

**Handwritten Digit Recognized**

**Academic Year: 2023-2024**

w

**Prepared by:**

Ouissal-ilham ZINOUNE

Hamza OKHADIR

Naoual KHELLOUFI

Mehdi EL AIMANI

Mustapha GHIATI

**Supervised by:**

Haut du formulaire

Bas du formulaire

Pr : BOUJIHA tarik

# Acknowledgements:

Physics Department

Master : Artificial Intelligence and Virtual Reality

We would like to express our heartfelt thanks to everyone who helped make this mini project possible. Our deepest gratitude goes to our professors for their continuous guidance and insightful feedback. We are also grateful to our colleagues and friends for their support and encouragement. Lastly, we extend our sincere thanks to our families for their patience and unwavering support throughout this journey.

# Table of contents:

[Acknowledgements: 2](#_Toc168960336)

[Table of contents: 3](#_Toc168960337)

[Table of figures: 4](#_Toc168960338)

[Introduction: 5](#_Toc168960339)

[Problem Statement: 6](#_Toc168960340)

[Context: 7](#_Toc168960341)

[Chapter 1: Neural Network Mechanics 8](#_Toc168960342)

[1. Introduction: 8](#_Toc168960343)

[2. The neuron: 9](#_Toc168960344)

[3. Forward Propagation 10](#_Toc168960345)

[3.1 Non-Vectorized Forward Propagation 10](#_Toc168960346)

[3.2 Non-Vectorized Forward Propagation 11](#_Toc168960347)

[4. Backpropagation 13](#_Toc168960348)

[4.1 Learn 13](#_Toc168960349)

[4.2 Error 13](#_Toc168960350)

[4.3 Gradient Descent 14](#_Toc168960351)

[4.5 Vectorized Backpropagation 18](#_Toc168960352)

[Chapter 2: Implementing a Simple Neural Network for Digit Recognition: 21](#_Toc168960353)

[1. Presentation of the MNIST Dataset 21](#_Toc168960354)

[1.1 Description of the dataset characteristics 21](#_Toc168960355)

[1.2 Exploration of the data and visualization of digit examples 21](#_Toc168960356)

[2. Neural Network Implementation 21](#_Toc168960357)

[2.1 Neural network architecture (layers, number of neurons, etc.) 21](#_Toc168960358)

[2.2 Cost function and optimization algorithm (gradient descent) 21](#_Toc168960359)

[2.3 Model training and performance evaluation 21](#_Toc168960360)

[3. Results and Discussion 24](#_Toc168960361)

[Conclusion 25](#_Toc168960362)

[References 27](#_Toc168960363)

# Table of figures:

[Figure 1: the Sigmoid activation function 9](#_Toc168961261)

[Figure 2: neural network 10](#_Toc168961262)

[Figure 3:network 12](#_Toc168961263)

[Figure 4:network 14](#_Toc168961264)

[Figure 5: Gradient descent 15](#_Toc168961265)

[Figure 6: the MNIST training 24](#_Toc168961266)

[Figure 7: Results 25](#_Toc168961267)

# Introduction:

Recent advances in artificial intelligence have significantly enhanced the ability of machines to understand and analyze images. This project focuses on the recognition of handwritten digits using convolutional neural networks (CNNs). The objective is to develop a model capable of effectively distinguishing between different handwritten digits.

# Problem Statement:

Automatic recognition of handwritten digits presents numerous challenges, primarily due to natural variations in individual writing styles. These variations make it difficult for machines to accurately recognize handwritten digits. A system capable of addressing this issue has wide-ranging practical applications, such as document digitization and automated validation systems.

# Context:

Recently, tremendous progress has taken place in the field of artificial intelligence, especially in the areas of machine learning and deep learning. These developments have paved the way for many exciting applications, such as handwritten digit recognition, which combines image processing with neural network architecture design and optimization. High accuracy can be achieved through the application of optimization methods.

The following report presents a solid system for identifying handwritten numbers by neural network, demonstrating such effectiveness. Deep research has taken place in neural networks, from the basic idea of how to set neurons to the complex process of forward and backward propagation. Moreover, the report encompasses all the basic issues regarding data collection and preprocessing techniques, model design and setup, problems related to training, and methods of evaluating the testing process.

All these parts will be examined in the following report in order to carry out a responsive conception of the techniques necessary for working out a handwritten digit recognizer. Detailed pedantic discussions on the methodologies and findings are presented, showing not only practical applications but also opportunities for further improvement of numeral recognition systems using deep learning.

From this detailed study, there come out very important issues, such as the great impact of deep learning on the field of image recognition, to serve as a very strong platform for further progress. The knowledge received within the frames of this project may be used to improve the real applicative efficiency and accuracy of the system for digit recognition to be developed later on in the future, which makes the following study very important and constructive in the stream of further developing artificial intelligence technologies.

# Chapter 1: Neural Network Mechanics

## Introduction:

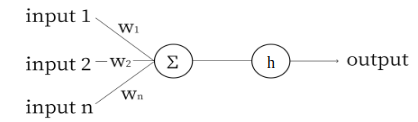
The origins of modern neural networks lie in the perceptron, an artificial neuron model introduced by Frank Rosenblatt in 1958. The perceptron is the basic building block of neural networks, providing a simple yet powerful classification algorithm. It can be modeled as a decision function fwhich takes as input a vector, or tensor x = [x1, x2, .... , xn]T, and whose output is a scalar y determining the class to which the input x belongs.

The decision function f can be broken down into two steps:

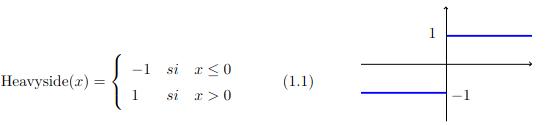
• A pre-activation that corresponds to the linear combination of x with a weight vector w=[w1,w2, .... , wn]T,such that a(x)=wTx = i=1nwixi.

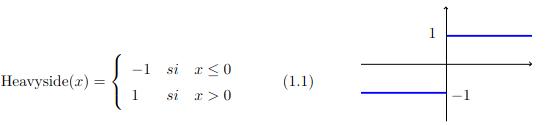
• An activation h which takes as input the output of the pre-activation adding a bias term b and whose output will correspond to  y ,such that  y=h(a(x))=h(wTx+b) .

To simplify notation, we'll refer to the parameter vector as:  =[b,w1, ....,wn]T so that we can write:  y=f(x;)=h(a(x))=h(wTx+b).



In the original form of the perceptron, h was the Heavyside function defines as below:

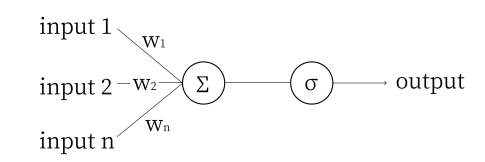




The use of the perceptron is limited to binary classification problems where one of the class will correspond to Y=0 and the other to Y=1. In addition, the perceptron can only be used on classification problems where the input variables X are linearly separable.

## The neuron:

A single node of a neural network (a neuron) differs from a perceptron in one way: the activation function. Consider this diagram of a neuron:



The symbol σ represents the Sigmoid activation function (x)= 11 + e-x

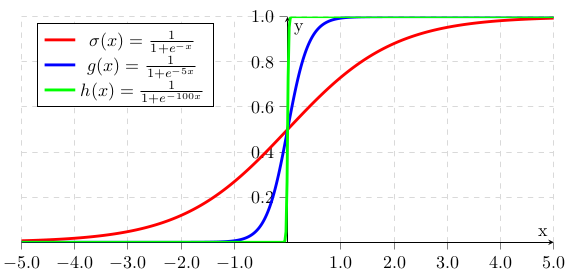


Figure 1: the Sigmoid activation function

Observe that as the coefficient of x increases toward infinity, (x) gradually approximates a step function. We use (x) (the sigmoid function) because it is differentiable, which is essential for learning in neural networks. Other common activation functions include tanh(x) and ReLU, but for our examples we will focus on the sigmoid function.

The rest of the neuron works similarly to a perceptron: each input is multiplied by its corresponding weight, the products are summed along with the bias, and then the activation function is applied to this sum.

## 3. Forward Propagation

### 3.1 Non-Vectorized Forward Propagation

Forward Propagation is an advanced term for the estimation of the output of a neural network. Before we move on to the third layer, we need to calculate the values of all the neurons in the second layer. Consider the network below:

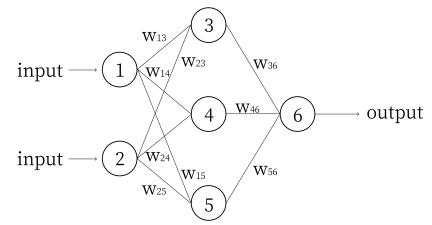


Figure 2: neural network

We denote the value of node i as i , and the bias of node i as bi . Computing the network using these variables, we get:

3=(w13n1+w232+b3)

 4=(w14n1+w242+b4)

 5=(w15n1+w252+b5)

6=(w36n3+w464+w565+b6)

Continuing this example of forward propagation, let’s assign some numbers and compute the output of this network. Let 1=0.2 and 2=0.3. Let w13=4, w14=5, w15=6, w23=5, w24=6, w25=7, w36=9, w46=10 and w56=11. Let all the biases b3..6=1 (input nodes do not have biases, the ”input nodes” are simply values given to the network). In practice, weights and biases of a network are initialized randomly between −1 and 1. Given these numbers, we compute:

3=(4\*0.2 +5\*0.1+1)=(3,3)=0.964

4=(5\*0.2 +6\*0.3+1)=(3,8)=0.978

5=(6\*0.2 +7\*0.3+1)=(4,3)=0.987

6=(9\*0.964+10\*0.978+11\*0.987+1)=(30.313)=1

This example actually illustrates one of the weak points of the Sigmoid function: it quickly approaches 1 for large numbers. The reason for using the Sigmoid function will be shown in the section on backpropagation.

### 3.2 Non-Vectorized Forward Propagation

Look again at these nodes of the network:

3=(w13n1+w232+b3)

 4=(w14n1+w242+b4)

 5=(w15n1+w252+b5)

We can rewrite this as

https://lh7-us.googleusercontent.com/docsz/AD_4nXdbyvV_YfHmKsB5H9hj4dZhPdFdn4VHBY6vWzseWzlzzroxNoYp9ms-koB-ghw_AdPrmKXSO_RAKQCLhNurxwdNfJNg-4ASQW4eGUZwJzBnHH-xO5BZaVdafFEKZmIEEexbLh22AxPUTj4cuRZg7tTOK77w?key=atALHG2SoDan_jDkn2IYaQ

Notice how the nodes in each layer of the network are in their own column vector, in the order they appear. Let’s relabel this network by layers:

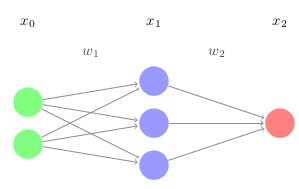
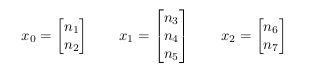
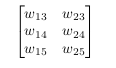


Figure 3:network

Here, x0 and x2 represent the input and output layers, and x1 is the middle layer (called a hidden layer). Mathematically speaking, these are represented as column vectors of dimension n × 1, where n is the number of nodes in the layer. Thinking back to the non-vectorized network in section 3.1



w1 and w2 are the weight matrices. Thinking back to the non-vectorized network in section 3.1, w1 corresponds to



and w2 refers to

https://lh7-us.googleusercontent.com/docsz/AD_4nXeF6ImRDBGTCmDapJemZ5JtzwTCpxxJ7JMgxt1St4pTt1fcb8qiFWnJYAx_jg5zxq0umo1rDfUNIwhmnHIky8pJLUIUxGH2Z8iXcHsQJeQFm7CbR-QOybITa0739Bg6uJoBFCg_KToaEjwb3LjRU0eyu_c?key=atALHG2SoDan_jDkn2IYaQ

Each layer (except the input) also has a bias vector, which has the same dimension as the layer itself (each node has a bias). Again thinking back to the non-vectorized network in section 3.1, we define b1 to be

https://lh7-us.googleusercontent.com/docsz/AD_4nXcSnqCg9Zp1Me7tvh2HO96OL4pvIMdCSKK5UOkDurpwyEz0obiSJyw6sxul_e-tNGSwDasYy60ifbvVH0wEFIF3ngfUrqwVX18ZLo7zMRpc9Y4VRaDKsrE7-CBMskm2RyP4Jz-w6Z168IztgSwiBhEt360?key=atALHG2SoDan_jDkn2IYaQ

and b2 to be

https://lh7-us.googleusercontent.com/docsz/AD_4nXc50ivdZnYqonUChzb7QPYzZygKiCymPNDUG3BUQjG5uTAzKLo4cc5HFLlKRsWNkz00ElqI7HzO0uMjYLVPLY9K-A8OkZXMiFaxIs1x1uES-XRrAaDzDqhdKExWdAGOPRkhGKcnZBj_K1iU8U6odmk8xxWP?key=atALHG2SoDan_jDkn2IYaQ

We can now re-write the forward propagation formula in a far more compact form. In any nlayer network, for a given layer xi+1 (assuming0i<n-1):

xi+1=(wixi+bi+1)

## Backpropagation

Backpropagation is an algorithm used in artificial intelligence to fine-tune mathematical weight functions and improve the accuracy of anartificial neural network’s outputs.It calculates the mathematical gradient of a loss function with respect to the other weights in the neural network. The calculations are then used to give artificial network nodes with high error rates less weight than nodes with lower error rates.

An important goal of backpropagation is to give data scientists insight into how changing a weight function will change loss functions and the overall behavior of the neural network. The term is sometimes used as a synonym for "error correction.

### 4.1 Learn

A neural network learns by being provided with training data and corresponding labels. The data (inputs) can take various forms, such as text, images, or numbers. The label represents the ground truth, which is the correct answer for the given input. By training on a sufficient number of data-label pairs, the network can learn to generalize the relationship between the input data and the labels. After the training phase, the network is tested or validated on a separate set of data that it has never encountered before (i.e., data not included in the training set). The validation accuracy indicates how well the network has learned to generalize from its training. Backpropagation is the technique used to update the weights and biases of the network to minimize the error during training.

### 4.2 Error

Consider the following network:

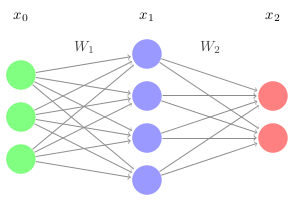


Figure 4:network

For the input x0 , let y represent the target vector, or the ground truth. We define the error as

E=12x2-y2

Essentially, this is the magnitude of the difference between the target and the network’s output. In order for a network to become more accurate, we want to minimize this error.

Let’s think of E as a function. Only x2 can vary, and we can only control this by changing the weight matrices (and the bias). Thus, for a neuron with n weights and a bias, the error can be graphed as an n+2 dimensional function (y=f(x) has 1 input, so it is graphed in two dimensions). For this network, each of the weights (3\*4+4\*2=20) and the biases (6) determine the error, so the error has many, many dimensions. If we get to the minimum of this function, we have minimized the error and trained the network.

### 4.3 Gradient Descent

Let's pretend we are working with a three dimensional function. So we can get to the minimum we will  use gradient descent.

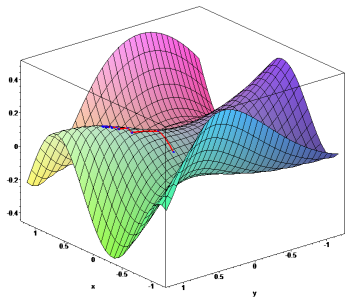
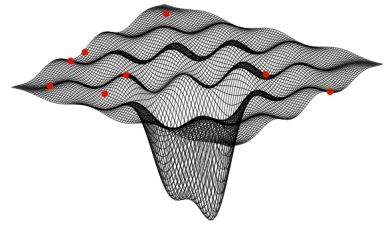


Figure 5: Gradient descent

Gradient descent is simple: Starting at some point, we move in the direction of steepest decline for a certain length. Then, at our new point, we again compute the direction of steepest decline, and move in that direction for a certain length. We repeat this process over and over until every single direction is an incline, at which point we are at the minimum.

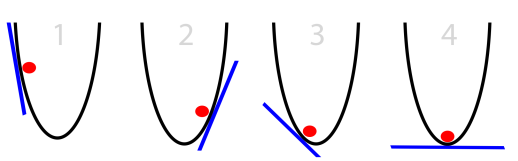
This has three issues. First, how do we know how long our steps are? Take a step too long, and we could overshoot the minimum. Take a step too short and it will take us many steps to reach the minimum. The step length is actually just a constant set by the programmer, and normally ranges from 0.1 to 0.0001. Adjusting the constant to get the best result is an important practical topic for getting the best result.

Secondly, what if there are multiple minimums, and we just happen to land in a local minimum, like the many in the function below?



Getting out of local minima to reach the global minimum is another important machine learning topic.For the purposes of explaining gradient descent, we’ll just pretend we’re working with an error function with  one minimum.

The third and final issue is: how do we know which direction is the steepest? We can’t just sample each direction, as there are infinite possibilities. Instead, we mathematically compute the best direction. Let’s consider a simple two-dimensional parabola:



From elementary calculus, we know that:

f(x)=x2

f'(x)=2x

The derivative gives us the instantaneous rate of change for any x. If we have a function in terms of x and y, we can take the derivative of f(x,y) with respect to x to find the rate of change in the x direction, and the derivative with respect to y to find the rate of change in the y direction. These are called partial derivatives. We treat the other variables like we would any other constant.”&éaq

Let’s do an example. Given f(x,y)= 2x2+3xy+y3 , the partial derivatives are:

fx=4x+3y

fx=3x+3y2

The gradient of f(x,y), or ∇f(x,y)  is just the vector:

(fx,fy)

For our example, the gradient is:

(4x+3y,3x+3y2)

This is the direction of steepest ascent. How do we know that? First, let's consider the directional derivative. ∇u f(x0,y0) is the rate of change of f(x,y) at the point (x0,y0) in the direction     . It is also defined in terms of the gradient as:https://lh7-us.googleusercontent.com/docsz/AD_4nXf12ugEmzzt0pBHnIzn37CepEVkAC89YJ2OEtxDXVZzsi2ZLlec72nuNUsOhIJmjJbiDntL87sz2pw5ngYGtpmSyb0gFUmwCufMmg-TIzbpvIU-IsySbBG9kCLXZ_LEd96sdUDN-TjJa6Pt5p2aQ8amplQP?key=atALHG2SoDan_jDkn2IYaQhttps://lh7-us.googleusercontent.com/docsz/AD_4nXf12ugEmzzt0pBHnIzn37CepEVkAC89YJ2OEtxDXVZzsi2ZLlec72nuNUsOhIJmjJbiDntL87sz2pw5ngYGtpmSyb0gFUmwCufMmg-TIzbpvIU-IsySbBG9kCLXZ_LEd96sdUDN-TjJa6Pt5p2aQ8amplQP?key=atALHG2SoDan_jDkn2IYaQ

∇u f(x0,y0)=∇f(x,y). .

We know from our standard dot product rule:

https://lh7-us.googleusercontent.com/docsz/AD_4nXcGQcNY1cHD6lWKsmiSOE6CFTMj9c2XYTRaL1UJ1ctjqn7SpDxh7-yV6pYoetw9ffwZfZFduBX5kAYZ2hR_68Iaf7b-xfILMliTLMSsEfGtHpAKP85z_tsYp2w09BOL05m0n1OORf0B9SAk6fg89DxEyZ4?key=atALHG2SoDan_jDkn2IYaQ

And cos() is maximized at =0. Thus, when two vectors are in the same direction, their dot product is maximized. From this information, the maximum of the directional derivative must be when ∇f(x,y) and      are in the same direction. This means that the direction of the steepest ascent (maximum rate of change) is the direction of the gradient. Great! Now our third issue has been solved. In order to find the minimum of a multi-dimensional function, we just need to compute the gradient, move in that direction for a certain length, and repeat until the gradient is 0. The only problem is.... how do we compute the gradient? Our function ishttps://lh7-us.googleusercontent.com/docsz/AD_4nXcKl1o9lcy9vLE5Ylhcu7Q_5QqusrQwKmuSx0czkmIJfkwKb4PLTIUmbJN4UEgbV8QL79Bk-SELkryNL_2PHA1cOx3ili3DFSs9aWPCvmEDfB5NHM64pq4v38ydEzjrlS4b-O8NFUZ5eUFZivA8kVFxgNM?key=atALHG2SoDan_jDkn2IYaQ

E(W,b)=12o - t 2

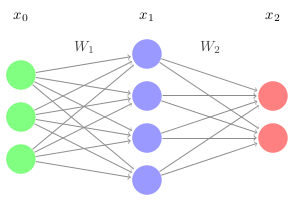
Where o is the network output at t is the target. Since the error is in terms of the weights and biases, that means that we need to compute:

(EW1,EW2,.....,Ebn)

This is why backpropagation is a fundamental concept in machine learning. It allows us to compute this gradient in a computationally efficient manner.

### 4.5 Vectorized Backpropagation

Consider the previous network:



Ignoring biases (which we will see follow a relatively simple rule), we know from forward propagation that:

x1=(W1x0)

x2=(W2x1)

And the error is, assuming some 2 × 1 target vector y:

E=12x2-y 2

Let’s first take the partial derivative of E with respect to W2 . This is just like taking a normal derivative (using the chain rule).

EW2=(x2-y)((W2x1))W2

EW2=[(x2-y)'(W2x1)](W2x1)W2

Here, is the Hadamard product, or element wise multiplication (we are working with vectors).To be simple , lets define:

2=(x2-y)'(W2x1)

Then, we can rewrite the partial as

EW2=2(W2x1)W2=2x1T

Note that x1T means that the x1 vector has been transposed (i.e. it is a row vector). This is essential for the dimensions to work out, which we can check now.

Since the whole point is to update the weights by some factor every time we backpropagate in the direction of fastest descent to minimize the error, we want to subtract the partial matrix (since it is in the direction of fastest ascent):

Wi=Wi-EW2

where alpha is the learning rate. This requires EW to be the same dimensions as Wi . Using W2 as an example, we know that

x2=(W2x1)

where x2 is a 2×1 vector, x1is a 4×1 vector, so W2 is a 2×4 matrix. Thus, both EWi and 2x1T are also 2 × 4 matrices. Since 2=(y-(W2x1))'(W2x1) and we know y is a 2 × 1 matrix, 2 has dimensions 2 × 1. If 2 is 2 × 1, then it must be multiplied by a 1 × 4 vector to create a 2 × 4 matrix. Since x1 is 4 × 1, it must be transposed to become 1 × 4.

Let’s continue to the next weight matrix.

EW1=(x2-y)((W2x1))W1

EW1=[(x2-y)'(W2x1)](W2x1)W1

EW1=2(W2x1)W1=W2T2x1W1

Substituting in for x1 , we get:

EW1=W2T2((W1x0))W1

EW1=[W2T2'(W1x0)](W2x0)W1

Again, we simplify this:

1=W2T2'(W1x0)

and we finish with

EW1=1(W1x0)W1

EW1=1x2T

We can generalize this for any layer. The only difference is the delta for the last layer:

L=(xL-y)'(WLxL-1)

The delta for every other layer is:

i=Wi+1Ti+1'(Wixi-1)

And the gradient for every weight matrix are calculated and the weight matrices are updated as follows:

EWi=ixi-1T

Wi=Wi-EWi

For biases, the rule is simpler:

bi=bi-i

That is the essence of backpropagation. Note that these formulas work for any activation function. The reason sigmoid is used to teach is because its derivative is fairly straightforward:

'(x)=(x)(1-(x))

Chapter 2: Implementing a Simple Neural Network for Digit Recognition:

## **Presentation of the MNIST Dataset**

### 1.1 Description of the dataset characteristics

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used benchmark for handwritten digit recognition. It consists of 70,000 grayscale images of handwritten digits (0-9), divided into 60,000 training examples and 10,000 test examples. Each image is 28x28 pixels in size, representing a digit centered and normalized to fit in a fixed-size grid.

### 1.2 Exploration of the data and visualization of digit examples

To better understand the dataset, we can explore the data and visualize some of the digit examples. This can help us gain insights into the variability and complexity of the handwritten digits, which will inform the design of our neural network model.

## **Neural Network Implementation**

### 2.1 Neural network architecture (layers, number of neurons, etc.)

The neural network architecture for this task consists of an input layer with 784 neurons (representing the 28x28 pixel image), a hidden layer with 128 neurons, and an output layer with 10 neurons (corresponding to the 10 digit classes). The hidden layer uses a ReLU activation function, and the output layer uses a softmax activation to produce probability distributions over the digit classes.

### 2.2 Cost function and optimization algorithm (gradient descent)

The cost function used is the mean squared error between the predicted outputs and the true labels. To minimize this cost function, we employ the gradient descent optimization algorithm, computing the gradients of the cost with respect to the model parameters and updating them iteratively.

## 2.3 Model training and performance evaluation

The provided image contains Python code that is part of a handwritten digit recognition project. The code starts by importing the necessary libraries, including NumPy, Pandas, and Matplotlib. It then loads the training data from a CSV file located at the path '/kaggle/input/digit-recognizer/train.csv'.

After loading the data, the code performs several preprocessing steps. It first converts the data into NumPy arrays using the np.array() function. It then extracts the shape of the data, which is stored in the variables m and n. The code then shuffles the data before splitting it into development and training sets, using the np.random.shuffle() function.

The development and training sets are stored in separate variables, with the development set data\_dev and its corresponding labels Y\_dev, and the training set data\_train and its labels Y\_train. The input features for the development and training sets (X\_dev and X\_train) are then normalized by dividing them by 255, to scale the pixel values between 0 and 1.

Finally, the code calculates the shape of the training set input features and stores it in the variable m\_train. This prepared data is likely to be used in the subsequent steps of the handwritten digit recognition project, such as model building and training.



The neural network model is trained on the MNIST training set for 10 epochs, using a learning rate of 0.01. After training, the model's performance is evaluated on the held-out MNIST test set, and the classification accuracy is reported.

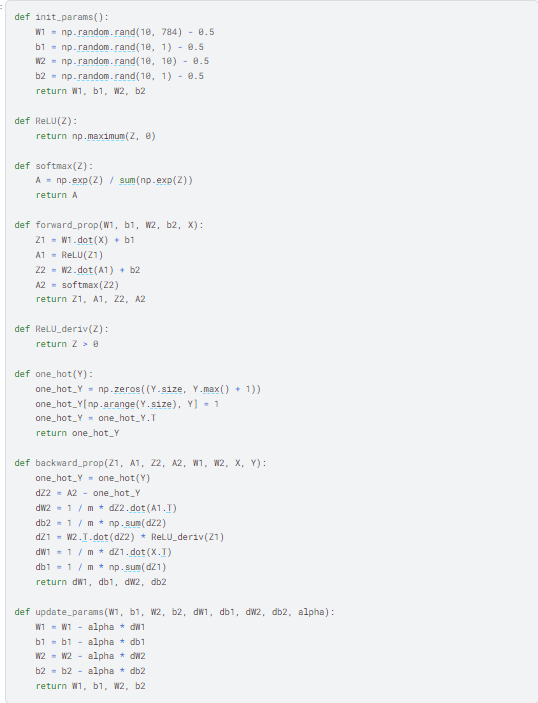


Figure 6: the MNIST training

## **Results and Discussion**

### **3.1 Model performance on the test set**

The results obtained show that the model achieves an accuracy of 62.6% on the MNIST test set. While this performance is lower than that of more complex deep learning models, it demonstrates the ability of a simple neural network to tackle the digit recognition problem.

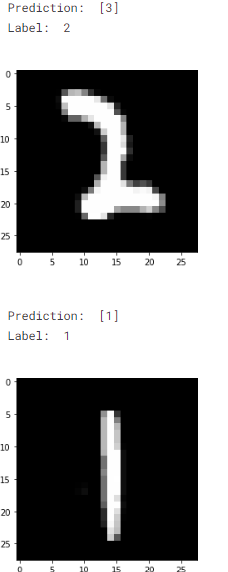


Figure 7: Results

Conclusion:

This project demonstrates the effectiveness of convolutional neural networks in handwritten digit recognition. The results obtained show high accuracy, proving that CNNs are well-suited for this type of task.

References:

https://www.kaggle.com/code/wwsalmon/simple-mnist-nn-from-scratch-numpy-no-tf-keras

https://user.tjhsst.edu/2018nsardana/nn2.pdf