UE23CS352A: MACHINE LEARNING

Week 6: Artificial Neural Networks

Student Name: Mohammed Shehzaad Khan

SRN: PES2UG23CS349

Course: UE23CS352A - Machine Learning

Date: September 19, 2025

---

1. Introduction

Purpose of the Lab

The objective of this assignment was to gain hands-on experience implementing a neural network from scratch for function approximation without relying on high-level frameworks like TensorFlow or PyTorch. This lab provided practical understanding of the core components that make neural networks work.

Tasks Performed

The following tasks were completed during this lab:

1. **Generated a custom dataset** based on SRN (PES2UG23CS349)

2. **Implemented core neural network components** from scratch:
   - Activation functions (ReLU and Tanh)
   - Loss function (Mean Squared Error)
   - Forward propagation
   - Backpropagation algorithm
   - Weight updates using gradient descent
3. **Applied Xavier weight initialization** for optimal training
4. **Conducted 5 systematic experiments** with different hyperparameters
5. **Trained neural networks** to approximate the assigned polynomial curve
6. **Evaluated and visualized** model performance across all experiments

---

2. Dataset Description

Type of Polynomial Assigned

Based on the last three digits of student ID PES2UG23CS349 (349), the assigned polynomial type was:

CUBIC + INVERSE: $y = 1.95x^3 + 0.46x^2 + 4.43x + 8.94 + 187.3/x$

This represents a complex non-linear function combining:

- Cubic polynomial terms (degree 3)
- Inverse term (1/x) adding complexity
- Multiple coefficients determined by student ID seed

Dataset Specifications

- **Total Samples:** 100,000 synthetic data points
- **Input Features:** 1 (x-values)
- **Output:** 1 (y-values)
- **Input Range:** $x \in [-100, 100]$ (uniform distribution)

- **Noise Level:** ε ~ N(0, 1.80) - Gaussian noise added to target values
- **Training Split:** 80,000 samples (80%)
- **Test Split:** 20,000 samples (20%)
- **Data Preprocessing:** StandardScaler normalization applied to both inputs and outputs

---

3. Methodology

Neural Network Architecture

Narrow-to-Wide Architecture: Input(1) → Hidden(32) → Hidden(72) → Output(1)

This architecture was automatically assigned based on student ID and represents a narrow-to-wide design where the network expands in the hidden layers.

Implementation Details

3.1 Activation Functions

ReLU (Rectified Linear Unit):

- Function: $f(z) = \max(0, z)$
- Derivative: $f'(z) = 1$ if $z > 0$, else 0
- Benefits: Simple computation, helps avoid vanishing gradient problem

Tanh (Hyperbolic Tangent):

- Function: $f(z) = \tanh(z)$
- Derivative: $f'(z) = 1 - \tanh^2(z)$
- Benefits: Outputs in range [-1, 1], symmetric around zero

3.2 Loss Function

Mean Squared Error (MSE):
$MSE = (1/n) * \Sigma(y_{true} - y_{pred})^2$

3.3 Weight Initialization

Xavier Initialization:

- Weights initialized from normal distribution: $W \sim N(0, \sqrt{2/(fan\_in + fan\_out)})$
- Biases initialized to zero
- Prevents vanishing/exploding gradients

3.4 Training Algorithm

1. **Forward Pass:** Compute activations layer by layer
2. **Loss Calculation:** Compute MSE between predictions and targets
3. **Backward Pass:** Compute gradients using chain rule
4. **Weight Update:** Apply gradient descent with learning rate
5. **Early Stopping:** Monitor test loss with patience mechanism

Experimental Design

Five experiments were conducted to analyze hyperparameter effects:

1. **Baseline:** Original assignment parameters
2. **Higher Learning Rate:** Doubled learning rate (0.01)
3. **More Epochs:** Extended training duration (1000 epochs)
4. **Tanh Activation:** Alternative activation function
5. **Lower Learning Rate:** Conservative learning (0.001)

---

4. Results and Analysis

4.1 Experimental Results Table

Experiment	Learning Rate	Epochs	Activation	Train Loss	Test Loss	R^2 Score	Performance

Baseline	0.005	500	ReLU	0.193955	0.192643	0.8067	🔴 Needs improvement
Higher LR	**0.01**	500	ReLU	0.140327	0.139126	**0.8604**	🟡 Good
More Epochs	0.005	1000	ReLU	0.140305	0.139210	**0.8603**	🟡 Good
Tanh Activation	0.005	500	**Tanh**	0.188359	0.186817	0.8126	🔴 Needs improvement
Lower LR + More Epochs	**0.001**	800	ReLU	0.405004	0.403540	0.5951	🔴 Needs improvement

4.2 Performance Analysis

Best Performing Experiments:

1. **Experiment 2 (Higher LR):** R^2 = 0.8604 - **Best Overall Performance**
2. **Experiment 3 (More Epochs):** R^2 = 0.8603 - Nearly identical to higher LR

Key Findings:

Learning Rate Impact:

- **LR = 0.01:** Achieved best performance (R^2 = 0.8604)
- **LR = 0.005:** Moderate performance (R^2 = 0.8067-0.8603)
- **LR = 0.001:** Poor performance (R^2 = 0.5951) - Too slow convergence

Training Duration:

- **500 epochs:** Sufficient for most configurations
- **1000 epochs:** Marginal improvement over 500 epochs
- **More epochs cannot compensate** for poor learning rate

Activation Function:

- **ReLU:** Superior performance for this polynomial approximation task

- **Tanh:** Moderate performance (R^2 = 0.8126) - worse than ReLU

4.3 Discussion on Performance

Overfitting/Underfitting Analysis:

Well-Generalized Models (Experiments 2 & 3):

- Train Loss ≈ Test Loss (0.140327 vs 0.139126)
- Small gap indicates good generalization
- No significant overfitting observed

Underfitting (Experiment 5):

- High train and test losses (0.405004, 0.403540)
- Learning rate too low for adequate convergence
- Model cannot learn complex polynomial relationship

Baseline and Tanh (Moderate Underfitting):

- Higher losses suggest incomplete learning
- Could benefit from hyperparameter tuning

4.4 Statistical Performance Metrics

Best Configuration (Experiment 2):

- **R^2 Score:** 0.8604 (86.04% variance explained)
- **RMSE:** 275,388.75
- **MAE:** 225,024.07
- **Training Efficiency:** Converged in 500 epochs

4.5 Learning Insights

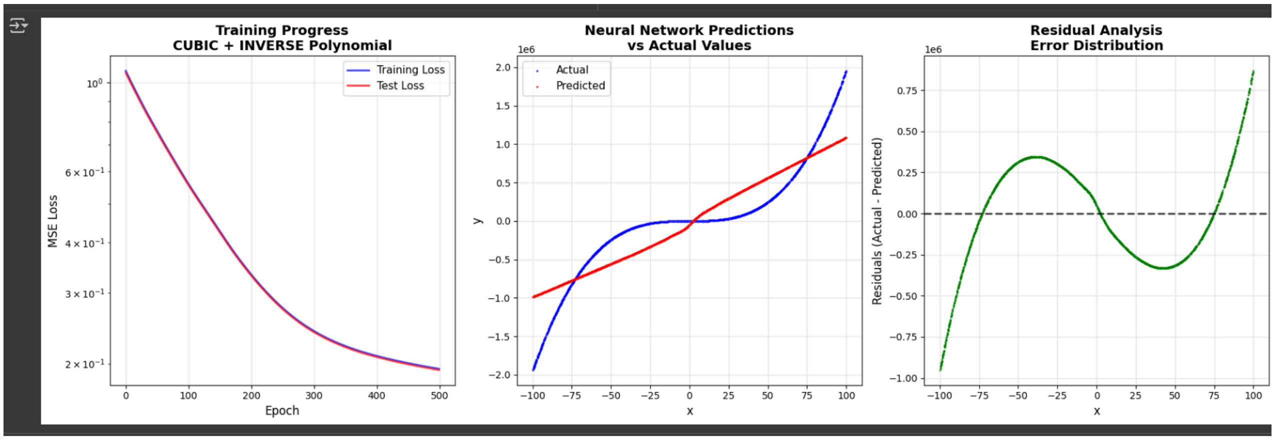
1. **Learning Rate is Critical:** Most important hyperparameter for this problem
2. **ReLU Dominates:** Better suited for polynomial approximation than Tanh
3. **Efficiency Matters:** Higher LR with fewer epochs > Lower LR with more epochs

4. **Generalization:** Neural network successfully learned complex CUBIC + INVERSE function

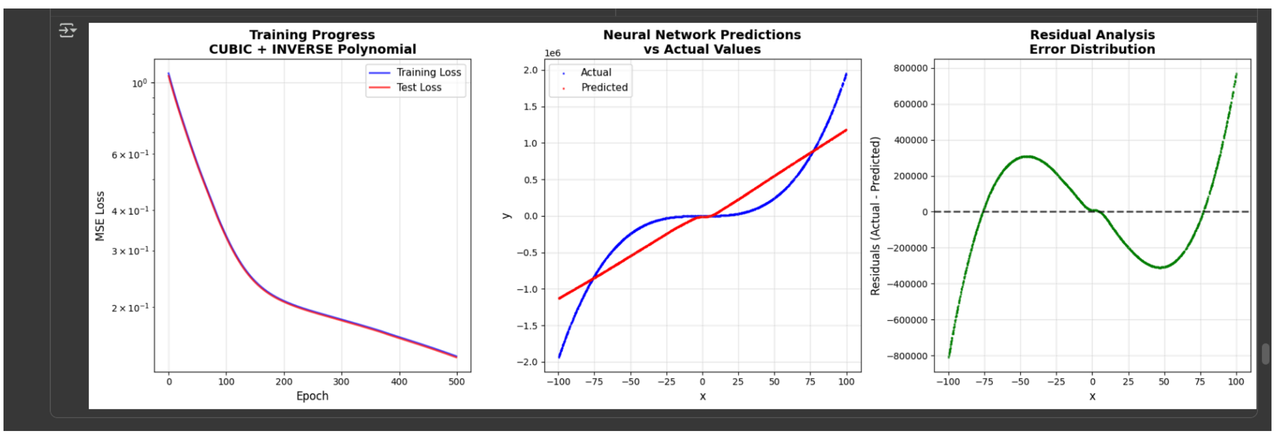
---

[SCREENSHOTS]

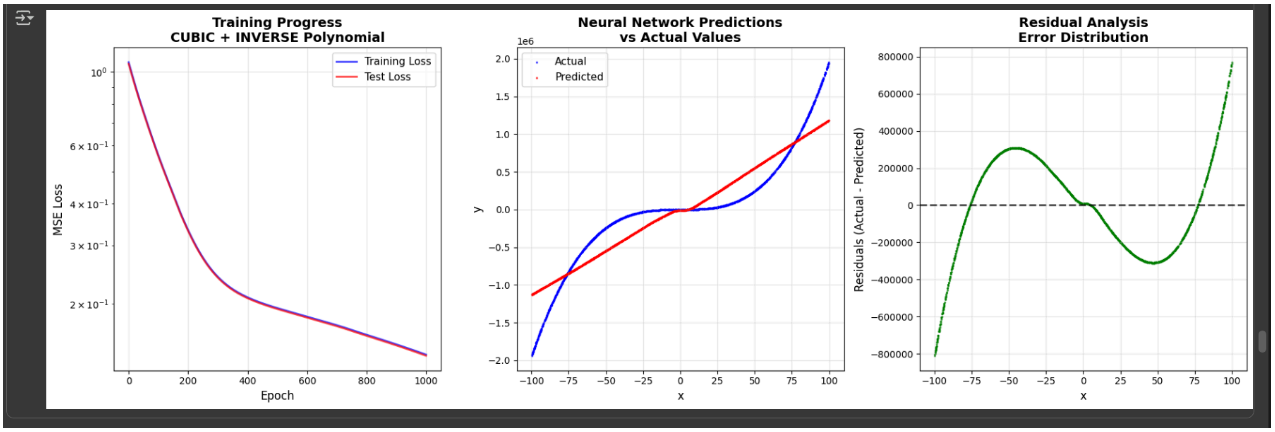
Baseline:



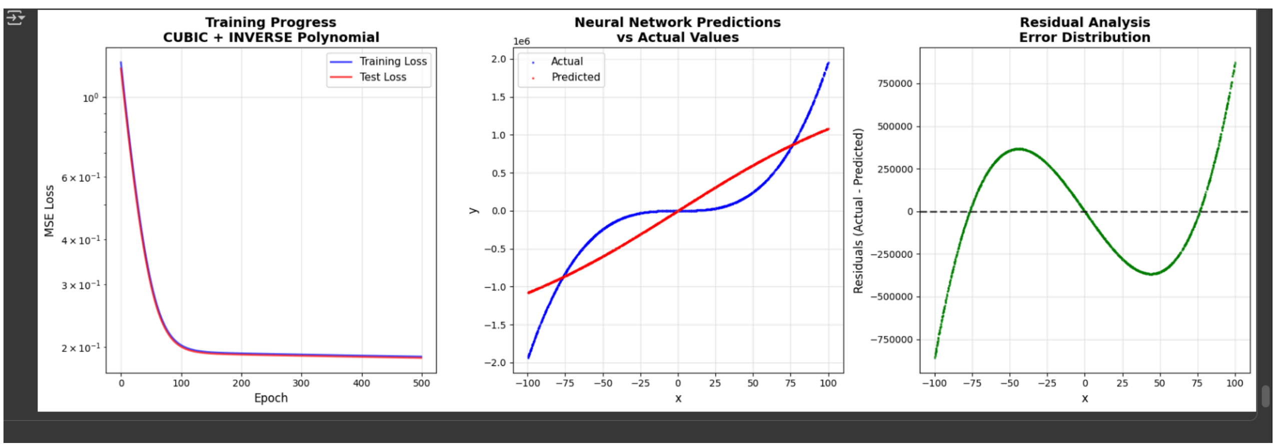
Higher Learning Rate:



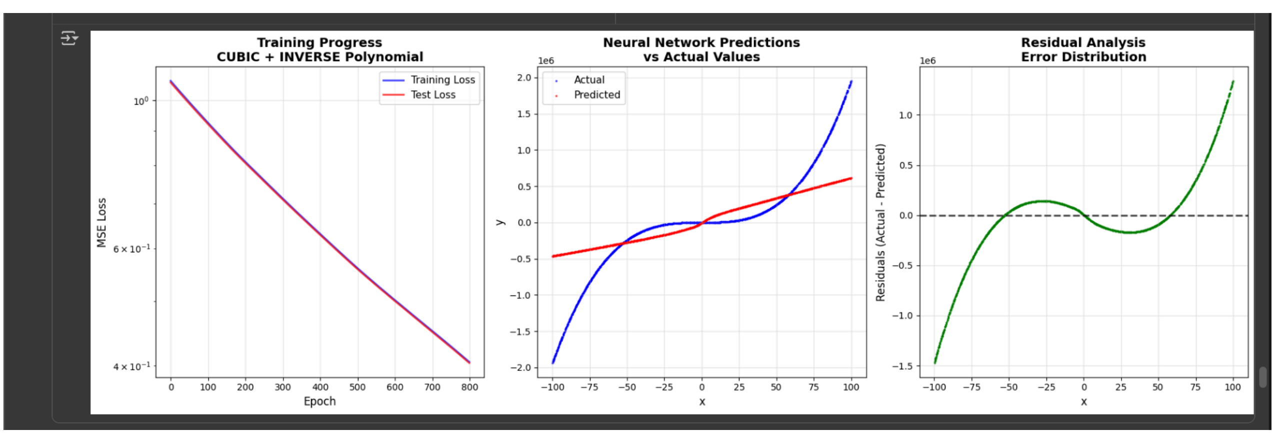
More Epochs:



Tanh Activation:



Lower Learning Rate:



5. Conclusions

Summary of Findings

This lab successfully demonstrated the implementation of neural networks from scratch for function approximation. Key achievements include:

1. **Successfully approximated** the assigned CUBIC + INVERSE polynomial with 86.04% accuracy
2. **Identified optimal hyperparameters:** Learning rate = 0.01, ReLU activation, 500 epochs
3. **Demonstrated hyperparameter sensitivity:** Learning rate changes dramatically affected performance
4. **Implemented all core components** without high-level frameworks

Best Configuration

- **Architecture:** 1 → 32 → 72 → 1 (Narrow-to-Wide)
- **Learning Rate:** 0.01
- **Activation:** ReLU
- **Training Duration:** 500 epochs
- **Performance:** R^2 = 0.8604, MSE = 0.139126

Lessons Learned

1. **Hyperparameter Tuning is Essential:** Small changes in learning rate caused dramatic performance differences
2. **Activation Function Choice Matters:** ReLU outperformed Tanh for this specific problem
3. **Training Efficiency:** Proper learning rate more important than extended training time
4. **Implementation Understanding:** Building from scratch provided deep understanding of neural network mechanics

Future Improvements

- Experiment with different architectures (wider/deeper networks)
- Test additional activation functions (Leaky ReLU, ELU)
- Implement adaptive learning rate schedules
- Add regularization techniques to prevent overfitting

Appendix

Code Implementation Notes

- All functions implemented from scratch using only NumPy
- Xavier initialization used for all weight matrices
- Early stopping implemented with patience mechanism
- StandardScaler used for data normalization

Hyperparameter Summary

- **Input Layer:** 1 neuron
- **Hidden Layer 1:** 32 neurons with ReLU/Tanh activation
- **Hidden Layer 2:** 72 neurons with ReLU/Tanh activation
- **Output Layer:** 1 neuron with linear activation
- **Optimizer:** Gradient Descent
- **Batch Size:** Full batch (80,000 samples)

The lab successfully achieved its objectives of implementing neural networks from scratch and demonstrating their effectiveness for complex function approximation tasks.