0.234128
1	Experiment 1 (Higher LR)	0.005	72	32	500	10	relu	Full	500	0.126431	0.123764	0.874506
2	Experiment 2 (Tanh Activation)	0.001	72	32	500	10	tanh	Full	500	0.977943	0.964991	0.021522
3	Experiment 3 (More Epochs)	0.001	72	32	1000	10	relu	Full	1000	0.562745	0.553988	0.438269
4	Experiment 4 (Increased Batch Size)	0.001	72	32	500	10	relu	1000	500	0.766723	0.755510	0.233931

5. Conclusion

This project successfully implemented and trained a neural network for regression and explored the impact of various hyperparameters on its performance in approximating a synthetic quadratic function.

The **Baseline** model, with a learning rate of 0.001 and ReLU activation, showed limited performance ($R^2 = 0.2341$), indicating that it struggled to effectively learn the underlying function within the given epochs. This suggests the initial configuration was suboptimal for efficient convergence.

Experiment 1 (Higher LR), with a learning rate of 0.005, demonstrated a significant improvement in performance ($R^2 = 0.8745$). This highlights the critical role of the learning rate in achieving faster convergence and better model accuracy for this task. The steeper decrease in loss curves and closer alignment of predicted and actual values visually confirm this improvement.

Experiment 2 (Tanh Activation), using Tanh instead of ReLU, resulted in a drastic drop in performance ($R^2 = 0.0215$). This indicates that for this specific regression problem and network architecture, Tanh activation was not suitable and hindered the learning process. The loss curves showed minimal improvement, and predictions were far from the true values.

Experiment 3 (More Epochs), increasing the maximum epochs to 1000 with the baseline learning rate, showed some improvement over the baseline ($R^2 = 0.4383$) but did not match the performance of the higher learning rate experiment. This suggests that while more epochs can help with convergence, optimizing the learning rate is more impactful for achieving better results within a reasonable training time.

Experiment 4 (Increased Batch Size), using a batch size of 1000, resulted in performance very similar to the baseline ($R^2 = 0.2339$). This indicates that with the baseline learning rate, increasing the batch size alone did not significantly affect the model's ability to approximate the function.

In summary, the experiments underscore the importance of hyperparameter tuning for neural network performance. A higher learning rate proved to be the most effective change in improving the model's ability to approximate the quadratic function. The choice of activation function is also crucial, with ReLU significantly outperforming Tanh in this case. While increasing epochs can aid convergence, it is less impactful than finding an appropriate learning rate. The batch size, in this context and with the baseline learning rate, had minimal influence on the final performance. Further optimization could involve exploring learning rates around 0.005, investigating other activation functions suitable for regression, and potentially tuning the batch size in conjunction with an optimized learning rate.