Artificial Neural Networks

NAME: NANDITA R NADIG
ID: PES2UG23CS365
COURSE NAME: MACHINE- LEARNING (UE23CS352A)
SUBMISSION DATE: 16.09.2025

Introduction

The purpose of this lab was to get practical experience with building an Artificial Neural Network (ANN) from the ground up to perform function approximation. Instead of relying on pre-built tools, the focus was on understanding and implementing each part of the network manually.

The tasks performed in the lab included:

- Generating a synthetic polynomial dataset based on the student SRN.
- Implementing the ReLU activation function, MSE loss function, forward pass, and backward pass.
- Training the network with early stopping to approximate the given polynomial function.
- Evaluating model performance using training and test loss, R^2 score, and visualization of predictions versus ground truth.
- Conducting hyperparameter experiments to study the effect of learning rate, batch size, epochs, and architecture choices.

Dataset description

The dataset used was a synthetic quadratic polynomial function $y = 1.46x^2 + 6.21x + 11.85$ with added Gaussian noise.

- **Total samples:** 100,000
- **Training set:** 80,000 samples
- **Test set:** 20,000 samples
- **Features:** 1 input feature (x) and 1 target (y)
- **Noise level:** Gaussian noise with standard deviation ≈ 2.09 (variance ≈ 4.37)

Methodology

1. **Weight Initialization (Xavier Initialization):**
    - All weights were initialized using Xavier initialization, so that the weights are not too big or too small, so the values don't explode or vanish when passing through layers.
    - Biases were initialized to zero.
2. **Activation Function:**
    - The *Rectified Linear Unit (ReLU)* was used for both hidden layers. It outputs 0 if the input is negative, otherwise it outputs the input itself.

$$\mathrm{f(z)} = \max\,(0, \mathrm{z})$$

    - Its derivative, required for backpropagation, is 1 for positive inputs and 0 otherwise.

$$f'(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$$

    - ReLU introduces non-linearity and helps avoid vanishing gradient issues.
3. **Loss Function:**

- Model performance was measured using the *Mean Squared Error (MSE)*.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- MSE works by squaring the differences between predicted and actual values, which means that larger errors count more and have a bigger impact on the final loss.

4. **Forward Propagation:**
   - Input data passes through the first hidden layer (ReLU), then the second hidden layer (ReLU), and finally the output layer (linear, since this is regression).
   - Each layer performs a linear transformation ($z = XW + b$) followed by the activation function.
5. **Backpropagation:**
   - Gradients of the loss with respect to weights and biases were calculated using the chain rule.
   - The derivative of ReLU was applied in the hidden layers during backpropagation.
   - These gradients guided the updates of weights and biases.
6. **Gradient Descent Optimization:**

   Parameters were updated iteratively:

$$W = W - \eta \cdot \frac{\partial W}{\partial L}$$

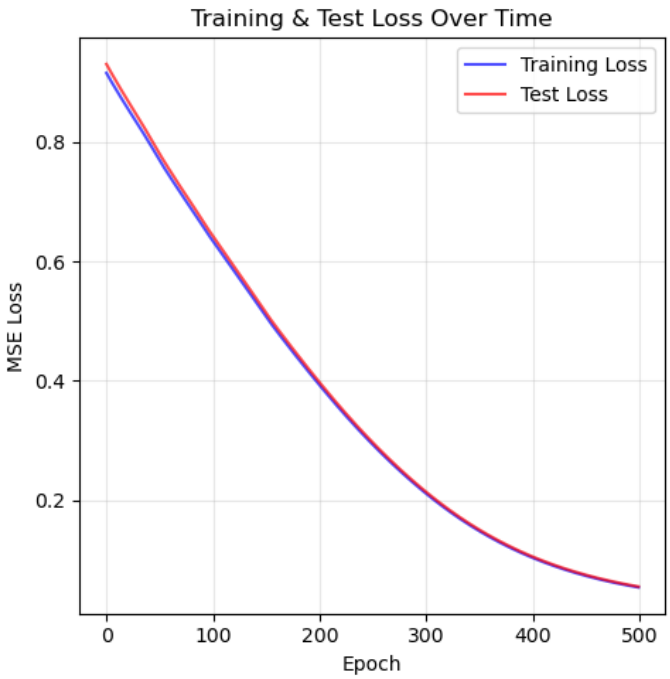
$$b = b - \eta \cdot \frac{\partial b}{\partial L}$$

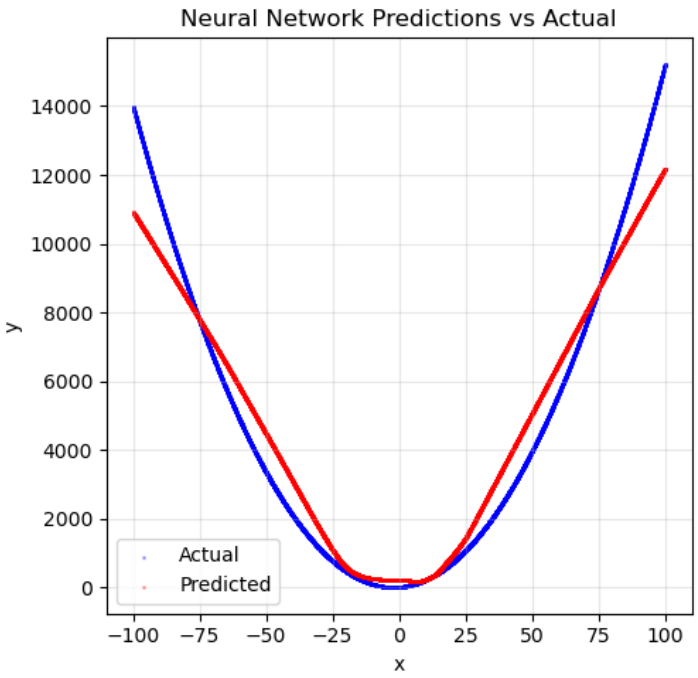
   where η is the learning rate (set to 0.005 in the baseline).
7. **Training Loop with Early Stopping:**
   - The model was trained for up to 500 epochs, tracking training and test loss at each iteration.
   - Early stopping was implemented with a patience value, halting training when test loss stopped improving.

Results and Analysis

Screenshots:

Experiment	Learning Rate	Batch Size	No of epochs	Optimizer	Activation Function	Training Accuracy	Test Accuracy	Training Loss	Test Loss	Observations
Baseline	0.005	80000	500	Gradient Descent	ReLU	0.947	0.9459	0.053891	0.055209	The training and test losses were very similar, which suggests the model generalizes well and isn't overfitting. When making a prediction at x = 90.2, the model had a relative error of about 12.96%.






```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 10,830.47
Ground Truth (formula):    12,442.45
Absolute Error:            1,611.97
Relative Error:            12.955%
```

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.053891
Final Test Loss:     0.055209
R² Score:            0.9459
Total Epochs Run:    500
```

Analysis:

- The model performed well in learning the quadratic polynomial.
- Since the training and test losses are very close, there is no sign of overfitting.
- The model also did not underfit because the R^2 score shows that about 95% of the variance in the data was explained.
- Small prediction errors are present at extreme values of x, but overall the model achieved good accuracy and stability.

Conclusion

The results show that neural networks can successfully approximate complex polynomial functions, but the performance strongly depends on the choice of hyperparameters. The baseline model achieved good accuracy and stability, with training and test losses remaining close throughout training. Hyperparameter tuning, such as adjusting the learning rate, batch size, number of epochs, or network architecture, plays a key role in improving performance and avoiding issues like underfitting or overfitting.