EXPERIMENT 2

Implementation of Neural Network from Scratch using Numpy

OBJECTIVE

To implement a Feedforward Neural Network with backpropagation using NumPy and train it on the MNIST dataset for handwritten digit classification. The model supports multiple activation functions and includes weight initialization, gradient-based learning, and evaluation metrics.

DATA PREPROCESSING

MNIST Dataset Conversion

- The MNIST dataset was provided in binary format (`idx3-ubyte` and `idx1-ubyte`) and converted to CSV using a custom `mnist_to_csv` function.
- Each image consists of 28x28 (784) grayscale pixels.
- Output labels range from 0 to 9, representing digits.
- Inputs are normalized between 0 and 1 by dividing by 255.
- Labels are extracted and stored as integers.

NEURAL NETWORK IMPLEMENTATION

Architecture

- The model supports **multiple hidden layers** with custom activation functions.
- Implemented using Python classes:
    - `Layer` – stores weights, biases, and activation type.
    - `NN` – manages training, forward and backward pass, and evaluation.

```
model.add(Layer(784, 128, 'relu'))
model.add(Layer(128, 64, 'relu'))
model.add(Layer(64, 10, 'softmax'))
```

Weight Initialization

- He initialization for ReLU/Leaky ReLU.
- Xavier initialization for Sigmoid/Tanh/Softmax.

Activation Functions

Name	Used In
ReLU	Hidden Layers
Sigmoid	Binary Classifier
Softmax	Multiclass Output
Tanh	Optional
Leaky ReLU	Optional

TRAINING CONFIGURATION

Hyperparameter	Value
Epochs	1000
Learning Rate	0.01
Loss Function	Cross-Entropy
Batch Size	Full Batch
Optimizer	SGD (manual)

Training Logic

- **Forward Pass**:
  - Calculates `z = Wx + b`, followed by activation.
  - Stores intermediate activations and `z` values for use in backpropagation.

- **Loss Calculation**:
  - For Softmax: Categorical Cross-Entropy
  - For Sigmoid: Binary Cross-Entropy

- **Backward Pass**:
  - Computes gradients of weights and biases.
  - Updates weights using Gradient Descent.

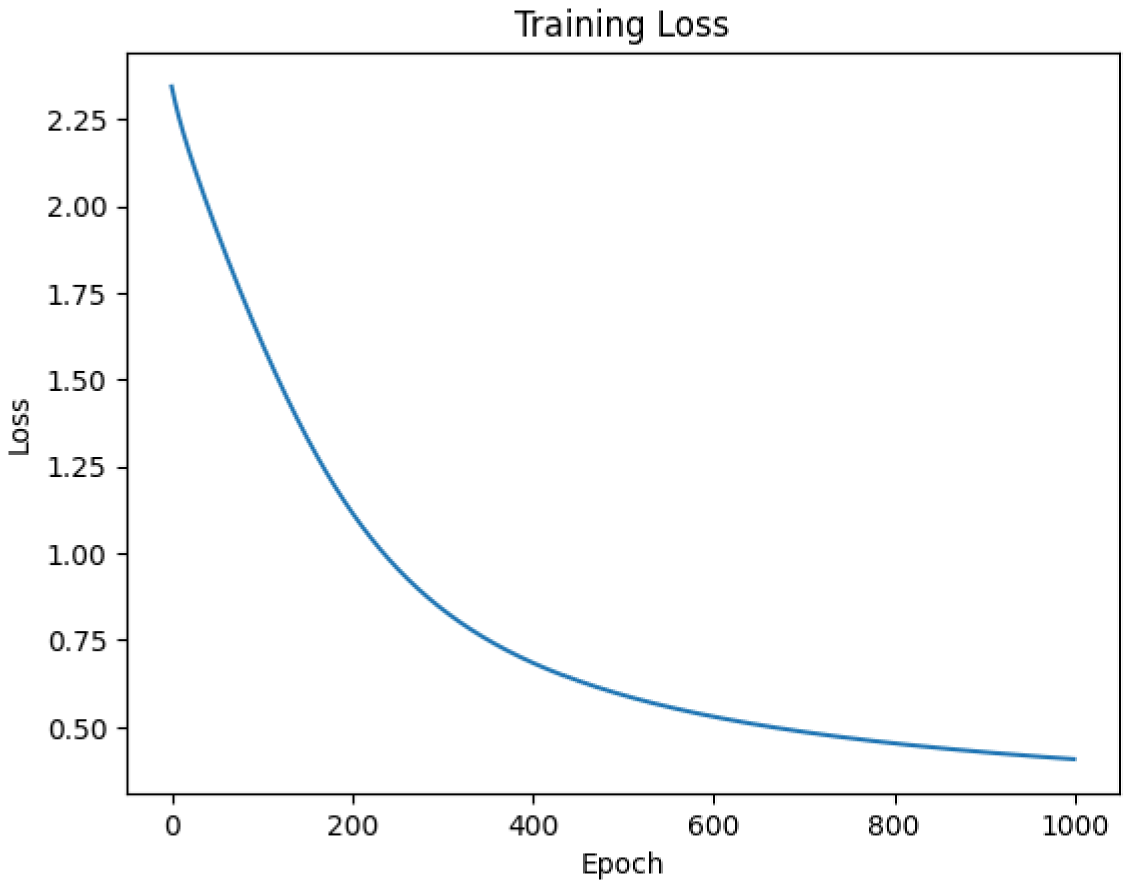
MODEL SUMMARY

Layer (type)   Output Shape   Param #
==

Layer 0	(128,)	100480
Layer 1	(64,)	8256
Layer 2	(10,)	650

==
Total params: 109386

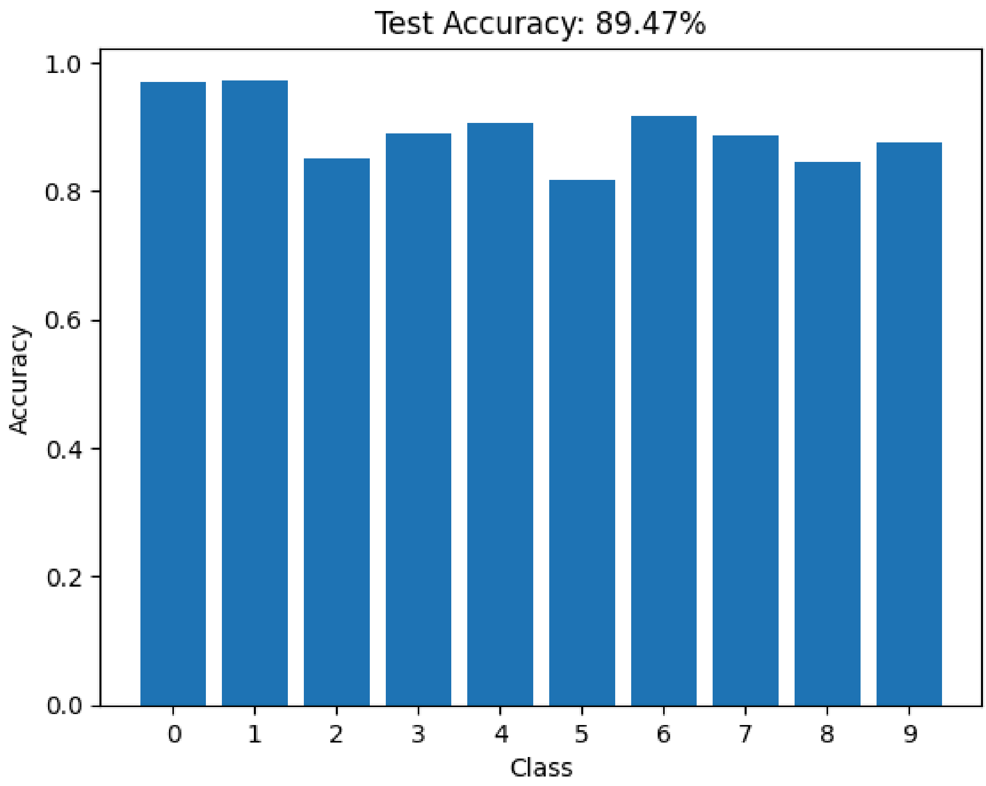
TRAINING PERFORMANCE



- Loss decreased steadily across epochs.
- Achieved stable convergence by epoch 800+.
- Final model saved to disk for later use.

model.save_model('./results/models/doubleLayer.pkl')

RESULTS & VISUALIZATION



Training Loss Plot

A graph was generated using `matplotlib`:

- Shows rapid decrease in loss early on.
- Slower convergence after epoch 500.

Test Accuracy

- Accuracy calculated by comparing predictions to ground truth.
- Class-wise accuracy charted for digits 0–9.

Metric	Value
Overall Test Accuracy	89.47%
Best Class Accuracy	>98% (digit '1')
Worst Class Accuracy	~85% (digit '5')

CONCLUSION

- The model generalizes well on the MNIST dataset with simple preprocessing and basic architecture.
- ReLU and Softmax activation functions yield competitive results.
- The manual implementation of backpropagation deepens the understanding of gradient flow and optimization.

FUTURE IMPROVEMENTS

- Add Dropout or L2 Regularization.
- Use Mini-Batch Gradient Descent for faster convergence.
- Extend to CNN for spatial feature learning.