# Neural Networks

Oliver Temple

January 4, 2022

**Abstract**

Now commonplace in todays world, neural networks are becoming increasingly popular, but what actually are they, and how do they work? This paper will dive into the basic concepts behind neural networks, the history of them, how they are used in the real world and how the most simple neural networks are made.

# Contents

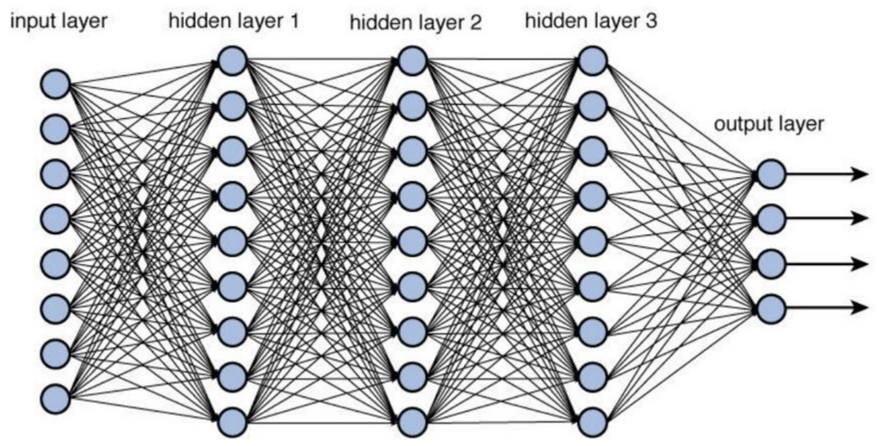


Figure 1: Neural Network

# 1 Introduction

## 1.1 What is a neural network?

A neural network is a computational model that can be used to perform a wide range of tasks, such as pattern recognition, image recognition, speech recognition, and computer vision. Neural networks are constructed from a set of interconnected neurons, which are capable of processing and responding to inputs and producing outputs. A representation of a neural network is shown in Figure 1.

## 1.2 What are neural networks used for?

In today's society, neural networks are used constantly to classify and predict a multitude of things. A common application is in social media, where complex models are used to predict which content the user will like, with the goal of increasing the user's time spent on the application.

More recently, neural networks have been used to create hypothesis that scientists had not thought of. A team of scientists that are researching new battery technologies used a neural network to find material combinations that are more likely to work. This has saved the team a lot of time and money, as the number of material combinations to test has been reduced.

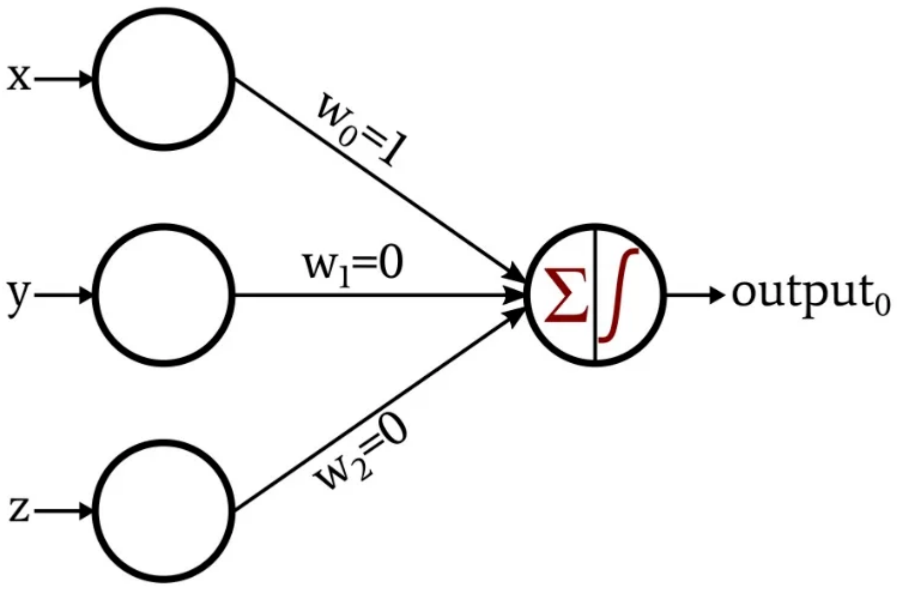


Figure 2: Perceptron

## 1.3 What are the different types of neural networks?

There are many different types of neural networks, and each types has its own properties that make them more specialized and useful. The most common types of neural networks are:

- Perceptron
- Feed Forward Neural Network
- Multi-Layer Perceptron
- Convolutional Neural Network
- Recurrent Neural Network
- Long Short-Term Memory

### 1.3.1 Perceptron

A perceptron model, shown in Figure 2, is one of the simplest and oldest neural networks. It is the smallest unit of a neural network. It accepts weighted inputs, to which the activation function is applied to produce an output.

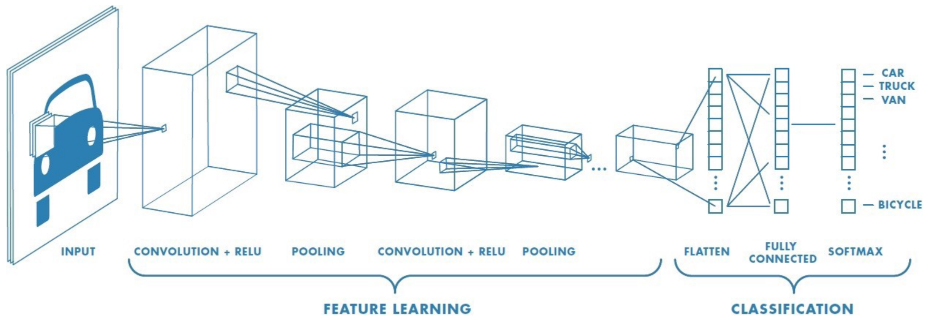


Figure 3: Convolutional Neural Network

### 1.3.2 Feed Forward Neural Network

A feed forward neural network is a neural network that is constructed from a set of interconnected layers. The layers are connected to each other, and the output of each layer is used as the input to the next layer. The data travels in only one direction, making them useful only for linear classification.

### 1.3.3 Multi-Layer Perceptron

A multi-layer perceptron is a neural network that consists of interconnected layers of perceptrons. It has bi-directional propagation, meaning that the data can travel forwards to predict and backwards to adjust. These are used for deep learning, where the network can learn to classify complex data. Figure 1

### 1.3.4 Convolutional Neural Network

A convolutional neural network(shown in Figure 3) contains a three dimensional arrangement of neurons. Each neuron on the first layer only processes information from a small part of the field of view.

### 1.3.5 Recurrent Neural Network

In a recurrent neural network, the output of a layer is saved and fed back into the input to help with prediction. This is used for time series prediction, where the network can learn to predict the future. For example, given the price history of a stock, the network could attempt to predict the future price.

### 1.3.6 Long Short-Term Memory

A Long Short-Term Memory (LSTM) neural network is a recurrent neural network that uses a long term memory to predict the future. LSTM neurons include a memory cell that can maintain information for long periods of time. A set of gates are used to control when information enters the memory, when it's output, and when it's forgotten.

### 1.4 Motivation

Artificial Intelligence is a field of study that is developing rapidly. The vast uses for AI make it extremely diverse and a very useful topic to understand. As neural networks are one of the most common forms of AI, I believe that this project will give me an insight into how they work, as well as a better understanding of their applications.

### 1.5 Project Goal

The goal of this project is to understand how neural networks work at the most basic level. This will allow me to build my knowledge in later projects to understand more complex models. I would also like to create a neural network from scratch to recognize handwritten digits from 0-9 using the MNIST dataset and the knowledge that I gained from this project.

## 2 The History of Neural Networks

Research into neural networks first began in the 1940s, when scientists were trying to explain how neurons in the brain might work. In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on how neurons might work, modeling a simple neural network using electrical circuits.

In 1949, Donald Hebb wrote "The Organization of Behavior", which discussed how neural pathways are strengthened, a concept that is essential to how humans learn, and is often summarized as "Cells that fire together, wire together".

In the 1950s, which improved computation power, Nathanial Rochester from the IBM research laboratories attempted to simulate a hypothetical neural network, however he failed to do so.

In 1959, Bernard Widrow and Marcian Hoff from Stanford developed the first neural network that was applied to a real world problem. They developed two models called "ADALINE" and "MADALINE" - Multiple ADAptive LINear Elements. ADALINE was developed to recognize binary patterns, sot that if it was reading streaming bits from a phone line, it could predict the next bit. MADALINE was the first neural network to be used to solve a real world problem. It used an adaptive filter that eliminated echos on phone lines and is still in commercial use.

In 1962, Widrow and Hoff developed a learning procedure that was based on the idea that one active perceptron might have a large error, one can adjust the weight values to distribute the error across the network. Applying this rule does not eliminate the error if the line before the weight is 0, however this will eventually correct itself. If the error is distributed across of the weights, the error will be eliminated. The procedure that was developed can be written as:

$$Weight\ Change = Pre - Weight\ line\ value \times \frac{Error}{Number\ of\ Inputs} \tag{1}$$

Despite the success that neural networks achieve later, traditional von Neumann architecture took over the computing scene, and neural network research was left behind, even though von Neumann himself suggested the imitation of neural functions by using telegraph relays or vacuum tubes.

At the same time, a paper was published that suggested that neural networks could not be extended to more that one layer. Futhermore, many people in the field were using a learning function that was fundamentally flawed, as it was not differentiable across the entire network. As a result, research and funding decreased dramatically.

Adding to this, the early success of some neural networks led to an exaggeration of the potential of neural networks, leading to promises that could not be fulfilled. Furthermore, philosophical questions led to fear, with writers pondering the effect that the so-called ”thinking machines” would have on humanity, ideas which are still around today.

In 1982, John Hopfield of Caltech published a paper theorising that connections between neurons could be two way, instead of the one way that they had been until this point. This paper renewed interest int he field. This was coupled with Reilly and Cooper using a ”Hybrid Network” in the same year, with multiple layers using different problem solving strategies.

Also in 1982, at a US-Japan conference, Japan announced a new effort on neural networks, and US papers generated worry that the US could be left behind. This led to more funding and more research in the field.

In 1986, with more research into multi-layer neural networks, the problem was how to extend the Widrow-Hoff rule (equation 1) to multiple layers. Three independent groups of researchers came up with similar ideas, which are now called back propagation networks, which used many layers and are slow learners, needing possibly thousands of iterations lo learn.

In modern technology, neural networks are used in many applications. The fundamental idea behind the nature of neural networks is that if it works in nature, it must be able to work on computers.

# 3 Bias in Artificial Intelligence

Despite the many benefits of neural networks, there are still many problems that are not solved. The most common problem is that bias in the training data can cause bias in the predictions of the network, however societal bias is also an issue, where our assumptions and norms as a society cause us to have lind spots in out thinking. These biases usually reflect widespread societal biases about race, gender, biological sex, age and culture.

## 3.1 What causes bias in AI?

### 3.1.1 Data Bias

As stated above, bias in AI can be caused by a bias in the training data. For example, an AI model that is being trained to detect faces, could be trained with more photos of a specific gender or race, making it better at detecting that group of people.

### 3.1.2 Societal Bias

Many people think that computers are impartial, and while this is true for your mirowave or your vaccum, an computer that can "think" may not be impartial. Take humans for example, upringing, experiences and culture are just some of the things that shape people, and they internalize certain assumptions about the world around them. AI is much the same. Algorithms built by humans can show the bias of the people who built them, AIs tend to "think" they way they have been taught. The algorithms and data that outputs the biased results often appear unbiased, but their outputs show that the bias is present.

## 3.2 Examples of Bias in AI

### 3.2.1 PortraitAI Art Generator

The portrait AI art generator, `https://ai-art.tokyo/en/`, takes an image, and renders an image of you in the manner of Baroque and Renaissance art. The results are great for white people, however, due to most of the paintings from this era being of white Europeans, the training data would have been biased towards white people.

The company who owns the service has said:

> *Currently, the AI portrait generator has been trained mostly on portraits of people of European ethnicity. We're planning to expand our dataset and fix this in the future. At the time of conceptualizing this AI, authors were not certain it would turn out to work at all. This generator is close to the state-of-the-art in AI at the moment. Sorry for the bias in the meanwhile. Have fun!*

### 3.2.2 Twitter Photo Cropping

Despite the size of a company like Twitter, AI bias is still an issue. Twitter recently apologized after uses claimed that its image-cropping algorithm was racist. The goal of the algorithm is to crop the image, centering on a human face. However, it only works if you're white (and more so if you're male). If your image contained a white persona and a black person, the white person would be centered in the preview. Even animal and cartoon faces received preferential treatment over black faces. In response, a Twitter spokesperson said:

> *Our team did test for bias before shipping the model and did not find evidence of racial or gender bias in our testing. But it's clear from these examples that we've got more analysis to do. We'll continue to share what we learn, what actions we take, and will open source our analysis so others can review and replicate.*

## 4 How Neural Networks Work

At its core, a neural network is a collection of neuron, formed into layers, that are all interconnected. Each neuron has its own bias, and each connection has a weight. These parameters are what determine the output of the network. As stated above, for my project I will be using a multilayer perceptron (Figure 4) as it allows for back propagation to be used, while not becoming to complex, like a convolutional neural network.

The basic steps of training a neural network are:

Step 1: Initialize the weights and bias with small random values.

Step 2: Propagate all the values in the input layer, through all of the hidden layers, until the output layer (Forward Propagation).

Step 3: Update weights and bias in the inner layers using the loss function (Back Propagation).

Step 4: Repeat Steps 2 and 3 until the network accuracy is satisfactory.

### 4.1 Forward Propagation

The purpose of forward propagation is to get an output from our neural network. To calculate it we take the weighted sum of the inputs, and add the bias. The output of the $n^{th}$ layer can be calculated using the formula in the equation 2.

$$output_n = \sum_{i=1}^{j}(Weight_{ni} * \sigma(output_{(n-1)i})) + bias_n \tag{2}$$

where $n$ is the layer, $i$ is the neuron, $j$ is the number of neurons in the layer and $\sigma$ is the activation function.

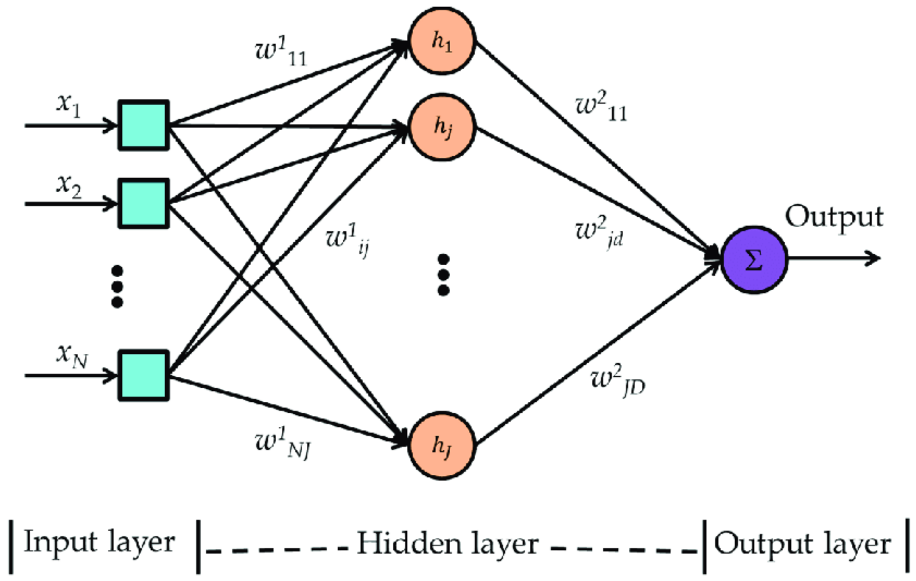


Figure 4: Multilayer Perceptron

## 4.2 Loss Function

The loss function is used to calculate the error of the network. To calculate it we take the difference between the desired output and the output of the network. There are many different loss functions to choose from, some of the more common ones are:

- Mean Squared Error (MSE)
- Cross Entropy Loss (or Log Loss)

### 4.2.1 Mean Squared Error

The MSE loss function is calculate as the average of the squared difference between the network's output and the desired output. It is used when there is only one output for the network. The equation is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(output_i - desired\ output_i)^2 \tag{3}$$

where $n$ is the number of samples we are testing against.

### 4.2.2 Cross Entropy Loss

Cross entropy loss, often called log loss, is a loss function that is used for more complex models. Each predicted output is compared to the desired output and a score is calculated that penalizes the output based on the difference between the two. The penalty is logarithmic, meaning smaller score for smaller differences, and larger score for larger differences. The equation is:

$$CEL = -\sum_{i=1}^{n}(desired\ output_i \times \log(output_i)) \tag{4}$$

where $n$ is the number of samples we are testing against.

## 4.3 Activation Functions

The purpose of an activation function is add some non-linearity to the network, to prevent it from becoming an overcomplicated linear regression. It is applied to the input of each neuron, before the output is calculated. There are many different activation functions to choose from, some of the more common ones are:

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU

### 4.3.1 Sigmoid

The sigmoid function(Figure 4.3.1) takes any input, $x$ and translates it to a value between 0 and 1. The equation is:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{5}$$

### 4.3.2 Tanh

Similar to the sigmoid function, the tanh function(Figure 4.3.2) takes any input, $x$ and translates it to a value between 0 and 1. The equation is:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6}$$

### 4.3.3 ReLU

Unlike the sigmoid function and the tanh function, the ReLU function(Figure 4.3.3) takes any input, $x$ and if it is negative, it returns 0, otherwise it returns the input. The equation is:

$$ReLU(x) = \max(0, x) \tag{7}$$

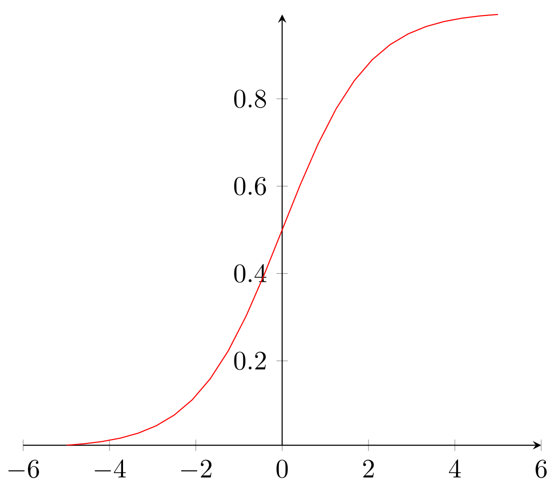


Figure 5: Sigmoid Function

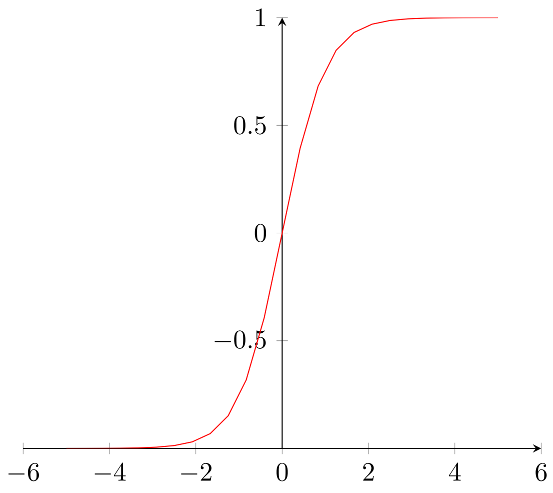


Figure 6: Tanh Function

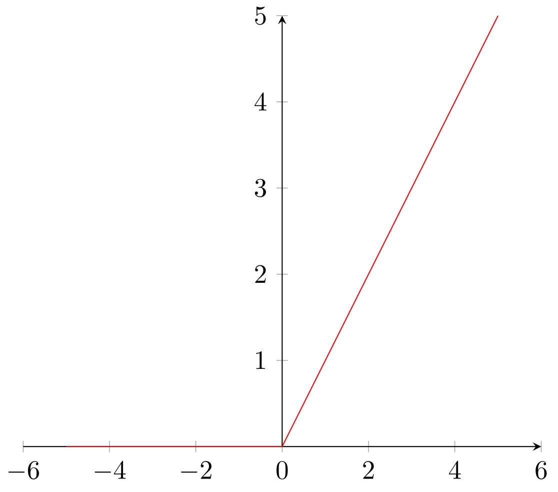


Figure 7: ReLU Function

#### 4.3.4 Leaky ReLU

The Leaky ReLU function(Figure 4.3.4) takes any input, $x$ and if it is negative, it returns a scaled down input, otherwise it returns the input. The equation is:

$$Leaky\ ReLU(x) = \max(0.1x, x) \tag{8}$$

## 4.4 Backpropagation

Backpropagation is the process where one propagates backwards through the network, adjusting the weights and biases to minimize the loss function. To do so, one must first calculate the derivative of the loss function, as this will allow us to know weather to increase or decrease the weights and biases, and by how much.

Imagine we plot a graph of loss against weights, as shown in Figure 4.4, to find the minimum of the loss with respect to weight, we must find the derivative of the loss function. We can calculate this using the chain rule. For example, to calculate the derivative of the cross entropy loss function (equation 4) with respect to the weights, we can use the chain rule:

**TODO: Finnish back propagation**

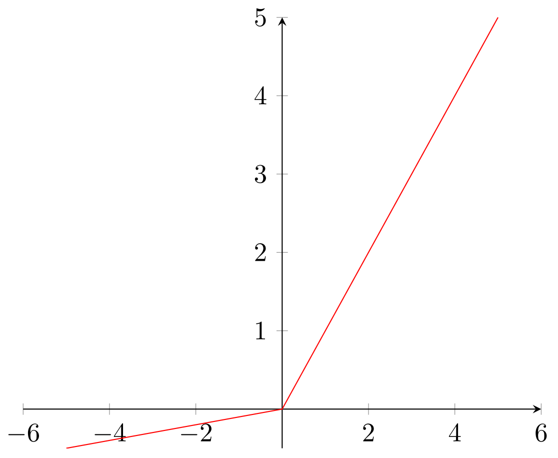


Figure 8: Leaky ReLU Function

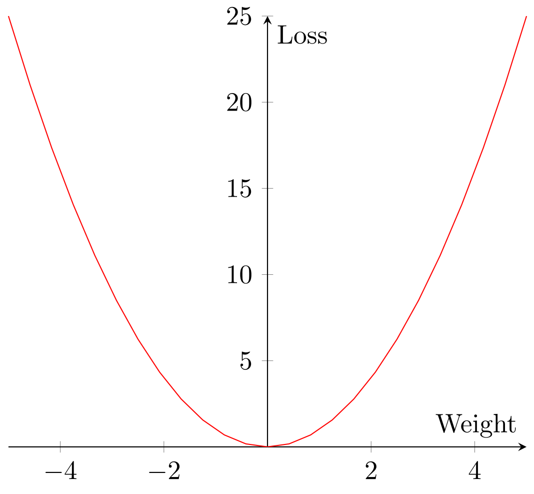


Figure 9: Weights against Loss

# 5 Coding a Neural Network

## 5.1 Data

When training a neural network, one of the most important assets is the data that will be used to train and test the neural network. For my project, I will be using the MNIST dataset, which is a set of labelled hand written digits from 0-9. The data is stored in a CSV file, with the images represented as a list of 784 pixels (28x28 pixel images) with values between 0 and 255 to represent the the brightness of the pixel.

## 5.2 Splitting the Data

When creating a neural network it is important to split out data set into a ”train” set and a ”test” set. This is so that we can test our neural network on data that it hasn’t seen before, which will give us an idea of the accuracy of the model.

## 5.3 Training the Model

To train the model, you must iterate through the training data set, with each iteration called an epoch. Each epoch will be a single pass through the entire training data set.

## 5.4 Testing the Model

## 5.5 Discussion of Results

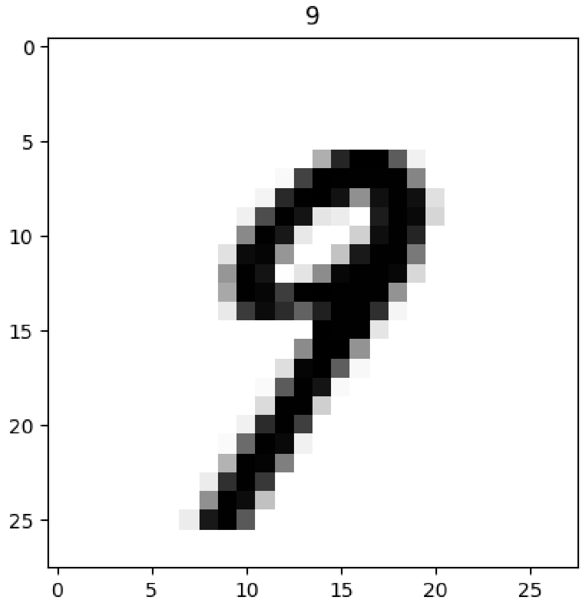
# 6 Conclusion

# 7 Appendix

## 7.1 MNIST Dataset

An example of a training sample from the dataset is shown below:

label: 9



```
pixel patters (input to network): [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.3137254901960784, 0.8549019607843137, 0.996078431372549,
0.996078431372549, 0.6392156862745098, 0.054901960784313725, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.023529411764705882, 0.7372549019607844, 0.996078431372549, 0.9921568627450981,
0.9921568627450981, 0.9921568627450981, 0.9921568627450981, 0.47843137254901963,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.043137254901960784, 0.8352941176470589, 0.9921568627450981, 0.996078431372549,
0.9254901960784314, 0.43529411764705883, 0.9333333333333333, 0.9921568627450981,
0.9333333333333333, 0.12156862745098039, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.058823529411764705, 0.6980392156862745,
0.9921568627450981, 0.9215686274509803, 0.10196078431372549, 0.07450980392156863,
0.0, 0.8901960784313725, 0.9921568627450981, 0.9607843137254902, 0.1607843137254902,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.4627450980392157, 0.9921568627450981, 0.9019607843137255, 0.09019607843137255,
0.0, 0.0, 0.1843137254901961, 0.9450980392156862, 0.9921568627450981, 0.8549019607843137,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.11764705882352941, 0.9568627450980393, 0.9803921568627451, 0.4117647058823529,
0.0, 0.0, 0.23137254901960785, 0.8980392156862745, 0.9921568627450981, 0.9921568627450981,
0.5215686274509804, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.4117647058823529, 0.9921568627450981, 0.9215686274509803,
0.00392156862745098, 0.10588235294117647, 0.45098039215686275, 0.9725490196078431,
0.9921568627450981, 0.9921568627450981, 0.9725490196078431, 0.15294117647058825,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.3411764705882353, 0.9921568627450981, 0.9764705882352941, 0.7607843137254902,
0.9921568627450981, 0.996078431372549, 0.9921568627450981, 0.9921568627450981,
0.9921568627450981, 0.4196078431372549, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.08627450980392157, 0.7686274509803922,
0.9568627450980393, 0.8274509803921568, 0.611764705882353, 0.8745098039215686,
0.9921568627450981, 0.9921568627450981, 0.8156862745098039, 0.0392156862745098,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.996078431372549, 0.9921568627450981, 0.9921568627450981,
0.10196078431372549, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.4549019607843137, 1.0,
0.996078431372549, 0.4235294117647059, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.12941176470588237,
0.9803921568627451, 0.996078431372549, 0.6313725490196078, 0.027450980392156862,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0196078431372549, 0.6431372549019608, 0.9921568627450981,
0.9215686274509803, 0.023529411764705882, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1411764705882353,
0.9921568627450981, 0.9921568627450981, 0.18823529411764706, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.054901960784313725, 0.8313725490196079, 0.9921568627450981,
0.7529411764705882, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0196078431372549, 0.5803921568627451,
0.9921568627450981, 0.9607843137254902, 0.054901960784313725, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.29411764705882354, 0.9921568627450981, 0.984313725490196,
0.5019607843137255, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.07450980392156863, 0.8117647058823529,
0.9921568627450981, 0.7803921568627451, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.43529411764705883,
0.9921568627450981, 0.9529411764705882, 0.24313725490196078, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.07450980392156863, 0.9019607843137255, 0.9921568627450981,
0.6509803921568628, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

## References

[1] **History of Neural Networks** `https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec` `https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html`

[2] **Bias in Artificial Intelligence** `https://www.lexalytics.com/lexablog/bias-in-ai-machine-learning`

[3] **Loss Functions** `https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e1`

[4] **AI generated hypothesis** `https://www.scientificamerican.com/article/ai-generates-hypotheses-human-scientists-have-not-thought-of/`