A Simple MNIST Digit Classifier Neural Network Made from scratch in Python

Thiruwaran Kalvin (T-Kalv)
Undergraduate Computer Science Student

August 7, 2025

1 Abstract

This research paper presents an implementation of a basic feed-forward neural network for classifying/recognising handwritten digits from the MNIST dataset, built from scratch in Python using only core libraries such as NumPy, Matplotlib, Pandas, and tqdm. This feed-forward neural network model, also known as a multilayer perceptron (MLP), achieves a classification probability accuracy of appoximately 88% for both the training and testing datasets. This demonstrates the viability of simple neural networks that recognise handwritten numerical digits without the need for external heavy libraries such as PyTorch or TensorFlow, instead utilising only core Python libraries and Juypter Notebook.

2 Introduction

Numerical Digit Classification is a specific field of Image Processing that focuses here on this paper, classifying and recognising handwritten low-level black and white numerical digit images using the MNIST Dataset[1]. This particular task of recognising handwritten digits is vital in the real-world as previously Bank Tellers played a crucial role in processing Cheques by manually verifying (looking) at the authenticity of the Cheques and verifying the Currency amount on the Cheques. This method was time-consuming and in-efficient due to manual Human validation as well as being prone to Human errors. As the level of technology/computation advanced, the need to accurately read hand-written numerical digits especially in sectors such as the Banking industry became more crucial to reduce the likelyhood of errors and reduce processing time. This results in Financial institutions diversting resources more effectivly to other services. Through the advancements in Artificial Intelligence and Machine learning, we can develop systems that can accuracy recognise hand-written digits much more efficient and effectively using hand-written datasets such as MNIST.

MNSIT (Modified Natual Institute of Standard and Technology) database is dataset that consits of handwritten numerical digits that is commonly used for image processing training and contains 60,000 training images used to train the Machine Learning Neural Network model and 10,000 testing images to validate the accuracy of the model. In this paper, we use this famous MNIST dataset which is the standard benchbark for Image Classification algorithms. As the MNIST database was constructed before the turn of the 21st Century, there have many implementations/existing solutions using the MNIST Dataset to classify and recognise hand-written numerical digits such as using tradition methods (including but not limited to KNN (K-Nearest Neighbours), SVM (Support Vector Machines)) and even Deep-Learning methods (including but not limited to CNN

(Convultional Neural Network), RNN (Recurrent Neural Network)). Arguably, one of the most famous implementations it the LeNet-5 architecture [3] which uses CNN consiting of 2 convolutions, 2 subsamplings and 3 fully connected layers with around 60000 trainable paramaters.

This paper proposes a light-weight simple MNIST Digit Classifier Neural Network [4] consiting of 2 layers with 784 neurons, built without the use of external heavy deep-learning libraries (only using libraries such as NumPy, Matplotlib) with a particular emphasis on understanding and manually implementing algorithms and functions. This work presents such algorithms and functions such as Forward Propagation, Backpropagation, ReLu (Rectified Linear unit) activation function, Softmax activation function.... This results in a reproducible implementation of simple MNSIT Digit Classifier Neural Network, with clear, hand-on demonstration of 'behind-the-scenes' of the network as well as the training procedure and performance benefits/outcomes.

This paper is organised as follows: Section 3 discusses related work, followed by Section 4 which outlines the Methodology and architecture, Section 5 presents Experimental Evaluation and finally Section 6 concludes with findings and future work.

3 Related Work

Before working on this 'Simple MNIST Digit Classifier Neural Network', previously a 'Simple Neural Network' was developed from scratch in Python for XOR [4]. This previous project implemented key concepts such as Forward Propagation, Backpropagation, Mean-Squared Error Loss and the Sigmoid Activation function using only core libraries such as NumPy and Matplotlib. The sucess of XOR having a probability of appoximately 95% helped to inspire the next challenge of handling a more real-world dataset MNIST, implementing these key concepts on this 'Simple MNSIT Digit Classifier Neural Network'.

Previous research into using the MNIST dataset for classifying and recognising handwritten digits has produced a wide-range of models with a variety of complexity and performance benefits to the traditional manual, human-validation of handwritten digits. LeNet-5 [3], introduced in 1998 having achived appoximately a 1%[5] test error probability which resulted in appoximatly 99% probability accuracy with the MNSIT dataset, implemented using CNN (Convultional Neural Networks). While LeNET-5[5], achived a respectable, high probability accuracy using CNN, Max Pooling with 2 convultions, 2 subsampling, 3 fully connected layers and around 60000 trainable paramaters, this paper investigates the performance of a simple multilayer perceptron (MLP) using a feed-forward neutal network trained from scratch without the need of heavy, high-level frameworks. However, more modern implementations do use these heavy, high-level frameworks such as PyTorch [6], Keras [7], TensorFlow [8] utilising the performance benefits of GPUs (Graphics Processing Unit) and TPUs (Tensor Processing Units) which help to optimise the speed of learning/training as well with the use of advanced optimisers (including but not limited to Adam). Here due to the added cost of utilising such performance enhancers, the system here presented in this paper will instead utilise modern CPUs for computation and training.

4 Methodology

4.1 Dataset

4.2 Network Architecture

4.3 Forward Propagation

4.4 Backpropagation and Loss Function

4.5 Training

5 Experimental Evaluation

5.1 Accuracy Results

5.2 Observations

5.3 Limitations and Anomalies

6 Conclusion and Future Work

Appendix

References