Mathematical analysis of artificial neural networks

Ioannis Theocharides (957865)

September 4, 2020

Supervisor: Dr Pawel Dlotko

Dept.Maths
Swansea University

Contents

The paper will start with an introduction to neural networks and explain the approach taken to cover the topic. We will then discuss neural networks from the biological perspective and consider how this relates to a mathematical view. Continuing, we will start explaining the mathematical aspects to the neural networks which is the key aspect of this project and will lead into a practical example using the MNIST dataset and a brief conclusion.

1 Preface

This project will cover the topic of neural networks starting from the core concepts behind a neural network and moving onto explaining their applications in real life applications like the MNIST database (a database of handwritten grey scale numerical digits of size 28 by 28 pixels) with a heavy focus on the mathematical aspects. This project is suitable for anyone wanting an introduction to neural networks and reinforcement learning in the ever growing world of data science. The majority of this project will take influence from Simon Haykin's book *Second Edition, Neural Networks A Comprehensive Foundation* (1). Most topics will be taken from this book and if not will be otherwise stated.

2 Intoduction

The idea for artificial neural networks first started by looking at how the human mind processes information. Standard computers process information in a much more straightforward manner and can't really interpret what they are processing. Humans, on the other hand, have a much greater ability to take information from the environment that they inhabit and are able to process and interpret it. One of the main factors behind this is that humans have a nervous system. The nervous system can be broken down into 3 main parts: receptors, neural network and effectors.

Receptors are responsible for taking information about what is happening inside the body or from the local environment that the body is in. They then pass the information to the neural network which is located inside the brain. The neural network interprets the information presented to it then sends the interpreted information to the effectors. The effectors take the appropriate information and provide a response which they deem appropriate given the information. After the response has been given, the effectors take information on the outcome of the response and feed information back to the neural network which takes in the information to see if it is a desirable outcome. The information is then sent back to the receptors on the quality of the information provided.

This forms a cyclic process with information being continuously processed from one part of the nervous system to another. A real life example of this is if you touch an object of extreme heat. When you do your receptors send information to your neural network that you are touching a hot object. The neural network sends information to your effectors explaining to them the situation and the effectors give the desired response of pulling your hand away from the hot object. The effectors then send feedback to the neural network on the outcome of the situation and the neural network sends information to the receptors on the quality of the information provided and if any information needs changing.

A block diagram to help visualise the process.

The main way that the human brain has the ability to be able to process such large amounts of information is through a cell called a neuron. Simplified neurons are information processing units which pass on information from one neuron to an another. The brain has billions of neurons in it, so it needs to know which neurons to use to transfer the appropriate information. It does this with the help of synapses which are a medium between neurons. Each synapse has an allocated weight value which defines the so-called "importance" between neurons. This is a core concept of neural networks and the start of neural networks from a mathematical aspect, and we will begin describing neurons as functions starting from the next chapter.

3 Artificial neural networks through the lens of mathematics

3.1 Directed Graphs

Before we define artificial neural networks from a mathematical perspective we must first define directed graphs to show the process of how information is processed through an artificial neural network. It is important to have these graphs because they show the topology of the network (the whole structure of the network). Before we can go onto directed graphs we must cover signal flow graphs since this leads into directed graphs. Signal flow graphs are simple diagrams that illustrate the flow of information from one part of a diagram to another. From (1) we have the following definition.

Definition 3.1. A signal-flow graph is a network of directed links (branches) that are interconnected at certain points called nodes. A typical node j has an associated node signal x_j. A typical directed link originates at node j and terminates on node k; it has an associated transfer function or transmitter that specifies the manner in which the signal y_k at node k depends on the signal x_j at node j.

The way the signal travels through the graph can be defined by three different properties:

1. A signal travels along a link only in the direction that is given to it by an arrow (we can have two types of links a "synaptic" link and an "activation" link).

2. The signal of a node is equivalent to the sum of all the signals entering the relevant node via the different links

3. The signal at a node is transmitted to each outgoing link originating from that node, with the transmitter being entirely independent of the transfer functions of the outgoing links.

The following diagrams show these properties to help understand the process more clearly. These diagrams are inspired by (1).

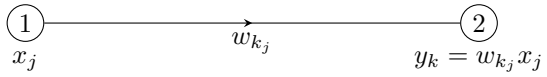


Figure 1: First Diagram

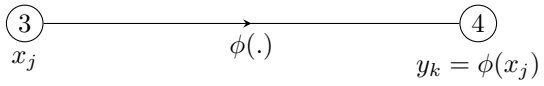


Figure 2: Second Diagram

The first two diagrams show property 1 in progress by showing what is inputted and outputted in terms of synaptic links as in the first diagram (from node 1 to 2) and activation links (from node 3 to 4) in the second.

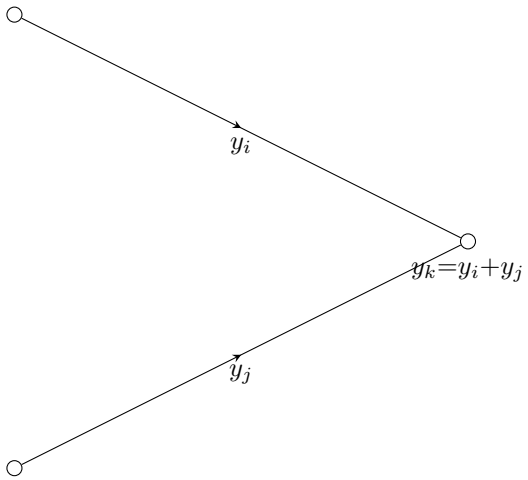


Figure 3: Third diagram

The third diagram helps show the second property by illustrating how the information combines to give a linear output.

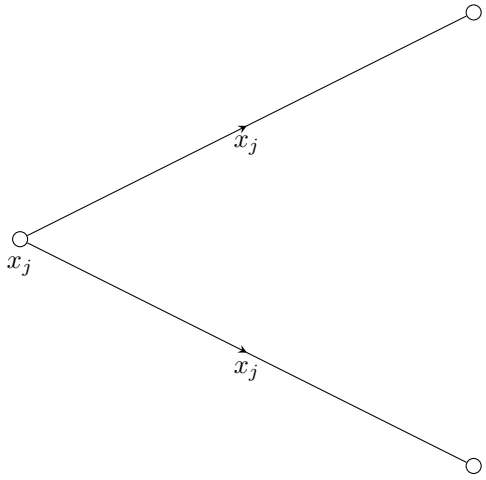


Figure 4: Diagram 4

The fourth diagram shows how the third property works in terms of how the signal is disbursed.

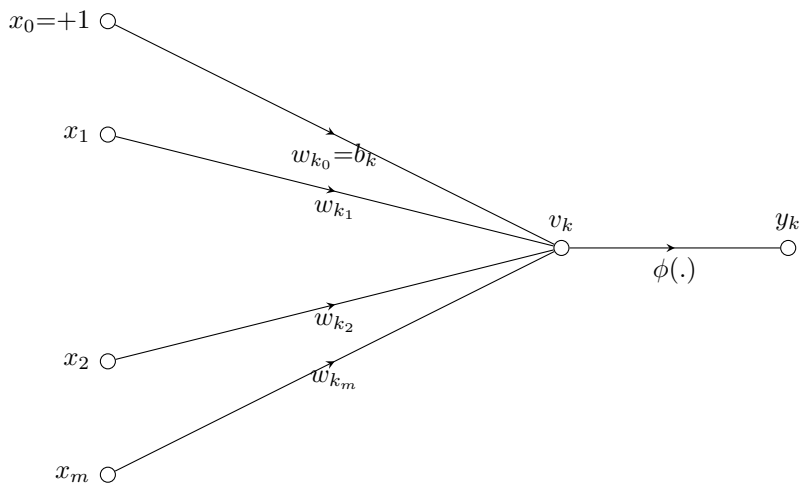


Figure 5: Diagram 5

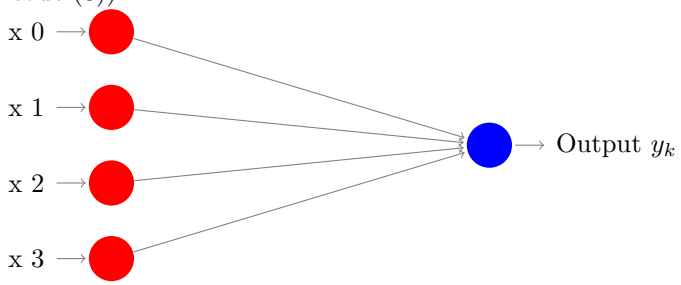
Above is the fifth diagram demonstrating a signal flow graph of a neuron.
This is the definition of a directed graphs from *Math insight* (2)

Definition 3.2. A directed graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are directed from one vertex to another. A directed graph is sometimes called a digraph or a directed network. In contrast, a graph where the edges are bidirectional is called an underacted graph.

Below is the definition of a directed edge.

Definition 3.3. A directed edge is an ordered pair of vertices with the ability that signals can travel from one vertex to another in a single direction.

A directed graph is a signal graph that not only shows the signal flow outside the neuron but also inside the neuron as well.
When it is necessary to only show the signal flow from neuron to neuron we can represent it as an architectural graph that is shown below (this and subsequent neural network diagrams are inspired by this code (3)).



There are also block diagrams which show a much more practical description of how an artificial neural network works. An example of this can be seen in the next section.

3.2 Basics of artificial neural networks

We will start this section by first defining what a neural network is from a mathematical perspective. Taken from (1) we have the following definitions.

Definition 3.4. A neural network is a directed graph consisting of nodes with interconnecting synaptic and activation links, and is characterised by four properties:

1. Each neuron is represented by a set of linear synaptic links, an externally applied bias, and a possibly nonlinear activation link. The bias is represented by a synaptic link connected to an input fixed at +1.
2. The synaptic links of a neuron weight their respective input signals.
3. The weighted sum of the input signals defines the induced local field of the neuron in question.
4. The activation link squashes the induced local field of the neuron to produce an output.

From the previous chapter we also determine the definition of a neuron.

Definition 3.5. An artificial neuron in a neural network is an information processing unit which models the biological neuron by having a mathematical function which processes information from the inputs it receives and applying the inputs to the mathematical function to send out a single output.

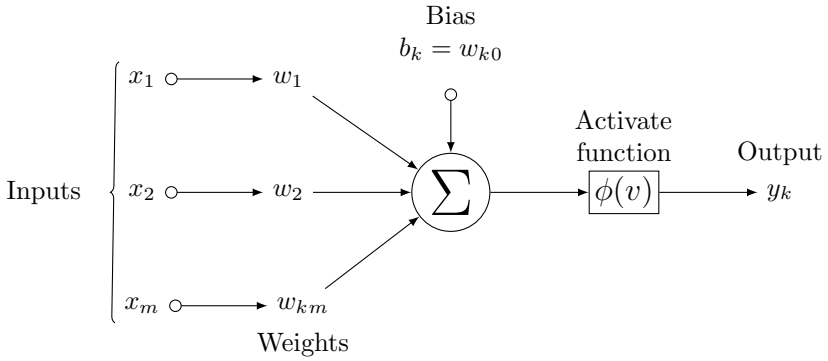
Artificial neurons can be represented by these two equations.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

$$y_k = \phi(u_k + b_k)$$

These are the values for the 1st equation:
The value $\boldsymbol{k}$ represents a neuron that information is processed into in a neural network
The value $\boldsymbol{j}$ represents another neuron in the neural network which is connected to the neuron $\boldsymbol{k}$
The value x_j represents the input signals onto neuron $\boldsymbol{k}$ from neuron $\boldsymbol{j}$
The value w_{kj} represents the synaptic weights of neuron $\boldsymbol{k}$
The value u_k is what is defined as the linear combiner of neuron $\boldsymbol{k}$

These are the values for the 2nd equation:
The value u_k represents the linear combiner from the first equation
The value b_k represents an externally applied bias onto neuron $\boldsymbol{k}$
The value ϕ denotes the activation function
The value y_k represents the output signal of neuron $\boldsymbol{k}$



The figure above is a block diagram which of a neuron which was mentioned in the previous chapter and is inspired from (4).

The first equation represents the information being processed into the neuron k and the second equation represents what information is being processed out of neuron k. Information from neuron j is

being processed into neuron k denoted by x_j. The synaptic weight describes the importance of the information being given from neuron j to neuron k denoted by w_{kj}. The sum of all the signals and weights is taken and is outputted as the linear combiner u_k. For the second equation ϕ is the activation function which is used to see if neuron k has relevant information to pass onwards. The value b_k acts as an external bias by either increasing or decreasing the net input of the activation function. This is demonstrated through the above diagram.

Input signals $x_1, x_2, x_3, ..., x_m$ have values from the interval $[0, 1]$. From the definition the externally applied bias has a value of 1. The bias may then be written as an input signal $x_0 = 1$ with the weight being defined as $w_{k0} = b_k$. We may then rewrite the first two equations as

$$u_k = \sum_{j=0}^{m} w_{kj} x_j$$

$$y_k = \phi(u_k)$$

We will now explore the properties of the activation function.

3.3 Activation Functions

Definition 3.6. The activation function which is denoted by $\phi(v)$, defines the output of a neuron in terms of the induced local field v.

The term "induced local field" is used in the place of the linear combiner (including the bias as well). There are five main types of activation functions:

1. Threshold Function
2. Piecewise-Linear Function
3. Sigmoid Function
4. ReLU
5. Softmax Function

The Threshold Function is defined by:

$$\phi(n) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

This activation function is the simplest of the activation functions but is also the least used since results from using this function are the least accurate. The logic behind this function is that if the induced local field is of a positive value the function activates and sends the value 1 on. If the value is negative then the function will not meet the requirements for activation.

The Piecewise-Linear Function is defined by:

$$\phi(n) = \begin{cases} 1 & \text{if } v \geq +\frac{1}{2} \\ v + 0.5 & \text{if } +\frac{1}{2} > v > -\frac{1}{2} \\ 0 & \text{if } v \leq -\frac{1}{2} \end{cases}$$

For this function we can see that it is similar to the threshold function apart from the difference that if the value for v is between positive and negative $\frac{1}{2}$ then the information sent on is of the value $v + 0.5$.

The Sigmoid Function is defined by:

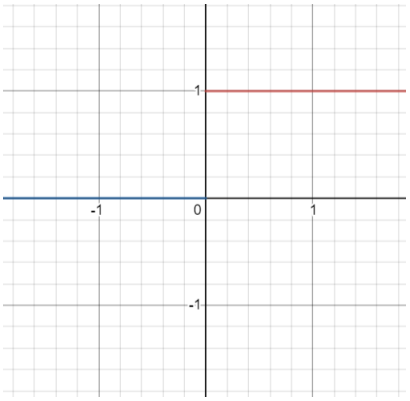


Figure 6: Threshold function (5)

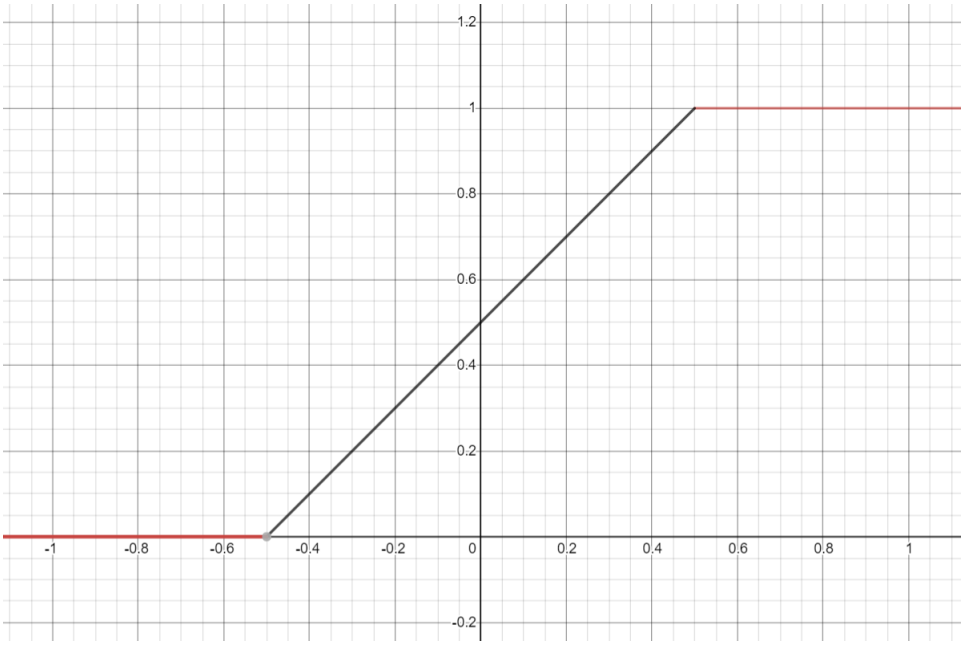


Figure 7: Piecewise-Linear function (6)

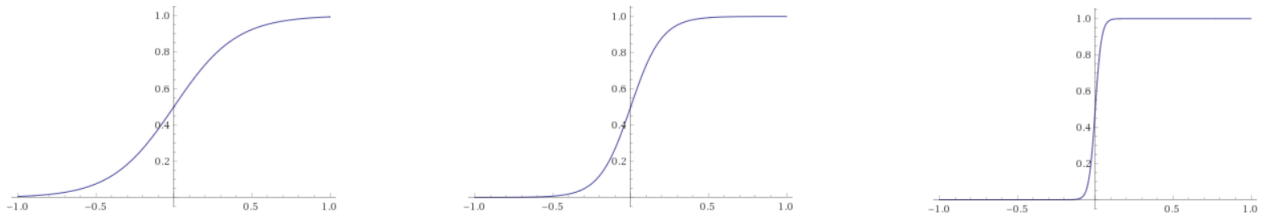


Figure 8: Sigmoid function with values a=5,10,50.

$$\phi(n) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases}$$

The sigmoid function was previously a very popular activation function, however data scientists have created more accurate functions based on its framework. The most common and well known sigmoid function is the of logistic function defined below as:

$$\phi(n) = \frac{1}{1 + exp(-av)}$$

The reason that the function has the a value is that it acts as the "slope parameter" which changes the angle of the slope depending on its value as we can see in Figure 8 (Figure 8 created in Wolfram Alpha (7)). We may also note that if the slope parameter increases to infinity then the sigmoid function takes on the values of a threshold function. A few reasons that the sigmoid function is more popular than the threshold function is that the threshold function only takes the constant values of 0 and 1 while the sigmoid function can take a much larger range of values form 0 to 1. The sigmoid function can be differentiated while the threshold can not, which is very useful when using neural networks. We may also notice that for the previous two functions we want values from a range of $[0, 1]$ while the sigmoid function takes on of range of values from $[-1, 1]$. This is because for the sigmoid function it is much more desirable for it to presume an anti symmetric form with respect to the origin. A sigmoid function might also be represented in the form of

$$\phi(n) = \tanh v$$

ReLU:

Taken from (8) ReLU stands for rectified linear unit. The equation for ReLU is given by

$$y = max(0, x)$$

The graph below made in wolfram alpha (7) is the graph of the ReLU equation.

ReLU is the most popular activation function used in neural networks today. It is a very simple equation so the computational power required to evaluate the function is very minimal, and the processing time is smaller due to it being such a straightforward function. Since all negative values of the function are set to zero, there are many neurons that won't activate (this is a desirable feature since we might have an overflow of information if many neurons activate at once). We will use the ReLU activation function in the MNIST neural network example used towards the end of the project.

The Softmax Function is defined by:

$$\sigma(z)_j = \frac{\exp z_j}{\sum_{k=1}^{K} \exp z_k}$$

To understand the Softmax Function it is important to understand what the practical application of Softmax is. Taken from (9) we see that the purpose of this activation function is to make the total

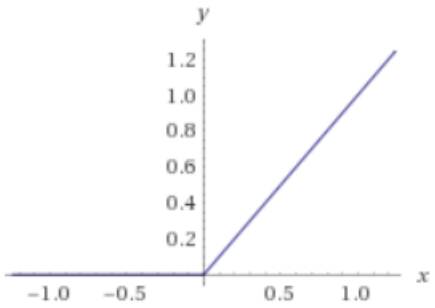


Figure 9: y=max(0,x)

output of the neurons equal 1. The purpose of this is to give each neuron a value corresponding to a certain output with the largest value being the value that the function chooses as its answer. For example lets say you define a neural network to tell the difference between apples, bananas and pears. We then define z as being a vector of inputs to the output layer (since we have 3 elements z would be defined as 1,2,3). The purpose of j is that it indexes the units so we have j= 1,2...K (this is dependent on how many fruit we have to test). It is important to note that the output of the Softmax function is the equal to the categorical probability distribution. Let's say we show our network a banana and it assigns the values [apple=0.2,banana=0.5,pear=0.3]. Since the emphasis is on banana, the function will output decide that the image most likely represents banana and thus picking banana over apple and pear. We will use the Softmax Function in the MNIST application later in the project.

3.4 Loss functions

This section will explain the purpose of loss functions in neural networks, as well as different loss functions used in neural networks. This section is based on (10).

The purpose of loss functions in neural networks is to help the artificial neural network minimize the loss it has by trying to make the weight parameters (covered in section 3.2) as accurate as possible. Loss is calculated by comparing the target value that it is trying to reach versus the predicted value assigned by the network during training. Gradient decent may then be used to improve the weights of the neural network.

The main loss functions used are the following:

1. Mean squared error
2. Binary Crossentropy
3. Categorical Crossentropy and Sparse Categorical Crossentropy

The Mean Squared Error is defined by:

$$Loss = \frac{1}{n}[\sum(\hat{Y} - Y)^2]$$

The value $\boldsymbol{n}$ represents the vector of n predictions from a sample of n data points.
The value $\hat{\boldsymbol{Y}}$ represents the vector of the predicted label.
The value $\boldsymbol{Y}$ represents the vector of the true label.
The output of this loss function is a whole number since it takes the mean of a vector of values and gives an output. This loss function is used, for example, when comparing the price of cars with respect to their engine size, number of seats e.t.c.

Binary Crossentropy is defined by:

$$Loss = -\frac{1}{n}\sum_{i=0}^{n}(Y * log(\hat{Y}_i) + (1-y) * log(1-\hat{Y}_i))$$

The values for the Binary Crossentropy are the same as for the previous loss function. It is important to note that the Binary Crossentropy works best when the output layer uses a sigmoid function (which was covered in 3.3) and the output value will be a binary 0 or 1. This is best used for neural networks when the output is a "yes" or "no" answer, for example if we want to identify if a picture is a cat or a dog (it's either a cat "yes" or a dog "no").

Categorical Crossentropy and Sparse Categorical Crossentropy have the same loss function as Binary Crossentropy with the only difference being how the value Y_i is used in the neural network. We use Categorical Crossentropy when samples can have multiple outputs, e.g a range of probabilities. We use Sparse Categorical Crossentropy when the neural network has one specific output for each label (we will use this loss function for our MNIST application).

3.5 Probabilistic Processes

Definition 3.7. A deterministic process is a process that if run again will always give the same outcome.

The activation functions stated previously are deterministic processes. In some neural networks it is sometimes wanted to have the process of the neuron activating to be a random process. This process is a probabilistic process defined as:

Definition 3.8. A probabilistic process is a process that if run again will not always give the same outcome because it is dependent on probabilities.

An example of a probabilistic process is if we have a six sided die and we want to roll to get the number 3. We know that the probability of the number 3 is $\frac{1}{6}$, however we don't know when in the throwing of the die the number 3 will occur. It could be on the first roll or on any subsequent roll.

In this process we want the neuron to be either -1 or for it to be of 1 i.e for it to be either activated or be deactivated. Let us take x to be the state of a neuron. It is mathematically represented as:

$$x = \begin{cases} +1 & \text{with probability } P(v) \\ -1 & \text{with probability } 1 - P(v) \end{cases}$$

The function $P(v)$ denotes the probability of that specific neuron activating. The value v denotes the induced local field of the neuron.

One of the common choices to use for the probability functions is a sigmoid shaped function denoted by:

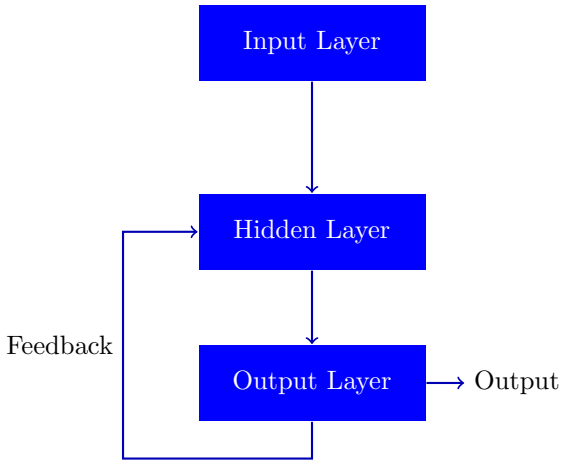
$$P(v) = \frac{1}{1 + exp(-v/T)}$$

The value of T is used as a temperature parameter to describe the thermal fluctuation representing the results of synaptic noise (note: this is not a physical temperature value). Synaptic noise occurs in biological neurons when there is a lot of processes happening in the synapses. The probabilistic models described above are used to represent the stochastic models of a neuron.

3.6 Iterative systems in artificial neural networks

Going back to the introduction it was mentioned that in biological neural networks that the process of information being sent to brain was a cyclical process. This is in regard to the fact that after the information that was sent to the brain is processed, the brain sends a signal back describing the quality of the information that was provided. A similar process happens in artificial neural networks known as feedback.

Definition 3.9. Feedback is said to exist in a dynamic system when the output of an element in the system impacts partially the input of that element in the system.



The above block diagram (inspired from (11)) shows the process of feedback in a two-layer network (one hidden layer, one output layer). The hidden layer sends signals to the output layer, where the output layer sends an output signal as well as feedback to the hidden layer for it to make amendments to get more accurate results by adjusting the weight values of the neurons in the hidden layer.

It is important to note that not all neural networks have a feedback system. The specific name for these networks is *iterative systems*. Let us take the case of a single loop feedback system. We have the input signal $x_j(n)$ with an internal signal $x'_j(n)$ and an output signal of $y_k(n)$ which are functions described as discrete-time variables n. From the Signal-flow diagram it is much easier to see the process of feedback since we have the forward path characterised by the operator X and the feedback path characterised by the operator Y. We will now describe this process of the input and outputs with these two equations:

$$y_k(n) = X[x'_j(n)]$$
$$x'_j(n) = x_j(n) + Y[y_k(n)]$$

The first equation describes the output process from the internal signal to the output signal. The second equation describes the process of the internal signal being influenced by the input signal and the output signal. The square brackets show that X and Y act as operators. Since both equations have the function x'_j we can combine both equations to give the following equation:

$$y_k(n) = \frac{X}{1 - XY}[x_j(n)]$$

We have X/(1-XY) to be described as the close-loop operator of the system and XY as the open-loop operator. In most cases the open loop operator is noncommutative i.e $XY \neq YX$. Let us now substitute the value of X for a fixed weight w and for Y, substitute with the unit-delay operator z^{-1} (the output of the unit-delay operator is delayed with respect to the input by one time unit). We may now describe the closed-loop operator of the system as

$$\frac{X}{1 - XY} = \frac{w}{1 - wz^{-1}} = w(1 - wz^{-1})^{-1}$$

By applying the binomial expansion to $w(1 - wz^{-1})^{-1}$ the closed loop operator is described as:

$$\frac{X}{1 - XY} = w \sum_{l=0}^{\infty} w^l z^{-l}$$

Therefore by substituting the summation into the equation for $y_k(n)$ we derive the following equation:

$$y_k(n) = w \sum_{l=0}^{\infty} w^l z^{-l}[x_j(n)]$$

Notice that we have included again the square brackets to signify that z^{-1} is an operator. More specifically, since we know that z^{-1} is a unit-delay operator, we also derive this equation:

$$z^{-l}[x_j(n)] = x_j(n-l)$$

We express $x_j(n-l)$ as a sample of an input signal delayed by l time units. In accordance with the above, we can express $y_k(n)$ as an infinite weighted summation of current and previous samples of the input signal $x_j(n)$, described as:

$$y_k(n) = \sum_{l=0}^{\infty} w^{l+1} z^{-l} x_j(n-l)$$

We now observe that the dynamic behaviour of the system is managed by the weight w. Specifically, we are able to describe two specific cases as described in (1):

1. The value $w < 1$, for which the output signal $y_k(n)$ is exponentially *convergent*; that is, the system is *stable*.

2. The value $w \geq 1$, for which the output signal $y_k(n)$ is *divergent*; that is, the system is *unstable*. If w=1 the divergence is linear, and if w> 1 the divergence is exponential.

The concept of stability is very important in the research of feedback systems. When we have $w < 1$ it is consistent with a system of *infinity memory* such that the output of said system is dependent on previous input samples extending all the way back to infinity. The memory is also described as fading i.e as time n increases the relevance of older samples decreases compared to newer samples.

This will conclude this section since we now have the required knowledge to understand neural networks on a basic level. The next section will see the network architecture and how neurons work together in a broader sense.

3.7 Network architectures in artificial neural networks

Until now in this project we have discussed and shown the properties of neurons in artificial neural networks. For this chapter we will discuss how we put neurons together in an architectural structure to get the desired outcome. Before we continue we must answer the question "What is the main reason we structure neurons in a network?" The answer to this is to create supervised machine learning algorithms.

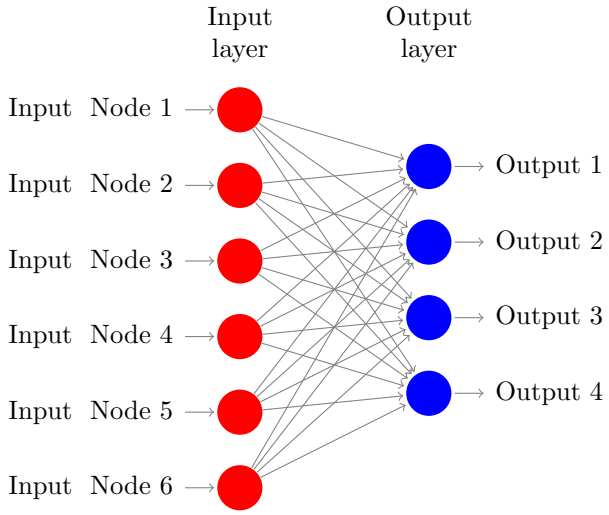
Definition 3.10. Supervised machine learning performs the task of learning a function that maps input labelled data to output labels based on input-output pairs.

The term label is interpreted differently depending on definition. For example if you want to interpret a set of integers, ex the units [1 2 3 4 5] you could set your label such that each unit is its own label. I.e 1 is a label, 2 is a label e.t.c. Or you may set your label to be the whole set [1 2 3 4 5]. Depending on what you want your learning algorithm to learn.

The architecture we use for a supervised machine learning algorithm will vary largely depending on what way we want the supervised machine learning algorithm to solve the problem. For the purpose of this project, we will not go into the specifics of which network architecture is suitable for each purpose since that is beyond the scope of this project. Outlined below (from (1)) are three main architectural structures we can use for supervised machine learning as well as how they function.

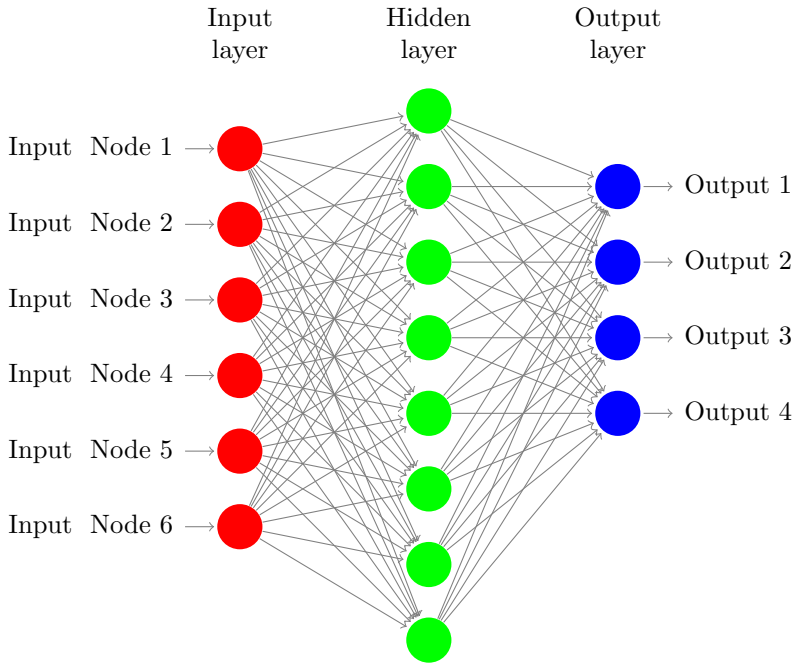
1. Single-Layer Feedforward Networks
2. Multilayer Feedforward Networks
3. Recurrent Networks

Single-Layer Feedforward Networks: These are the simplest forms of networks. The reason we describe these networks as having layers is due to the nature of how neurons are arranged in these networks.



From the above diagram there are source nodes (the input layer) which have a mapping onto the output layer of neurons (computational nodes). There is no mapping from the output layer to the input layer though, which means there is no feedback in this network and, hence, it is called a feedforward network (sometimes also known as an acyclic). The name *single-layered network* is used since we are only interested in the output layer; not the input layer since there are no computations happening in the input layer.

Multilayer Feedforward Networks: Multilayered feedforward networks are very similar in structure and function of their single layered counter parts. They have an input and an output layer and are feedforward in structure so they don't have any feedback. The main difference is that there exist extra layers of computation within the network.

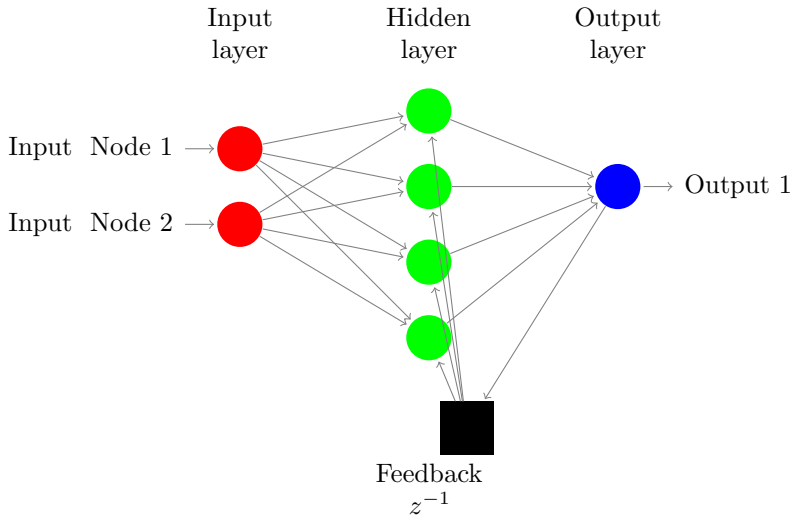


From the above diagram there is an extra layer of neurons which are referred to as the *hidden layer* of neurons. This layer exists to act between the input and output layers with the purpose of helping the learning algorithm by adding more computations within the network. As explained in (1) the purpose of having more than 1 layer of computation is to be able to to extract higher-order statistics. This is to help the learning algorithm have a greater understanding of the process and be able to interpret the higher-order statistics coming into the layers in a more accurate manner. When there are many input nodes it is extremely useful to have hidden layers to help extract higher-order statistics.

The logic behind the process of a multilayered feedforward network is to have the source nodes in the input layer give the respective components of the input vectors which, in turn, account for the input signals applied to the neurons in the second layer. The process is then repeated with the output of the second layer mapped as the input for the third layer, e.t.c. The output of the final layer of neurons is used to summarise the response of the network as a whole with how the activation pattern sent signals through the network (activation pattern is supplied by the source node at the beginning of the network).

Specifically for the above diagram we have one input layer, one hidden layer and one output layer. Since there are six source node, eight hidden layer neurons and four output neurons we may call this network a 6-8-4 network to illustrate the nodes and neurons in the network. If the above diagram had a second hidden layer with ten neurons we could call it a 6-8-10-4 network. We define a neural network as fully connected if every node in each layer is connected to each node in the next layer which the signal is being sent towards. If this is not the case then it is defined as a partially connected neural network.

Recurrent Networks: Recurrent neural networks are different from their feedforward counterparts by having at least one feedback loop in the network. This feedback loop can take place in any part of the network.



From the above diagram we can see that there is a feedback loop in the network from the output layer back to the hidden layer. There is also a unit-delay element in the feedback loop as described in the previous chapter on the function of feedback in neural networks. It is worth mentioning that we can have a feedback network with only one layer of neurons and no hidden layers and source nodes.

3.8 Knowledge representation in artificial neural networks

In the previous section there was a short introduction to the term *labels* and a short description of their context in terms of the system they are used in. This section will go more in depth on how knowledge is represented in neural networks and the rules that must be followed to represent knowledge correctly.

Firstly (taken from (1)) we define the term knowledge as:

Definition 3.11. The term knowledge (used in artificial neural networks) refers to the stored information that is used to predict, interpret and appropriately respond to the outside world.

There are two main aspects to knowledge representation:

1. The information that is made clear to the network
2. How information is actually encoded for later use

We can see that, naturally, knowledge representation is determined towards a goal. In real-world examples of machine learning the better our representation of knowledge then better our solution will be. Because neural networks can be used for a large variety of different problems, designing the correct network can be difficult due to the large number of options available to us.

All neural networks need an environment to work in. This environment can range from the real world to an environment that is artificially set up for the neural network to work in. For example, we can set up a learning algorithm in a neural network to try and learn how to beat the first Super Mario game. The environment in this case would be the Super Mario game. From this we can gather that there are two types of information:

1. The state of the world that is currently known, which is shown by facts about what the world is and what has been known of it; this is referred to as *prior information*.
2. Observations of the world which are taken by the use of sensors which are made to investigate the environment in which the neural network operates. The observations taken give a variety of information which are used as examples used to train the neural network.

In the previous section it was touched briefly on labeled data and defined supervised machine learning. Above when we mention observations, the observations themselves can give a variety of information. The examples can be either labeled or unlabeled data. Unlabeled data is mostly used for unsupervised learning which is machine learning purely from the environment and without labeling any data. It is worth mentioning that the machine is still given goals to achieve not the data on how to do that. The process that supervised machine learning network uses to be able to gather information and give a desired result is called learning and generalization. This area of machine learning is still growing with many uncertainties so the application of machine learning we will describe will be mostly based on supervised machine learning.

Let us first take a set of handwritten digits (i.e, a set of handwritten numbers from 0 to 9). This is the most common example of how neural networks are trained and will be the practical application used later on. We must first define input-output pairs of data.We do this by defining within the network a set of handwritten digits and how each digit corresponds to a specified output. The network then trains to replicate the desired result. Next, the network is presented with another set of handwritten digits, but the network must predict which handwritten digit corresponds to which digit. This is defined as the generalization process. From section 3.2, synaptic weights and biases are part of the process of assigning values to the knowledge that is gathered from the environment to be used in the neural network for the desired outcome.

There are four general rules (taken from (1)) that networks follow to be able to represent knowledge in the most truthful way which follow a logical pattern.

Rule 1: When we have inputs that are alike and are from alike classes, they should usually produce similar outcomes in the network and should be categorised as being part of the same group.

This is logically sound since if there are similar inputs then they should have similar outputs and should be categorised together. The question arises: what makes two inputs as being "similar"? One way of approaching this is by defining similarity using euclidean distance.

Let us first start by defining x_i as an m-by-1 vector denoted as:

$$x_i = [x_{i1}, x_{i2}, x_{i3}, ..., x_{im}]^T$$

where the elements are real and T is the transposition of the matrix. The vector x_i is defined as being a point in an m-dimensional space, which is referred to as Euclidean space, and takes real values $\mathbb{R}^m$. When we have a pair of vectors x_i and x_j the Euclidean distance between them is defined as follows:

$$d(x_i, x_j) = ||x_i - x_j|| = [\sum_{k=1}^{m} (x_{ik} - x_{jk})^2]^{\frac{1}{2}}$$

We define x_{ik} and x_{jk} to be the kth elements assigned to the input vectors x_i and x_j. The term *reciprocal* is given as being the similarity between the Euclidean distance $d(x_i, x_j)$ of the vectors x_i and x_j. The closer that vectors x_i and x_j are to each other then the smaller the Euclidean distance $d(x_i, x_j)$ will be and the more similar the vectors will be to one another. If they are significantly close to each other then they will categorised in the same group as stated by Rule 1. There are other methods of testing for the similarity between vectors, such as using the dot product, or, if we wanted to measure the distance between populations of data we could use the Mahalanobis method.

Rule 2: When we have inputs that are not alike in the and are from separate classes then they should have a large difference in the output of the network. This rule is simply the logical opposite of the first rule.

Rule 3: When there is a feature that has a value of great importance then the representation of this in the neural network should be done with a large number of neurons.

Going back to the example of handwritten digits, if we have a feature that is unimportant (such as a blank space) then there will be fewer neurons dedicated to the representation of that blank space. If there is more important feature (such as a straight line in the number 1) then there will be more neurons to represent that feature since it is of a higher importance.

Rule 4: The design of a neural network should include and be informed by invariances and prior information in order to simplify its design. These should then not have to be learned by the network during training.

Rule 4 is particularly desirable since by following it we get neural networks that have a specialized structure. Biologically, the visual and auditory systems are specialized networks since they focus on taking the most important visual and auditory information from the environment.

4 Real life example of artificial neural networks in MNIST

4.1 Introduction

This chapter will go over how we apply artificial neural networks to the MNIST while showing the coding process behind it and referring to the previous chapters on how the mathematics links with the coding aspect of it. The code was written in python using Google Collab and Spyder and will use some aspects from the Tensorflow 2 quick start for beginners (12) but was otherwise designed by the author of this project.

4.2 Running the code

This section will cover the installation and setup of an environment for running the example in the next section. If you do not plan to run the example the next section will explain it in full detail. To run the code you will require an installation of the latest version of Anaconda for Python as well as the Keras machine learning library. You can find the latest installer here (13). When Anaconda has been installed open the Anaconda 3 command prompt (on Windows you can find this by searching Anaconda Prompt (Anaconda3)). Install the Keras library by typing `conda install keras`.

Once the installation process completes obtain a copy of the example code from here (14). Open Spyder (included with Anaconda Navigator) and navigate to the example code directory. Open the file `mnist_network_code.py` and run it by clicking the `Run file` button or by pressing `F5` on the keyboard. The output of the program is in the integrated terminal located by default on the right-hand side.

4.3 Explanation of code

This section will explain the code and how many of the key ideas used are talked about in the previous sections.

With reference to the code in your editor or in 10, lines 9 to 11 begin by importing the Keras library as a backend for machine learning into our project. This library includes many useful and common machine learning functions and tools used in the construction of artificial neural networks. This greatly simplifies the task of creating a neural network since Keras includes many building blocks as simple function calls. For example instead of downloading and importing the MNIST dataset ourselves, Keras includes a function for accessing the MNIST dataset directly as seen on line 14.

Using the dataset from line 14, we separate the data into

1. `x_train`: A list of image pixel values representing each pixel of each image of handwritten numerical digits. The list is of `28*28*n` where `n` is the number of images in the training set.

2. `y_train`: A list of integers representing the value of each digit in the training set. The list is of size `n` where `n` is the number of images in the training set.

3. `x_test`: A list of image pixel values representing each pixel of each image of handwritten numerical digits. The list is of `28*28*n` where `n` is the number of images in the testing set.

4. `y_test`: A list of integers representing the value of each digit in the testing set. The list is of size `n` where `n` is the number of images in the testing set.

For each element in `x_train` and `x_test` we divide the value by 255 to normalise each value into a range of 0 to 1 as done by line 18. We divide by 255 because the current range is from 0 to 255 as this is the domain of pixel values. We verify the operation has been done successfully by printing the output of the normalisation to the terminal on line 20.

We now begin by creating the model for the neural network. As seen on line 23, we use a `Sequential` model type as this is the simplest for the purpose of our example. In previous sections we referred to the structure of neural networks as being layered; lines 26 and 27 add the first layer to our architecture. This layer, known as a `Flatten` layer, takes a list of values and transposes it into a `28*28` pixel image.

Lines 30 and 31 implement and add the first hidden layer of the neural network. This is a `Dense` layer with 128 neurons and a ReLU activation function. The purpose of this layer is to take the output of the `Flatten` layer and incorporate the learned weights into the model by applying those weight values to the neurons. Lines 34 and 35 apply a `Dropout` operation which is used to reduce the possibility of overfitting while training the model. Outfitting is when the network performs well on training data and reports a high accuracy but under-performs on test data and real life data. Dropout is used to destroy a proportional random subset of the neurons in the previous layer. Note that this isn't classified as a full layer according to the definitions of layers in the previous sections.

Lines 38 and 39 implement the output layer of the network. This is done by using another `Dense` layer, though this time with the activation function Softmax which was defined previously in the project. We use Softmax because we desire to have each neuron output a value between 0 and 1 such that the sum of all values equals 1 for a given input image. The output value can then be used as a likelihood for the integer that the image represents. Since we want to define the numbers between 0 and 9 as output classes, we use 10 neurons in our layer.

Using the `compile` function on line 42, the model is assembled using the layers we defined previously. We assign to the model a loss function which, in this case, is Sparse Categorical Crossentropy which was defined previously in the project. Sparse Categorical Crossentropy is used because our outputs can be represented as probabilities, as mentioned above. We also assign an optimiser to the model. An optimiser is used to help the neural network adjust its weight values to a more accurate value at a faster rate. Optimisers are outside the scope of this project; for the purpose of our example we will use the Adam optimiser which is based on a technique known as stochastic gradient descent. Lastly we also specify that we wished our output of our network to include the accuracy metric so we can evaluate its performance.

Using the training sets defined earlier in the code, we train the network on line 45. We train the network over six iterations of the training data to improve accuracy during training. The number of iterations was determined empirically for best accuracy in this example. Lines 48 and 49 test and report the performance and accuracy of the network on the previously defined testing data. This output can be seen in 11 or by running the code. Notice that there are six epochs of training where accuracy is increasing with every unit. Each epoch takes roughly 5 seconds to complete training on the training set which consists of 60000 images. Final accuracy on the testing set of 10000 images is around 0.977.

5 Conclusion

The main-take away from this project is to get an introduction to the world of artificial neural networks and to see how the mathematical aspect links to the practical side by referencing the functions used in the neural network and showing the real life output of it through a practical coded example. The practical code shows us that we can take a simple concept (identifying hand written numerical values) and use neural networks to identify them. The code itself for such as task is also short which makes it easy to understand and replicate. We specified the neural network to train using 60000 individual images and it computed this in around 5 seconds, showing how quickly it can tackle large datasets. This is only the tip of the iceberg for the world of neural networks and data science since this only covers the fundamentals. There is still much more to explain, such as optimizing the structure of neural networks and integrating new algorithms to be implemented into the network. Data science is very new

```python
import keras
from keras.layers import Flatten,Dense,Dropout
from keras.models import Sequential

# Load mnist dataset
mnist =keras.datasets.mnist
#x_train is the list of images y_train is the labels assigned to each image
(x_train,y_train),(x_test,y_test) = mnist.load_data()
# Normalise values to range (0,1)
x_train,x_test = x_train/255.0,x_test/255.0

print(x_train.shape)

# model=model.keras.Sequential()
model = Sequential()
# optimizer='adam'
#(28,28) represents the dimensions of image in pixels
input_layer=Flatten(input_shape=(28,28))
model.add(input_layer)

#Activation function is relu
hidden_layer_1=Dense(128,activation='relu')
model.add(hidden_layer_1)

#Percentage of nodes destroyed
hidden_layer_2=Dropout(0.3)
model.add(hidden_layer_2)

#Activation function is softmax
output_layer=Dense(10,activation='softmax')
model.add(output_layer)

#Building model with appropiate loss function and optimizer. Metrics is values you want to show i.e in this case accuracy
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

#Training sets for code with 6 iterations of training
model.fit(x_train,y_train,epochs=6)

#The final test set checking the models perfomance vs actual test data
score=model.evaluate(x_test,y_test)
print(' accuracy ',score[1])
```

Figure 10: MNIST code

```
Epoch 1/6
60000/60000 [==============================] - 5s
82us/step - loss: 0.3222 - accuracy: 0.9062
Epoch 2/6
60000/60000 [==============================] - 5s
77us/step - loss: 0.1627 - accuracy: 0.9522
Epoch 3/6
60000/60000 [==============================] - 5s
78us/step - loss: 0.1287 - accuracy: 0.9617
Epoch 4/6
60000/60000 [==============================] - 5s
78us/step - loss: 0.1068 - accuracy: 0.9678
Epoch 5/6
60000/60000 [==============================] - 5s
80us/step - loss: 0.0942 - accuracy: 0.9707
Epoch 6/6
60000/60000 [==============================] - 4s
74us/step - loss: 0.0839 - accuracy: 0.9736
10000/10000 [==============================] - 0s
32us/step
 accuracy  0.9772999882698059
```

Figure 11: Results from MNIST network

so there is still much more to explore in this field that will take an even bigger influence in the job market as time progresses.

6 Bibliography

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*. USA: Prentice Hall PTR, 2nd ed., 1998.

[2] "Directed-graph-definition." https://mathinsight.org/definition/directed_graph. September 4, 2020.

[3] "Neural-network-diagram-reference." http://www.texample.net/tikz/examples/neural-network. September 4, 2020.

[4] "Block-diagram-reference." https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network. September 4, 2020.

[5] "Threshhold-function." https://www.desmos.com/calculator. September 4, 2020.

[6] "Piecewise-linear-function." https://www.desmos.com/calculator. September 4, 2020.

[7] "Wolfram-alpha." https://www.wolframalpha.com. September 4, 2020.

[8] "A-practical-guide-to-relu." https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7. September 4, 2020.

[9] "Softmax-function." https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions. September 4, 2020.

[10] "Loss-functions." https://towardsdatascience.com/understanding-different-loss-functions-for-neural-n September 4, 2020.

[11] "Feedback-diagram." https://tex.stackexchange.com/questions/502559/feedback-diagram. September 4, 2020.

[12] "Tensorflow-quick-start-guide-for-beginners." https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/beginner.ipynb. September 4, 2020.

[13] "Anaconda-download-link." https://www.anaconda.com/distribution/. September 4, 2020.

[14] "Github-link." https://github.com/Intern097/Dissertation_code. September 4, 2020.