Neural Network from Scratch with NumPy: Project Report

Harsh Rana

June 5, 2025

Abstract

This report details the implementation, training, and evaluation of a feedforward neural network built from scratch using only NumPy. The network is applied to the MNIST dataset for handwritten digit recognition. We discuss the mathematical foundations, implementation steps, training process, results, and provide visualizations of both training metrics and sample predictions.

Contents

1 Introduction

Neural networks are a cornerstone of modern machine learning, capable of learning complex patterns from data. While high-level frameworks like TensorFlow and PyTorch abstract away much of the underlying complexity, implementing a neural network from scratch using NumPy offers valuable insight into the mechanics of deep learning. This project focuses on classifying handwritten digits from the MNIST dataset using a simple neural network.

2 Project Objectives

1. Implement a feedforward neural network using only NumPy.
2. Explain and demonstrate the mathematical operations behind each component.
3. Train and evaluate the network on the MNIST dataset.
4. Visualize training metrics and prediction results.
5. Provide a foundation for further exploration and extension.

3 Dataset Description

The MNIST dataset consists of 70,000 grayscale images of handwritten digits (0-9), each of size 28×28 pixels. For this project:

- The dataset is split into training, validation (dev), and test sets.
- Each image is flattened into a 784-dimensional vector.
- Labels are integers from 0 to 9.

4 Neural Network Architecture

The neural network implemented in this project is a fully connected feedforward network with the following structure:

- **Input Layer:** 784 neurons (one per pixel).
- **Hidden Layer:** h neurons (hyperparameter, e.g., 64, 110, 128), with ReLU activation.
- **Output Layer:** 10 neurons (one per digit class), with softmax activation.

4.1 Parameter Initialization

Weights are initialized using He initialization for improved convergence:

$$\begin{aligned} W_1 &\sim \mathcal{N}(0, \sqrt{2/784}) \\ b_1 &= 0 \\ W_2 &\sim \mathcal{N}(0, \sqrt{2/h}) \\ b_2 &= 0 \end{aligned}$$

5 Mathematical Foundations

5.1 Forward Propagation

Given input $X \in \mathbb{R}^{784 \times m}$ (where m is the number of samples), the forward propagation steps are:

$$\begin{aligned} Z_1 &= W_1 X + b_1 \\ A_1 &= \text{ReLU}(Z_1) \\ Z_2 &= W_2 A_1 + b_2 \\ A_2 &= \text{softmax}(Z_2) \end{aligned}$$

where

$$\text{ReLU}(z) = \max(0, z)$$

and

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

5.2 Loss Function

The cross-entropy loss for multi-class classification is used:

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^{m} \log(A_2[y_i, i])$$

where y_i is the true label for sample i.

5.3 Backward Propagation

The gradients for updating the parameters are computed as follows:

Output Layer:

$$\begin{aligned} dZ_2 &= A_2 - \text{one\_hot}(Y) \\ dW_2 &= \frac{1}{m} dZ_2 A_1^T \\ db_2 &= \frac{1}{m} \sum dZ_2 \end{aligned}$$

Hidden Layer:

$$dZ_1 = W_2^T dZ_2 \odot \text{ReLU}'(Z_1)$$
$$dW_1 = \frac{1}{m} dZ_1 X^T + \frac{\lambda}{m} W_1$$
$$db_1 = \frac{1}{m} \sum dZ_1$$

where $\odot$ denotes element-wise multiplication and

$$\text{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

5.4 Parameter Update

Parameters are updated using gradient descent:

$$W := W - \alpha \cdot dW$$
$$b := b - \alpha \cdot db$$

where α is the learning rate.

5.5 Regularization

L2 regularization is applied to the weights to prevent overfitting:

$$dW_1 := dW_1 + \frac{\lambda}{m} W_1$$

where λ is the regularization parameter.

6 Implementation Details

6.1 Data Preparation

The MNIST CSV file is loaded using pandas. Data is shuffled and split into training, validation, and test sets. Pixel values are normalized to the range [0, 1].

6.2 Network Functions

The implementation includes:

- Functions for parameter initialization.
- ReLU and softmax activation functions, with their derivatives.
- Forward and backward propagation routines.

- Utility functions for predictions, accuracy, and loss calculation.
- A training loop with loss and accuracy tracking.
- Hyperparameter tuning for hidden layer size and regularization.
- Visualization of training accuracy and loss.
- Model saving and loading using pickle.

7 Training Process

7.1 Hyperparameter Tuning

The hidden layer size and L2 regularization parameter (λ) are tuned by evaluating performance on the validation set. The best combination is selected for final training.

7.2 Training Metrics

Training accuracy and loss are tracked and plotted over iterations. The plot `training_metrics.png` shows both metrics.

8 Results

8.1 Performance

The final model achieves the following performance (replace with your actual results):

- **Validation (Dev) Accuracy:** 0.92
- **Test Accuracy:** 0.91

8.2 Visualization of Training Metrics

Figure 1 shows the evolution of training accuracy and loss.

8.3 Sample Predictions

To qualitatively assess the model, we visualize several test images along with their predicted and true labels. Figures 2 show examples.

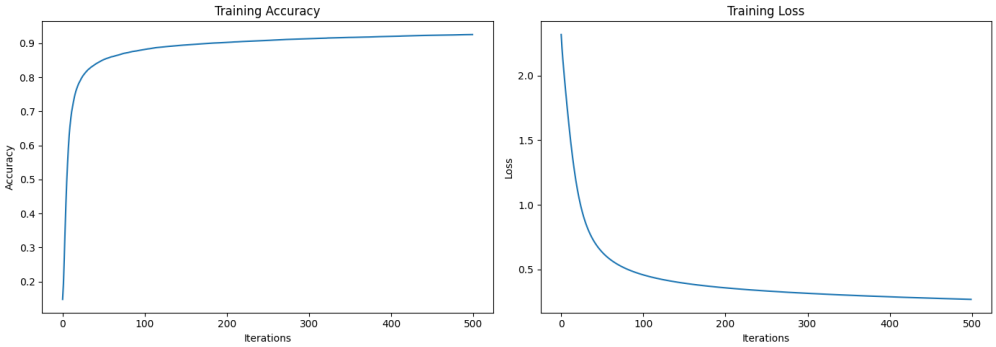

Figure 1: Training accuracy (left) and loss (right) over iterations.

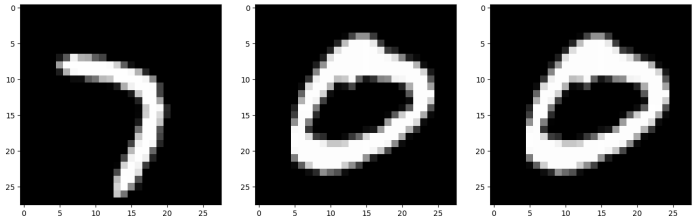

Figure 2: Sample predictions: [7,0,0] True Labels: [7,0,0]

9 Discussion

The project demonstrates that a neural network built from scratch using NumPy can achieve strong performance on the MNIST dataset. The implementation provides full transparency and control over each step, making it an excellent educational tool. However, it is not optimized for speed or large-scale datasets and is limited to shallow networks. Future work could include adding support for deeper architectures, advanced optimization algorithms, and experimenting with other datasets.

10 Conclusion

This project provides a thorough understanding of the mathematical and computational steps involved in implementing a neural network from scratch. The network achieves strong performance on the MNIST dataset and serves as a solid foundation for further exploration in deep learning.

References

[1] Artificial Neural Network - Wikipedia

[2] Backpropagation - Wikipedia

[3] Softmax Function - Wikipedia

[4] The MNIST Database

Author: Harsh Rana