Artificial Neural Networks

Strengths and Weaknesses

MATH4069 Statistical Machine Learning

Group project report

2022/23

School of Mathematical Sciences

University of Nottingham

Group D:

Joshua Ranson

Jakub Niekrasz

Ahmadreza Omidvar

We have read and understood the School and University guidelines on plagiarism. We confirm that this work is our own, apart from the acknowledged references.

Abstract

Artificial Neural Networks (ANN) are a class of Machine Learning algorithms that attempt to imitate the function of neurons in the biological brain. Taking input data through a system of interconnected nodes with different weightings for each input and neuron. This process can produce simple results from complex input data. ANNs have flexibility to be applied to different complex issues, such as Facial Recognition, Medical Diagnosis and Automated Trading Strategies. In this report we discuss the history and development of neural networks, the mathematics that underpin their operation, the application of neural networks to toy and real-world data as well as the strengths and weaknesses of neural networks as a machine learning algorithm. In this report we tested the performance of a multi-layer feed forward neural network against PCA and LDA on a toy dataset; performing a classification task and found that due to the toy datasets small size, the neural network was unable to perform better than the traditional machine learning algorithms. However, when applied to the real-world data which contained 10,000 data points and 10 features, our neural network was able to surpass PCA.

Contents

1 Introduction

Artificial Neural Networks (ANNs) are machine learning algorithms capable of taking in highly complex and large data sets, discovering patterns within the data and using said patterns to come up with useful output and predictions based on the data. Neural networks were inspired by neural networks of mammal brains and are structured as follows. The first component of an ANN is the input layer where the network receives data. This is followed by one or more hidden layers which contain 'neurons' that perform weighted sums and become 'activated' based on an activation function. If each neuron in a layer is connected to every neuron in the previous layer, then it is called dense. If a network has two or more (definition varies) hidden layers, it is called a deep neural network. The final layer in an ANN is the output layer which gives us our result. This report aims to provide an overview of the history and development of ANNs, explain the mathematical formulation of the model and apply the model to both a toy and real-world dataset. Furthermore, we hope to determine the strengths and weaknesses of ANNs as well as the most appropriate situations in which an ANN may be deployed. We began by discussing how neural networks developed from a single M-P Neuron to the more sophisticated Mark-I Perceptron; and then to modern day networks (Section 2). In these sections we found out how problems such as training speed and the inability of contemporary networks to simulate the XOR gate have, in the past, made neural networks a sub-par machine learning algorithm for tasks such as classification of linearly separable patterns. Then we moved on to describing the mathematics behind how current networks function (Section 4). We covered; the formulation of a single layer feed-forward neural network, the significance of activation functions (why they are required, why certain functions are preferred etc.), loss functions and how they allow us to not only determine model performance but also how they facilitate model learning, the back propagation algorithm and how it is used in conjunction with the loss function to adjust weights and in turn improve model performance, and finally feature scaling and why it is important. We then trained a shallow and a deep neural network on a toy dataset and compared its performance to PCA and LDA (Section 5. We found that LDA and PCA were more appropriate to use for this data set because of its small size. Afterwards we fitted 3

different networks to a dataset containing bank customer data. The aim of this was to use a neural network to predict whether a person was likely to close their bank account based on 10 data variables (bank balance for example). We fitted 3 networks, a shallow network, an intermediate network, and a deep network and illustrated how a deeper network may over-fit data when the number of features isn't very large . We then compared our network to PCA and concluded that it was more appropriate. Furthermore, we employed hyper-parameter tuning and drop-out to showcase how the performance of an ANN can be improved. Finally, we discussed issues in training as well as the importance of proper initialisation where we found that initializing weights incorrectly can cause exploding or vanishing gradients which prevents the network from functioning as intended.

2 History of Neural Networks

2.1 Humble Beginnings

The story of artificial neural networks (ANNs) began in 1943 with the foundational research of Warren McCulloch and Walter Pitts. They developed a computational model based on physical connected circuits which made use of "threshold logic"; mimicking the function of neurons in the brain to simulate complex behaviours such as learning [1]. Here, "threshold logic" refers to a variant of diode/transistor logic circuits (mainly used in high noise applications) where threshold values are compared to the input of a logic gate to determine if the input into the neuron will be 1 or 0 [2]. The model proposed by the two researchers is commonly referred to as "The McCulloch-Pitts (M-P) Neuron" or as it would be more commonly referred to as in the future, the "Perceptron", and its function is akin to that of the modern logic gate [3] [4]. Though the definitions of the M-P neuron and perceptron differ, M-P neuron accept inputs of either 0 or 1 and perceptrons accept inputs $\in$ (0,1). The way the M-P neuron model determines an output is simple:

$$y = \begin{cases} 1, & \sum_{i=0}^{n} x_i \geq \theta \\ 0, & \text{otherwise} \end{cases}$$

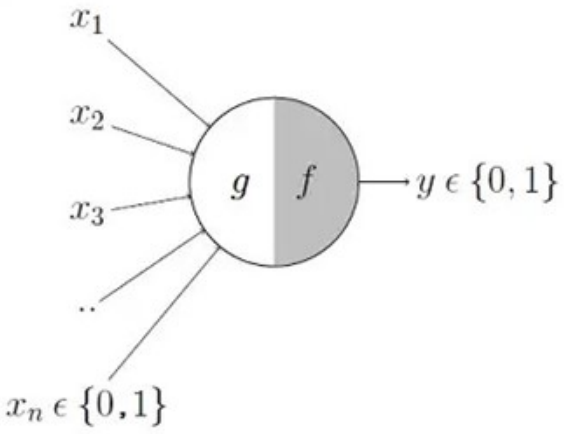
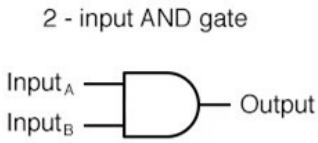


Figure 1: Diagram of an M-P Neuron [3]

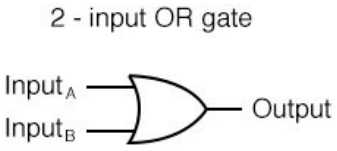


A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2: Two-Input AND Gate [5]

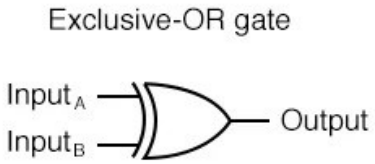
In Figure 1 we see a more generalised diagram of the M-P Neuron model. Here g represents the aggregation function, which in our case is the sum of the inputs $(\sum_{i=0}^{n} x_i)$ and f represents the activation function, which determines if the output should be 1 or 0. In our case, $y = f(g(x)) = 1$ if $g(x) \geq \theta$ and $y = f(g(x)) = 0$ if $g(x) < \theta$. By taking only 2 inputs x_1, x_2 and setting $\theta = 2$ we essentially create what is most referred to as a two-input AND gate, a diagram of which can be seen in Fig 2 below. It is called an AND gate because the output is only equal to 1 if both x_1 AND x_2 are also equal to 1 [3].

Furthermore, by setting $\theta = 1$ we are able to reproduce an OR gate (Fig 3), which is called an OR gate because the output is equal to 1 if $x_1 = 1$ OR $x_2 = 1$ OR $x_1 = x_2 = 1$. However, there is one major flaw with the M-P model, which would cause huge problems



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Figure 3: Two-Input OR Gate [5]



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4: Two-Input XOR Gate [5]

in the further development of neural networks during the 1950s and 1960s. Due to the fact that the model is a linear classifier, it is unable to reproduce the XOR or Exclusive-OR gate (Fig 4). This issue will be discussed in more detail when we begin covering basic perceptrons and the 1969 research by Minsky and Papert [6].

2.2 Hebbian Learning

During the early 1940s, advancements into neuroscientific theory were made by Donald Hebb which allowed the proposition of a hypothesis based on neuroplasticity (the ability of the brain to form and reorganize connections, especially as a response to learning or injury - paraphrased from Oxford Dictionary); Hebbian Learning [7]. Hebbian Learning attempts to explain how neurons in the brain adapt during the learning process, and thus the theory claims that increases in synaptic efficiency ("the strength of communication between neurons" [8]) arise from repeated stimulation of postsynaptic cells by presynaptic cells. In short, "cells that fire together, wire together" [9]. Many regard Hebbian theory as the neuro-scientific basis of unsupervised learning [1]. Researchers Farley and Clark (1954) first applied the ideas of Hebbian Learning to simulate a Hebbian network (neural network based on Hebbian theory) using computational machines, which at the time were referred to as "calculators" [1]. However, a far more influential advancement in the field of neural network research would be made just four years later in 1958 at the Cornell Aeronautical Laboratory; the creation of the Mark-I Perceptron, the first physical

implementation of the M-P neuron model proposed in 1943 by McCulloch and Pitts [6].

2.3 A Glimpse Into the Future: How Rosenblatt 'solved AI'

Using the M-P neuron model and with funding from the US Office of Naval Research, Frank Rosenblatt was able to construct a machine called the Mark-I Perceptron (Fig 5) which was intended for the purpose of image recognition [6]. The machine functioned slightly differently than the example of an M-P neuron outlined previously, though the perceptron was still a binary classifier. A binary classifier is an algorithm that can classify data into two groups. The machine consisted of a 20 by 20 array of cadmium sulphide photocells/photosensors which acted as a camera that could produce a 400-pixel image. A value indicating light intensity for each pixel $\in (0,1)$ would then be fed into the perceptron, where the value of each pixel would be multiplied by a weight which was encoded using potentiometers (dial, knob, variable resistor etc.) [6]. The weighted sum would then be fed into a step function which would in turn classify the captured image into one of two predetermined groups. Updates to weights during the learning process were made using electric motors (thankfully not by hand).

Even though this invention was an overwhelmingly important steppingstone for the field of machine learning and AI, Rosenblatt did not manage to showcase the machine without causing controversy. Based on his statements during a 1958 press conference, The New York Times reported that the perceptron was "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence" [6]. From this quote one might conclude that both Rosenblatt and the Navy were wholeheartedly convinced that they had solved the problem of creating not only artificial intelligence but also artificial consciousness using perceptrons, which makes it exceedingly clear why these statements caused such an uproar at the time. Although the perceptron demonstrated great potential, in under a decade it was proven that they were only capable of classifying linearly separable patterns, thus their usefulness was severely limited. Nevertheless, the perceptron still forms the basis of many modern feed-forward neural networks, though current methods employ a multilayer approach rather than a single layer one. A multilayer perceptron can overcome some of the drawbacks

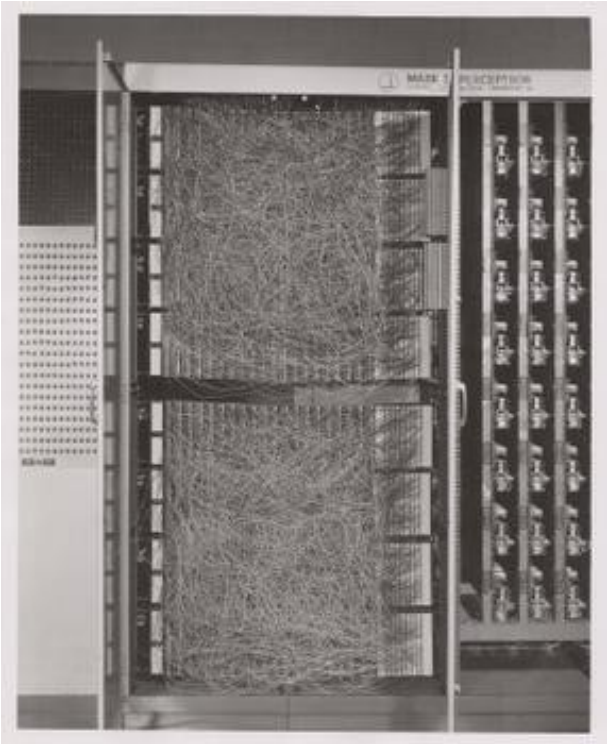
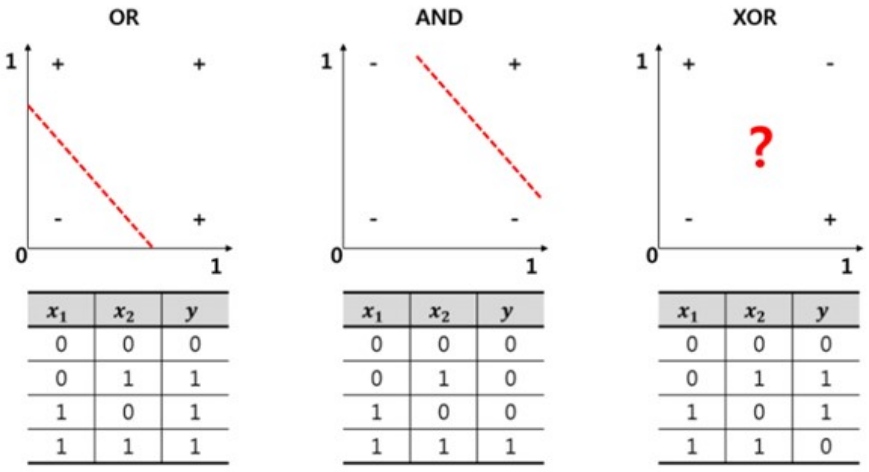


Figure 5: The Mark-I Perceptron [6]

of Rosenblatt's single layer application for example, with a multilayer perceptron it is possible for the network to learn the XOR function [6].

2.4 Stanford's Love for Acronyms

In 1959 the first commercially viable neural networks were implemented to help filter out noise and echoes in phone lines. Bernard Widrow and Marcian Hoff of Stanford developed both the "ADALINE" and "MADALINE" models, the names of these models being acronyms for Multiple ADAptive LINear Elements. ADALINE was developed to detect patterns in a binary bit stream, such as that of a phone line, to predict the next bit in the sequence. While "MADALINE was the first neural network applied to a real world problem" [10]. Even though the MADALINE system seems comparatively ancient when looking at modern neural networks, it is still used extensively in commercial applications to this day [10]. Though advancements in the field of ANNs were seemingly being made on a continuous basis during the 1940s and 50s, the period of unprecedented growth was about to come to a screeching halt.



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 6: OR, AND and XOR Gate Example Diagrams [12]

2.5 The End of An Era: AI "Winters"

The term "AI Winter", coined as an analogy of a nuclear winter, refers to a period where the interest and funding of AI research was greatly diminished. The first AI Winter would begin with the Automatic Language Processing Advisory Committee (ALPAC) report of 1966, where the committee concluded that machine translation was "more expensive, less accurate and slower than human translation". After this report was published, funding would be pulled by the National Research Council (NRC) and further research into machine translation would sadly be put on hold until the modern day, where we now have multitudes of different machine translation-based programs such as Google Translate [11]. Another major event which cemented the inevitable decline in public and scientific interest in the field of AI was the publishing of the book "Perceptrons" by Marvin Minsky and Seymour Papert in 1969 (as mentioned previously). The main takeaway from this book was that contemporary neural network computational machines were insufficiently powerful for the task of simulating large neural networks and more importantly, that these machines (namely single layer perceptrons) were incapable of reproducing the exclusive-OR or XOR circuit. The reason behind why this was not possible can be illustrated with a simple diagram, in this diagram assume we are working with a single layer 2-input perceptron with both weights equal to 1 (i.e., the simple logic gate example outlined previously) [11] [6].

As we can see in Figure 6, the single layer perceptron is able to classify the inputs correctly in the case of the OR and AND gates because the inputs can be separated into +'s and −'s using a single straight line however, this is not possible in the case of the XOR gate. There two lines would be needed therefore a multilayer perceptron would be required, which at the time was infeasible due to hardware limitations [12]. Interest in neural networks was rekindled briefly in 1974 by Werbos' rediscovery of the back-propagation algorithm [13] which would finally enable multi-layered networks to be trained using the hardware that was available at the time. The back-propagation algorithm was discovered and rediscovered numerous times during the 1960s however, the first published appearance within the context of optimizing multi-layered neural networks appears to be from the 1969 work of A. E. Bryson and Y. C. Ho where the algorithm was described as a "multi-stage dynamic system optimization method" [4]. However, it was not until 1985 that the back-propagation algorithm would be noticed by anyone. In 1985 Parker published a report on the algorithm which in turn led to the algorithms "re-re-discovery" [9] by Rumelhart, Hinton and Williams, who published a concise and detailed formulation of the algorithm. This event helped bring about an end to the first AI winter. Despite the optimism felt after the rediscovery of the back-propagation algorithm, renewed interest was short lived and yet another AI Winter began in 1987 with the collapse of the LISP machine market [11]. LISP machines were a type of computer that was optimized specifically for the LISP programming language, which was the preferred programming language for AI [14]. Many other events occurred during the 1980s, and 1990s which would further diminish people's interest in AI in general, with the lowest point occurring somewhere in the 1990s [11]. All hope was not lost however, AI research began to gain traction once again during the 2000s and by the 2010s, neural networks were being widely implemented thanks to advances in hardware.

2.6 Resurrection: The AI Spring of the 2010s

As computers continued to become exponentially more powerful, new avenues into neural network research opened, making previously impossible problems completely feasible. For example, thanks to advancements in GPU (graphics processing unit) power, neural

network training could be accelerated using GPUs which in turn made training multi-layer perceptrons (with back-propagation) a much simpler task than it once was [11]. Currently all the FAANG companies(Facebook (Meta), Amazon, Apple, Netflix, Google (Alphabet)) employ some form of artificial intelligence in their products, and for now it seems that development into ANNs, AI and machine learning technologies will not be stopping any time soon.

3 Network Description and Visualisation

3.1 Description

Artificial neural networks (ANNs) are a subset of machine learning algorithms that (to a certain degree) mimic the behaviour of neurons in the brain. ANNs typically consist of several different elements: an input layer where our data features get passed into the network, one or more hidden layers which consist of nodes or neurons that make up the bulk of the network, and an output layer. A neuron in an ANN is essentially just a perceptron (described in 2.1) that takes a weighted sum of inputs, feeds this weighted sum into an activation function (see 4.2) and then outputs either 0 or 1 a value calculated using the activation function. With the weights for the weighted sum being adjusted using the back-propagation algorithm for example.

There are many different types of neural networks such as:

- Single Perceptrons
- Feed-Forward Neural Networks (FFNN) - neural networks where data travels only in one direction i.e. from input layer, through the hidden layers, into the output layer. These types of networks can be further classified based upon the number of hidden layers (single or multi-layered FFNN) [15].
- Multilayer Perceptrons (MLP) - a special case of the FFNN where every single node is connected to every neuron in the next layer, making the network "fully connected". MLPs as the name suggests are made up of multiple hidden layers and make use of back-propagation [15].

- Convolutional Neural Networks (CNNs) - another special case of the FFNN most commonly used for image detection. CNNs in the case of image detection, essentially split up the image into many small pieces, then calculate the convolution between the pixel data matrix of each small piece of the image with a filter matrix, reducing the dimension of the problem [16].

- Recurrent Neural Networks - often used for sequence generation e.g. text auto suggestion. These types of networks are not feed forward, they contain loops and recurrent layers. Each node in a recurrent layer stores a state or 'memory' and takes a vector of states from the previous layer as an input as well as a vector of states from its own layer (from the previous time step)[16].

- And many more...

3.2 Visualisation

Since the multilayer perceptron is the type of ANN we will be using for much of this project, we will now go into more depth on the description of this type of network. We will also look at a diagram which will help us visualise how it functions. As stated previously, MLPs are constructed from multiple hidden layers i.e., layers of nodes/neurons which are in between the input and output layers. Each neuron in a layer takes in a weighted sum of either the inputs to the network (data features) or the weighted sum of outputs of the previous layer, these weighted sums are then fed into an activation function which produces either an output which is used in the next layer or are fed through an activation layer which determines the output at the output layer. Each layer in a MLP is said to be 'dense', which means that each neuron in each layer receives an input from EVERY neuron in the previous layer - this is why MLPs are said to be fully connected. Figure 7 gives us a simple example of what a multilayer perceptron may look like. In this example network, there are 3 inputs, 2 dense layers having 5 neurons each and a single output. Since this example network has more than 1 hidden layer it may be referred to as a deep neural network, though the definition of 'deep' in the context of neural networks seems to change depending on the source. Each node in the first layer takes a weighted sum

of inputs, which it then passes through an activation function (ReLU for example, see 4.2.2) and determines if the neuron should be activated. Each neuron in the second layer then takes a weighted sum of the outputs of the first layer's neurons and these weighted sums are once again passed through an activation function to determine how much of the second layer's neurons become activated. The outputs of the second layer are then passed to yet another activation function which determines the final output. The weights used for each weighted sum are adjusted using the back-propagation algorithm. The way this algorithm functions is it attempts to find the weights which minimise the loss function. Loss functions measure how 'off' the model's predictions are in comparison to the true outputs from the training data. The weights are adjusted to minimise the lost function by computing the gradient of the loss function using the chain rule. The gradient is computed one layer at a time and is iterated backwards from the last layer to the first layer. The process continues iterating until the gradient converges. We can also take a closer look at the mathematical relationships between weights, activation functions and neurons in a multilayer perceptron (Fig 8).

Fig 8 is a visualisation of an MLP with 2 inputs, 2 dense layers (with 4 neurons each) and a single output. The diagram shows that the output of each neuron of a given layer is the dot product between the weight vector (with each neuron having a different weight vector) and the input vector plus a constant bias, all fed into an activation function. Each layer in this diagram has its own activation function; layer 1 has g_1, layer 2 has g_2 and the output layer has g_3. Layer 1 produces an output vector y_1 which is then fed into layer 2 where layer two does essentially the same thing layer 1 did with the input vector. Layer 2 then produces an output vector y_2 in the same fashion and this is then fed into the final output layer where the weighted sum (plus a bias) is fed into the final activation function. This produces a scalar output y, which is our final output.

4 Mathematics behind Neural Networks

Single hidden layer feed-forward Neural Networks are the most basic form of Neural Networks. In this section we will set out the mathematics that allows Neural Networks to create insightful analysis. For this study into Neural Networks, we will be concentrating

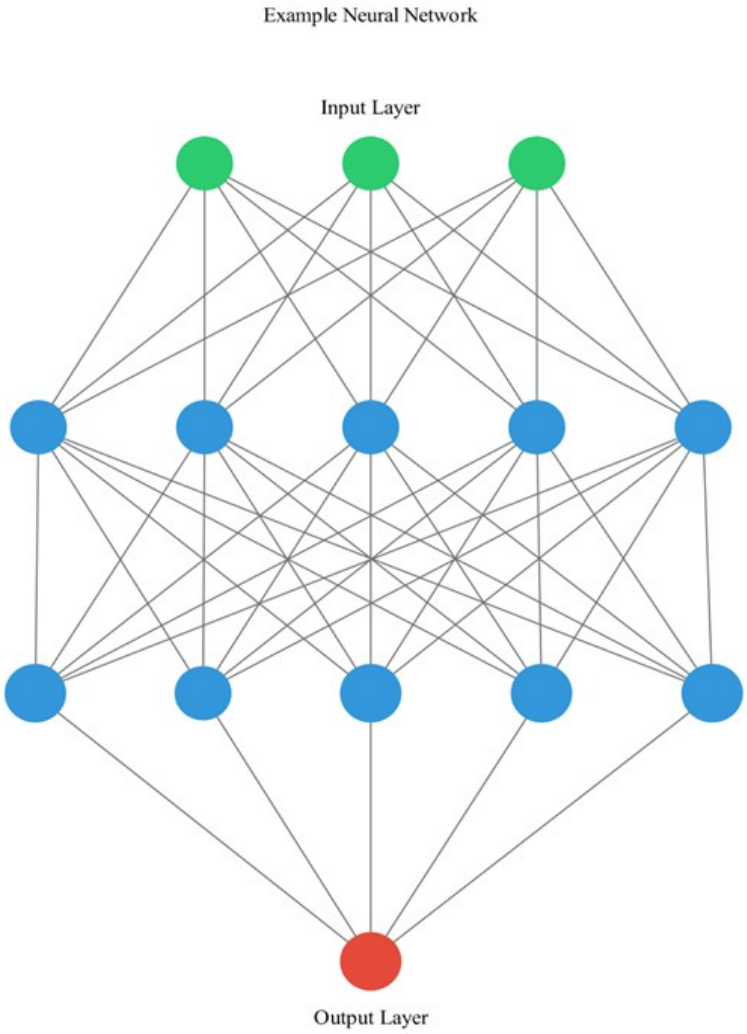


Figure 7: Diagram of a Multilayer Perceptron (Produced by authors using ANNVisualizer)

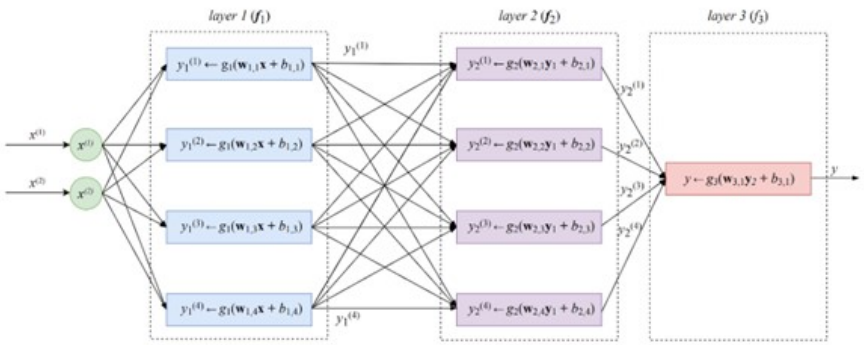


Figure 8: More Mathematically Thorough MLP Diagram [16]

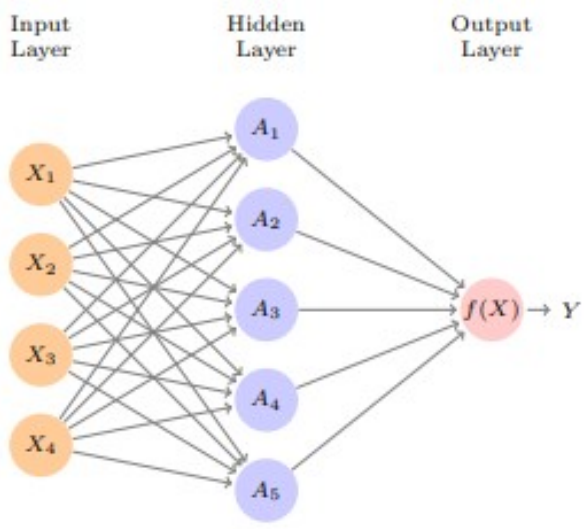


Figure 9: Single layer feed-forward neural network with format 4;5;1 [17]

on classification problems. The mathematical analysis will be looking into a two-stage classification network.

4.1 Single Layer Neural Networks

For the input layer, there are p different input variables $X = (X_1, X_2, ..., X_p)$. These inputs are fed into the hidden layer of the NN, with each individual input being connected to every node, see Figure 9. We then need to calculate a non-linear function $f(X)$ to predict the response variable Y. The non-linear function is of the form:

$$\begin{aligned} f(X) &= \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) \\ &= \beta_0 + \sum_{k=1}^{K} \beta_k g(w_k 0 + \sum_{j=1}^{p} w_{kj} X_j) \end{aligned}$$

Within the hidden layer are the activation nodes A_k, these are functions created from the inputs $X_1, X_2, ..., X_p$:

$$A_k = h_k(X) = g(w_k 0 + \sum_{j=1}^{p} w_{kj} X_j)$$

Note: $g(\cdot)$ is a non-linear activation function.

Each node within the hidden layer represents a different transformation of the original input values. The K different nodes from the hidden layer $(A_1, A_2, ..., A_k)$ then feed into the output layer to create our final output:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

Note: This is similar to a linear regression model

4.2 Activation Functions

When using NNs for classification problems, we need to estimate all the parameters in the model $\beta_0, ..., \beta_K$ and $w_1 0, ..., w_{Kp}$. These estimates are then transformed by the activation function contained within the hidden layer. Activation functions are used in NNs because they map the estimate values from $(-\infty, \infty) \rightarrow (-1, 1)$ or $(0, 1)$ depending upon the function used [18]. Activation functions serve two main purposes for NNs; to model interaction effects between input variables and to model non-linear effects. Without activation functions NNs would just be a linear regression model.

4.2.1 Tanh Function

The first activation function we are going to look at is the Tangent Hyperbolic function. This function is useful because it centres the data around 0 and has a higher gradient than the other functions we can use [19]. A higher gradient results in higher values from the activation functions and larger updates to the weights of the network. Resulting in greater progress after each epoch[1].

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

$$\mathbb{R} \rightarrow (-1, 1)$$

[1]An epoch is an iteration of a Neural Network, from input to updating weights and biases using back-propagation

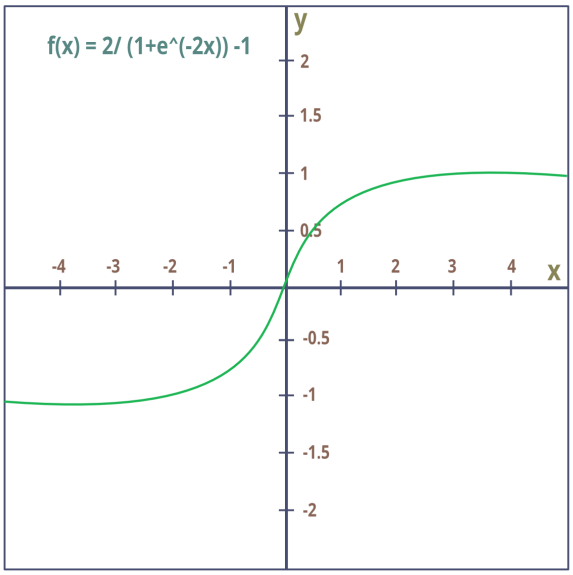


Figure 10: Tangent Hyperbolic Function [20]

4.2.2 Rectified Linear Unit

The ReLU is a non-linear function which is a combination of two linear functions, $y = 0 \in (-\infty, 0]$ and $y = x \in [0, \infty)$. As the number of neurons within the hidden layer increases we are able to more accurately model any continuous function on $\mathbb{R}^m$.

$$g(z) = (z)_+ = \begin{cases} 0, & \text{if } z < 0 \\ z, & \text{otherwise} \end{cases}$$

The ReLU function is preferred because it is more resource efficient to compute and requires a fraction of the storage space. This results in NNs using ReLU functions learning much faster than other activation functions. Theorem 4.2.2 implies that as the number of neurons increase within the hidden layer the better our NN model will be able to represent any function, linear or non-linear [21].

Theorem 1. *Denote all L^p functions on $\mathbb{R}^m$ as L^p. If the non-linear activation function $\phi(\cdot)$ is the ReLU function, then Λ_N is dense in L^p as $N \to \infty \Rightarrow lim_{N\to\infty}\Lambda_N = L^p$* [2]

[2]This theorem is too in-depth to be covered in this project but please see [22] and [23] for more information.

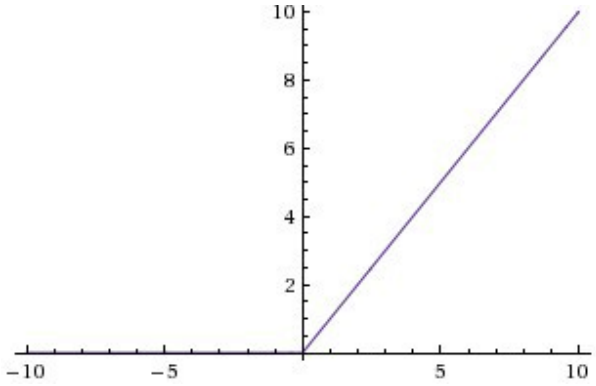


Figure 11: Rectified Linear Unit [20]

4.2.3 Sigmoid and Softmax Functions

There are several different sigmoid functions (Fig 12) we could use as an activation function, such as the Logistic function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$\mathbb{R} \to (0, 1)$$

The closer the function is to 1 the more likely the node will be activated. This function is helpful when predicting probabilities as the results are within (0,1). This function would generally be implemented in the output layer of a binary classification network (y_1 or y_2) with the following final output:

$$f(X) = \begin{cases} y_1, & \frac{1}{1+e^{-x}} \geq 0.5 \\ y_2, & \text{otherwise} \end{cases}$$

The Softmax activation function is a non-linear function that is helpful for multi-class classification problems because it's output is $\in (0, 1)$. Therefore the output is equivalent to the probability of the entry being each class:

$$Outputlayer = (Pr(f(X) = y_1), Pr(f(X) = y_2, ..., Pr(f(X) = y_n))$$

Softmax function

$$\sigma(\mathbf{z}_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\mathbb{R}^K \to (0, 1)^K$$

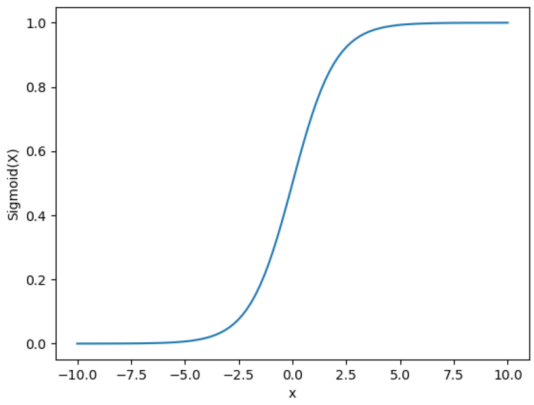


Figure 12: Sigmoid Function [20]

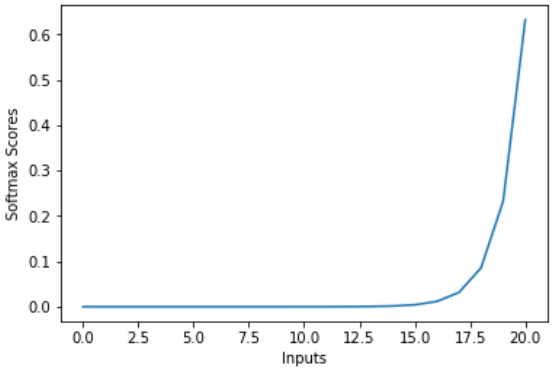


Figure 13: Softmax Function [20]

4.3 Loss Functions

Loss functions are used in the training of Neural Networks because they allow us to compare the output given by our model to the predictor variable in the training data. We then tune the hyper-parameters to minimise the following average loss function [24].

$$J(w^t, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

There are several different Loss functions we can use for NNs, the function we should use depends upon the question we are trying to model.

4.3.1 MSE and MAE

Mean Squared Error (MSE) is one of the simplest Loss functions we have available. It calculates the difference between the predicted output and the actual output. MSE is a good loss function because it facilitates gradient decent optimisation due to its convex nature and globally defined minimum.

$$MSE : L(y) = \frac{1}{n}\sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

The Mean Absolute Error (MAE) is used as an alternative to MSE because it overcomes MSE's sensitivity to outlier variables.

$$MAE : L(y) = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}|$$

Both MSE and MAE loss functions are utilised in regression problems.

4.3.2 Binary Cross-Entropy

This function is used for classification models where there are only two available outcomes. Binary Cross-Entropy works by comparing the outcome of the model, 0 or 1, with the results from the training dataset, combined with the probability of these outcomes being achieved.

$$CELoss = \frac{1}{n}\sum_{i=1}^{n} - (y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i))$$

4.3.3 Categorical Cross-Entropy

Categorical Cross-Entropy is used in classification cases where we have more than 2 outcomes available.

$$CELoss = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \cdot log(p_{ij})$$

This is the loss function that is implemented in our analysis of ANN classification problems.

4.4 Back Propagation

To allow our NN to learn we need to apply the Back Propagation algorithm to find the gradient of the loss function with respect to the parameters. To explain the process, we need to set-up a single hidden layer MLP. We need to use a design matrix $\mathbf{X}$ along with a vector of class labels $\mathbf{y}$. The hidden layer is then computed as $\mathbf{H} = max\{0, \mathbf{XW}^{(1)}\}$, with a ReLU function producing $max\{0, \mathbf{Z}\}$ element-wise. The result of this NN is un-normalised log probabilities for each class, given by $\mathbf{HW}^{(2)}$. We then use Categorical Cross Entropy (see Subsection 4.3.3) to calculate the loss function J_{MLE} [25]. We also need to add a regularisation term, resulting in total loss (4.1). This is visualised in the computational graph Figure 14.

$$J = J_{MLE} + \lambda(\sum_{i,j}(W_{i,j}^{(1)})^2 + \sum_{i,j}(W_{i,j}^{(1)})^2) \tag{4.1}$$

To train the MLP NN we need to compute $\bigtriangledown_{\mathbf{W}^{(1)}} J$ and $\bigtriangledown_{\mathbf{W}^{(2)}} J$. There are two paths to the total cost: cross-entropy and weight-decay. Firstly, weight-decay will always add $2\lambda\mathbf{W}^{(i)}$ to the gradient on $\mathbf{W}^{(i)}$, this is from the regularisation term. For cross-entropy, let $\mathbf{G}$ be the gradient of un-normalised log probabilities ($\mathbf{U}^{(2)}$). This term comes from the cross-entropy operation.

The algorithm now follows two separate branches. The first branch adds $\mathbf{H}^T\mathbf{G}$ (The hidden layer multiplied by the gradient) to the gradient on $\mathbf{W}^{(2)}$. The second branch requires several steps, firstly computing $\bigtriangledown_{\mathbf{H}} J = \mathbf{GW}^{(2)T}$ (the gradients multiplied by the hidden layer output weights), then using the ReLU operation (Subsection 4.2.2) to zero components of the gradient corresponding to negative entries in $\mathbf{U}^{(1)}$, $\mathbf{G}' = max\{0, \mathbf{G}_i\}$

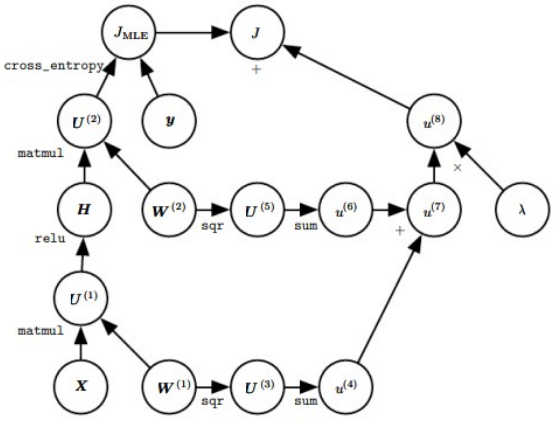


Figure 14: Computational graph used to compute loss [25]

$\forall\ i \in \mathbf{U}_i > 0$. Finally we need to add $\mathbf{X}^T\mathbf{G}'$ to the gradient on $\mathbf{W}^{(1)}$ (the weights from the input to the hidden layer). We then need to utilise the gradient decent algorithm to update the weightings [25].

4.5 Gradient Descent

In this section we are aiming to minimise equation 4.1, once we are at the global minimum for this function our model will reach its prediction optimum. We need to find $J' = 0 \Leftrightarrow J$ is minimised. To be able to achieve this we need to find the direction of J that is decreasing the fastest. Let $\mathbf{u}$ be a unit vector for the slope of J in direction u, we can then find the directional derivative:

$$\min_{\mathbf{u},\mathbf{u}^T\mathbf{u}=1} \mathbf{u}^T \bigtriangledown_x f(\mathbf{x}) \tag{4.2}$$

The directional derivative is minimised when the unit vector $\mathbf{u}$ is orientated in the opposite direction of the gradient. We decrease f by following the direction of negative gradient, following this process new weightings and biases are proposed, for a weight w_{ij} we would follow the steps:

$$\bigtriangledown_x f(\mathbf{x}) = \frac{\delta L}{\delta w_{ij}} = \frac{\delta L}{\delta h_{ij}} \frac{\delta h_{ij}}{\delta w_{ij}} \tag{4.3}$$

$$w'_{ij} = w_{ij} - \epsilon \bigtriangledown_x f(\mathbf{x}) \tag{4.4}$$

Cat (1) or Dog (0)	Weight (grams)	Length (meters)
1	4,250	0.46
1	5,100	0.39
0	56,000	0.91

Figure 15: Example Training Dataset

Cat (1) or Dog (0)	Weight (scaled)	Length (scaled)
1	0	0.13
1	0.016	0
0	1	1

Figure 16: Scaled Example Training Dataset

We use ϵ to signify the learning rate of the NN, how large we would like each learning step to be, often a small constant. We continue to iterate the NN until we reach the global minimum [25].

4.6 Feature Scaling

Feature scaling refers to a family of methods which aim to bring features of a dataset closer together in scale to improve the performance of machine learning algorithms applied to the data. For example, say our training dataset looks like Figure 15.

If we wanted to perform feature scaling on this dataset, we would first decide on a suitable method which would bound our features (weight and length) to values which would allow our classification algorithm to work optimally. Say we wanted to use a feed-forward neural network to classify animals into dogs or cats based on the weight and length, we would normalize (will be discussed further in the following sections) our features i.e. scale them to be bound between [0, 1]; resulting in the scaled dataset (Fig 16).

4.6.1 Why we need to feature scale

Feature scaling is arguably one of the most important steps of data preparation (in the context of machine learning). Proper scaling often decides whether the fit of your machine learning model is accurate. The reason for this is that machine learning algorithms only see

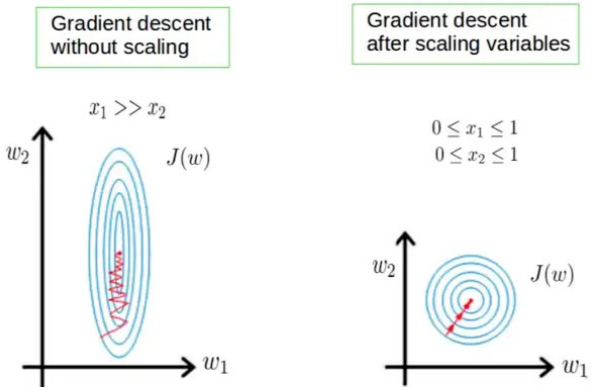


Figure 17: Diagram Showing Gradient Descent With and Without Scaling [27]

numbers, they cannot process the context and meaning behind said numbers. When Fig 15 is fed into a machine learning algorithm (for example), many different algorithms (neural networks especially) would assume that because Weight is much greater than Length, Weight has a much greater significance in the classification of the animal. Therefore weight would play a much more decisive role in the training of the model.

Furthermore, the gradient descent algorithm (which is the backbone of back-propagation) converges at a much faster rate when features are scaled. Which means that the process of training a neural network will be significantly faster when using feature scaling [26].

In Fig 17 we can see why this is the case. When one feature is much greater than the other, the approximation to the gradient tends to 'bounce around' or "oscillate inefficiently [26]". This suggests that the algorithm would take much longer to converge without scaling. Also, we can see that with scaling the approximation does not 'bounce around' and converges directly to the optimal gradient.

4.6.2 Common feature scaling techniques

1. Normalisation

   (a) One of the two most common methods of feature scaling

   (b) The method used for producing Fig X.1

   (c) Scales data feature to fit the range [0,1]

   (d) Formula: $x' = (x - x_{min})/(x_{max} - x_{min})$, where x refers to a column from the

dataset, x' to the scaled column, x_{min} to the column minimum and x_{max} to the column maximum

2. Standardisation
   (a) Second most common method of feature scaling
   (b) Assumes data within each feature is normally distributed, and scales them to have mean 0 and standard deviation 1
   (c) Formula: $x' = (x - \mu)/\sigma$
3. Max Abs Scaler
   (a) Scales each feature individually much that the maximum absolute value of each feature is 1
   (b) Does not center the data so sparsity is preserved.
4. Robust Scaler
   (a) Robust to outliers in dataset (hence the name)
   (b) Uses the interquartile range of a feature for scaling

5 Application

Although the neural network is today's popular machine learning model, it has its pros and cons. In this section, a Neural Network will be used to train two different datasets, and its strengths and weaknesses will be discussed accordingly.

5.1 Toy Data

This dataset is a subset of data used in [28], 16 by 16-pixel digital images of handwritten digits. In this cut-down image of the dataset, only digits 1, 3, and 8 are used (Fig 18).

The data consists of 90 data points with 256 features (16*16) and is split into 3 parts which 2 parts are used in training and the rest for validation. As it is known, this dataset can be classified well using classical statistical machine learning methods. Figure

Figure 18: Samples of toy data

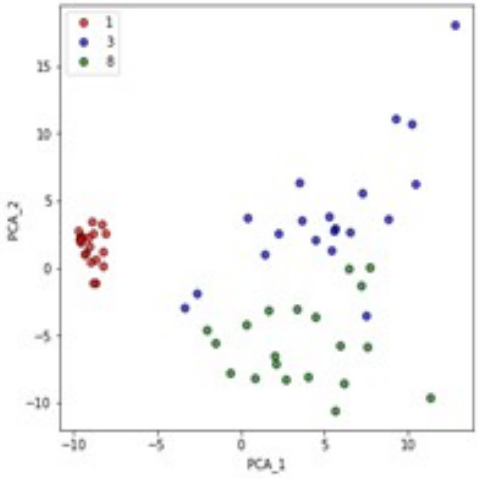


Figure 19: Training data classified using PCA

19 shows how data is classified using the PCA algorithm using just the first two principal components. It can be seen in Figures 20 and 21 that the LDA algorithm has worked even better than PCA in this small dataset on training and test data.

What about Neural Networks? Can this powerful algorithm work better in such a small dataset? First of all, categorical labels have to be encoded in such a way to be useful for NN frameworks such as TensorFlow [29]. For this problem, one-hot encoding is used to transfer categorical labels. A very shallow network using just two hidden layers, each with 5 neurons (Fig 22) and a deeper model (Fig 23), is for training the model.

As the results in Figures 24 & 25 demonstrate, over-fitting occurs easily after a few training epochs. It causes the model not to be generalised enough to work well on unseen data. This is one of the drawbacks of neural networks when working with a small number of data points compared to traditional machine learning algorithms.

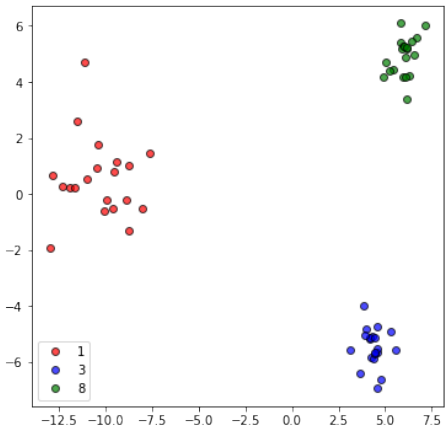


Figure 20: Training data

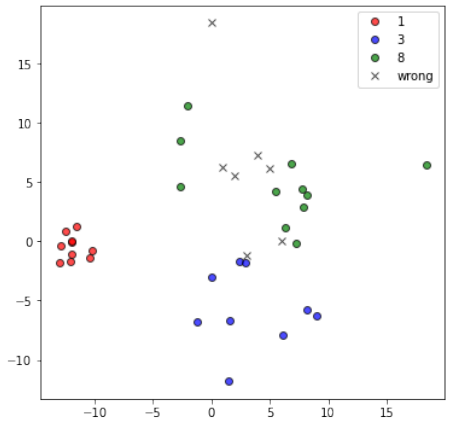


Figure 21: Validation data classified (LDA)

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 5)	1285
dense_1 (Dense)	(None, 5)	30
dense_2 (Dense)	(None, 3)	18

Total params: 1,333
Trainable params: 1,333
Non-trainable params: 0

Figure 22: Shallow NN model used to train toy data

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 10)	2570
dense_4 (Dense)	(None, 20)	220
dense_5 (Dense)	(None, 40)	840
dense_6 (Dense)	(None, 20)	820
dense_7 (Dense)	(None, 10)	210
dense_8 (Dense)	(None, 3)	33

Total params: 4,693
Trainable params: 4,693
Non-trainable params: 0

Figure 23: Deep NN model used to train toy data

Training loss (y) against epochs (x) [3]

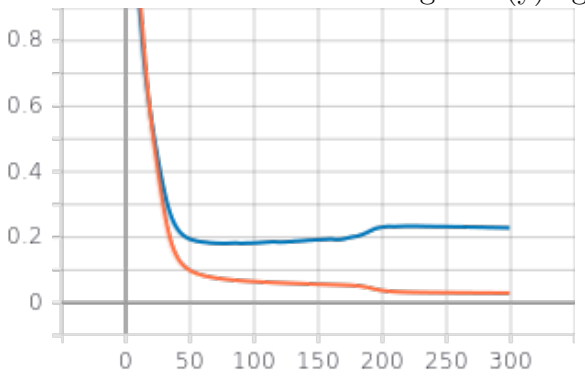


Figure 24: Shallow NN

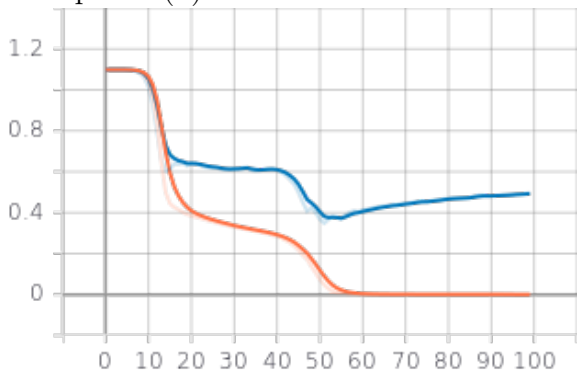


Figure 25: Deep NN

RowNumber	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
1	619	France	Female	42	2	0.00	1	1	1	101348.88	1
2	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
3	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
4	699	France	Female	39	1	0.00	2	0	0	93826.63	0
5	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
6	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
7	822	France	Male	50	7	0.00	2	1	1	10062.80	0
8	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
9	501	France	Male	44	4	142051.07	2	0	1	74940.50	0
10	684	France	Male	27	2	134603.88	1	1	1	71725.73	0

Figure 26: Bank customer data

5.2 Bank Customer Data

The question is, how is the performance of NN models in large datasets compared to other statistical models? To investigate this, a data set containing details of a bank's customers reflecting the fact whether the customer left the bank (closed his account) or continues to be a customer is used [30]. The dataset consists of 10,000 data points and 10 features, including Credit score, Gender, Age, account balance and etc. Target values are binary variables 1 shows the customer left the bank, and 0 shows the customer continues to be a bank customer (Fig 26). It is obvious that most of the customers of a bank will remain in the bank; hence the data would be imbalanced. (Fig 27)

5.3 Feature Scaling

Weights in deep learning models are initialized using small random variables and used to calculate each layer output. Therefore, the scale of the input of our model is a significant

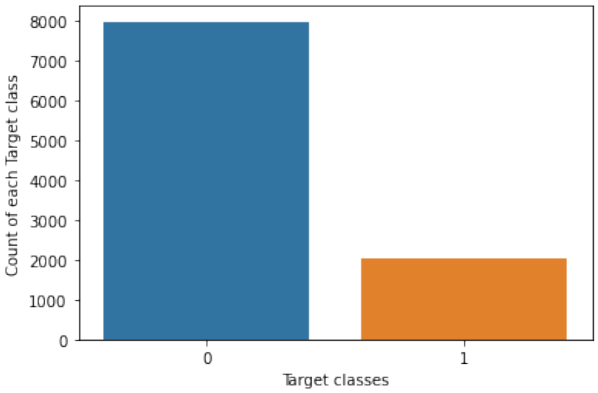


Figure 27: Target classes distribution

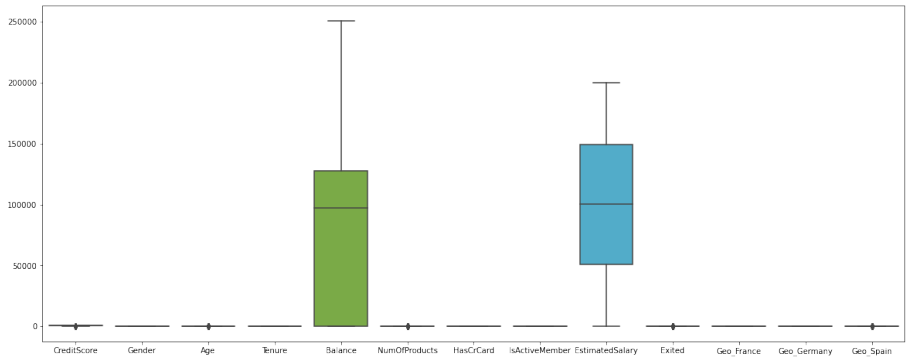


Figure 28: Input features scales

factor in the training process. Unscaled input variables, such as the bank data (Fig 28), can result in an unstable learning process. As a rule of thumb, the input variables would be scaled to have mean zero and variance of 1.

5.4 Classification with PCA

Despite the previous training in the handwritten digit dataset, PCA could not capture the characteristics of our data to classify them correctly. Plotting the data point using the first two principal components can show this clearly (Fig 29).

5.5 Classification with Neural Networks

In this problem, three different networks are used:

1. Figure 30: A shallow network with 2 small hidden layers.

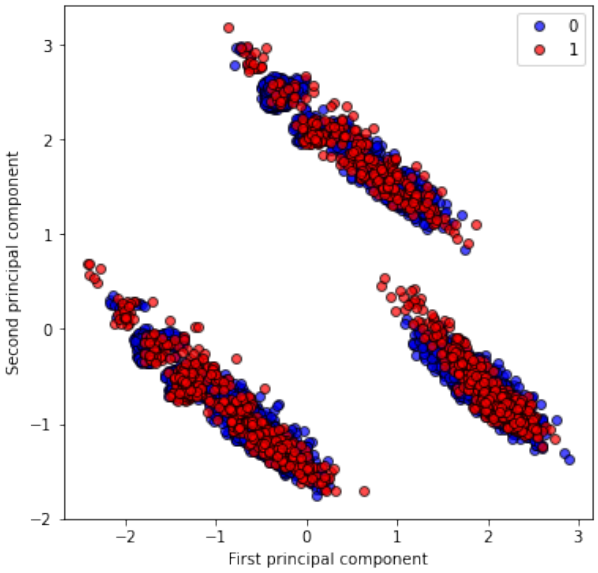


Figure 29: PCA applied to Bank Customer dataset

2. Figure 31: A network with 4 hidden layers.

3. Figure 32: A deep network with 7 hidden layers and larger number of neurons.

5.5.1 Parameters

In all our models, ReLU is used as the activation function of the hidden layers, and the Sigmoid activation function is used for the output layer. The models are trained using various learning rates (Section 6.2), and uniform random initialisation is used to initialise the weights. Binary cross entropy is used as the loss function for the back-propagation process.

5.5.2 Results

As shown in Figures 33, 34 & 35, minimum validation loss could be reached using the network with 4 layers, but it tends to over-fit after about 50 epochs. The shallow network did not suffer from the over-fitting problem, but it did not learn well in the training process. The deep network learned the training data very well but it is not general enough on validation data due to model complexity.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 5)                 65
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 30
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 6
=================================================================
Total params: 101
Trainable params: 101
Non-trainable params: 0
```

Figure 30: Shallow Network

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 20)                260
_________________________________________________________________
dense_17 (Dense)             (None, 20)                420
_________________________________________________________________
dense_18 (Dense)             (None, 10)                210
_________________________________________________________________
dense_19 (Dense)             (None, 6)                 66
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 7
=================================================================
Total params: 963
Trainable params: 963
Non-trainable params: 0
```

Figure 31: Intermediate Network

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 20)                260
_________________________________________________________________
dense_9 (Dense)              (None, 40)                840
_________________________________________________________________
dense_10 (Dense)             (None, 60)                2460
_________________________________________________________________
dense_11 (Dense)             (None, 80)                4880
_________________________________________________________________
dense_12 (Dense)             (None, 60)                4860
_________________________________________________________________
dense_13 (Dense)             (None, 40)                2440
_________________________________________________________________
dense_14 (Dense)             (None, 20)                820
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 21
=================================================================
Total params: 16,581
Trainable params: 16,581
Non-trainable params: 0
```

Figure 32: Deep Network

Training Loss (y) against Epochs (x) [4]

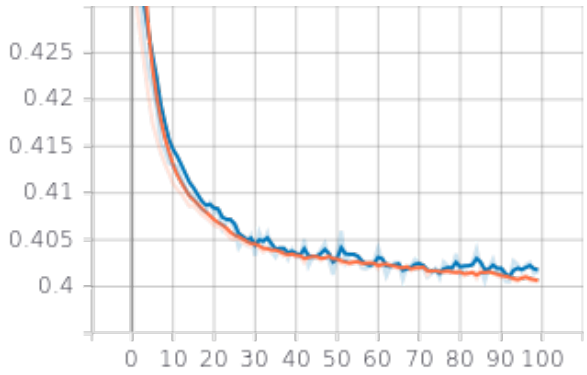


Figure 33: Shallow Network

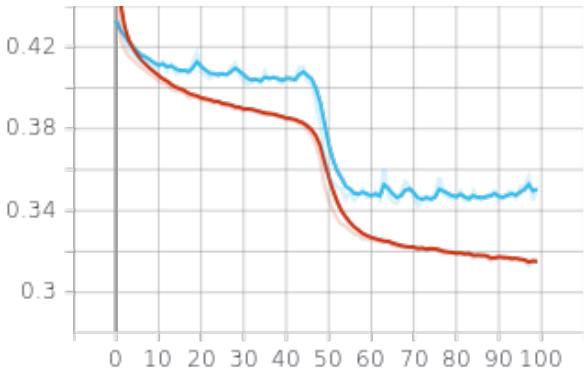


Figure 34: Intermediate Network

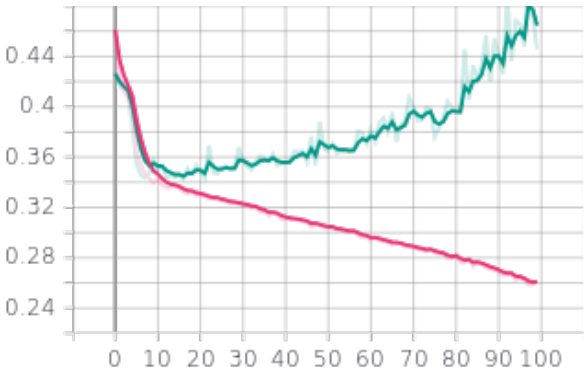


Figure 35: Deep Network

6 Issues in Training

6.1 Initialisation

Weight initialisation is one of the most important parameters to be considered when training neural networks. Networked parameters known as weights are used to calculate the weighted sum of input in each perceptron(node) of the neural network. In the optimisation process, weight should be specified at the starting point to begin the learning of the network. Training deep models is a sufficiently difficult task that most algorithms are strongly affected by choice of initialisation [25]. Different performances can be expected when using different ways to initialise the weights. By choosing the weight carefully, the problem of vanishing and exploding gradient can be avoided. These issues happen during the back propagation process, in deep ANNs we must back propagate and therefore partially differentiate between several different hidden layers. This can cause gradient vanishing as the partial differential of the loss function can $\rightarrow 0$ this means that the weights and biases will not be updated and will therefore not learn. A similar issue happens when the partial differential gets larger and larger causing the model to diverge, this is called gradient exploding. The question is that why don't we set the weight to zero values instead of random values? The point is that the parameters should break the symmetry between the neurons to enable the gradient descent algorithm to update the parameters in each training round. As an illustration, suppose two units with the same activation function are connected to the same input values. In this scenario, using the same weights will lead to the same result from each unit, and consequently, they will be updated equally. Hence, the network will not work better than a linear model.

6.1.1 Initialisation Methods

The initialisation of the weights of neural networks is a whole field of study, as the careful initialisation of the network can speed up the learning process [31]. In general, the weights will be initialised randomly from a Gaussian or Uniform distribution. In the modern deep learning framework, initialisation can be chosen from different available options, which are mostly variations of random initialisation, such as Xavier [32] or Kaming [33].

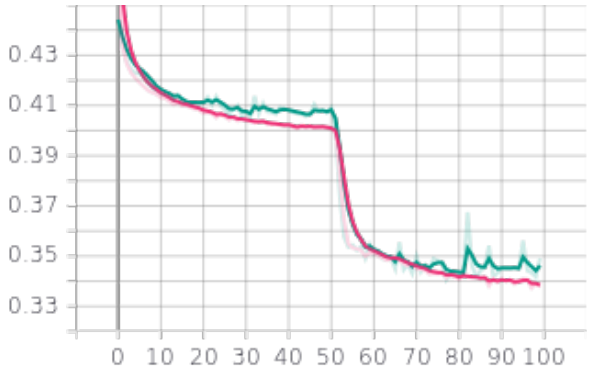


Figure 36: Loss VS Epochs in training and evaluation stage using L2 regularisation

6.2 Over-fitting and Hyperparameter Tuning

As the number of parameters increases, the neural network would be more powerful to learn different complex data, which would cause them to suffer from over-fitting easily. There are a few techniques to overcome the over-fitting problem while maintaining training accuracy. Some of these techniques will be discussed and applied here.

6.2.1 L2 Regularisation

In this technique, large values of weights will be penalized by adding a weight penalty term to the loss function. This would normally lead to the stability of the network and better prediction performance. In this study, L2 regularisation is used, adding the sum of squared values of the weights to the loss function 6.2.1.

$$-\lambda \sum_{j=1}^{p} \beta_j^2 \tag{6.1}$$

In Figure 36 the effect of the regularisation term in reducing the evaluation loss and preventing over-fitting of the model can be seen.

6.2.2 Drop-out

Drop-out is one of the most effective techniques to prevent over-fitting. During training, the network will randomly ignore or drop out some number of nodes. In effect, during each epoch, the network deals with a different view of the base configuration. This will

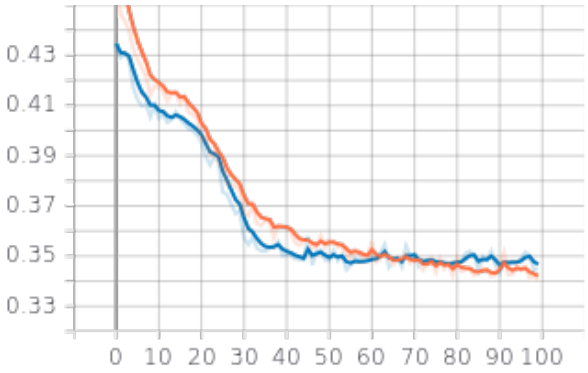


Figure 37: Loss vs Epoch in model with 20% Dropout

Effect of drop-out on learning

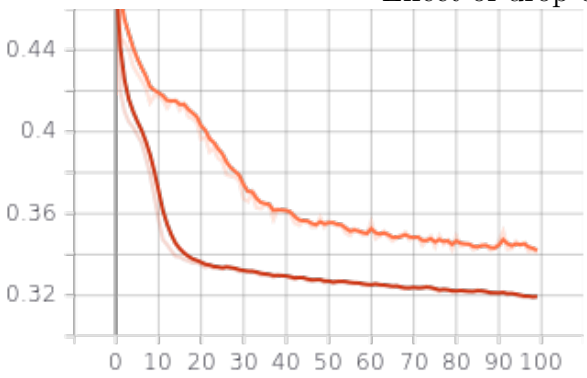


Figure 38: Training loss

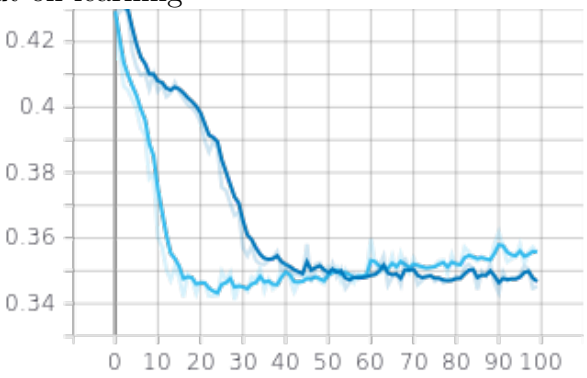


Figure 39: Validation loss

force the nodes to take on different responsibilities in the training process, which will lead to a more generalised network. Drop-out will be implemented per layer in the network except for the output layer which all nodes are required for prediction. The ratio of the nodes to be dropped out is a hyper-parameter which should be tuned during the training and validation process. The other important point is that drop-out will be used just in training, and during the validation or prediction process, all nodes will be used to gain the output result. Figure 37 illustrates the result of applying 20% drop-out to our base model, which shows how the model learned to be less complex and predict the validation data better than before.

To compare to the base model, curves are shown in Figures 38 & 39, comparing training and validation loss with and without drop-out.

7 Conclusions

The main strength of Artificial Neural Networks is that they can solve complex non-linear problems. Through the process of back-propagation we can create and train accurate predictive models. The model output from the creation process will be highly accurate as the loss function will be at its global minimum. ANNs are especially effective when trained on large datasets. However, there are several situations where implementing an ANN would be a sub-optimal choice. From the analysis in our project, ANNs struggle to create accurate models if the training dataset is too small, with small datasets it is more effective to use other machine learning techniques such as PCA. There are other more crucial issues that need solving to make ANNs more effective such as gradient vanishing and exploding.

On reflection of our project we believe we have been able to create a succinct overview of ANN for the scope and depth we were tasked with. Our group approached this project with different levels of knowledge in Machine Learning, but during the coursework we were able to effectively communicate and perform as a team, setting tasks according to the strengths of different individuals. We found that we had to be critical of the topics we wanted to include in the project because the more we researched ANN we realised the breadth of the subject. Due to the time/ resources available for this coursework we made sure to concentrate on the core areas of ANN to guarantee that these topics were covered to a high level. If we were to repeat this project we would increase the thoroughness/explanation for the mathematics of ANN especially around the topics of Back-Propagation and Gradient Descent, these are two key topics for ANN but their complexity made it difficult to get a complete understanding and to be able to explain them clearly.

If we had more time to explore ANNs, several areas we could investigate are:

- Convolutional Neural Networks
- Transfer Learning
- Sequence Models
- Natural Language Processing models

Finally, it is worth mentioning that nowadays, projects are focused on very large networks, we could look into interesting fields such as:

- Large Language Models (LLM)
- Reinforcement Learning
- Autoencoder and GAN
- Transformers

References

[1] "History of artificial neural networks - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/History_of_artificial_neural_networks

[2] "High-threshold logic - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/High-threshold_logic

[3] C. Akshay, "Mcculloch-pitts neuron — mankind's first mathematical model of a biological neuron — by akshay l chandra — towards data science," 7 2018. [Online]. Available: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

[4] "Ali ghodsi lec5 model selection, neural networks - youtube." [Online]. Available: https://www.youtube.com/watch?v=769aJ5DWn-E&t=1814s

[5] "Multiple-input gates — logic gates — electronics textbook." [Online]. Available: https://www.allaboutcircuits.com/textbook/digital/chpt-3/multiple-input-gates/

[6] "Perceptron - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Perceptron

[7] "Hebbian theory - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Hebbian_theory

[8] J. Lines, A. Covelo, R. Gómez, L. Liu, A. Araque, K. Poskanzer, and M. Santello, "Synapse-specific regulation revealed at single synapses is concealed when recording multiple synapses," *Cellular Neuroscience*, 11 2017. [Online]. Available: www.frontiersin.org

[9] Jaspreet, "A concise history of neural networks — by jaspreet — towards data science," 8 2016. [Online]. Available: https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec

[10] "Neural networks - history." [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html

[11] "Ai winter - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/AI_winter#The_abandonment_of_connectionism_in_1969

[12] "[deep learning] 2 history of artificial intelligence / xor problem of perceptron / artificial neural network (ann)." [Online]. Available: https://extsdd.tistory.com/217

[13] "Backpropagation - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Backpropagation#History

[14] "Lisp machine - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Lisp_machine

[15] "Types of neural networks and definition of neural network," 11 2022. [Online]. Available: https://www.mygreatlearning.com/blog/types-of-neural-networks/

[16] B. A. Burkov, "The hundred-page machine learning."

[17] G. James, D. Witten, T. Hastie, and R. Tibshirani, "An introduction to statistical learning with applications in r second edition," 2021.

[18] S. Sharma, "Activation functions in neural networks — by sagar sharma — towards data science," 9 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[19] P. Antoniadis, "Activation functions: Sigmoid vs tanh — baeldung on computer science," 11 2022. [Online]. Available: https://www.baeldung.com/cs/sigmoid-vs-tanh-functions

[20] "Activation functions in neural networks - geeksforgeeks," 11 2022. [Online]. Available: https://www.geeksforgeeks.org/activation-functions-neural-networks/

[21] H. Jiang, "Machine learning fundamentals."

[22] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1 1991.

[23] B. Asadi and H. Jiang, "On approximation capabilities of relu activation and softmax output layer in neural networks."

[24] "Loss functions and their use in neural networks — by vishal yathish — towards data science." [Online]. Available: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9

[25] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning."

[26] B. Roy, "All about feature scaling. scale data for better performance of... — by baijayanta roy — towards data science," 4 2020. [Online]. Available: https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35

[27] L. Zhang, "Why scaling data is very important in neural network(lstm) - stack overflow," 10 2017. [Online]. Available: https://stackoverflow.com/questions/46686924/why-scaling-data-is-very-important-in-neural-networklstm/46688787#46688787

[28] T. Hastie, R. Tibshirani, and J. Friedman, "Springer series in statistics the elements of statistical learning data mining, inference, and prediction."

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, and G. Research, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." [Online]. Available: www.tensorflow.org.

[30] "Bank customer churn dataset — kaggle." [Online]. Available: https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset

[31] J. Brownlee, "Why initialize a neural network with random weights? - machinelearningmastery.com," 8 2022. [Online]. Available: https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/

[32] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference*

on Artificial Intelligence and Statistics, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.