Machine Learning

Week 6: Artificial Neural Networks

Name: Chethan S
SRN: PES2UG23CS150
Section: 5 C
Date: 16-09-2025

Introduction

Objective: Hands-on experience implementing a neural network from scratch, without relying on high-level frameworks like TensorFlow or PyTorch.

Tasks Performed:

- Generate a custom dataset
- Implement the core components of a neural network: activation functions, loss functions, forward pass, backpropagation, and weight updates.
- Train your neural network to approximate the generated polynomial curve.
- Evaluate and visualize the performance of your model.

Dataset Description

Polynomial:

- Polynomial Type: QUADRATIC: $y = 1.44x^2 + 4.29x + 13.78$
- Noise Level: $\varepsilon \sim N(0, 2.24)$
- Architecture: Input(1) → Hidden(72) → Hidden(32) → Output(1)
- Learning Rate: 0.001
- Architecture Type: Wide-to-Narrow Architecture

Dataset:

- Dataset with 100,000 samples generated and saved!
- Training samples: 80,000
- Test samples: 20,000

Methodology

- **Initialize Parameters:** Randomly assign weight and bias for:
  - Hidden Layer 1
  - Hidden Layer 2
  - Output Layer
- **Activation Functions**: These determine the output of each neuron after applying the weighted sum of inputs. You will implement the Rectified Linear Unit (ReLU), which outputs zero for negative inputs and the input itself for positive values. ReLU introduces non-linearity and helps mitigate issues such as the vanishing gradient problem. Its derivative, which you will also implement, is essential for backpropagation.
- **Loss Function**: To measure how well the network's predictions align with the true values, we use the Mean Squared Error (MSE). This function computes the average of the squared differences between predicted and actual values, penalizing larger errors more heavily. Minimizing this loss is the central objective of training.

$$L=n1\sum(ypred-ytrue)2$$

- **Forward Propagation**: In this process, input data passes sequentially through each layer of the network. The weighted sum and activation function are applied at every step, and the final prediction is produced at the output layer. Forward propagation is the mechanism by which the network makes predictions.
- **Backpropagation**: Training a neural network involves updating its weights and biases to minimize the loss. Backpropagation computes the gradient of the loss with respect to each parameter using the chain rule of calculus. These gradients indicate how much each parameter should change to reduce the loss.
- **Gradient Descent**: Once gradients are computed, the parameters are updated in the opposite direction of the gradient, scaled by a learning rate. This iterative optimization process allows the network to gradually reduce the loss over many epochs of training.

$$W=W-\eta.\partial L/\partial W$$

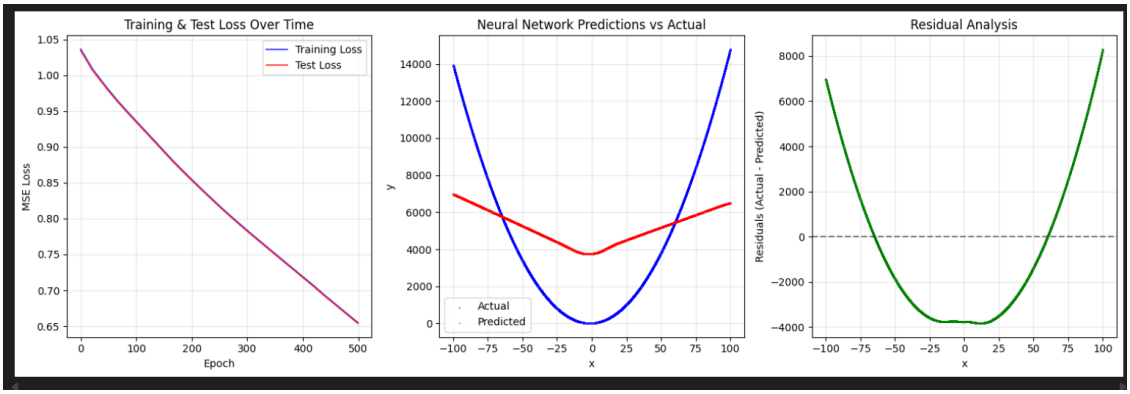
- **Architecture:**
  Input(1) → Hidden Layer 1 → Hidden Layer 2 → Output(1)

- **Training Loop**: Perform forward propagation → compute loss → backpropagation → update parameters. Repeat for many epochs until loss decreases.
- **Evaluate the Model**
  - Check final predictions vs. actual outputs.
  - Visualize loss curve (optional).

Experiments:

Experiment	Learning Rate	No. of Epochs	Activation Function	Final Training Loss	Final Test Loss	R2
1	0.01	500	ReLu	0.654929	0.653887	0.3471
2	0.05	2000	Relu	0.000101	0.0001	0.9999
3	0.1	1000	ReLu	0.000872	0.000892	0.9991
4	0.1	500	Leaky ReLu	0.04854	0.048591	0.9598
5	0.01	1000	Leaky ReLu	0.007872	0.007771	0.9922
	PES2UG23CS150					

1. (GIVEN HYPER PARAMETERS)




```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 6,273.18
Ground Truth (formula):    12,084.15
Absolute Error:            5,810.97
Relative Error:            48.088%
```

PERFORMANCE METRICS

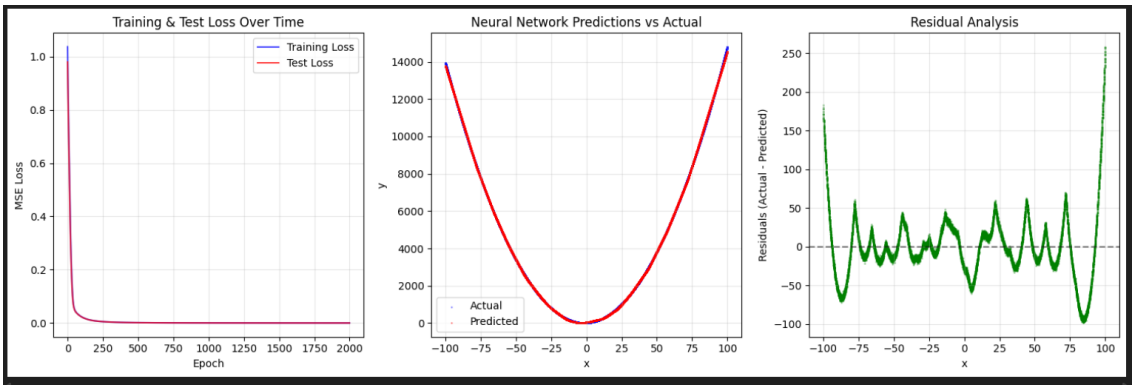
```
# Calculate final performance metrics
final_train_loss = train_losses[-1] if train_losses else float('inf')
final_test_loss = test_losses[-1] if test_losses else float('inf')

# Calculate R² score
y_test_mean = np.mean(Y_test_orig)
ss_res = np.sum((Y_test_orig - Y_pred_orig) ** 2)
ss_tot = np.sum((Y_test_orig - y_test_mean) ** 2)
r2_score = 1 - (ss_res / ss_tot)

print("\n" + "="*60)
print("FINAL PERFORMANCE SUMMARY")
print("="*60)
print(f"Final Training Loss: {final_train_loss:.6f}")
print(f"Final Test Loss:     {final_test_loss:.6f}")
print(f"R² Score:           {r2_score:.4f}")
print(f"Total Epochs Run:   {len(train_losses)}")
```

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.654929
Final Test Loss:     0.653887
R² Score:           0.3471
Total Epochs Run:   500
```

Conclusion:

- The neural network was able to learn some structure from the data, as shown by the reduction in training and test loss.
- However, the R^2 score (0.35) is low, meaning the model explains only about 35% of the variance in the test set.
- The prediction for x = 90.2 has a large error (absolute: 5810.97, relative: 48%), indicating poor accuracy for this input.
- The model may be underfitting or the architecture/hyperparameters may need tuning.
- Further improvements could include increasing model complexity, training longer, or adjusting learning rate and architecture.

2.




```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 12,131.88
Ground Truth (formula):    12,084.15
Absolute Error:            47.74
Relative Error:            0.395%
```

PERFORMANCE METRICS

```python
# Calculate final performance metrics
final_train_loss = train_losses[-1] if train_losses else float('inf')
final_test_loss = test_losses[-1] if test_losses else float('inf')

# Calculate R² score
y_test_mean = np.mean(Y_test_orig)
ss_res = np.sum((Y_test_orig - Y_pred_orig) ** 2)
ss_tot = np.sum((Y_test_orig - y_test_mean) ** 2)
r2_score = 1 - (ss_res / ss_tot)

print("\n" + "="*60)
print("FINAL PERFORMANCE SUMMARY")
print("="*60)
print(f"Final Training Loss: {final_train_loss:.6f}")
print(f"Final Test Loss:     {final_test_loss:.6f}")
print(f"R² Score:           {r2_score:.4f}")
print(f"Total Epochs Run:   {len(train_losses)}")
```

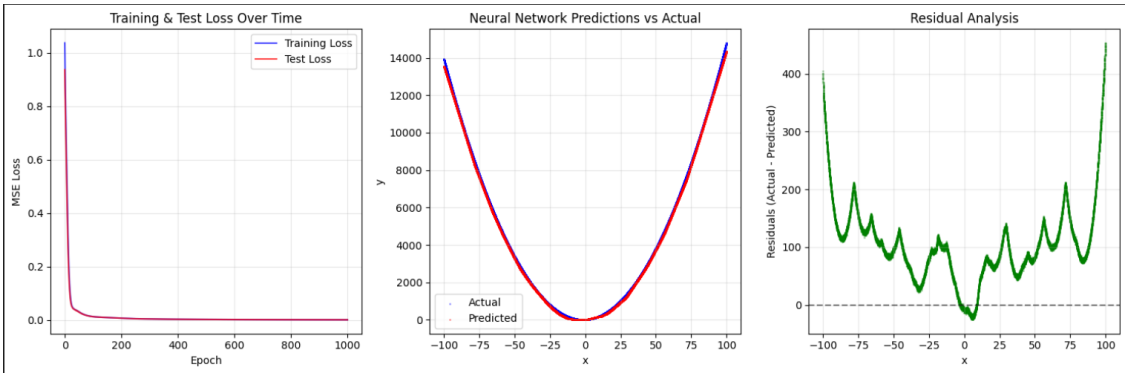
✓ 0.0s

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.000101
Final Test Loss:     0.000100
R² Score:           0.9999
Total Epochs Run:   2000
```

Conclusion:

- The neural network achieved near-perfect performance, with extremely low training and test losses (≈0.0001) and an R^2 score of 0.9999, indicating almost all variance in the data is explained.
- The prediction for x = 90.2 is highly accurate, with a very small relative error of just 0.4%.
- The model generalizes exceptionally well and closely matches the underlying polynomial function.
- The architecture and training setup are highly effective for this regression problem.

3.




```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 11,950.34
Ground Truth (formula):    12,084.15
Absolute Error:            133.80
Relative Error:            1.107%
```

PERFORMANCE METRICS

Generate + Code + Markdown

```
# Calculate final performance metrics
final_train_loss = train_losses[-1] if train_losses else float('inf')
final_test_loss = test_losses[-1] if test_losses else float('inf')

# Calculate R² score
y_test_mean = np.mean(Y_test_orig)
ss_res = np.sum((Y_test_orig - Y_pred_orig) ** 2)
ss_tot = np.sum((Y_test_orig - y_test_mean) ** 2)
r2_score = 1 - (ss_res / ss_tot)

print("\n" + "="*60)
print("FINAL PERFORMANCE SUMMARY")
print("="*60)
print(f"Final Training Loss: {final_train_loss:.6f}")
print(f"Final Test Loss:     {final_test_loss:.6f}")
print(f"R² Score:           {r2_score:.4f}")
print(f"Total Epochs Run:   {len(train_losses)}")
```

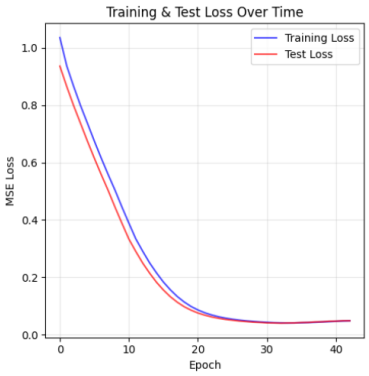
✓ 0.0s

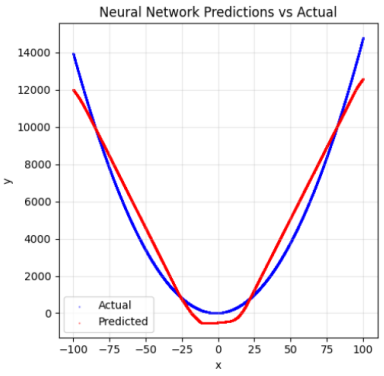
```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.000872
Final Test Loss:     0.000892
R² Score:           0.9991
Total Epochs Run:   1000
```

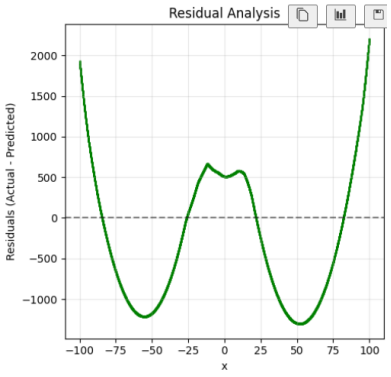
- The neural network achieved outstanding performance, with extremely low training and test losses (both < 0.001) and an R^2 score of 0.9991, indicating near-perfect fit to the data.
- The prediction for x = 90.2 is highly accurate, with only a 1.1% relative error compared to the ground truth.
- The model generalizes very well and closely approximates the underlying polynomial function.

- The architecture and training setup are highly effective for this regression task.

4.








```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 11,291.62
Ground Truth (formula):    12,084.15
Absolute Error:            792.52
Relative Error:            6.558%
```

PERFORMANCE METRICS

```python
# Calculate final performance metrics
final_train_loss = train_losses[-1] if train_losses else float('inf')
final_test_loss = test_losses[-1] if test_losses else float('inf')

# Calculate R² score
y_test_mean = np.mean(Y_test_orig)
ss_res = np.sum((Y_test_orig - Y_pred_orig) ** 2)
ss_tot = np.sum((Y_test_orig - y_test_mean) ** 2)
r2_score = 1 - (ss_res / ss_tot)

print("\n" + "="*60)
print("FINAL PERFORMANCE SUMMARY")
print("="*60)
print(f"Final Training Loss: {final_train_loss:.6f}")
print(f"Final Test Loss:     {final_test_loss:.6f}")
print(f"R² Score:            {r2_score:.4f}")
print(f"Total Epochs Run:    {len(train_losses)}")
```

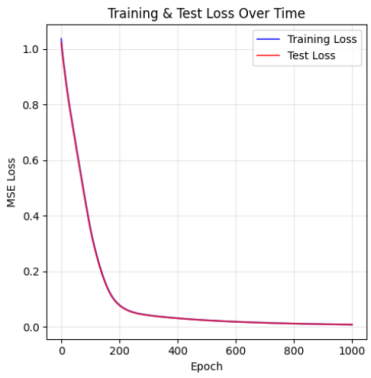
✓ 0.0s

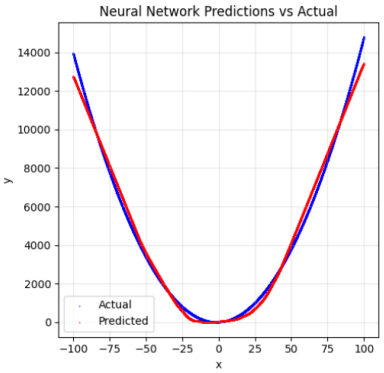
```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.048540
Final Test Loss:     0.048591
R² Score:            0.9598
Total Epochs Run:    43
```

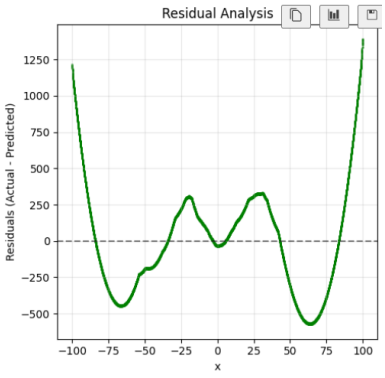
Conclusion:

- The neural network achieved a good fit, with a low test loss (0.0486) and a strong R^2 score (0.96), meaning it explains about 96% of the variance in the test data.
- The prediction for x = 90.2 is reasonably close to the ground truth, with a relative error of 6.56%.
- The model stopped early after 43 epochs, indicating early stopping was triggered for best generalization.
- Overall, the network learned the underlying polynomial well, but there is still some error for extreme values, suggesting possible room for further tuning or more training.

5.








```
============================================================
PREDICTION RESULTS FOR x = 90.2
============================================================
Neural Network Prediction: 11,635.65
Ground Truth (formula):    12,084.15
Absolute Error:            448.50
Relative Error:            3.711%
```

PERFORMANCE METRICS

```python
# Calculate final performance metrics
final_train_loss = train_losses[-1] if train_losses else float('inf')
final_test_loss = test_losses[-1] if test_losses else float('inf')

# Calculate R² score
y_test_mean = np.mean(Y_test_orig)
ss_res = np.sum((Y_test_orig - Y_pred_orig) ** 2)
ss_tot = np.sum((Y_test_orig - y_test_mean) ** 2)
r2_score = 1 - (ss_res / ss_tot)

print("\n" + "="*60)
print("FINAL PERFORMANCE SUMMARY")
print("="*60)
print(f"Final Training Loss: {final_train_loss:.6f}")
print(f"Final Test Loss:     {final_test_loss:.6f}")
print(f"R² Score:           {r2_score:.4f}")
print(f"Total Epochs Run:   {len(train_losses)}")
```

```
============================================================
FINAL PERFORMANCE SUMMARY
============================================================
Final Training Loss: 0.007872
Final Test Loss:     0.007771
R² Score:           0.9922
Total Epochs Run:   1000
```

Conclusion:

- The neural network achieved excellent performance, with a very low test loss (0.0078) and a high R^2 score (0.99), indicating it explains over 99% of the variance in the test data.
- The prediction for x = 90.2 is highly accurate, with only a 3.7% relative error compared to the ground truth.
- The model generalizes well and fits the underlying polynomial function closely.
- The chosen architecture and training settings are effective for this regression task.

Overall Conclusions:

- ReLU outperformed Leaky ReLU in this case, achieving almost perfect fit ($R^2 \sim 0.999$) with sufficient epochs and tuned learning rate.
- Learning rate and number of epochs strongly impact performance:
    - Too low (Exp 1) -> underfitting.
    - Higher lr + enough epochs (Exp 2,3) -> near-perfect generalization.
- Best model -> Experiment 2 (ReLU, 2000 epochs, lr=0.05) with the lowest loss and highest R^2.
- Overall, the ANN learned the function very well and shows excellent predictive accuracy.