Machine Learning Journal

Assessment Task 2 – Mitchell Kijurina – 14274334

Option 1: *study a fundamental machine learning model*

Table of Contents

[1. Introduction 3](#_Toc148092142)

[1.1 Purpose of project 3](#_Toc148092143)

[1.2 Objectives 3](#_Toc148092144)

[1.3 Scope and Limitations 3](#_Toc148092145)

[2. Background and Related Concepts 4](#_Toc148092146)

[2.1 What is a Neural Network? 4](#_Toc148092147)

[2.2 Introduction to Multi-Layer Perceptrons (MLPs) 5](#_Toc148092148)

[2.3 Overview of Activation Functions 5](#_Toc148092149)

[2.4 Understanding Loss Functions 5](#_Toc148092150)

[2.5 Basics of Training Algorithms 6](#_Toc148092151)

[3. Dataset 7](#_Toc148092152)

[4. Theoretical Foundations 8](#_Toc148092153)

[4.1 Neural Network Architecture 8](#_Toc148092154)

[4.2 Forward Propagation 8](#_Toc148092155)

[4.3 Backward Propagation 8](#_Toc148092156)

[4.4 Loss Function (Cross-Entropy) 8](#_Toc148092157)

[4.5 Gradient Descent Algorithm 8](#_Toc148092158)

[4.6 Weights and Biases 8](#_Toc148092159)

[5. Implementation Details 9](#_Toc148092160)

[5.1 Input, Hidden and output Layers. 9](#_Toc148092161)

[5.2 Activation Functions 10](#_Toc148092162)

[5.3 Optimisation function 11](#_Toc148092163)

[5.4 Loss Function 11](#_Toc148092164)

[5.5 Training Loop 11](#_Toc148092165)

[6. Results and Discussion 12](#_Toc148092166)

[6.1 Model Performance Metrics 12](#_Toc148092167)

[6.2 Loss and Accuracy Plots 12](#_Toc148092168)

[6.3 Interpretation of Results 12](#_Toc148092169)

[7. Conclusion 13](#_Toc148092170)

[7.1 Summary of Findings 13](#_Toc148092171)

[7.2 Implications 13](#_Toc148092172)

[7.3 Future Work 13](#_Toc148092173)

[Sources 14](#_Toc148092174)

[ChatGPT logs 14](#_Toc148092175)

# 1. Introduction

## 1.1 Purpose of project

The primary purpose of this project is to facilitate a deep, hands-on understanding of the fundamental concepts that drive the functioning of Multi-Layer Perceptrons (MLPs), a type of artificial neural network. By implementing an MLP from scratch, this study aims to bridge the gap between abstract theoretical principles and their concrete realization in code. The project serves as a pedagogical exercise to explore key aspects of neural network learning, such as hypothesis space, learning algorithms, and loss function design, within a practical context. This project aims to provide insights into how MLPs learn to make predictions or decisions based on data, thereby acting as a learning guide to both the implementation and theory of neural networks.

## 1.2 Objectives

The main objectives of this study are to:

* To explore the theoretical fundamentals of MLPs
* To implement an MLP model from scratch to demonstrate a clear understanding of its technical aspects.

## 1.3 Scope and Limitations

The study will focus solely on the architecture, theory, and implementation of MLPs, particularly within the context of the learning theory framework. The primary dataset for practical implementation will be the MNIST dataset for digit recognition. However, the study will not delve into variations of neural networks like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs). Further the project will not evaluate the impact of adjusting the hyper-parameters for optimisation of the network, as the focus of the study is solely on the implementation of the algorithm.

# 2. Background and Related Concepts

This section will cover a high level overview of coming topics that are explored more in-depth when looking at the implementation of the program.

## 2.1 What is a Neural Network?

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are crucial to machine learning and deep learning algorithms. Their structure is inspired by the human brain, mimicking the way biological neurons signal to each other [1].

The neuron is the fundamental unit of the neural network. Each neuron, or a node, has the following components [2].

* Inputs
* Weights
* Bias
* Activation Function

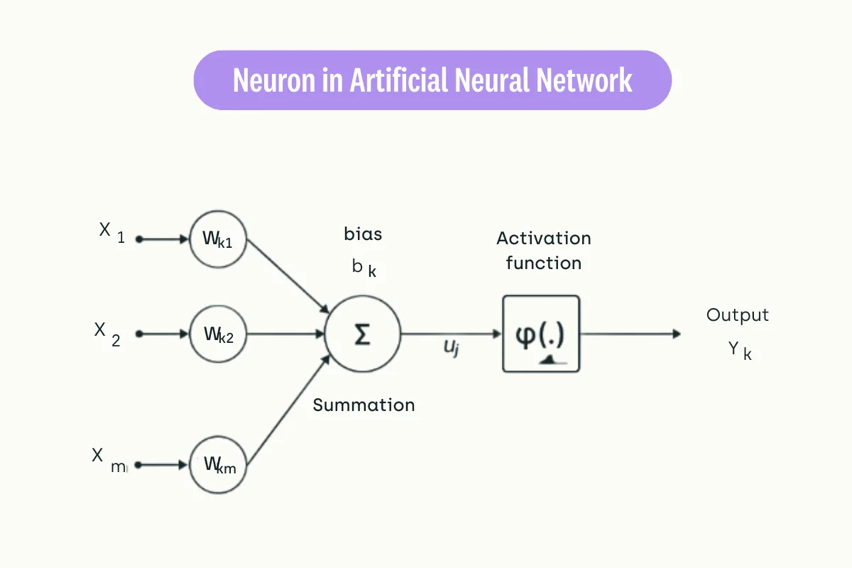


Figure 1 - Diagram of a neuron [2]

## 2.2 Introduction to Multi-Layer Perceptrons (MLPs)

Multi-Layer Perceptrons (MLPs) are a type of feedforward neural network consisting of at least three layers: an input layer, one or more hidden layers, and an output layer. They can recognize more complex patterns with the inclusion of more layers and nodes per layer [2].

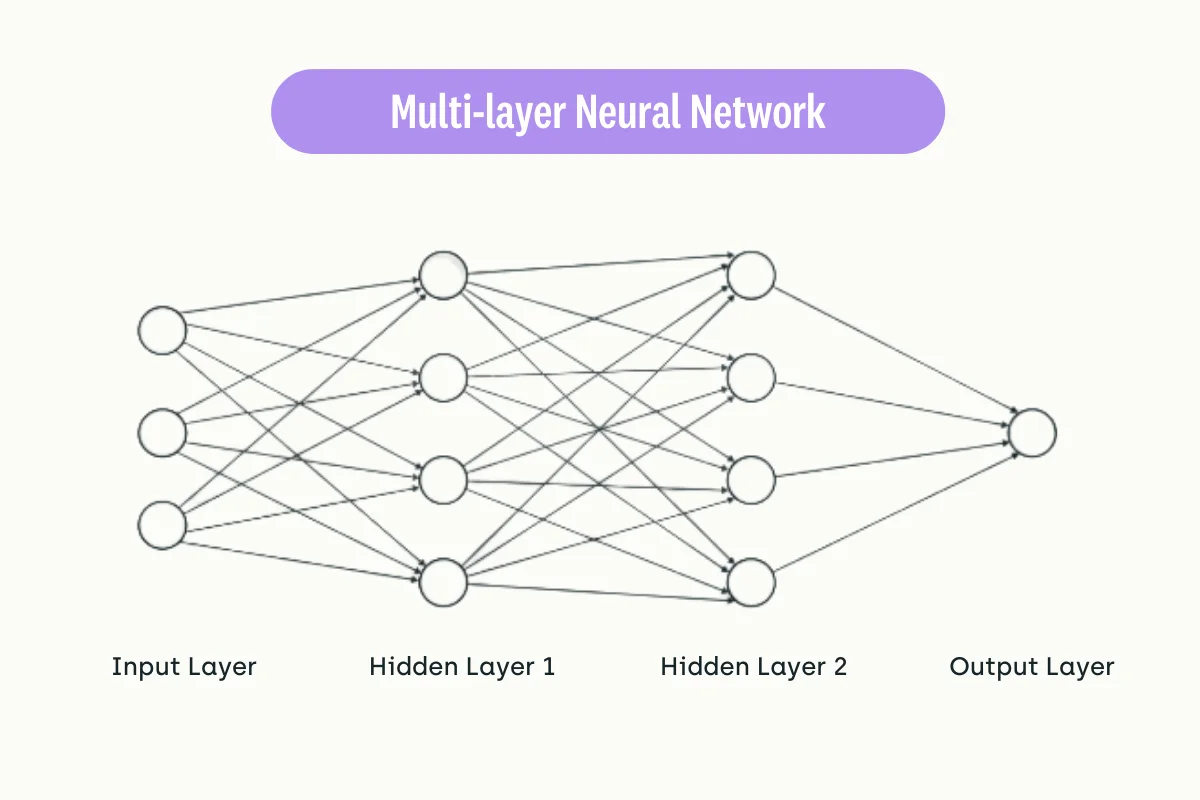


Figure 2 - Diagram of a MLP [2]

## 2.3 Overview of Activation Functions

Activation functions introduce non-linear properties to the network, enabling it to learn from the error and to handle complex problems. Common activation functions include Rectified Linear Unit (ReLU), Sigmoid, and Hyperbolic Tangent (tanh). These functions are crucial for neural network's learning process as they help in adjusting the weights during the training phase, aiding in the propagation of useful information through the network.

## 2.4 Understanding Loss Functions

Loss functions are pivotal as they measure the disparity between the predicted values and actual values, guiding the optimisation of the network's weights. The choice of a loss function, like Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks, can significantly affect the network's performance and convergence speed.

## 2.5 Basics of Training Algorithms

Training algorithms aim to minimise the loss function by iteratively adjusting the weights of the network. Backpropagation, combined with optimisation algorithms like Gradient Descent, is commonly used for this purpose. During training, the algorithm iteratively adjusts the weights to find the loss function's minimum, effectively "training" the network to improve its predictions over time.

# 3. Dataset

The Modified National Institute of Standards and Technology (MNIST) dataset is one of the most well-known and widely used datasets in the machine learning community, particularly for benchmarking classification algorithms. It consists of 60,000 grayscale images for training and an additional 10,000 images for testing. Each image in the dataset is 28x28 pixels and represents a handwritten digit from 0 to 9.

**Characteristics of MNIST:**

* **Grayscale Images**: The images are grayscale, simplifying the complexity as compared to coloured images. Each pixel has a single intensity value between 0 (black) and 255 (white).
* **Uniform Size**: All images are resized to a uniform 28x28 pixel size, thereby eliminating the need for additional pre-processing steps related to image resizing.
* **Digit Labels**: Each image comes with a corresponding label indicating the actual digit it represents, making it a supervised learning problem.
* **Balanced Classes**: The dataset is relatively balanced, meaning that there are almost an equal number of images for each of the 10-digit classes.

**Relevance to the Project:**

The MNIST dataset provides an ideal ground for implementing and understanding the Multi-Layer Perceptron model. Its simplicity and well-defined structure allow for a focus on the learning algorithm rather than data pre-processing. Moreover, the nature of the task—digit recognition—serves as a practical yet challenging problem, aligning well with the objectives of dissecting the intricacies of neural network learning both in theory and in practice.

**Source of dataset:**

The dataset was souced using the platform Kaggle; which is a machine learning and data science community driven site: <https://www.kaggle.com/datasets/hojjatk/mnist-dataset>

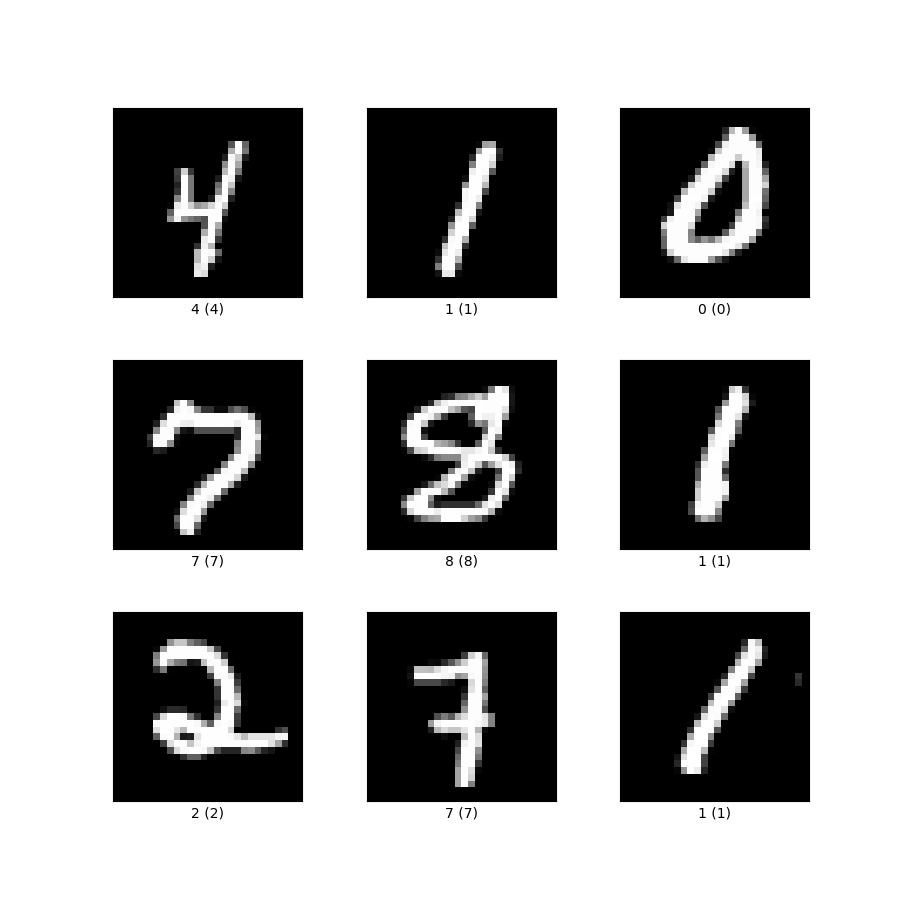


Figure 3- MNist Dataset [3]

# 4. Theoretical Foundations

## 4.1 Neural Network Architecture

## 4.2 Forward Propagation

## 4.3 Backward Propagation

## 4.4 Loss Function (Cross-Entropy)

## 4.5 Gradient Descent Algorithm

## 4.6 Weights and Biases

# 5. Implementation Details

## 5.1 Input, Hidden and output Layers.

**In a neural network, layers are structured units of nodes or neurons that transform the input data. Each layer performs specific operations dictated by activation functions and modifiable parameters (weights and biases). The network architecture in the code comprises:**

**Input Layer: This is the entry point of the network where each neuron corresponds to one feature of the dataset. In the code, the input layer implicitly has 784 neurons, matching the number of features in the MNIST dataset.**

**Hidden Layer: Hidden layers reside between the input and output layers, performing transformations on the input data. In this network, there is one hidden layer with 128 neurons, characterized by ReLU (Rectified Linear Unit) activation. The variables W1 and b1 hold the weights and biases for this layer, respectively.**

**Output Layer: This is the final layer of the network, and it typically transforms the values from the last hidden layer into output values that make sense for the given problem. In this case, the output layer has 10 neurons, each representing a class of digits (0-9). The Softmax function is applied to convert these values into probabilities. The weights and biases for this layer are stored in W2 and b2, respectively.**

**Each layer in the neural network serves to gradually transform the raw input into a form that makes it easier to produce the desired output. The transformations are governed by the layer's activation functions and its learnable parameters.**

## 5.2 Activation Functions

Activation functions introduce non-linearity into a neural network, enabling it to learn complex mappings from the input data. They act as the "gating mechanisms" that either allow or restrict information to flow through the network. Different activation functions offer various properties that can affect the performance of a neural network.

**Types of Activation Functions**

1. **ReLU (Rectified Linear Unit)**

The ReLU function replaces all negative values in the input with zero. ReLU introduces non-linearity while solving some issues that other activation functions like sigmoid or tanh might face, such as the vanishing gradient problem. It's computationally efficient because it only requires a simple thresholding at zero.

1. **SoftMax**

The SoftMax function is often used in the output layer of a classifier to represent probability distributions of target classes. Given a vector of raw scores (or logits), SoftMax squashes the values between 0 and 1 and ensures they sum up to 1. Softmax is advantageous when you want to classify an input into one of multiple classes. It returns the probability distribution of the classes, which is convenient for not just determining the most likely class but also understanding the model's confidence in its prediction.

In the forward propagation process, ReLU is used in the hidden layer while Softmax is employed in the output layer.

## 5.3 Optimisation function

The optimisation algorithm used in the code is a basic form of Gradient Descent, specifically Mini-batch Gradient Descent. This is seen from the loop structure within the train\_neural\_network() function, where the model parameters (weights and biases) are updated incrementally for each mini-batch of data.

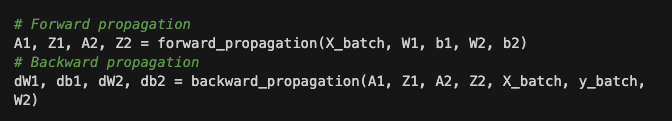
Mini-batch Gradient Descent strikes a balance between these two approaches. It divides the dataset into smaller batches and updates the model parameters for each batch.

In the train\_neural\_networks() function

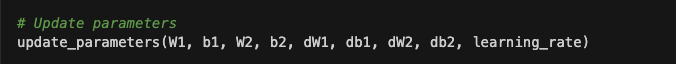
A black screen with white text

Description automatically generated

Forward and Backward Propagation: For each batch, the network undergoes a forward and a backward propagation to compute the gradients.



Parameter Update: After obtaining the gradients, the parameters are updated.



The learning\_rate controls the step size during the optimisation process. It's set prior to the training loop and is used in the update\_parameters() function to adjust the weights and biases.

## 5.4 Loss Function

## 5.5 Training Loop

# 6. Results and Discussion

## 6.1 Model Performance Metrics

## 6.2 Loss and Accuracy Plots

## 6.3 Interpretation of Results

# 7. Conclusion

## 7.1 Summary of Findings

## 7.2 Implications

## 7.3 Future Work

# Sources

1. <https://www.ibm.com/topics/neural-networks#:~:text=Neural%20networks%2C%20also%20known%20as,neurons%20signal%20to%20one%20another>
2. <https://kili-technology.com/data-labeling/machine-learning/neural-network-architecture-all-you-need-to-know-as-an-mle-2023-edition#:~:text=4,feature%20instead%20of%20different%20ones>
3. <https://www.tensorflow.org/datasets/catalog/mnist>

## ChatGPT logs

* Code implementation chat