Mathematical Analysis of Multilayer Perceptron Training
Independent Study

Robert Davidson

BSc Mathematics and Computer Science
r.davidson1@universityofgalway.ie

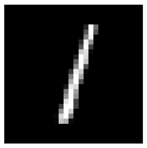
Date: March 9, 2025

Abstract

This paper provides a comprehensive mathematical derivation and implementation of a Multilayer Perceptron (MLP) for the classification of handwritten digits from the MNIST dataset, starting from fundamental principles. We begin by exploring the biological inspiration behind artificial neurons, establishing a clear mapping between biological and artificial neural components. We then develop the mathematical representation of a single artificial neuron, progressing to the construction of perceptron layers and the full MLP architecture. The necessity of hidden layers for non-linear separability is discussed, alongside the selection and application of appropriate activation functions. We detail the forward propagation process using matrix-vector notation, followed by an in-depth explanation of cost functions and the error backpropagation algorithm for gradient computation. Optimization and training techniques, including batch-based methods and strategies for improving convergence, are examined. We address the critical issue of regularization to mitigate overfitting and underfitting, including the influence of initialization strategies. Finally, we present the implementation of a fully functional MLP for MNIST classification, concluding with a discussion of extensions to deeper architectures and modern practices in neural network development. This work aims to provide a clear, step-by-step mathematical foundation for understanding and building MLPs from the ground up.

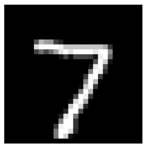
Introduction

The MNIST (Modified National Institute of Standards and Technology) dataset consists of 70,000 handwritten digit images (60,000 for training and 10,000 for testing). It is commonly used for training and evaluating machine learning models in image recognition tasks. Each image is a 28x28 pixel grayscale image, and the task is to classify the digit (0-9) represented by the image.



Sample MNIST digit 1

Sample MNIST digit 2



Sample MNIST digit 7

A Multilayer Perceptron (MLP) is a type of artificial neural network composed of multiple layers of nodes (neurons), each fully connected to the next layer. The MLP processes input data through a series of transformations, with the final layer producing an output. It is one of the fundamental types of neural networks and is used in a variety of applications, including image recognition.

In this paper, we aim to build the mathematical foundations necessary to understand and implement a Multilayer Perceptron (MLP) for the MNIST digit recognition task.

Contents

Preliminaries

Here we will cover some basic mathematical concepts that will be used throughout the paper. These will be derived from first principles, so don't worry if you're not familiar with them.

Vectors

A vector in mathematics is an ordered list of numbers, represented as a bold lowercase letter, e.g. $\mathbf{v}$. A vector can be represented as:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

We can refer to the first element as v_1, the second as v_2, and the ith element as v_i.
Each element of $\mathbf{v}$ corresponds to a coordinate in some n-dimensional space, where n is just the number of elements in the vector. The vector points to the location of these coordinates in space. For example, take the vector in 2-dimensional space:

$$\mathbf{v} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

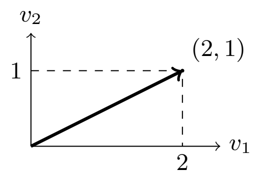


Figure 1: A 2-dimensional vector $\mathbf{v}$

Vectors

A vector is an ordered list of numbers, reprsent as a bold lowercase letter, e.g. $\mathbf{v}$. A vector can be represented as:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

We can refer to the first element as v_1, the second as v_2, and the ith element as v_i.
Each element corresponds to to a coordinate in some n-dimensional space, where n is just the number of elements in the vector. For example, we can represent a $2-$dimensional vector as:

$$\mathbf{v} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{or more commonly} \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Where $\mathbf{v}$ is a vector in 2-dimensional space, which we just refer to as $\mathbb{R}^2$.

The Transpose of a Vector

Consider a vector $\mathbf{v}$:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

The transpose of $\mathbf{v}$, denoted $\mathbf{v}^T$, is the same vector, but flipped on its side:

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}.$$

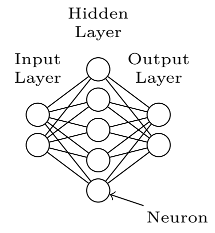
The transpose of a vector is used to convert a row vector into a column vector, or vice versa.

Multiplying Vectors : The Dot Product

Say we have a vector $\mathbf{x} = \left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]$, we say the dimension of this is 2×1. In order to multiply $\mathbf{x}$ with another vector we need the vector to be of dimension 1×2.
Let $\mathbf{v} = \left[\begin{smallmatrix} v_1 \\ v_2 \end{smallmatrix}\right]$. We note that the dimension of $\mathbf{v}$ is 2×1. We can use the transpose of v to turn it into a 1×2 vector.
Given two vectors $\mathbf{w}$ and $\mathbf{v}$:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \quad \text{and} \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

1 Motivation for the Multilayer Perceptron



The perceptron is composed of multiple artificial neurons. Each of which makes a simple decision based on its inputs. These neurons are subsequently assembled into layers, with each layer functioning as a distinct 'area' of the brain. In this way, the Multilayer Perceptron serves as an abstract model inspired by the brain's structure.

1.1 The Structure and Function of a Biological Neuron

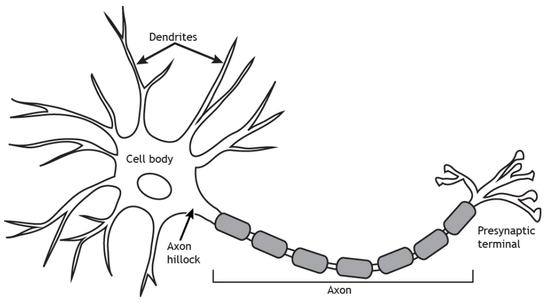
Neurons, the brain's fundamental processing units, transmit information throughout the nervous system. The human brain contains about 86 billion neurons [1], each comprising:

- **Dendrites:** Branch-like structures that receive input signals from other neurons.
- **Cell Body (Soma):** Combines incoming signals and determines whether the neuron should fire.
- **Axon:** Carries the electrical impulse (if the combined signal at the soma exceeds the threshold) to other neurons.
- **Synapse:** Junction between neurons; connects one neuron's axon to another's dendrites.
- **Firing Threshold** The neuron fires only if the total input exceeds a certain threshold.

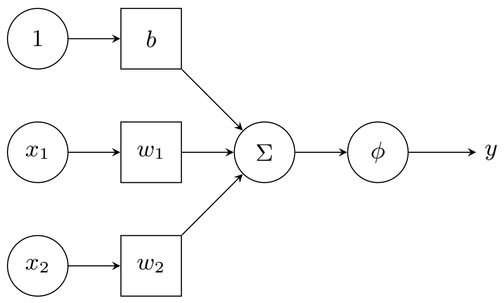
1.2 Mapping the Biological Neuron to the Artificial Neuron

The perceptron, proposed by Frank Rosenblatt in 1958 [3], is a simplified model of the biological neuron. Its components are:

- **Inputs:** $x_1, x_2, \ldots, x_n$, analogous to dendrites.
- **Weights:** $w_1, w_2, \ldots, w_n$, indicating each input's importance.
- **Summation:** $\sum w_i x_i + b$, analogous to the soma, with b as a bias term.
- **Activation Function** (ϕ)**:** applies a mathematical function to the weighted sum to determine the output y, analogous to the neuron's firing mechanism.
- **Output:** The final signal from the perceptron.



(a) Human neuron [2]



(b) Artificial neuron

Figure 2: Comparison of biological and artificial neurons.

2 Artificial Neurons

2.1 Mathematical Representation

As we discussed, biological neurons receive multiple weighted signals and fire when the combined input surpasses a threshold. In the artificial neuron, we mathematically represent this process as follows:

2.1.1 Inputs and Weights as Vectors

We need some way to represent the list of inputs signals to the A.N (artificial neuron). We can do this by using a vector, which is a mathematical object that represents a list of numbers.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

where each x_i represents an individual input signal.
Similarly, the weights associated with each input are also represented as a weight vector:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Each weight w_i corresponds to the input x_i and represents the strength of that input's connection to the neuron - analogous to synaptic strength in a biological neuron. A larger magnitude of w_i indicates a stronger influence of x_i on the neurons output.

2.1.2 Weighted Sum

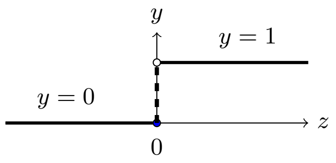
The neuron calculates a weighted sum of its inputs, analogous to the combined signals reaching the soma of a biological neuron. This summation, denoted as z, is defined as:

$$\begin{aligned} z &= w_1x_1 + w_2x_2 + \ldots + w_nx_n + b \\ &= \sum_{i=1}^{n} w_ix_i + b \end{aligned}$$

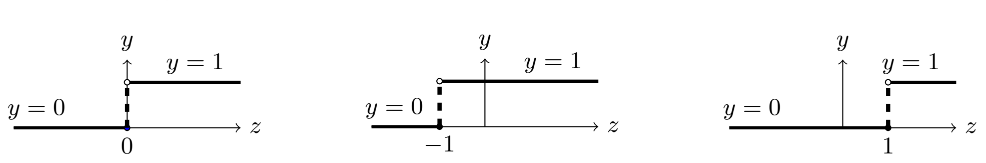
2.1.3 Activation Function and the Bias Term

To determine the output of the artificial neuron, we use an activation function. Analogous to the threshold firing of a biological neuron, the activation function, denoted ϕ, decides whether the neuron should fire based on the weighted sum of its inputs, z. In the artificial neuron, we use a simple step function as the activation function:

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$



Although, in practice we often use a different activation function, such as the sigmoid or ReLU function, the step function is a good starting point for understanding the basic concept of an activation function.
The bias term, b, is introduced to shift the activation threshold. Biologically, this is analogous to the neuron's resting membrane potential, which influences how easily the neuron will fire.



2.2 Learning

The core idea of learning is, given an unfavorable outcome, we look back and try to figure out what we could've done differently to achieve a better outcome. We can only do this if we know how wrong our decision was, and what we should have done instead.

Consider, for a moment , getting a college test result back. If we wanted to get 90% on the test and only got 40%, we know we need to give a much higher weight to studying next time. Whereas, if we got 88% we know we're on the right track, and only need to make minor adjustments.

In an artificial neuron, we can compare the difference between the correct output and the output produced by the neuron. The larger the error, the greater need for adjustment. The learning rate η controls how sensitive we are to these errors. A high learning rate means we make large adjustments, while a low learning rate means we make small adjustments.

The question then becomes, why not always use a high learning rate? Let's revisit the college test scenario. Imagine you aimed for 90% but only scored 40%. A large error signal indicates a significant adjustment is needed. With a high learning rate, you would react very strongly to this error. You might drastically increase your study time, perhaps even overcompensating and studying excessively. This could lead to burnout or neglecting other important aspects of life. Conversely, with a low learning rate, even with a 40% score, your adjustment would be small. You might only slightly increase your study time, potentially not enough to significantly improve your score on the next test. The learning rate, therefore, is a delicate balance. It needs to be high enough to enable learning in a reasonable time, but low enough to prevent wild oscillations or overreactions to errors. The learning rate, is something we can adjust to find the right balance between making large adjustments when we are far off, and making small adjustments when we are close to the correct answer.

Mathematically, we can represent the learning process as follows. We define the error as the difference between the predicted result and the actual result. This error indicates how far off our prediction was from the correct answer, and can be positive (overprediction) or negative (underprediction)

$$\text{Error} = y_{\text{actual}} - y_{\text{predicted}}$$

We then define how much we should change the weight of each input (x_i) as:

$$\Delta w_i = \eta \times \text{Error} \times x_i$$

Here, η is the learning rate, controlling our sensitivity to errors. The multiplication by $x_i$is important because it scales the weight adjustment by the magnitude of the input x_i. Inputs with larger values have a proportionally larger influence on the output, and therefore their weights should be adjusted more significantly for a given error. We also calculate the change for the bias term as: We also calculate how much we should change the bias term as:

$$\Delta b = \eta \times \text{Error}$$

The intuition behind these adjustments is that if the prediction is consistently too low, we increase weights and bias to make firing easier (like studying more). If consistently too high, we decrease them to make firing harder (like studying less). We then update the weights and bias as follows:

$$w_i = w_i + \Delta w_i$$

$$b = b + \Delta b$$

2.3 What does the Artificial Neuron Actually do? - Example: Mushroom Classification

Imagine we're in a field filled with all kinds of mushrooms. Some have dozens of little white spots, and tall stems. While others have very few spots and short terms. You know from past experience that the mushrooms with many spots are poisonous, while the ones with few spots are safe to eat.

But there's a catch. Sometimes we find mushrooms that are hard to classify - they have medium length stems, or only a few spots. Where exactly do we draw the line between poisonous and safe mushrooms? You may not know from a glance, and you certainly don't want to eat a poisonous mushroom to find out.

This is exactly the problem the artificial neuron helps solve. It learns to draw a line in the mushroom field that separates the poisonous mushrooms from the safe ones.
Lets define the two inputs to our neueron as:

$$x_1 = \text{number of spots} \quad \text{and} \quad x_2 = \text{length of the stem}$$

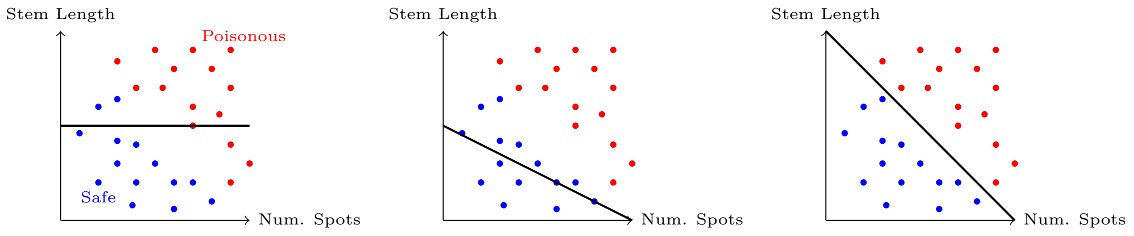
The artificial neueron then forms a linear combination of these inputs:

$$z = \sum_{i=1}^{2} w_i x_i + b = w_1 x_1 + w_2 x_2 + b$$

where our weights scale the importance of each input. The neueron then decides whether the mushroom is poisonous or not:

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

where, $y = 1$ means the mushroom is poisonous, and $y = 0$ means it is safe to eat. The perceptron learns the weights w_1 and w_2 to draw the best line that separates the two classes of mushrooms.



(a) Before training (b) After 100 training passes (epochs) (c) Final decision boundary

Figure 4: Artificial Neuron learning to separate safe and poisonous mushrooms.

We refer to this line as the decision boundary, as it separates the two classes of mushrooms. We now have a simple model, that given the number of spots and the length of the stem, can predict whether a mushroom is poisonous or safe to eat. This is the essence of what an artificial neuron does - it learns to draw decision boundaries in the input space to separate different classes of data.

2.4 Putting it all together

We can define the single artificial neuron as a linear classifier that separates two classes of data by learning the optimal weights and bias.

3 |UNCOMPLETE PAST THIS | Beyond the single-unit perceptrons

3.1 From a Neuron to a Perceptron Layer

Discuss the XOR problem and how it cannot be solved by a single layer perceptron and linear separability problems Discuss how a layer of neuerons allows us to learn more complex decision boundaries, a layer of 2 neuerons can split the input space into 2 regions, a layer of 3 neuerons can split the input space into 3 regions.

3.2 Hidden Layers

Introduce the concept of hidden layers and how they enable learning of complex, hierarchical representations Discuss feedforward computation in multilayer architectures

3.3 Multilayer Perceptrons

4 Activation Functions for Deeper Networks

4.1 Common Activations

sigmoid, tanh, ReLU, Leaky ReLU pros/cons, relus simplicity, sigmoids tendence to saturate, tanh zero centered, leaky relu to prevent dying neurons

4.2 When/where to use each

provide typical guidlanes, relu for hidden layers, tanh for output, leaky relu for deeper networks, softmax for multiclass classification

5 Forward Propogation

5.1 Matrix Vector Notation

show how each layers output is computer: $z^(l) = W^(l)a^(l-1) + b^(l)$ and $a^(l) = \phi(z^(l))$ emphasis on vectorization for efficiency emphasise how this extends to multiple layers

6 Cost Functions and Error Metrics

6.1 choice of cost function

Mean Squared Error, Cross Entropy explain the difference between the two, i.e. MSE for regression, cross entropy for classification

6.2 Error Backpropogation

how the network distributes the error back through the network deriving partial derivaties for weights and biases at each layer

7 Backpropogation

7.1 Gradient Computation

derivaties of common activation functions chain rule application from output layer back to input layer

7.2 Update rules

how to update weights and biases using the computed gradients the role of the learning rate

8 Optimization and Training

8.1 Batch Based Methods

Stochastic Gradient Descent, Mini Batch Gradient Descent Explain the difference between the two, i.e. SGD for noisy data, Mini Batch for more stable convergence

8.2 Improving convergence

Momentum, RMSProp, Adam Explain the role of each, i.e. momentum to prevent oscillations, RMSProp to adjust learning rates, Adam as a combination of the two Explanation of the hyperparameters for each optimization method

9 Regularization

9.1 Overfitting and Underfitting

Explain the concepts of overfitting and underfitting Techniques to prevent overfitting, i.e. dropout, L1/L2 regularization, early stopping

9.2 Initalization

Why naive (all zeros) initialization fails Recommended schemes for initialization, i.e. Xavier, He

10 Building the Multilayer Perceptron for MNIST

- Data input: Organize your data and labels
- Network structure: define layers, number of
- Forward Propogation: Implement the sequence of matrix multiplications, biases and activations
- Loss calculation: e.g. cross entropy for classification tasks
- Backpropogation: Compute gradients layer by layer
- Weight updates: Incorporate an optimization method (SGD, Adam, etc)
- Training loop: Shufflee data, iterate over epochs, track training and validation loss

11 Extensions and Next steps

11.1 Depper architectures

convoulition nueral networks, recurrent neural networks, transformers

11.2 Modern practices

Batch normalization, advanced regularization, learning rate schedules

References

[1] Azevedo et al. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain, 2009.

[2] Physio-pedia.com. hhttps://www.physio-pedia.com/neurone.

[3] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain, 1958.