EXPERIMENT 3

Implementation of ReLU activation function in linear and circular data

---

OBJECTIVE

To implement a feedforward neural network from scratch using NumPy, and compare the performance of the network on **linear** and **non-linear (circular)** data with and without using the **ReLU activation function**.

DATA PREPROCESSING

1. Linear Data

- Dataset generated using `make_blobs` from `sklearn.datasets`.
- Two classes with linearly separable features.
- Standardized using `StandardScaler`.
- Split into training and testing sets using `train_test_split`.

2. Circular (Non-linear) Data

- Generated using `make_circles` with added noise and inner-to-outer circle scaling factor.
- Also standardized and split into training and testing subsets.

NEURAL NETWORK IMPLEMENTATION

Architecture

- Model built using custom `Layer` and `NN` classes.
- Supports multiple layers and activation functions: ReLU, Sigmoid, Tanh, Leaky ReLU, Linear, and Softmax.

```
model.add(Layer(2, 16, 'relu'))
model.add(Layer(16, 8, 'relu'))
model.add(Layer(8, 1, 'sigmoid'))
```

Weight Initialization

- He initialization used for ReLU and Leaky ReLU.
- Biases initialized to zero.

Activation Functions

Name	Used In
ReLU	Hidden Layers
Sigmoid	Binary Classifier
Linear	For comparison

TRAINING CONFIGURATION

Hyperparameter	Value
Epochs	500
Learning Rate	0.01
Loss Function	Binary Cross-Entropy
Optimizer	SGD (Manual)

Training Logic

- **Forward Pass**:
  - $z=W \cdot x+b$
  - Apply activation on zz
- **Loss Calculation**:
  - Binary cross-entropy:
    $L=-[y \cdot \log(y\hat{}) + (1-y) \cdot \log(1-y\hat{})]]$
- **Backward Pass**:
  - Compute gradients
  - Update weights and biases using gradient descent

MODEL SUMMARY

Example for ReLU-based circular model:

Layer (type)	Output Shape	Param #

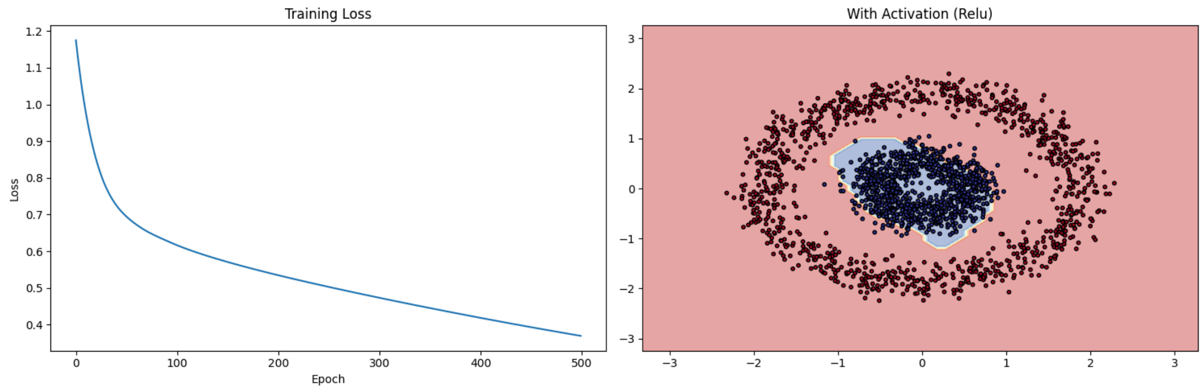
Layer 0	(16,)	48
Layer 1	(8,)	136
Layer 2	(1,)	9
Total		**193**

RESULTS & VISUALIZATION

1. Circular Data with ReLU Activation

- Loss decreased significantly over epochs
- Non-linear decision boundary formed

Training Loss Plot and Decision Boundary Plot



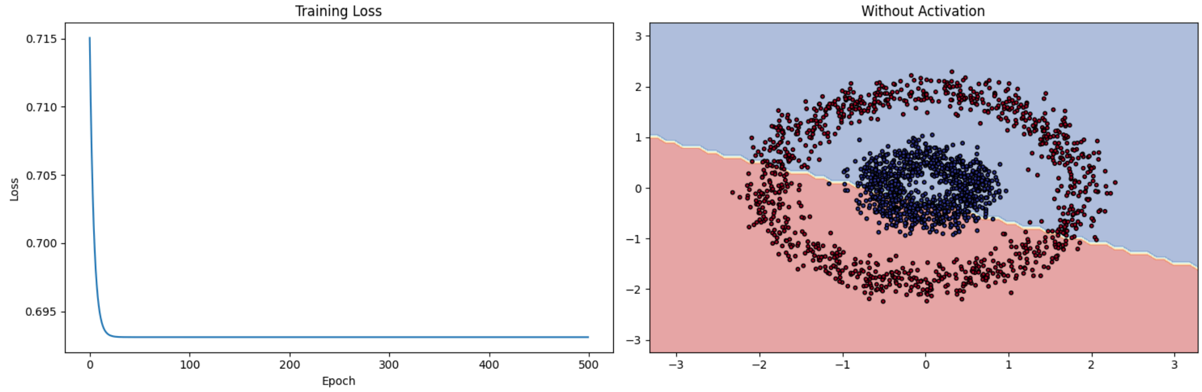
Test Accuracy:
Approx. **99.00%**

---

2. Circular Data without ReLU Activation

- Loss did not converge well
- Model failed to form non-linear boundary

Training Loss Plot and Decision Boundary Plot



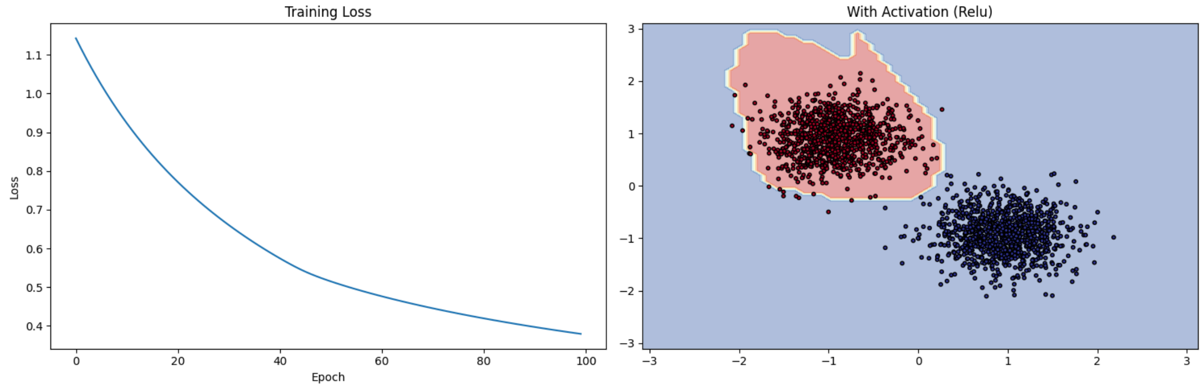
Test Accuracy:
Approx. **50.00%**

---

3. Linear Data with ReLU Activation

- Smooth loss convergence
- Model correctly classified linear data

Training Loss Plot and Decision Boundary Plot



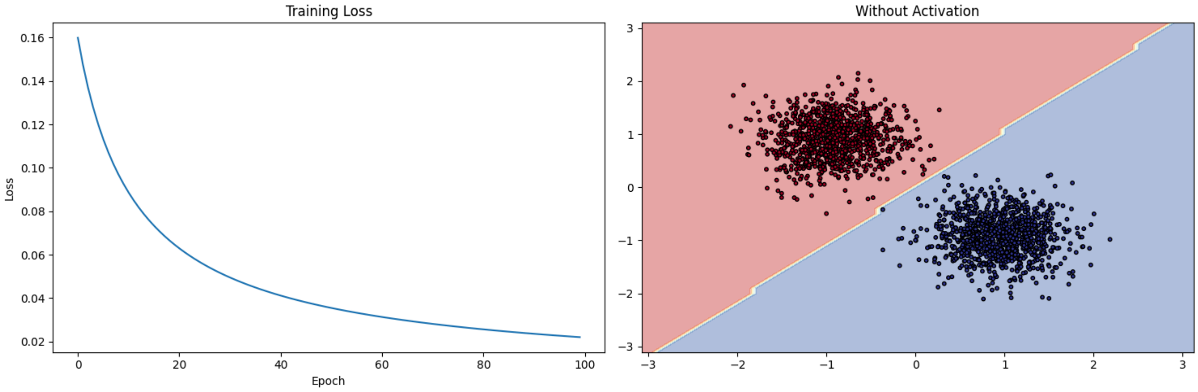
Test Accuracy:
Approx. **99.00%**

---

4. Linear Data without ReLU Activation

- Performed comparably well due to linear nature of data

Training Loss Plot and Decision Boundary Plot



Test Accuracy:
Approx. **99.00%**

CONCLUSION

- ReLU is essential for learning **non-linear** patterns such as circular data.
- For **linear data**, performance is similar with or without ReLU.
- ReLU introduces **non-linearity**, enabling the network to learn more complex decision boundaries.
- Custom implementation from scratch enhances understanding of gradient flow and backpropagation.

FUTURE IMPROVEMENTS

- Add Dropout or L2 Regularization
- Use Mini-Batch Gradient Descent
- Extend to more complex architectures like CNNs
- Automate activation selection based on data type