A Mathematical Introduction to Neural Networks in Machine Learning

Ansh Erinjery

Contents

Chapter 1

Introduction

> "Machine intelligence is the last invention that humanity will ever need to make."
>
> *Nick Bostrom*

1.1 The Importance of Machine Learning

Machine learning is revolutionising the way we interact with modern technology. It influences many sectors, such as finance, autonomous systems, healthcare and natural language processing. Machine learning involves developing algorithms that can learn from and make predictions based on data. The mathematical foundations of machine learning are crucial in providing the tools required to create these complex systems, ensuring they are both efficient and effective.

In an era where data is generated at a rapid pace, machine learning provides the means to process and analyse vast amounts of data and information. These algorithms use mathematical models to efficiently sift through large datasets to identify patterns and trends that would be practically impossible to detect with manual methods. [1]

1.1.1 The Uses of Machine Learning

Machine learning improves automation and efficiency across various industries. It significantly impacts the automation of repetitive tasks, optimises processes, and even makes decisions. For instance, machine learning algorithms detect fraudulent transactions in real-time, reducing financial crimes. In healthcare, machine learning models forecast disease outbreaks, personalise treatment plans, and help with diagnosis, improving patients' treatments.

1.1.2 The Importance of Mathematics

Every aspect of machine learning heavily relies on mathematics, from creating models to optimising algorithms. Linear algebra helps in understanding data structures, calculus provides the means for optimising models, and

probability theory supports the capacity to handle uncertainty and generate predictions in models. By utilising these mathematical principles, machine learning algorithms can uncover patterns in data, make informed decisions and enhance their performance over time through iterative learning processes. [2]

By understanding the role mathematics plays in machine learning, we can better understand the strengths and limitations of neural networks and other machine learning models to create more effective and innovative solutions in such a rapidly evolving field. This project aims to explore the fundamental concepts behind neural networks in order to gain a deeper appreciation of their mathematical foundations and to understand how these intriguing systems work.

1.2 The History of Neural Networks

Neural networks have a fascinating history spanning several decades. Their origins can be traced back to the early 20th century with the work of Warren McCulloch, a neuroscientist, and Walter Pitts, a logician, in 1943. They discovered that neurons in the human brain perform logical operations and have binary outputs that depend on specific thresholds. Based on this fact, they presented the idea of a mathematical model of an artificial neuron, demonstrating that these simple units could be combined to perform logical operations, laying the groundwork for this promising field. [3]

1.2.1 The Perceptron

Following this, further progress was made with the development of the perceptron by Frank Rosenblatt. The perceptron was a simple linear classifier that could learn from data through adjustment of its weights. [4]

However, the field faced a significant setback in 1969 when Marvin Minsky and Seymour Papert published “Perceptrons”. The book highlighted the flaws of single-layer perceptrons, particularly their inability to solve non-linearly separable problems, such as the XOR problem. [5] Scientists began to lose interest in the field due to a lack of progress and more promising methods available at the time, a period known as "AI Winter."

1.2.2 Multilayer Perceptrons

Neural networks became more popular in the 1980s with the development of multi-layer perceptrons and the backpropagation algorithm, discovered by several researchers. Backpropagation provided an efficient way of training multi-layer networks, overcoming the limitations identified by Minsky and

Papert. [6] This breakthrough drastically improved neural network performance and reignited interest in the field.

Throughout the 1990s and early 2000s, neural networks continued to evolve. They benefited from the increased computational power at the time and the availability of large datasets. Researchers explored various architectures, such as convolutional neural networks for image classification and recurrent neural networks for analysing sequential data. [7] These innovations demonstrated the potential of neural networks in many diverse applications.

1.2.3 Staggering Growth

The 2010s were the years of the greatest achievements and development of neural networks due to the advances in deep learning. Deep learning is a sub-field of machine learning involving training large neural networks with many layers or "deep" architectures. The development of powerful GPUs and algorithms, such as dropout and batch normalisation, enabled the training of these complex models. [8] Landmark achievements include the success of AlexNet in the ImageNet competition [9], producing remarkable results, and the development of AlphaGo by DeepMind [10], a computer program that plays the board game Go.

Neural networks are now at the forefront of AI research and applications. They continue to push the boundaries of current capabilities with advancements in model architectures, optimisation techniques and the understanding of their theoretical principles. [11] The history of neural networks is evidence of the immense promise this field holds for our future.

1.3 Applications of Neural Networks

Computer vision is one of the most widely used areas of application of neural networks. Convolutional Neural Networks are widely used in image recognition and classification tasks. They are used in facial recognition systems, object detection, and medical image analysis, where they can help diagnose diseases by analysing medical scans and images. [12]

Neural networks also play a crucial role in autonomous systems and robotics. Self-driving cars use neural networks to interpret sensory data from cameras (LiDAR) to navigate different environments safely. These networks help identify obstacles, predict the movement of other vehicles and pedestrians, and make decisions to ensure safe and efficient driving. [13]

In finance, neural networks are used for predictive analytics, fraud detection, and algorithmic trading. By analysing large datasets of financial transactions, neural networks can identify fraudulent activities and predict market trends.

This helps with risk management, investment decisions and optimising trading strategies. [14]

Neural networks have made great contributions to cybersecurity. By analysing network traffic and user behaviour, they can identify potential security breaches and take measures to prevent attacks. Their ability to adapt and learn from new data makes them effective in combating evolving cybersecurity threats. [15] The above applications are just a few among many practical uses of neural networks in the real world.

Chapter 2

The Basics of Neural Networks

> "A baby learns to crawl, walk and then run. We are in the crawling stage when it comes to applying machine learning."
>
> *Dave Waters*

2.1 What is an Artificial Neural Network?

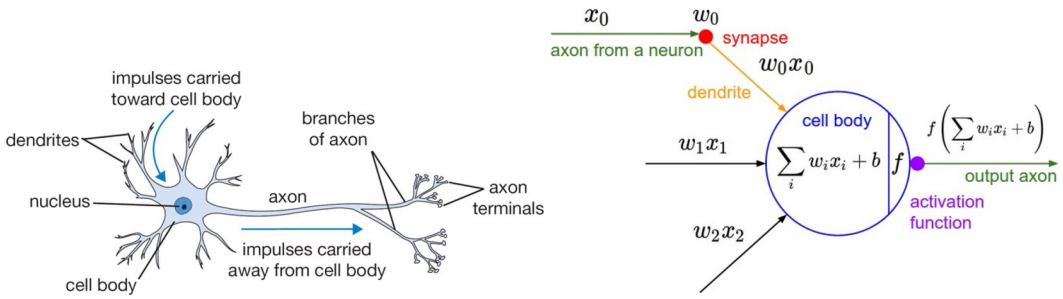
An artificial neural network (ANN) is a software implementation of the neural structures of our brain. These computational models consist of layers of interconnected nodes called neurons, which work together to process input data, recognise patterns, and make decisions. Like the brain's neurons, these artificial neurons receive inputs, apply a transformation, and pass the result to the next neuron. This interconnected structure allows ANNs to learn and adapt from data and improve their performance over time. [8]

2.1.1 Biological Inspiration

The inspiration for ANNs comes from the biological neural networks found in the human brain. In the brain, neurons are connected by synapses that transmit electrical signals. This allows for complex thought processes, learning, and memory. Similarly, in ANNs, artificial neurons are connected by weighted connections that transmit signals from one neuron to the next. [8] These weights are adjusted during the training process, like how the strengths of synaptic links change in the brain through learning experiences. This adjustment process is important for the network to learn from data and make accurate predictions.

2.1.2 Architecture

The architecture of an ANN includes an input layer, multiple hidden layers, and an output layer. The input layer receives data, which is then transformed and processed through the hidden layers. Each neuron in a hidden layer applies an activation function to its input, allowing the network to identify patterns. The output layer then produces the final predictions based on the processed data. [16]



(A) A neuron in the brain [17]

(B) An artificial neuron [18]

FIGURE 2.1: The relationship between human and mathematical neurons

2.1.3 Training

Training an ANN involves a method called backpropagation, which adjusts the weights of the connections based on the error of the network's predictions. This process begins with forward propagation, where input data is passed through the network to generate an output. The error between this output and the required result is calculated using a loss function. Backpropagation involves computing the gradient of this error with respect to each weight and updating the weights in order to minimise the error. [2] This iterative process allows the network to learn from its mistakes and gradually improve its accuracy.

2.2 Supervised learning

Supervised learning is a machine learning paradigm consisting of labelled data where each input is paired with an output label. The goal of supervised learning is to learn a mapping of inputs and outputs that an be applied to predict the outputs of new, unseen data. We can state the problem as follows:

The supervised learning problem [19] Given an unknown function $f : X \to Y$ between spaces X and Y, find a good approximation of f using only a dataset of N samples:

$$\mathcal{D} = \left\{ \left(x_i, y_i\right) \right\}_{i=1}^{N} \quad \text{with} \quad y_i = f(x_i) \quad \text{for all } i = 1 \dots N.$$

The space X is called the **feature space** and its elements, feature vectors. The space Y is called the **label space** and its elements, labels. [16]

Example 3.1 Let X be the space of all images of apples and oranges and f the classifier that maps X to $Y = \{\text{"orange"}, \text{"apple"}\}$. We can express this numerically as $X = [0,1]^{3\times H\times W}$ (i.e. an image of height H and width W with

3 colour options) and $Y = [0,1]^2$ (one probability for each of the two classes).

The way to approach this problem involves three main steps:

1. Choose a model $F : X \times W \to Y$ with a parameter space W (W for weights).
2. To quantify the error of the model's output, we need to choose a **loss function** $\ell : Y \times Y \to \mathbb{R}$, where $\ell(y_1, y_2)$ indicates "how different" y_1 and y_2 are.
3. Based on the dataset and the loss function, choose $w \in W$ so that $F(\,\cdot\,; w)$ is the best possible approximation of the target function f. [16]

However, this approach leaves us with some questions: How do we choose a suitable model and a loss function, and how can this be optimised?

These questions do not have straightforward answers and can be handled by trial and error at first. The choices made will have a significant impact on the final output, and the set of assumptions made to continue with the problem is called the **inductive bias**. [8] Here, we will focus on using neural networks as our model of choice and will answer the other questions later on.

2.2.1 The Universal Approximation Theorem

Theorem 3.2.1 (The Universal Approximation Theorem) [20] *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a non-constant, bounded, and monotonically increasing activation function. Then, for any continuous function $f : \mathbb{R}^n \to \mathbb{R}$, and any $\epsilon > 0$, there exists a neural network N with activation function σ such that:*

$$\sup_{x \in K} |f(x) - N(x)| < \epsilon$$

for all x in a compact subset K of $\mathbb{R}^n$.

This theorem shows that a neural network with a single hidden layer and a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$, given the activation function is non-constant, bounded, and monotonically increasing. The theorem specifically applies to feedforward neural networks with a single hidden layer. The number of neurons in this hidden layer must be sufficient to achieve the approximation within the desired precision ϵ.

The Universal Approximation Theorem is crucial because it establishes that neural networks with just one hidden layer can theoretically approximate any continuous function. This demonstrates the power of neural networks as universal approximators that can learn and represent any mapping between inputs and outputs. [11]

2.2.2 A Feed Forward Neural Network

Let us also define *a feed forward neural network:*

Example 3.2.1 *Let $L, N_0, \ldots, N_L \in \mathbb{R}$, and $\sigma_i : \mathbb{R} \to \mathbb{R}$ for any $i = 1, 2, \ldots, L$, let $A_i : \mathbb{R}^{N_{i-1}} \to \mathbb{R}^{N_i}$ be affine functions. A function $F : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined as*

$$F(x) = F_L \circ \cdots \circ F_1(x)$$

where

$$F_i = \sigma_i \circ A_i \quad \text{for } i = 1, 2, \ldots, L$$

is called a neural network. [8]

L is the total number of layers; $N_1, \ldots, N_{L-1}$ are the dimensions of the hidden layers and N_0, N_L denote the input and output layers respectively. The composition function demonstrates the passing forward of information from subsequent layers. We will discuss these various components in detail later to develop our understanding. *Feed forward* refers to the direction of data flow-forward through the layers of the network. [11]

2.3 Artificial Neurons

Neurons are the fundamental building blocks of neural networks. Each neuron receives one or more inputs from the previous layer of neurons, processes them and produces an output. Given inputs $x_1, x_2, \ldots, x_n$ to a neuron, each input x_i is weighted by a corresponding weight w_i and then a bias term b is added at the end. This can be represented as a single variable y. [11] An artificial neuron can be mathematically represented as:

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where:

- x_i represents the input signals,
- w_i denotes the corresponding weights,
- b is the bias term
- φ is the activation function.

The combination of weights, bias and activation function allows each neuron to perform a transformation on its inputs, which is fundamental to the learning process. [8]

There are different ways in which the neurons can process inputs::

1. **They can perform a scalar product**
   One way to handle the inputs is to use the scalar product of the inputs and the weights. This can be described by:

$$h(s) = \langle w, s \rangle$$

2. **They can calculate a distance**
   Another way is that neurons can calculate the distance between an input signal s and w. Therefore, for some norm $\|\cdot\|$: [21]

$$h(s) = \|w - s\|$$

These are the fundamental types of neurons, and they can be used in combination with one another across the various layers of a network.

2.4 Layers

In neural networks, layers are structured groups of neurons that process data at various levels. A neural network is made up of L layers, denoted as $l = 1, 2, 3, \ldots, L$, where layer 1 is the **input layer** that receives data, and layer L is the **output layer** that produces the final output, e.g. a prediction. The remaining layers, $2 \leq l \leq L - 1$, are called **hidden layers** because their outputs are the inner workings of the network, which help it to learn and reach the correct result. [22]

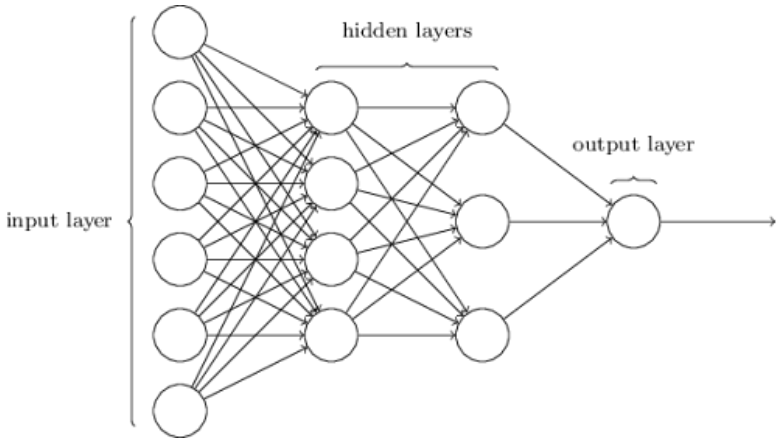


FIGURE 2.2: A neural network with an input layer, two hidden layers and a single output neuron [23]

The **input layer** is the first layer that receives the data. Each neuron in this layer represents an item of the input data. If the input vector is $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, the input layer consists of n neurons where each neuron directly corresponds to an element x_i of the input vector.

Hidden layers are located between the input layer and the output layer, where most of the network's computations occur. Each hidden layer applies transformations to the input data combinations of linear or non-linear operations. For a hidden layer with m neurons, the output of the j-th neuron can be expressed as:

$$h_j = \varphi\left(\sum_{i=1}^{n} w_{ij}x_i + b_j\right)$$

, where w_{ij} are the weights connecting the i-th input to the j-th neuron in the hidden layer, b_j is the bias term for the j-th neuron, and φ is the activation function. [24]

If there are multiple hidden layers, the output from one layer becomes the input to the next. For example, if a network has two hidden layers, the output of the first hidden layer is $\mathbf{h}^1 = [h_1^1, h_2^1, \ldots, h_m^1]$. The input to the second hidden layer is this output, and its neurons produce:

$$h_k^2 = \varphi\left(\sum_{j=1}^{m} w_{jk}^2 h_j^1 + b_k^2\right)$$

, where w_{jk}^2 are the weights connecting neurons from the first hidden layer to the second, b_k^2 are the biases, and h_k^2 are the outputs of the second hidden layer. [8]

The **output layer** produces the final prediction of the neural network. It can produce various types of outputs depending on the task it has been given. For classification tasks, the output layer often produces probabilities for each class, represented by different output neurons. [16] For regression tasks, it generates continuous values e.g. for a network used to predict house prices, the output would be the price of the house. For more complex tasks such as image generation or language translation, the output layer may contain vectors or sequences.

2.5 Weights and Biases

Weights and biases are parameters in neural networks that are adjusted during the training process to minimise the error between predicted and actual outputs.

Weights (w) represent the strength of the connection between neurons in different layers. For a given neuron, the input is a weighted sum of the outputs from the previous layer. These can be represented in a matrix.

Biases (b) are additional parameters that allow the model to shift the activation function to the left or right. This can help the network better fit the data

received. These can be represented as a vector:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

2.6 The Activation Function

Activation functions are crucial components in neural networks that introduce non-linearity into the model. They allow it to learn and represent complex patterns. They work by transforming the input signal of a neuron, allowing the network to capture patterns within the data. Without activation functions, neural networks would only be able to model linear relationships, greatly reducing their ability to solve real-world problems. [8] There are a wide variety of activation functions, some less commonly used than others.

One example of an activation function is the **Heaviside** function [25], given by:

$$\mathcal{H}(x) := \mathbb{1}_{x\geq 0}(x) := \begin{cases} 1, & if\, x \geq 0, \\ 0, & otherwise \end{cases}$$

A neuron that uses the Heaviside function as its activation function is known as a **perceptron**.

The Heaviside function is an example of a **scalar** or **pointwise** activation function. Let $\sigma : \mathbb{R} \to \mathbb{R}$ then we allow

$$\sigma\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}\right) \equiv \begin{bmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_n) \end{bmatrix}.$$

Here are some commonly used activation functions [26], as shown in the figure below.

- **Rectified Linear Unit** (ReLU) is one of the most used in modern neural networks:
$$\sigma(\lambda) = (\lambda) := \max\{0, \lambda\}$$
- **Sigmoid** also called the logistic sigmoid function:
$$\sigma(\lambda) := \frac{1}{1 + e^{-\lambda}}.$$
The sigmoid function was used as an activation function in early neural networks. This is the reason why activation functions are often labelled with a σ in general.

- **Hyperbolic tangent**- this is very similar to the sigmoid function:

$$\tanh(\lambda) := \frac{e^{\lambda} - e^{-\lambda}}{e^{\lambda} + e^{-\lambda}}.$$

- **Swish**- a more recent choice of activation function that is essentially a smooth variant of the ReLU. It is given by the input multiplied by the sigmoid function:

$$\text{swish}_{\text{fi}}(\lambda) := \lambda\, \sigma(\beta\lambda) := \frac{\lambda}{1 + e^{-\beta\lambda}},$$

  where $\beta > 0$. The β parameter is usually set as 1; however, it could be treated as a trainable parameter if required.

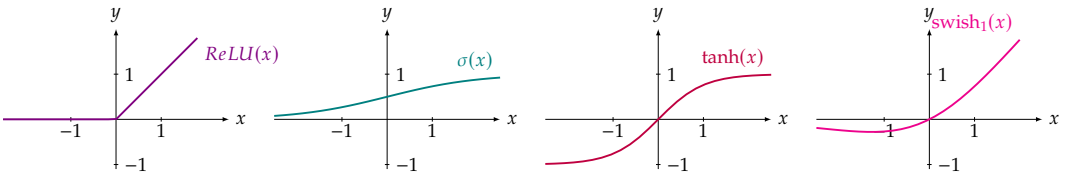


FIGURE 2.3: [19] From left to right: the rectified linear unit, the sigmoid, the hyperbolic tangent and the swish function with $\beta = 1$.

All the previous examples of activation functions are deterministic, but **stochastic** activation functions are also used. **Dropout** is a stochastic function that is sometimes used during the training process but is removed once the training is finished. It works by randomly setting individual values of a signal to zero with probability p:

$$\left(\text{dropout}_p(x)\right)_i := \begin{cases} 0 & \text{with probability } p, \\ x_i & \text{with probability } 1 - p. \end{cases}$$

All the activation functions above are fixed functions (swish and dropout have a parameter, but it is usually fixed to a chosen value) This means that the trainable parameters of a neural network are usually just the weights and biases. [8]

2.7 The Cost Function

The cost function, also known as the loss function, is a crucial component in training neural networks. It quantifies the distance between the network's predicted output and the actual target output. Reducing this distance allows the network to improve its accuracy and performance.

2.7.1 Mean Squared Error

[27] For a dataset with N training examples, where $\mathbf{x}^{(i)}$ represents the input vector and $y^{(i)}$ represents the corresponding target output for the i-th example, the cost function calculates the individual losses over all the examples. The most commonly used cost function is the Mean Squared Error (MSE), which calculates the average squared difference between the actual and predicted values. Monitoring and minimising the MSE during the training process helps determine the optimal conditions for the network. It is given as follows:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

where $\hat{y}^{(i)}$ is the predicted output for the i-th neuron, calculated as:

$$\hat{y}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$$

The MSE of an Estimator

[28] The MSE of an estimator $\hat{\theta}$ with respect to a parameter θ is defined as:

$$\text{MSE}(\hat{\theta}) = \mathbb{E}_\theta\left[\left(\hat{\theta} - \theta\right)^2\right].$$

It can also be written as the sum of the variance of the estimator and the bias squared. This is another useful way to calculate the MSE and shows that if the estimator is unbiased, the MSE is equal to the variance.

$$\text{MSE}(\hat{\theta}) = \text{Var}(\hat{\theta}) + \text{Bias}(\hat{\theta})^2$$

Proof of variance and bias relationship To prove this, we can use the formula that states for a random variable X,

$$\mathbb{E}(X^2) = \text{Var}(X) + (\mathbb{E}(X))^2.$$

By substituting X with $\hat{\theta} - \theta$, we have:

$$\begin{aligned}\text{MSE}(\hat{\theta}) &= \mathbb{E}[(\hat{\theta} - \theta)^2] \\ &= \text{Var}(\hat{\theta} - \theta) + (\mathbb{E}[\hat{\theta} - \theta])^2 \\ &= \text{Var}(\hat{\theta}) + \text{Bias}(\hat{\theta}, \theta)^2\end{aligned}$$

Understanding the relationship between MSE, variance and bias allows for the selection of an appropriate neural network architecture. If the MSE is high due to a high variance, regularisation techniques can be applied to reduce it. It can also be reduced by reducing bias through increasing the complexity of

the network, e.g. adding more neurons or layers. [16]

There are various different cost functions that can be used by different neural networks. For example, classification tasks often use the cross-entropy loss function, which measures the difference between the discovered probability distribution of a classification model and the network's own predicted values. [29]

Chapter 3

Training Neural Networks

> "Artificial Intelligence, deep learning, machine learning — whatever you're doing if you don't understand it — learn it. Because otherwise, you're going to be a dinosaur within 3 years."
>
> *Mark Cuban*

3.1 Forward Propagation

Forward propagation (or a forward pass) refers to the calculation of intermediate variables for the neural network starting from the input layer to the output layer. Inputs are passed to the network and is fed in the forward direction through the neurons, to generate an output. Consider the network shown below.

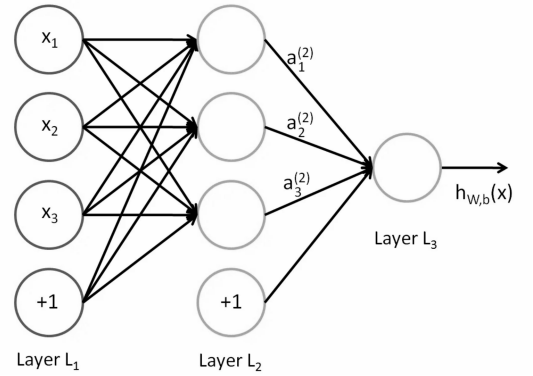


FIGURE 3.1: A neural network with three layers [30]

In the above picture, the three different layers of the network are shown, Each of the connections, shown as lines between the neurons, has a corresponding weight. In the following equations, the weight of the corresponding link will be denoted as $w_{ij}^{(1)}$ where i is the number of the node in layer $1+1$ and j is the number of the node in layer 1 e.g. the weight of the link between the first neuron of the first layer and the second neuron of the second later would be denoted as $w_{21}^{(1)}$. [8] The network also has a bias of 1 connected to all of the neurons in the next layer. The bias attached to each neuron will be denoted

as $b_i^{(1)}$ where i is the number of the node in layer $1+1$. [31]

The following system of equations can be used represent the above network:

$$\begin{aligned} h_1^{(2)} &= \varphi(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}) \\ h_2^{(2)} &= \varphi(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}) \\ h_3^{(2)} &= \varphi(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}) \\ h_{W,b}(x) = h_1^{(3)} &= \varphi(w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + w_{13}^{(2)}h_3^{(2)} + b_1^{(2)}) \end{aligned}$$

The first three equations are the outputs of the three neurons in the second layer. The last equation calculates the final output, taking the outputs of the second layer as inputs into the third layer. [32] These equations can be simplified as shown below, with $z_i^{(1)}$ representing the sum of inputs for neuron i in layer 1, including the bias:

$$z_1^{(2)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)} = \sum_{j=1}^{n} w_{ij}^{(1)}x_i + b_i^{(1)}$$

, where n is the number of nodes in the previous layer. The equations can be further simplified:

$$\begin{aligned} h^{(2)} &= \varphi(z^{(2)}) = \varphi(W^{(1)}x + b^{(1)}) \\ h_{W,b}(x) = h^{(3)} &= \varphi(z^{(3)}) = \varphi(W^{(2)}h^{(2)} + b^{(2)}) \end{aligned}$$

Here is what the general form for forward propagation looks like:

$$h^{(l+1)} = \varphi(z^{(l+1)}) = \varphi(W^{(l)}h^{(l)} + b^{(l)})$$

This shows how the output of layer l becomes the input for the layer $l+1$. These can also be represented as matrices and vectors, as shown below:

$$
\begin{aligned}
z^{(2)} &= \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix} \\
&= \begin{pmatrix} w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 \\ w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 \\ w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3 \end{pmatrix} + \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix} \\
&= \begin{pmatrix} w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)} \\ w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 + b_2^{(1)} \\ w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3 + b_3^{(1)} \end{pmatrix}
\end{aligned}
$$

This shows how neural networks use matrices and vectors to handle computations involved with forward propagation efficiently, especially when dealing with large datasets. Modern deep learning frameworks (like TensorFlow and PyTorch) are optimised for matrix operations on GPUs, further enhancing computational efficiency. [33]

3.2 Backpropagation

Backpropagation is a fundamental algorithm used for training neural networks. It involves propagating the error backwards through the network to update the weights and biases so that the cost function can be minimised. Backpropagation is simply a practical application of the chain rule and uses the idea that the derivative of a function can be calculated by working back through the layers of the network. [8]

In order to find the most suitable weights and biases for the neurons, we need to know how the cost function changes with respect to them. The chain rule can be applied to dine the derivative of the cost function with respect to the weight as shown:

$$
\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}
$$

, where C is the cost function, and z is the input to the neuron. [2] Now we need to find the following gradients:

$$
\frac{\partial C}{\partial \hat{y}} = ? \quad \frac{\partial \hat{y}}{\partial z} = ? \quad \frac{\partial z}{\partial w_1} = ?
$$

Starting with the gradient of the cost function (C) with respect to the predicted value ($\hat{y}$):

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = 2 \times \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

Let $y = [y_1, y_2, \ldots, y_n]$ and $\hat{y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n]$ be the row vectors of actual and predicted values. The above equation can now be simplified to [34]:

$$\frac{\partial C}{\partial \hat{y}} = \frac{2}{n} \times \text{sum}(\hat{y}_i - y_i)$$

To find the gradient of the predicted value ($\hat{y}$) with respect to z (for this example, we will be using the sigmoid activation function):

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial z} &= \frac{\partial}{\partial z} \sigma(z) \\
&= \frac{\partial}{\partial z} \left(\frac{1}{1 + e^{-z}} \right) \\
&= \frac{e^{-z}}{(1 + e^{-z})^2} \\
&= \frac{1}{(1 + e^{-z})} \times \frac{e^{-z}}{(1 + e^{-z})} \\
&= \frac{1}{(1 + e^{-z})} \times \left(1 - \frac{1}{(1 + e^{-z})} \right) \\
&= \sigma(z) \times (1 - \sigma(z))
\end{aligned}$$

The gradient of z with respect to the weight (w_i) is

$$\begin{aligned}
\frac{\partial z}{\partial w_i} &= \frac{\partial}{\partial w_i}(z) \\
&= \frac{\partial}{\partial w_i} \left(\sum_{i=1}^{n} (x_i \cdot w_i + b) \right) \\
&= x_i
\end{aligned}$$

Therefore, we get:

$$\frac{\partial C}{\partial w_i} = \frac{2}{n} \times \text{sum}(\hat{y}_i - y_i) \times \sigma(z) \times (1 - \sigma(z)) \times x_i$$

For the derivative of the cost function with respect to the bias (b), the same can be applied, but since the bias is a constant term, the derivative of z with respect to b is 1 and so it can be written as [34]:

$$\frac{\partial C}{\partial b} = \frac{2}{n} \times \text{sum}(\hat{y}_i - y_i) \times \sigma(z) \times (1 - \sigma(z))$$

3.2.1 Optimisation

Optimisation involves finding the set of parameters (weights and biases) to minimise the cost function. The most common optimisation algorithm is gradient descent and its variants. [8]

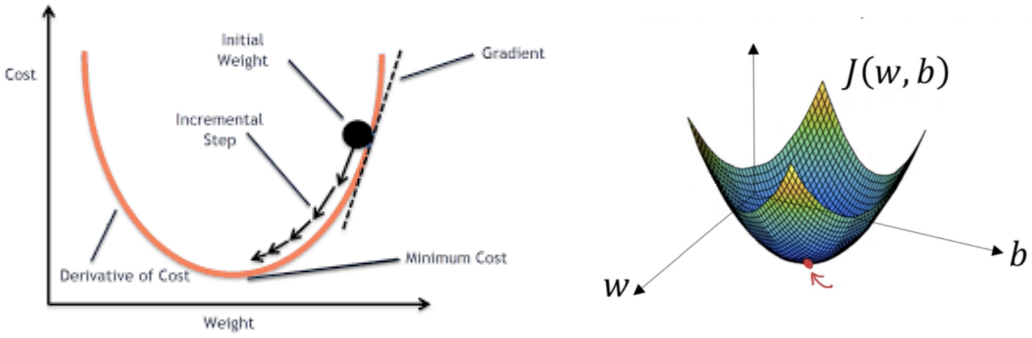
Gradient Descent: The weights and bias are updated as follows, and the backpropagation and gradient descent are repeated until convergence. [34]

$$w_i = w_i - \left(\alpha \times \frac{\partial C}{\partial w_i}\right)$$

$$b = b - \left(\alpha \times \frac{\partial C}{\partial b}\right)$$

, where α is the learning rate- a scalar value that determines how much the parameters change in each iteration of backpropagation. A high learning rate means that the weights change faster and lead to faster convergence; however, it may overshoot the optimal point, making it disadvantageous. [2]

By repeatedly applying these updates to the parameters, the algorithm converges to the minimum of the cost function, leading to the optimal conditions. [16] This ensures the network reduces its error as much as possible, making it very accurate.



(A) [35] (B) [36]

FIGURE 3.2: A visualisation of gradient descent

3.3 Initialisation

Proper initialisation of neural network weights is crucial for effective training. The choice of initialisation strategy can significantly impact the speed of convergence to the minimum cost and the final performance of the model.

If the initial weights are too small or too large, the gradients during backpropagation can become very small or grow exponentially, so learning may

be slow or not happen at all. Typically, weights are initialised from a Gaussian (normal) distribution or a uniform distribution [8]:

Gaussian Distribution : $w_{ij} \sim \mathcal{N}(0, \sigma^2)$ **Uniform Distribution** : $w_{ij} \sim \mathcal{U}(-a, a)$

, where σ and a are chosen to control the scale of the weights.

3.3.1 Xavier Initialisation

One example of an initialisation method is the Xavier initialisation. [2] This is designed to keep the variance of the weights approximately constant across layers. This method is suitable for activation functions like sigmoid and tanh, which can suffer from vanishing gradients if not properly initialised. For a neuron with n_{in} incoming connections and n_{out} outgoing connections, the weights are initialised as:

- **Uniform Xavier Initialisation**:

$$w_{ij} \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right)$$

- **Normal Xavier Initialisation**:

$$w_{ij} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right)$$

Practical Considerations: The choice of initialisation method is usually dependent on the type of activation function used. Different activation functions transform inputs differently, affecting the flow of signals (activations) and gradients (during backpropagation). Therefore, it is crucial that the correct one is selected. Biases are typically initialised to zero or small constants, ensuring all neurons behave symmetrically at the beginning of the learning stage [37].

Chapter 4

Conclusion

4.1 An Overview

The study of neural networks has proven to be an important aspect of machine learning. In this project, we have explored the biological inspiration of neural networks and their current applications, as well as the mathematical principles that govern their functioning and learning processes.

In the introduction, we explored the historical development of neural networks, from the initial model of the perceptron to the advanced multi-layer perceptrons. We then laid out the basics of neural networks, describing their structure and the roles of artificial neurons, layers, weights, and biases. These are the basic elements of neural networks, which are affected by mathematical ideas that make them work and perform well. We also discussed the significance of activation and cost functions, which are important components that determine how neural networks learn from data. The Universal Approximation Theorem highlighted the theoretical possibility of neural networks to approximate any function, provided the network is complex enough.

4.2 Training a Neural Network

The project then focused on neural network training. We explored forward propagation and backpropagation, the fundamental processes involved in training them. Forward propagation involves calculating activations across layers, while backpropagation uses the chain rule of calculus to propagate errors backwards, updating the weights to minimise the cost function.

The optimisation techniques discussed, such as gradient descent, are critical for efficient learning. Initialisation strategies were also explored to understand their impact on the training process. This ensures that the weights are set to prevent the common issues of vanishing and exploding gradients.

4.3 Final Thoughts

Mathematics plays a pivotal role in every aspect of neural networks, including their architecture and their real-world application. Therefore, the mathematical foundation will remain one of the key areas of research and development as the field continues to evolve, driving further applications and advancements.

In conclusion, exploring the mathematics of neural networks not only enriches our knowledge about these powerful tools but also provides the necessary knowledge to develop new models and improve upon existing ones. This project has given a broad overview of the mathematical concepts that underpin neural networks and make them a crucial tool in machine learning, paving the way for future research.

Bibliography

[1] R. Tibshirani T. Hastie and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed., Springer, 2009.

[2] M. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[3] M. A. Arbib. *Brains, Machines, and Mathematics*. Springer, 1964.

[4] F. Rosenblatt. "The Perceptron: A Perceiving and Recognizing Automaton". In: (1958).

[5] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[6] Geoffrey E. Hinton David E. Rumelhart and Ronald J. Williams. "Learning representations by back-propagating errors". In: (1986).

[7] D. Myszewski C. Clabaugh and J. Pang. *Neural Networks - History*. URL: cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html.

[8] Y. Bengio I. Goodfellow and A. Courville. *Deep Learning*. MIT Press, 2016.

[9] I. Sutskever by A. Krizhevsky and G. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: (2012).

[10] D. Silver. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: (2016).

[11] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer, 2018.

[12] G. Litjens et al. "A Survey on Deep Learning in Medical Image Analysis". In: *Medical Image Analysis* 42 (2017), pp. 60–88.

[13] IEEE Transactions on Intelligent Transportation Systems. "A Comprehensive Review on Autonomous Vehicle Perception using Deep Learning". In: (2020).

[14] D. M. West. *How Artificial Intelligence Is Transforming the Financial Industry*. Brookings, 2018.

[15] C. Chio and D. Freeman. *Machine Learning and Security: Protecting Systems with Data and Algorithms*. O'Reilly Media, 2018.

[16] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] Scientific Figure on ResearchGate. *Artificial neural network (ann) model with multilayer feedforward networks architecture*. URL: www.researchgate.net/figure/Neural-network-biology_fig1_332302093.

[18] Scientific Figure on ResearchGate. *Biologically inspired Neural Network*. URL: https://www.researchgate.net/figure/Biologically-inspired-Neural-Network-29_fig2_330276665.

[19] B. M.N. Smets. "Mathematics of Neural Networks (Lecture Notes Graduate Course)". In: *arXiv preprint arXiv:2403.04807* (2024).

[20] G. Cybenko. "Approximation by Superpositions of a Sigmoidal Function". In: (1989).

[21] A. Lichtner-Bajjaoui. "A Mathematical Introduction to Neural Networks". In: (2020).

[22] Y. Bengio Y. LeCun and G. Hinton. "Deep Learning". In: *Nature* 521.10 (2015), pp. 436–444.

[23] M. Nielsen. *Neural Networks and Deep Learning*. URL: images.app.goo.gl/a5gV9j1NwtGeMdYv7.

[24] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Pearson, 1999.

[25] Gabor Melli's Research Knowledge Base. *Heaviside Step Activation Function*. URL: https://www.gabormelli.com/RKB/Heaviside_Step_Activation_Function.

[26] P. Baheti. *Activation Functions in Neural Networks*. URL: www.v7labs.com/blog/neural-networks-activation-functions.

[27] H. Pishro-Nik. *Introduction to Probability, Statistics, and Random Processes*. Kappa Research, LLC, 2014.

[28] The Free Encyclopedia Wikipedia. *Mean squared error*. URL: en.wikipedia.org/wiki/Mean_squared_error.

[29] K. Pykes. *Cross-Entropy Loss Function in Machine Learning: Enhancing Model Accuracy*. URL: www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning.

[30] Scientific Figure on ResearchGate. *A layered network with three layers*. URL: www.researchgate.net/figure/A-layered-network-with-three-layers_fig2_324929594.

[31] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Microsoft Press, 2000.

[32] N. Raychev. "Mathematical foundations of neural networks. Implementing a perceptron from scratch". In: (2020).

[33] F. Cholletn. *Deep Learning with Python*. Manning Publications, 2018.

[34] D. K. Saravanan. *A Gentle Introduction To Math Behind Neural Networks*. URL: www.towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba.

[35] Analytics Vidhya. *Gradient Descent Algorithm: How Does it Work in Machine Learning?* URL: www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/.

[36] Niklas Donges. *Gradient Descent in Machine Learning: A Basic Introduction*. URL: www.builtin.com/data-science/gradient-descent.

[37] S. Ruder. "An overview of gradient descent optimization algorithms". In: (2016).