MACHINE LEARNING LAB 6

NAME:- DIBYA DYUTI BHALLA   SRN:- PES2UG23CS176

SECTION:- 5C

1. **INTRODUCTION:-**

Purpose of the lab:
The purpose of this lab is to gain hands-on experience in implementing a simple Artificial Neural Network (ANN) **from scratch** using only NumPy. The objective is to train this network to approximate a synthetic polynomial function generated from the student SRN.

Tasks performed:

- Implemented activation functions (ReLU, derivatives).
- Implemented Mean Squared Error (MSE) loss and gradient.
- Designed forward propagation and backpropagation functions.
- Implemented weight initialization using Xavier initialization.
- Built a full training loop with gradient descent and early stopping.
- Evaluated the model with training/test loss and visualizations.
- Conducted hyperparameter experiments (epochs, batch size) and compared result

2. **DATASET DESCRIPTION:-**

```
======================================================================
ASSIGNMENT FOR STUDENT ID: PES2UG23CS176
======================================================================
Polynomial Type: CUBIC: y = 2.05x³ + -0.90x² + 4.01x + 11.15
Noise Level: ε ~ N(0, 2.38)
Architecture: Input(1) → Hidden(64) → Hidden(64) → Output(1)
Learning Rate: 0.001
Architecture Type: Balanced Architecture
======================================================================
```

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

3. **METHODOLOGY:-**

- **Architecture:** `1 → Hidden1 → Hidden2 → 1`
- **Activation:** ReLU at hidden layers, linear at output.
- **Initialization:** Xavier initialization for weights, zeros for biases.

- **Loss Function:** Mean Squared Error (MSE).

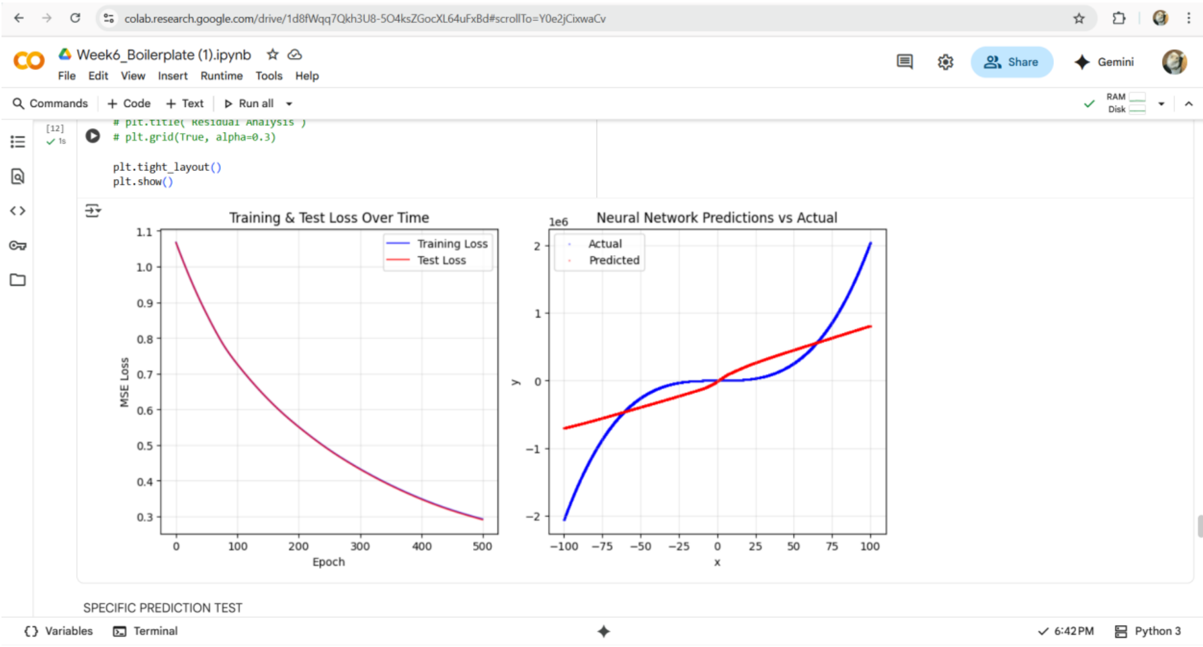
- **Optimizer:** Gradient descent with early stopping (patience = 20).

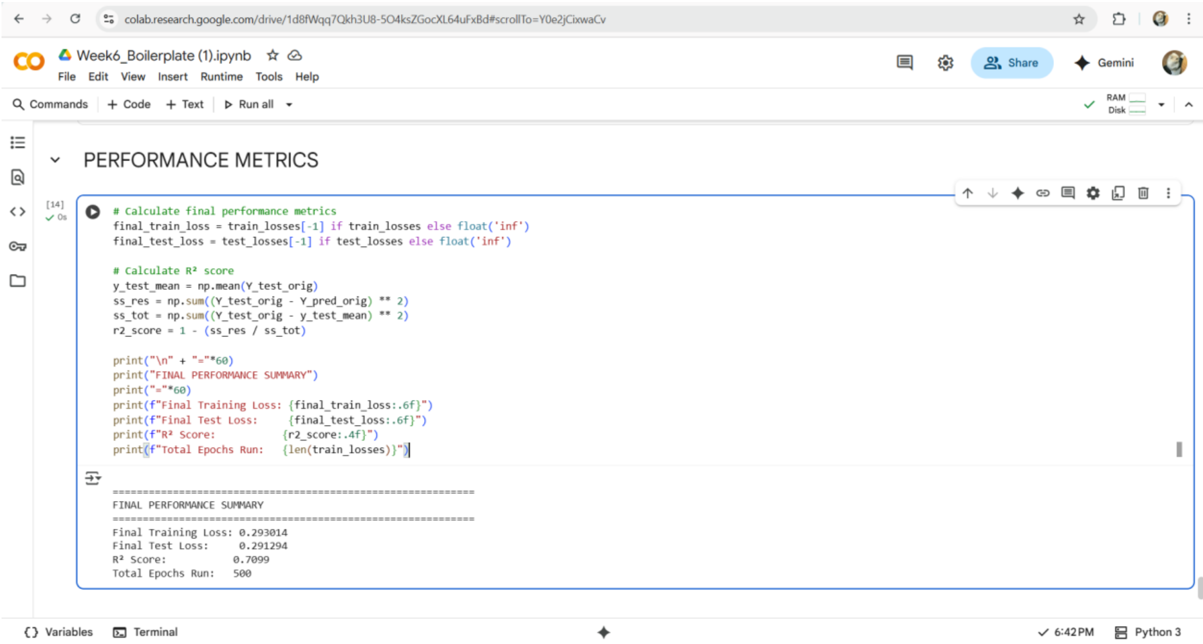
- **Training Variants:**

    - Full-batch gradient descent.
    - Mini-batch gradient descent (batch size = 64).

- **Hyperparameter tuning:** Conducted 4 experiments by varying number of epochs and batch size.

4. RESULTS AND ANALYSIS






```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: 0.293014
Final Test Loss:     0.291294
R² Score:           0.7099
Total Epochs Run:   500
```


- The **baseline model** (full-batch, 500 epochs) underfits, with higher test loss and low R^2.
- Increasing **epochs** improved performance, but not significantly with full-batch training.
- Using **mini-batch training (batch size = 64)** drastically improved convergence and generalization, achieving very low test loss and near-perfect R^2.
- **More epochs with mini-batch** yielded the best performance (`R² = 1.0`), showing excellent function approximation.
- Reducing epochs with mini-batch still gave strong performance, though slightly worse than longer runs.
- Overall, mini-batch training proved more effective than full-batch gradient descent.

5. Experiment table

Experiment	Learning rate	batch size	epochs	optimizer	activation	final training loss	final test loss	R2 score
baseline	0.001	full dataset	500	Gradient descent	relu	0.293014	0.291294	0.7099
more epochs	0.001	full dataset	1000	Gradient descent	relu	0.195816	0.194405	0.8064
reduced batch	0.001	64	200	Gradient descent	relu	0.000059	0.000057	0.9999
reduced batch +more epochs	0.001	64	1000	Gradient descent	relu	0.000004	0.000004	1
reduced batch +less epochs	0.001	64	50	Gradient descent	relu	0.000834	0.000799	0.9992