MNIST Classification Using a Neural Network Implemented From Scratch (NumPy Only)

Project Report

February 15, 2026

Abstract

This report presents the implementation and evaluation of a fully connected neural network trained from scratch (using only NumPy) for handwritten digit classification on the MNIST dataset. The notebook performs data loading, preprocessing, exploratory data analysis (EDA), model implementation, training with Adam optimization, and final evaluation including per-class metrics and visualization. The final model achieves a test accuracy of 98.38%, demonstrating strong performance without using deep learning frameworks.

1 Introduction

The goal of this project is to implement a multi-layer perceptron (MLP) entirely from scratch using NumPy and apply it to the MNIST handwritten digit dataset.

Unlike typical implementations using PyTorch or TensorFlow, this project manually defines:

- Weight initialization (He/Xavier)
- Forward propagation
- Backpropagation
- Cross-entropy loss
- Adam optimizer
- Regularization and early stopping

This provides full transparency into the mathematical foundations of neural networks.

2 Dataset and Preprocessing

2.1 Dataset

- **Name:** MNIST

- **Source:** OpenML `mnist_784` (version 1)
- **Total samples:** 70,000
- **Image size:** 28×28 grayscale
- **Flattened dimension:** 784

2.2 Train / Validation / Test Split

- Training set: 54,000 samples
- Validation set: 6,000 samples
- Test set: 10,000 samples

2.3 Preprocessing

- Pixel values normalized to range $[0, 1]$
- Labels converted to integer format
- 10% of training data used for validation

3 Exploratory Data Analysis (EDA)

The notebook includes:

- Class distribution visualization (balanced dataset)
- Sample images per class (3×10 grid)
- Mean image per digit class
- Pixel value distribution histogram

Observations:

- Classes are approximately balanced.
- Mean images clearly reveal structural digit patterns.
- Pixel values are heavily concentrated near zero.

4 Model Architecture

4.1 Network Structure

$$784 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 10$$

- Hidden activations: ReLU
- Output activation: Softmax
- Dropout rate: 0.3

4.2 Mathematical Formulation

For each layer l:

$$Z^{(l)} = W^{(l)} A^{(l-1)} + b^{(l)}$$

$$A^{(l)} = \text{ReLU}(Z^{(l)})$$

Output layer:

$$\hat{Y} = \text{softmax}(Z^{(L)})$$

Loss function (cross-entropy):

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

5 Training Procedure

5.1 Optimization

- Optimizer: Adam
- Learning rate: 0.001
- Batch size: 128
- L2 regularization: 10^{-4}
- Early stopping patience: 10 epochs

5.2 Training Progress

Training stopped at epoch 39 due to early stopping. Best validation accuracy was achieved at epoch 29.

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	0.3906	0.9650	0.1298	0.9607
5	0.0863	0.9888	0.0823	0.9742
10	0.0566	0.9938	0.0718	0.9785
15	0.0455	0.9941	0.0702	0.9803
20	0.0433	0.9966	0.0694	0.9807
25	0.0396	0.9976	0.0650	0.9822
29	–	–	0.0674	**0.9832**

Table 1: Training and Validation Performance

6 Test Set Evaluation

6.1 Overall Performance

- Test Accuracy: **98.38%**
- Test Loss: **0.0578**

6.2 Per-Class Metrics

Digit	Precision	Recall	F1 Score
0	0.986	0.992	0.989
1	0.993	0.992	0.993
2	0.985	0.980	0.983
3	0.983	0.980	0.982
4	0.981	0.978	0.979
5	0.988	0.988	0.988
6	0.989	0.981	0.985
7	0.982	0.981	0.981
8	0.980	0.980	0.980
9	0.970	0.986	0.978

Table 2: Per-Class Precision, Recall and F1 Scores

6.3 Observations

- Highest F1 score: Digit 1 (0.993)
- Lowest precision: Digit 9 (0.970)
- All classes achieve F1 ≥ 0.978
- Minor confusion occurs between visually similar digits

7 Conclusion

This project demonstrates that a fully connected neural network implemented entirely in NumPy can achieve 98.38% accuracy on MNIST. Adam optimization, dropout, L2 regularization, and early stopping effectively control overfitting while maintaining high generalization performance. The model performs consistently across all digit classes and provides strong empirical validation of the implementation.