Mathematical properties of neural networks trained on artificial datasets

Author Natalia Ślusarz

BSc (Hons) Mathematics and Computer Science

Final Year Dissertation

Supervised by Prof. Ekaterina Komendantskaya

Heriot-Watt University

School of Mathematical and Computer Sciences

Department of Computer Science

April 2021

I, Natalia Ślusarz, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Natalia Ślusarz

Date: 20.04.2021

Abstract

Machine learning and neural networks (NNs) are growing branches of computer science, applications of which are spreading to many areas of industry - from voice recognition, through manufacturing and healthcare to image processing.

NN accuracy can be checked as can be their performance on different tests set but there is no definitive way to check how they arrive at their conclusion. Their semantics and high level properties are often not clear.

The purpose of this thesis is to investigate the mathematical properties of NNs by using artificially generated datasets with known properties. It will use knowledge of the data distribution as well as the ability to control it with parameters to implement a new tool to extract more information about the state of NN during the training. This should allow to form more specific working hypotheses that will later be tested on other datasets.

This strategy will allow us to extend our understanding of NNs and the process they go through to arrive at their final prediction, in turn helping in ongoing NN verification (efforts that are currently in the works in many labs, LAIV among them).

Acknowledgements

I would like to thank my supervisor, Prof. Ekaterina Komendantskaya, for the time and effort spent on helping me; it's her patience and responsiveness have made this possible. I am also grateful for introducing me to LAIV: Lab for AI and Verification which has given me multitude of ideas and inspirations for this thesis.

I would also like to extend my gratitude to Daniel Kienitz whose previous research has inspired the topic of this thesis and whose expertise and advice have been a great help in the process.

Contents

Chapter 1

Introduction

1.1 Motivation

In the last years applications of Artificial Intelligence have been becoming increasingly popular and important at a high pace. Much of this progress was driven by the rise of neural networks (NNs) and related machine learning technology. From voice recognition in smartphones to image recognition in self-driving cars NNs have become a part of daily life.

Historically artificial NNs came to be as a mathematical experiment starting in 1940s when the first attempt was made to create a computational model of NNs [1]. As such they were a mostly theoretical exercise for a long while; no one was worried about their safety at the time. After a few decades the interest in the topic died out slowly with progress coming to a halt.

However in the 1980s with advancing technology came the idea of using many hidden layers [2] and backpropagation in NNs which made it possible to apply NNs to pattern recognition in the tasks that were otherwise too difficult for humans [3]. About a decade ago with new more powerful computers and breakthroughs such as convolutional NNs [4] or deep feed-forward NNs [5], NNs have once again gained popularity and this time their applications are going much further. They started to find their way into areas such as autonomous cars which rely on their sureness as consequences of their mistakes can be grave.

Example 1.1.1. One of the famous benchmarks is German Traffic Sign Recognition Benchmark (GTSRB) (see Figure 1.1a) [6]. It is an image classification problem with more then 40 classes and 50 000 entries which contain images of single traffic signs at various angles and different times of the day. It was created to test and develop technologies allowing for automatic recognition of traffic signs as seen in various conditions, simulating the data available to a car on the road.

Improving in this area is vital for development of autonomous and semi-autonomous cars.

Yet despite their popularity, NNs remain hard to understand from a technical point of view. Their verification is not a simple task and while there exists a multitude of tools that can be used the problem of verification is still a major one [7].

Accuracy, which is often used for that purpose, can be deceiving. For example if we had an illness or condition that statistically affects only 1% of the population a NN that always returns a negative result (i.e. does not have the affliction) would be 99% accurate – and yet it would not be considered a good NN. This is why accuracy should not be used as the sole judge of quality when it comes to NN - other parameters need to be used.

Even the truly good NNs will encounter problems with adversarial examples [8, 9] – data that is particularly hard for a neural net to correctly classify. Those in particular pose serious safety concerns as for example a small change, such as a black sticker put in the right spot of the stop sign could cause the image recognition software to fail to recognise it and cause an accident. There also exist digital methods of altering an image in such a way that a NN will classify it incorrectly while to human eyes it still looks the same. The fact that this happens even for networks with good accuracy is evidence of the brittleness of the learned features.

1.2 Aim

The aim of this project is to undertake systematic study of mutual dependency between

(a) a range of properties of a trained NN

(b) a range of properties of data on which the network is trained.

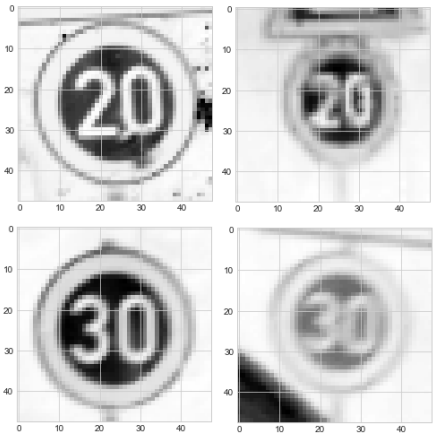
With the view to advance the state of the art and understanding of NN capabilities to model different types of data as well as generalise to unseen examples.

1.3 Methods

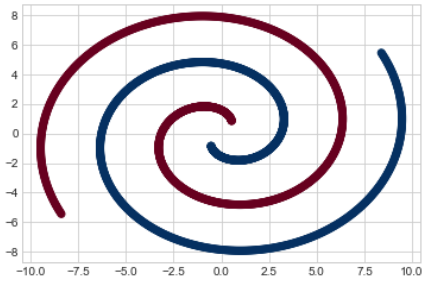
This project uses artificially generated, simplified data that can be controlled with parameters which give precise knowledge of the data distribution. This allows for constraints (codimensionality, distance between spirals and entanglement) that can be controlled in an attempt to understand how changing the data distribution affects the learned representation (measuring eigenvectors, weight matrix etc.).

This in turn can be a powerful tool for investigating the counter intuitive behaviour of the NNs, as learned representation gives information about the net's structure and behaviour when it comes to adversarial examples - which is one of the main goals of this work.

Figure 1.1: Example of real world (Figure 1.1a) and artificial (Figure 1.1b) data.



(a) German Street Sign Recognition Benchmark [6]



(b) Double spiral, defined by $S_1 : r = b\varphi$ and $S_2 : r = -b\varphi$ where S_1, S_2 are the two different spirals and $\varphi \in [\varphi_{min}, \varphi_{max}]$

1.4 Objectives

There are several objectives that need to be met:

O1 Define precisely the parameters - codimensionality, distance between spirals and entanglement - that change the data distribution in the spiral dataset and show how the three parameters generalise to other data (Section 2.3).

O2 Give a formal, mathematical definition to the intuitive idea of learned representation (Section 2.4.7).

O3 Create a tool that allows access to data from NN (Section 3.3).

O4 Conduct a series of experiments with the artificial datasets to discover correlation between the parameters and the newly defined notion of learned representation (Section 4.1).

O5 Conduct a series of experiments investigating the impact of activation function in a NN (Section 4.3).

O6 Study the applications of those concepts in adversarial robustness and neural nets verification (Section 4.3).

O7 Make a conclusion and suggest further experimentation and applications (Section 6).

Chapter 2

Background

2.1 Data

Datasets are an integral part of machine learning. Typically they are organised in matrix format $X \in \mathbb{R}^{n \times k}$ (often called the design matrix), where n is the number of entries (individual observations) and k is the number of attributes. It is a convenient format as many algorithms applied to it can make use of matrix algebra.

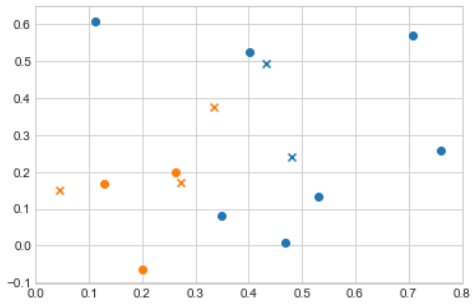
2.1.1 Data mining

Data mining is a process of gaining information from raw data using methods coming from machine learning and statistics, also known as a part of a greater process called Knowledge Discovery in Databases (KDD) [10]. Its goal is the discovery of useful patterns in batches of data which are expressed as joint probability distribution between set of attributes (inputs) X and set of labels Y with $p(x, y)$ where $y \in Y$ and $x \in X$ [11]. This can be directly taken from the given data - it is simply a measure of the likelihood of y and x occurring at the same time (containing the same information).

Variables can be independent or dependent - in the latter case the dependant variable will always change when the independent variable it is affected by does. We can check if variables X, Y are independent by seeing if the following holds $\forall\ x \in X, y \in Y : p(x \cap y) = p(x) \times p(y)$

Example 2.1.1. Taking a very simple generated dataset with independent variables (see Figure 2.1a) consisting of points where each point is either a dot or a cross and belongs to one of two groups signified by colour - Orange and Blue. Then we can see that $p(cross) = \frac{5}{15}$ (probability of point being a cross) and $p(blue) = \frac{9}{15}$ (probability of point being blue). Then joint probability

Figure 2.1



(a) Generated dataset with independent variables

(b) Dataset based on Gaussian distribution

$p(cross, blue) = \frac{5}{15} \cdot \frac{9}{15} = 0,2.$

In cases of most data however we cannot assume their independence (that is that information about x does not give any information about y) and in turn use formulae based on their conditional probability. For example see Figure 2.1b - for some variables x and y their joint probability is $p(x, y) = p(y)p(x|y)$. Let $p(x|y)$ be known - $\mathcal{N}(\mu = +1, \sigma = 1)$ for $y = 1$ and $\mathcal{N}(\mu = -1, \sigma = 1)$ for $y = 0$ - then by taking $p(y)$ as also known from data we can find the joint probability.

In classification the goal is learning a model that approximates $p(y|x)$ (in case of discriminative models) - how probable it is to get the outcome y given x is true - or $p(x|y)$ (describing generative models) - trying to describe the data distribution itself.

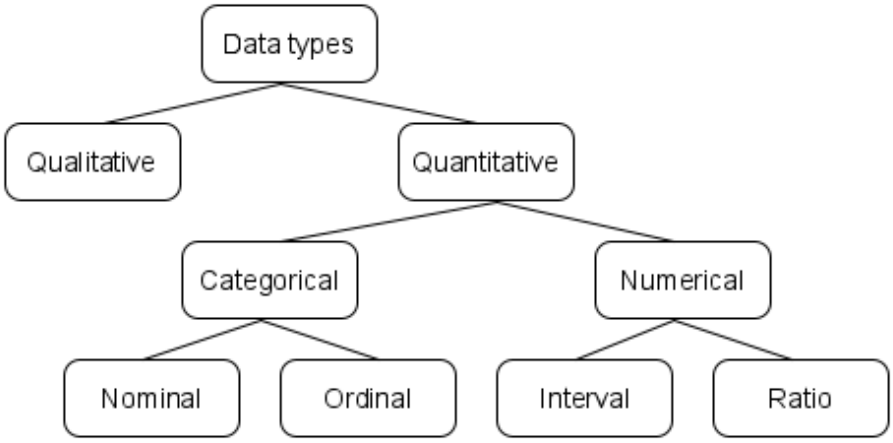
2.1.2 Types of data

Data is, at the highest level, divided into quantitative - focused on measurable quantities - and qualitative - encompassing data that is descriptive and cannot be objectively measured. In data mining and machine learning it is quantitative data that is of most importance and it is commonly further divided into more specific categories [11].

Nominal data takes values from a finite set that function like labels. They are used for values that can be differentiated between (e.g. red, green, blue) but distance between them cannot be measured and they can't be compared. Ordinal data, while still having values that are symbolic, allow for ordering but without the ability to say how much different two values are (e.g. small $<$ medium $<$ large). They can be compared but no other operations can be carried on them. These two kinds are often grouped together under the name "categorical" as they are quite similar and sometimes hard to distinguish between.

Next comes numerical data among which interval and ratio data can be distinguished. In-

Figure 2.2: Data types



terval data is measured along the scale in equal and fixed units; distance between values can be measured and compared but as the zero point is not defined no other operations can be performed whereas ratio data uses data that has a defined zero. Ratio quantities are effectively treated as real values and other mathematical operations can be performed on them.

However categorical and numerical data can be transformed into one another and vice versa which is quite important for machine learning algorithms. The simplest method is label encoding (converting labels such as 'A', 'B', 'C' into 0, 1, 2) which is rarely used, apart from perhaps decision trees and some ordinal values, as it has no connection to the actual data, simply being a simpler type of label. Among others there is frequency encoding, assigning value based on frequency with which value appears in the dataset or target encoding which assigns value based on values of target variable (usually by means of average). There are many more methods that can be used, as well as ones converting data types the other way around, depending on the particularities of the dataset - for example for categorical values "dummy variables" can be used, such as assigning a number value for each label.

2.1.3 Natural vs generated data

Another common distinction pertains to the source of data – that is whether it is natural (Figure 1.1a) or researcher generated (Figure **??**) [12]. Another good way to describe it is that natural data does not come with provided $p(x, y)$, while generated data does.

Natural data can be broadly described as data that occurred without any prompting and was simply collected by the researcher. A good example can be gathering data on plant char-

acteristics and their species - such information existed before the study and is not influenced by it.

Generated data is in contrast obtained by sampling from a known distribution. In it can be similar to the datasets used in this work which have been generated to fit certain parameters (as defined in Section 2.3) that can produce interesting results or, in case of qualitative data, would be interviews designed and held by researches since that data would not have existed without their involvement.

2.2 Mathematical definitions

This section involves the definitions for most of the mathematical concepts behind NNs, data and other topics that are relevant in this project as it draws quite heavily on the theory behind them.

As mentioned before, while accuracy is a useful measure for determining how good a NN is, it may not always be sufficient. Attempts to understand the way they work made in this work require understanding of the mathematics behind some of the related ideas.

2.2.1 Covariance matrix

Covariance matrix is an important object in data mining and NNs. First we need to look at the definition of covariance - it is a measure of variability of two random variables. Intuitively it has a positive value if when one variable increases, the other does the same (the more similar the change is the higher the covariance). On the other hand if one variable value rises and other falls than the covariance is negative.

Definition 1 (Covariance)**.** Given vectors $\mathbf{x} = x_1, ...x_n$, $\mathbf{y} = z_1, ...z_n$ the covariance $\sigma(\mathbf{x}, \mathbf{z})$ is given by

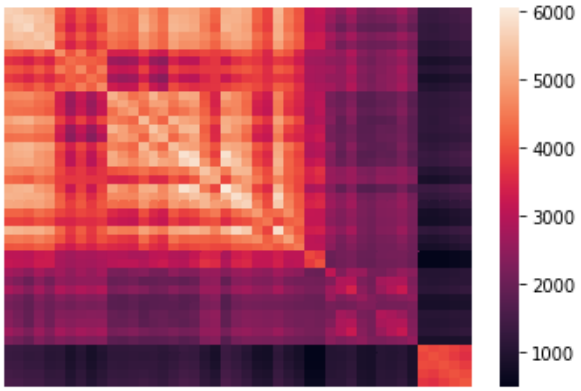
$$\sigma(\mathbf{x}, \mathbf{y}) = \frac{1}{n-1} \sum_{i=1}^{n} ((x_i - \overline{x})(z_i - \overline{z})) \tag{2.1}$$

where $\overline{x}, \overline{z}$ are means of $x_1, ...x_n$ and $z_1, ...z_n$ and n is the number of samples for $\mathbf{x}$, $\mathbf{z}$.

Covariance matrix is simply a square matrix $C \in \mathbb{R}^{n \times n}$ where n is the number of features in the dataset and the entries are covariances between the corresponding features. It is an important source of information about the relations between features in data mining.

As such it is one of the things constituting learned representation (see Section 2.4.7) of a

Figure 2.3: Covariance matrix in the form of a heatmap for a choice of pixels (features) from GTSRB (the darker colour signifies lower covariance) [6]



NN and claims about quality can be made based on its properties. For example weights that are strongly correlated or too large [13] may be an indication of the model being over fitted.

2.2.2 Singular value decomposition

Before we define singular value decomposition we need to define the singular values as well as eigenvalues and eigenvectors.

Definition 2 (Eigenvectors and eigenvalues)**.** Let A be a square, $n \times n$ matrix. An eigenvector $v \in \mathbb{R}^n$ of matrix A is a non-zero vector that satisfies $A\mathbf{v} = \lambda\mathbf{b}$. We call $\lambda \in \mathbb{R}$ that satisfies this equation an eigenvalue.

Definition 3 (Singular values)**.** Let A be a $m \times n$ matrix. Then consider $A'A$ which is a $n \times n$ symmetric matrix therefore its eigenvalues are real and non-negative.

Let $\lambda_1 ... \lambda_n$ be the eigenvalues of $A'A$ where A' is the transposed matrix, ordered so that $\lambda_1 \geqslant \lambda_2 \geqslant ... \geqslant \lambda_n \geqslant 0$. Then

$$\sigma_i = \sqrt{\lambda_i} \tag{2.2}$$

Then $\sigma_1, ..., \sigma_n$ are singular values of matrix A.

Singular Value Decomposition is, in simple words a factorization of a rectangular matrix. It is defined as [14]:

Definition 4 (SVD - Singular Value Decomposition)**.** Let A be a $n \times m$ matrix A of rank r. Let $V = [v_1, ..., v_r]$ and $U = [u_1, ..., u_r]$. Then A can be decomposed as

$$A_{n\times m} = U_{n\times n}\Sigma_{n\times m}V'_{m\times m} = \sigma_1 u_1 v_1' + ... + \sigma_r u_r v_r' \tag{2.3}$$

where $U'U = I_{n\times n}$ and $V'V = I_{m\times m}$ (U, V are orthogonal) and $\Sigma_{n\times m}$ is a diagonal matrix with singular values of A on its diagonal.

In general SVD is a powerful tool when working with a rectangular matrix but in machine learning it also has quite a few important applications - it is used in dimensionality reduction and a very popular tool when working with image compression [15].

SVD enables examination of eigenvectors of NN (of its weight matrix) which give significant information of its quality. Important notion when it comes to eigenvectors is that they can be seen as the vectors most "stretched" or "shrunk" by the matrix (either resulting in points being further or closer to each other from a spatial perspective). Since they represent the boundary of how "stretched" they can become they are a useful indicator. For example, intuitively large eigenvectors are less then optimal - they would map points "far" away from others in space and that is not a desirable outcome. It is one of the aims of this project to investigate the relation between eigenvectors of a particular NN and its quality (see Section 3.1).

2.2.3 Riemannian Manifolds

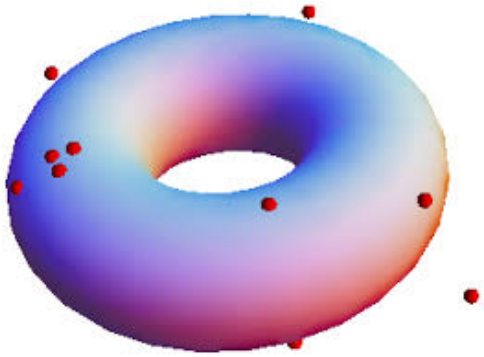
As manifolds are quite prominent in theory behind NNs it is important to understand that concept. A generic manifold is a topological space that locally resembles Euclidean space at each point.

Riemannian manifold (sometimes called Riemannian space) is a smooth manifold together with Riemannian metric - a positive-definite (inducting the norm) inner product for the tangent space (which can be informally described as a space created out of all tangent vectors attached to a set point on the manifold). The reason it is important is that the addition of a metric allows measuring distance and angles on the manifold [16].

Manifolds can help understand or visualise data. Quite often the operations involve "unravelling" the data in a way that will not introduce unwanted changes, heavily relying on methods from spatial geometry.

Manifold hypothesis Manifold hypothesis states that high-dimensional data tends to lie near a lower dimension manifold [17]. It is important for machine learning since it means that data such as images (can contain millions of pixels) is actually a problem dependent on much fewer attributes (and therefore less costly to run). Manifolds can be seen as connected region of the input space for a NN. Distance between points of data should not be arbitrarily large -

Figure 2.4: Data lying in the vicinity of a two dimensional torus
Source [17]



in more conversational terms they should be fairly "close" to each other.

2.2.4 Cross-entropy

Cross-entropy is a function that computes the difference between to probability distributions and as such is commonly used as error or loss function in NNs [18]. Mathematically it can be defined as:

Definition 5 (Cross entropy for discrete probability distribution)**.** Let p, q be discrete probability distributions both with the same support $\mathcal{X}$. Then cross entropy is as follows:

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x) \tag{2.4}$$

With discreet probability meaning a probability distribution for discreet variables (as opposed to continuous) - in this distribution one can calculate the probability for each value of the variable. For definition of support see Appendix A. In machine learning usually the two distributions in question would be the actual probability distribution ($p(x)$) and the predicted probability distribution ($q(x)$). Normally in a loss function a mean of entropy will be used. There exists a corresponding definition in case of continuous probability distribution.

2.2.5 Frobenius norm

Frobenius norm, also often called Hilbert–Schmidt norm, is a norm of a $n \times m$ matrix. There are three equivalent ways of defining it [19]; the one used in this work is based on the mathematical measure called trace which is defined as the sum of diagonal entries of a square matrix.

Definition 6 (Frobenius norm)**.** Let A be an $n \times m$ matrix and A' its transpose. Then the

Frobenius norm, denoted $\|A\|_F$ is defined as:

$$\|A\|_F = \sqrt{trace(A'A)} \tag{2.5}$$

Matrix norm is a useful tool when analysing the properties of a matrix - similarly to singular values and eigenvalues (see Section 2.2.2) it informs of the presence of large values however, due to providing a single number, it is easier to compare its value for larger amounts different matrices.

2.2.6 Diffeomorphism

Diffeomorphism is an isomorphism but defined for smooth manifolds. Formally:

Definition 7 (Diffeomorphism)**.** Given manifolds N and M, a differentiable map $f : M \to N$ is called a diffeomorphism when it is a bijection and its inverse, $f^{-1} : M \to N$ is also differentiable.

This notion is useful in more theoretical analysis of data as it is analogous with bijection simply redefined for manifolds - as mentioned previously there is room for analysis of neural networks by looking at data as manifolds and applying methods from spacial geometry [20].

2.3 Spiral dataset

The dataset that was chosen to be the base of experiments is based on a double spiral - and therefore referred to throughout this paper as spiral dataset. It was picked both because it can be easily controlled with parameters - codimensionality, distance between spirals and entanglement - and because it is also quite difficult for NNs to approximate.

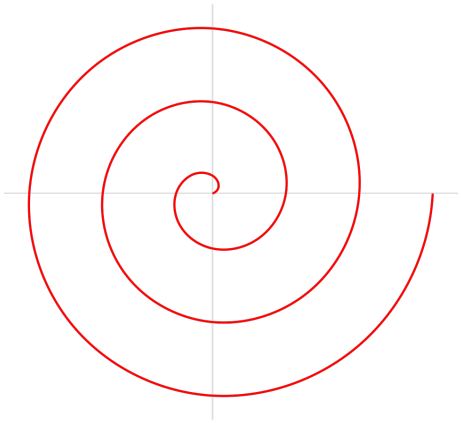
2.3.1 Mathematical definiton of a spiral

A spiral is quite a straightforward term which is easy to picture. Its mathematical definition is as follows.

Definition 8 (Spiral)**.** Two dimensional spiral is defined in polar coordinates where radius r is a monotonic continuous function of an angle φ:

$$r = r(\varphi)$$

Figure 2.5: Archimedean Spiral [21]



While there exit many types of spirals the ones used are Archimedean spirals - that is their equation is of the form: $r = a + b\varphi$ where $a, b \in \mathbb{R}$. Specifically the equation used is $r = b\varphi$ with $a = 0$ for simplicity although only a certain range of φ will be taken into account both to avoid clustering at origin and an infinite set that is the entire spiral.

The two spirals in the dataset are $S_1 : r = b\varphi$ and $S_2 : r = -b\varphi$

For our purposes of sampling data in the form of a matrix it is also useful to look at the coordinate representation on the x,y-axis:

$x_1 = r(\varphi)cos\varphi$

$x_2 = r(\varphi)sin\varphi$

2.3.2 Parameters

There are three parameters used to control the spiral dataset - codimensionality, distance between spirals and entanglement.

Codimensionality

Codimension is a term used to describe the difference between dimension of two certain objects; it is a relative term that can only be defined in relation to two objects [22].

Definition 9 (Codimension)**.** The codimension is the difference between the ambient space the data lies in and the dimensionality (dim) of the tangent space of the manifold $T_x(M)$.

$$codim = n - dim(T_xM) \tag{2.6}$$

This notion will also become quite important in case of projecting the set data into higher dimensional spaces.

Distance between spirals

Distance between spirals can be in broad terms described as the shortest distance between two points on the two spirals.

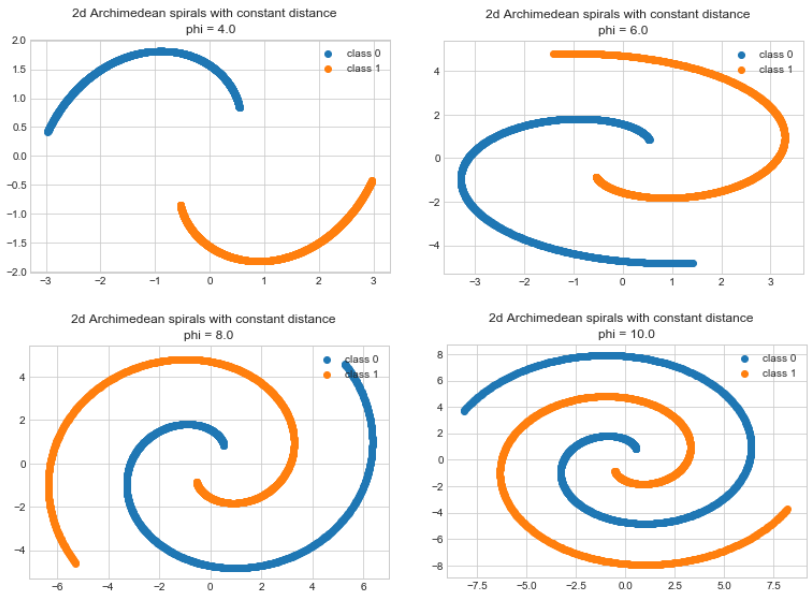
Definition 10 (Distance between spirals)**.** Given two spirals $C_1 : r_1 = r(\varphi)$ and $C_2 : r_1 = -r(\varphi)$ with $r \epsilon \mathbb{R}$ and $\varphi \epsilon [\varphi_{min}, \varphi_{max}]$ where $\varphi_{min}, \varphi_{max} \in \mathbb{R}$ the distance d between two spirals is defined as follows:

$$d_{spiral} = argmin_{x \in C_1, y \in C_2} d_{Euc}(x, y) \tag{2.7}$$

Entanglement

An intuitive description of entanglement would be a measure of how hard the spirals are to separate - say the number of line segments one would need to separate the two spirals. Another, slightly more complex way to describe it, would be through the amount of radial basis functions (a type of function the value of which depends only on distance from some fixed point) that would need to be put in space to be sufficient for all the data. In this case it is controlled by

Figure 2.6: Effect of changing maximum φ



the maximum φ of the specific spiral dataset which constitutes the range on which it is defined (as infinite spiral does not work as a small dataset).

With small φ the two spirals are very distinct and easy to classify with the difficulty of it increasing with φ.

2.3.3 Generalisation

While explained mostly on the example of the spiral dataset these three parameters (codimensiality, entanglement, distance between spirals) are a natural way to describe the topology of other data. Codimensionality is a well defined mathematical measure that is already often used in relation to data. Distance, intuitively, gives a good description of how close in space we can expect point to be in our dataset. Lastly, entanglement serves as a good measure of how hard to separate the data in question is. However these are not known for real datasets - and that is why there is a need to use an artificial dataset instead.

2.4 Neural Networks

2.4.1 Linear Classifiers

Linear classifiers are a type of simple hypothesis in supervised learning. It assumes the relation between the variable and the set of features is linear. More formally it can be described as a function of X such that [23]:

$$f(X) = \sum_{i=0}^{n} (w_i X_i) \tag{2.8}$$

where $X = X_1, ..., X_n$ is set of input features, $X_0 = 1$ and $\overline{w} = w_0, ..., w_n$ are weights.

The core quality of a good classifier is how well the weights are chosen. To achieve this we first take $e \in E$ where E is a set of examples and denote the observed value as $val(e, X)$. Now take our equation and define the predicted value as

$$pval(e, Y) = w_0 + \sum_{i=1}^{n} (w_i val(e, X_i)) \tag{2.9}$$

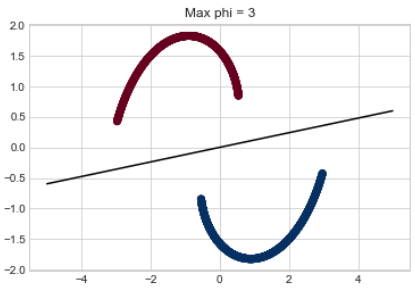
where $val(e, X_i)$ is the value of example e for feature X_i. Then choose the error metric - often the "sum-of-squares" error $Error(\overline{w}) = \sum_{e \epsilon E} (val(e, Y) - pval(e, Y))^2$ or cross-entropy (see Section 2.2.4) although others may be used. One way to achieve weights that attempt to minimise the error is iteratively, using algorithm called Gradient Descent (GD). GD starts from randomly assigned weights and adjusts them iteratively in proportion to partial derivative of the error as

well as η which is referred to as learning rate. More precisely the new w_i is changed as follows:

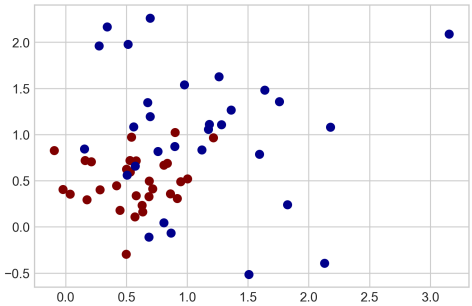
$$w_{i+1} = w_i - \eta \frac{\partial Error(\overline{w})}{\partial w_i} \tag{2.10}$$

This is updated after iteratively going through each example e either until a stopping condition is reached (which most often will be the change between old and new w_i being smaller then some set value).

Figure 2.7: Example of linearly separable and non-linearly separable data.



(a) Spiral dataset with max $\phi = 3$



(b) Non-linearly separable data

Example 2.4.1. While linear classifiers work for many problems, they fail for others. As we can see from some preliminary experiments they do work for a subset of the spiral dataset (Figure 2.7a) but clearly would not work for more entangled examples. There are many cases (Figure 2.7b) which are not linearly separable - although that can be circumvented for some data sets with the so-called "kernel trick".

2.4.2 Kernel trick

Kernel methods, often called kernel tricks, are a class of pattern recognition algorithms. It allows the programmer to work in high-dimensional data while avoiding computing coordinates of data in that space - instead computing inner products of images of pairs of entries. This is a more efficient way of projecting data into higher dimension [24]. It is a useful tool since while the data may be hard to classify in its current space it is possible that the problem would become easier in higher dimensions (though it is worth noting that this brings with it danger of overfitting [25]).

Kernel trick takes its name from mathematical idea of kernel function defined as [26]:

Definition 11 (Kernel function)**.** Let $\mathcal{X}$ be a non-empty set. Symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

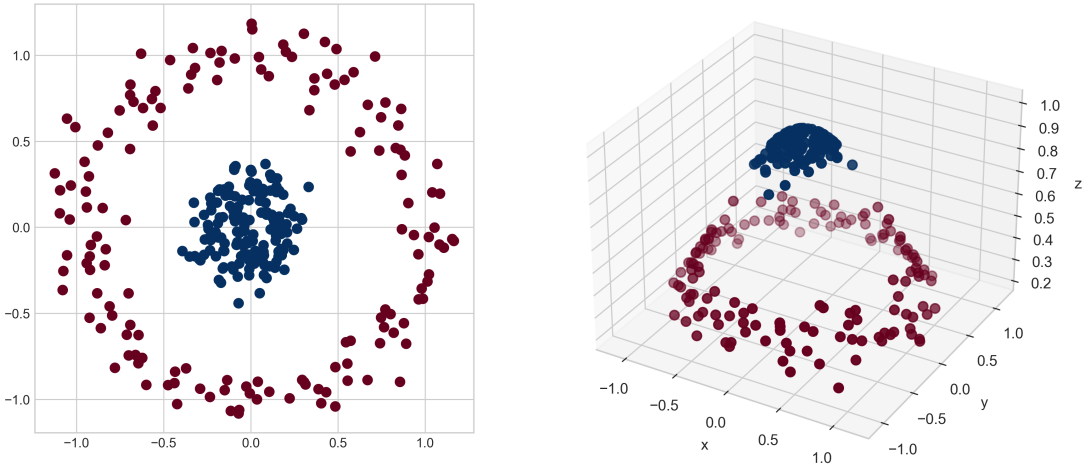
is called a positive-definitive kernel on $\mathcal{X}$ if:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \tag{2.11}$$

for any for any $x_1, \ldots, x_n \in \mathcal{X}$, given $n \in \mathbb{N}, c_1, \ldots, c_n \in \mathbb{R}, n \in \mathbb{N}, c_1, \ldots, c_n \in \mathbb{R}$.

There are many algorithms based on different kernel functions with some of the most popular being support vector machines (SVM) and Gaussian processes.

Figure 2.8: Kernel trick



(a) Data in 2D

(b) Data projected into 3D

Example 2.4.2. Take data in the form of two circles around the same origin - the 2D visualisation clearly shows it cannot be separated by a single line. Here there exists a simple solution - take $Z = X^2 + Y^2$ to project the data into 3D which (as seen on the right) makes linear separation possible.

NNs can be also used to find a kernel fitting the data to be used later in traditional kernel methods (relieving the user from having to choose it) in a process called Deep Kernel [27]. Furthermore the NN itself can be viewed as a kind of automatically learning kernel. In a way NN can be described as a linear classifier that in parallel also learns the feature space - which is explored in more detailed in Section 2.4.7.

2.4.3 Simple Neural Network

NNs, broadly defined, are computer systems inspired by biological NNs. Their structure is based around the collection of nodes - artificial neurons which are a mathematical construct

supposed to imitate, in a way, the working of a brain neuron.

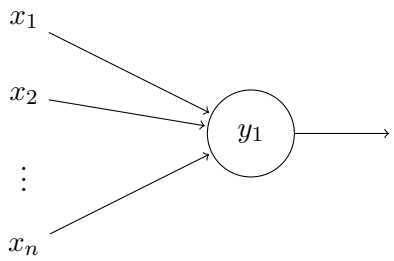
A neuron's output is defined as:

Definition 12. Given a neuron k, n inputs $x_1, ..., x_n$ the neuron's value is as follows

$$y_k = \psi \left(\left(\sum_{j=1}^{n} w_{kj} x_j \right) - \theta_k \right) \tag{2.12}$$

where w_{kj} are weights, ψ is an activation function and θ is a bias. Weights and bias can be trained.

Figure 2.9: Perceptron



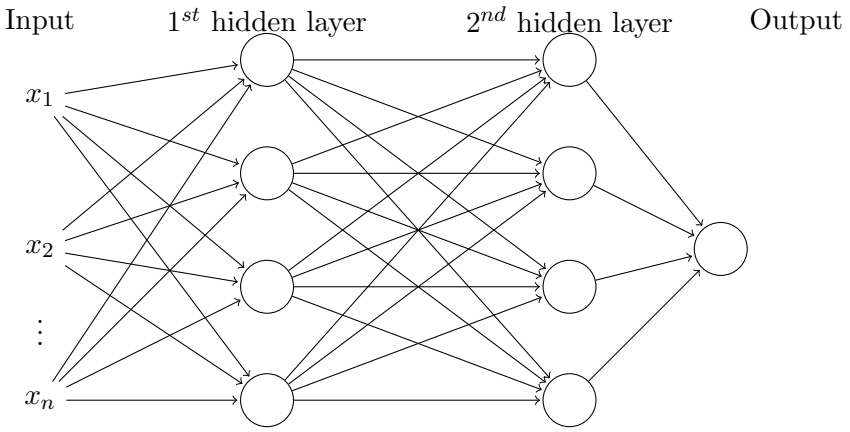
The similarities between this definition and the linear classifiers are clear - single layer NNs, also called perceptron, are a kind of linear classifier (as mentioned already in Section 2.4.2). There are also many activation functions in use, one of the more popular ones being sigmoid function $S(x) = \frac{1}{1+e^{-x}}$.

2.4.4 Backpropagation and Deep Neural Networks

As mentioned before, simple NNs, while remaining a backbone of other algorithms they do not solve all problems. A solution to this which became possible as the capabilities of computers have increased over time are deep feed-forward NNs - networks with multiple parallelly working neurons and multiple hidden layers which allows for catching different features and non-linear dependencies.

One does not need an incredibly deep or complicated network - it has been shown that a NN with only a single hidden layer is a universal approximator - that is that given sufficient width [28] it can approximate any function. However learning the weights is an NP-hard problem - new training algorithms improve the task but it remains a costly solution - more recent works have showed that wider networks need exponentially more parameters than deeper ones [29].

Figure 2.10: Sample NN with multiple hidden layers



The choice of depth and width of the network depends on the problem and computational constraints.

An often used algorithm for training such networks - that is updating the weights - is called backpropagation - it's simple, computationally efficient and works often [30] although finding the right combination of parameters - number of layers, nodes, learning rates etc. that will result in a well working NN can be a challenge.

Training of a typical feed-forward NN has multiple stages. First random weights are assigned for each neuron, then inputs are passed and predicted values computed; those can then be compared to actual values (since we are working with a training set). Each neuron's output is still described by very similar formulae as defined in Definition 12 - $y_k = \psi\left(\left(\sum_{j=1}^{n} w_{kj}x_j\right) - \theta_k\right)$ with activation function of choice - again, historically sigmoid function has often been used as it has a nice derivative which is useful for backpropagation. Suitable error metric has to be chosen as well. Then it is turn to update the weights - the goal being minimisation of the error, or loss function. The formula is based on Gradient Descent from Equation 2.10:

$$w_{i+1} = w_i + \Delta w_i \tag{2.13}$$

where

$$\Delta w_i = \eta \times gradient + \Delta w_{i-1} \tag{2.14}$$

These steps will be repeated until acceptable error threshold or maximum number of iterations is reached.

2.4.5 Convolutional Neural Networks

Convolutional NNs (CNN) are a specific kind of deep neural nets where not all the nodes are fully connected (that is not all neurons in consequent layers are interconnected). The name has its source in mathematical concept (as it often happens) of convolution - that is because convolution is used as an operation in at least one of their layers instead of typical matrix multiplication [18].

Definition 13 (Convolution)**.** Given two integrable function f and g, their convolution is:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\mathrm{d}\tau \tag{2.15}$$

where $t, \tau \in \mathbb{R}$

In machine learning the first function (f) is often called input and the second (g) kernel; the output of this operation may be referred to as feature map.

In practice it is often used on image processing therefore it can serve as a good example. In case of input image not every pixel would be connected to every neuron - instead each neuron would look at an area comprising of a few pixels - the neuron itself will learn the weights and bias in the same manner as before (the only difference being that now the neuron will be addressed by two indices as it is now represented by an array instead of a vector). Importantly the same weights are used for all the networks.

This way all the neurons learn the same feature at different locations of the input; result is often referred to as a feature map. If there exist multiple features that need to be classified, just as many feature maps need to be used.

After such a convolutional layer a pooling layer can be used to further simplify the output - it both further reduces the dimension as well as abstracts it from location (summarises features from a region of the feature map).

CNNs are often used for image processing - their nature allows for cutting down on memory requirements and works well with the structure of image data, taking into account the locality of it (since close laying pixels have a high chance of being correlated) [31].

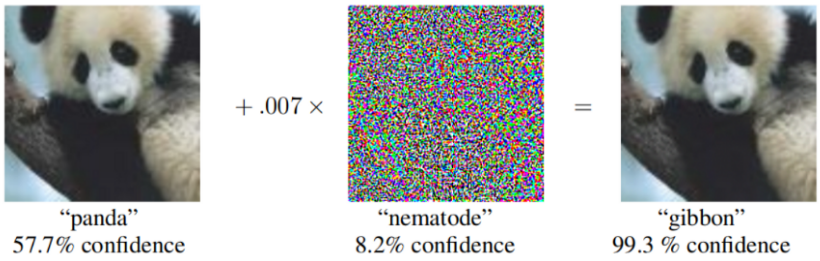
2.4.6 Verification

Software verification is the process of making sure that the developed software adheres to requirements, specifications or any other imposed restrictions.

NNs evaluation typically takes a standard form for machine learning - it focuses on accuracy on a new test set that needs to be sufficiently high to deem it acceptable. As mentioned in Section 1.1 it is not a sufficient criteria. That said there have been quite a few algorithms and methods investigating different ways of verifying NNs.

One of the interesting problems are adversarial attacks - data designed to be wrongly classified. An interesting subset of those are adversarial perturbations - entries that are very similar to ones that are classified correctly and they are given the wrong label. While there is no definitive method that can be used to check adversarial robustness of the network there a few available approaches.

Figure 2.11: Adversarial example
Source: [32]



One of the simplest is evaluation against designed adversary example [33] - unfortunately the task of designing one is a hard task in itself and using a weak adversary may cause one to falsely consider the network robust in that regard. Others attempt to create scalable verification models - finding the bounds on how much a certain property can be violated to create a computationally traceable measure [34]. There have also been studies into the relation between the high-dimensional geometry of the data and its other properties [35].

Another popular approach is formal verification - attempt to prove or disprove the correctness of the underlying algorithm and fix the fact that as of now there is little understanding of the decision making process of a NN [36]. The issue with that approach for now is the choice of function that is supposed to describe the network.

Verification is a large motivation behind this paper - using the approach of analysing a small, theoretical problem in hope that the result will provide insights into the workings of NNs that may be applicable to the problem of verification. The existence of adversarial examples is a proof that the learned representation of NNs is different to human perception - and accuracy is not enough to determine the difference. Therefore a much deeper understanding is needed - one of the ways it will be approached is investigation of learned representation (Section 2.4.7).

2.4.7 Learned representation

Learned representation in NNs contains a lot of information about the NN and can be used to determine more closely its quality.

First an understanding of the mathematical concept is needed - in algebra coordinate representation is a way of representing a vector as an ordered list in terms of a specific basis, defined as follows [37]:

Definition 14 (Coordinate representation)**.** Let $B = \{b_1, b_2, \ldots, b_n\}$ be an ordered basis for vector space V of dimension n. Then $\forall v \in V \exists \alpha_1, \ldots, \alpha_n$ such that $v = \alpha_1 b_1 + \ldots + \alpha_n b_n$. Then the coordinate vector (also called coordinate representation) is

$$[v]_B = (\alpha_1, \alpha_2, \ldots, \alpha_n) \tag{2.16}$$

Where a basis of a vector space X is a set of vectors such that each element of X can be expressed as a combination of those. For convenience it is often ordered by numbering the elements.

This becomes useful when considering weight matrices of a NN - the vector representing the weights of a specific neuron.

This is a useful tool for determining quality of the NN besides the accuracy. Several of the qualities that would be a good indicator are (arbitrarily) small weights and biases as well as no rank collapse which is the name of the occurrence when one or more singular values of the weight matrix have value of zero [38].

2.5 Related research

Use of toy datasets Analysis of NN behaviour on artificial datasets similar to the spirals used in this work is not the most common field of research since, as mentioned before, there is a significant challenge in scaling the results to other, real-world datasets. However there is still plenty of research being done on toy datasets - although most often ones of greater complexity such as MNIST since as some studies have pointed out even with all the progress in the field NNs remain susceptible to adversarial attacks and perturbations [39]. There is still plenty of room to improve both performance and understanding of NNs on such datasets.

Choosing data close enough to the real data that one wants to work with does mean that

results achieved on the artificial data are more likely to apply to real-world data. This method can often be used when looking to test a new approach to a certain dataset as it can give an indication as to whether it is likely to give good results [40]. This allows to limit the amount of tests that then have to be run on the actual dataset.

However the case for this paper is different as the spiral-dataset was chosen for the difficulty its structure presents to NNs and the ease with which it can be controlled by parameters. Both of those traits are vital when it comes to analysis of the process of NN training and the changes in it. This approach is not new - there are many artificial data generators of various types such as mlbench tool for R or wakefield [41] [42]. Such tools however are mostly used for testing limits of various NNs and other algorithms by controlling the available parameters but are too complex to allow for analysis of the way small change can affect a network.

Mathematical approach More mathematical approach to NNs is not uncommon - after all NNs as a concept have originated from mathematical experiments [1] before the technology was sufficient for any usable implementation. Considering the complexity and size of currently used NNs, especially Deep NNs which can have millions of neurons, most mathematical descriptions nowadays have to be simplifications and approximations but even that can lead to useful insights when it comes to the structure of the NN.

The beauty of this approach is that various mathematical fields and concepts can be applied to multiple parts of the machine learning process.

For example it was used in feature extraction, which is a way of identifying the most important features in data which can be later passed on to another network, substituting a large dataset with a smaller one while not losing the parts vital for classification. This approach, pioneered by Mallat and continued by various others, has resulted in creating better extractors achieving state-of-the-art results in classification tasks [43] [44].

In NN verification a geometrical approach has recently been developing; analysing the input and output data as manifolds (see Section 2.2.3) has indicated connection between the adversarial robustness and the shape of manifold modelled from input [20].

Analysis of the learning process The idea of de-mystifying the training process and understanding the it is an important notion in current NN research, especially in NN verification.

Currently NNs are far too often treated as "black-boxes", with most focus concentrated on their performance and its improvement. However with NNs being deployed in real-world

applications such as autonomous navigation it is important to understand how exactly they work since it is a good starting point for determining their quality (see Section 2.4.6).

For example formal verification seeks to define what properties need to be satisfied to prove its quality. As the NNs, due to the nature of machine learning, do not have given specification that must be fulfilled for the NN to be considered "correct" a deeper analysis is required to find the desirable properties that can be evaluated. Those can often differ depending on the type of the network as they are dependant on their architecture to some degree [45] [46].

Analysis of a specific NN, for example of an algorithm like ReSuMe (or Remote Supervised Method) can help further understanding on its performance and properties. It was first analysed theoretically in to determine promising features of the algorithm [47] - how its behaviour differed from other similar training methods and which of its features had impact on it - that were later explored and proven more practically [48].

Overall achieving better understanding of the nature of the learning process can be later translated into finding more optimised ways to use it and to improve performance and safety.

Chapter 3

Methodology

This chapter will provide the description of the project's workflow and research strategy as well as list the tools that will be used during the experiments.

3.1 Research hypotheses

The project will analyse the inner working of the NNs. Using a generated dataset allows for close monitoring of the data distribution and how it affects the learned representation.

The importance of the weight matrix when it comes to deductions about NN quality was already mentioned in Sections 2.2.1 and 2.4.7 and led to creation of the first two hypotheses to be investigated in this work that involve analysis of various values that can be extracted from the weight matrix of the NN.

Hypothesis 1. Value of the Frobenius norm rises with the complexity of the training dataset and has higher value in final layers.

Hypothesis 2. Norms of eigenvalues rise with complexity of the training dataset and have higher values in final layers.

The overall aim is to also investigate other relationships that may be discovered during experiments - hence the third hypothesis which is based on the behaviour observed in experiments involving the first two hypotheses.

Hypothesis 3 Classes are treated differently during training depending on activation function.

The final hypothesis of this thesis, divided into two parts, involves robustness of the NNs used throughout the experiments.

Hypothesis 4, Part 1. The choice of activation function impacts robustness to random noise.

Hypothesis 4, Part 2. The choice of activation function impacts adversarial robustness.

In summary this thesis connects analysis of the relationship between the weight matrix of the NN and its performance (robustness and nature of the data transformation) as well as the impact of the choice of activation function used in the NN.

3.2 Workflow

The aim of this project is to analyse the performance of NNs on a generated dataset controlled by three parameters. Therefore both the creation of data and the neural nets will be required before running the experiments initially outlined in Section 1.4.

Firstly the more theoretical aims should be achieved by creating precise definitions of the terms involved - such as parameters controlling the dataset.

Then sets of experiments can be run as described in Section 1.4 and further along in Section 3.7, with the results presented in Section 4. A tool will need to be developed to gain access to information necessary for obtaining conclusions (see Sections 3.3, 5).

Depending on the first results obtained further experiments will be designed to investigate the observed dependencies to help study those concepts in relation to adversarial robustness and more general NNs verification (see Sections 4.2, 4.3).

Based on all the information gathered through the project the final conclusion will be made. The project's aim is also discovery of potential application of the discovered relation as well as suggesting possible avenues of further research.

3.3 Functional Requirements

The concept of the tool which will be developed is inspired by existing software such as TensorBoard: TensorFlow (`https://www.tensorflow.org/tensorboard`) as well as TensorFlow Playground. TensorFlow is a robust machine learning library that comes with TensorBoard - a tool for visualisation of data, network structure and other parameters, projection of data to lower dimensions and many more functionalities. TensorFlow Playground is an online tool that provides an interactive visualisation of a NN - a small network that can have its structure changed and be tested on some prepared datasets in real time, making it a good educational

tool for gaining intuition about NNs. However both of these provide plenty of utilities and data about NNs, they do not give all the information needed in this project and therefore a new tool is necessary. The focus will be on implementing the functionalities listed below (although others may prove necessary as experiments progress):

1. Computing of the loss with respect to input - checking for vanishing gradient problem
2. Providing weight distributions and changes during training as well as their norms
3. Having the ability to compare how the points from the input data are mapped throughout training to achieve linear separability
4. Comparing performance of the NN depending on types of activation function, complexity of training data and network size
5. Generating adversarial examples for the NN
6. Providing values of codimension, entanglement, distance (defined in Section 2.3)

3.4 Non-functional Requirements

In addition to requirements listed above there is a number of non-functional requirements for this project:

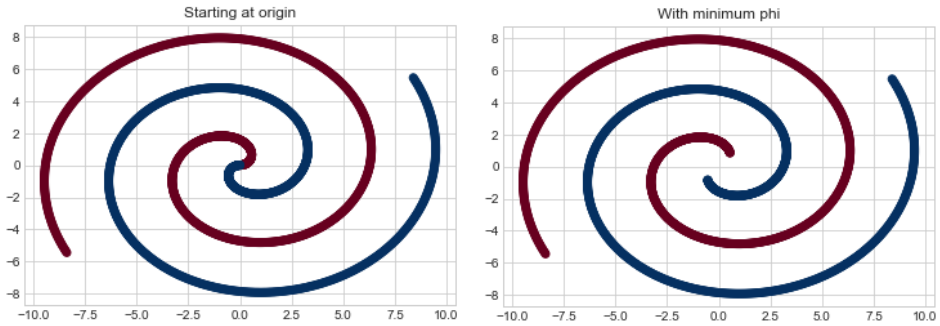
1. Licences of all software and libraries used should be respected (see Appendix E.3.1)
2. All of the experiments should be well documented and reproducible
3. All the code produced should be open source and well documented

3.5 Data Requirements

The data used will be based around the spiral dataset described in Section 2.3. Effort will be made so that they cover a large variation of parameters to analyse the difference in behaviour depending on their change and provide data that is of various difficulty to classify. Furthermore it is important that the sampling of points in from the spirals to the dataset itself is well distributed.

As mentioned in Section 2.3 some care needs to be taken when designing the spiral - for example to avoid clustering of points at origin the spiral should not start at $\varphi = 0$ instead being defined for some $\varphi_{min} > 0$ (see Figure 3.1).

Figure 3.1: Comparison of spirals with different φ_{min}



3.6 Software Requirements

Python is the main programming language used in this project for generating the spiral dataset as well the creation and running of NNs is **Python 3.7.0** within **Spyder 3.3.1** - due to its ease of use and useful libraries as it is a commonly used language for machine learning.

For package management as well as open-source distribution of Python **Anaconda Distribution 1.9.12** was used; it was chosen since contained all the tools needed for the project and provided a well designed GUI (Anaconda Navigator). Licenses of all software used in this project will be respected (see Appendix E.3.1).

The following important libraries are expected to be used in Python:

- **numpy (v. 1.15.4)** - its n-dimensional arrays and implemented mathematical operations were vital for operations on data
- **pytorch (v. 1.3.1)** - commonly used for machine learning, will be used for the NNs and other preliminary experiments related to classification
- **matplotlib (v. 2.2.3)** - for generating figures illustrating the dataset and some results for the purpose of this paper

3.7 Experiments

Several types of experiments are planned for this project.

The first set of experiments involves usage of variations of the spiral dataset where the data distribution will be controlled by the previously defined parameters. It will allow to observe the difficulty of the task depending on the set in question as well as the changes in learned representation. For those experiments networks with different activation functions will be used for comparison (see Section 4.1).

The goal is discovery of any correlations between the change in specific parameters and the performance of the NNs as well as gaining insight on how the data gets transformed by the neural net. Since the experiments are done on generated data which we know the distribution of unlike with natural data it will be possible to analyse the changes in detail.

The second set of experiments will aim to investigate further the difference between networks depending on activation function by analysing the specific way input points are mapped at the end of training during networks attempts to make them linearly separable. Furthermore it will analyse possible causes of observed behaviour (see Section 4.2).

Lastly a final set of experiments will involve testing both types of networks to determine their robustness in the face of random noise as well as adversarial examples (see Section 4.3).

The overall aim of the experiments is to learn more about the transformations that the NN performs by using a dataset the distribution of which we can control and analyse.

3.8 Evaluation Strategy

Due to the nature of this project, combining research and tool implementation its evaluation has multiple stages to it.

Implementation As specified in Section 3.3 a simple tool will be developed for the purposes of the experiments and will be made available as a public repository on GitHub.

The following will be used as measures of success in regards to the development of the tool:

T1 The implementation is complete and fully functional

T2 All of the functionalities specified in 3.3 have been implemented.

T3 The methods provided in the tool are flexible enough to allow for experimentation

T4 The visualisation functions are sufficient

Experimentation As the core of this project is research and experimentation the performance in this regard is a major part of the evaluation.

As this project is quite theoretical in nature and any obtained results cannot be directly applied to other problems, the evaluation of all sets of experiments will be based on the results gathered during them. They will be considered successful if some more general conclusions can be drawn that can be later tested against other datasets (once relevant measures of complexity for them can be determined). A further measure is whether a probable explanation - or a venue for future experiments if further investigation is needed - is identified for the observed behaviours.

More specifically the measures that determine the success of this project in this part are as follows:

E1 All the hypotheses specified in 3.1 are investigated.

E2 There is a sufficient number of experiments to support or disprove each othe the hypotheses

E3 There is sufficient mathematical reasoning to explain the observed results

E4 The experiments are well described for the purposes of reproducibility

E5 The initial conclusions are general in nature and can in the future be investigated on other datasets

Reasoning The final part of the evaluation is quite relative in nature as it involves determining whether the results and interpretations present in this thesis.

Hence the the following will need to be used for evaluation as well:

R1 The motivation behind the topic of this thesis is explained sufficiently

R2 The reasoning behind and the purpose of the experiments is explained clearly

R3 Conclusions and hypotheses are well linked to existing literature

Summary The project will be deemed successful if all or most of the measures presented in this section have been fulfilled. Thanks the hybrid nature of this thesis, involving research questions and implementation all of its parts need to be evaluated positively for it to be considered successful.

Chapter 4

Results and Evaluation

This chapter presents the main results of this thesis.

Firstly, Section 4.1 focuses on the relationship between the weight matrix of a NN, involving various parameters of the matrix such as its eigenvalues or Frobenius norm (see Section 2.2.5), and the complexity of the training dataset.

Then Section 4.2 looks to investigate the difference in NN behaviour depending on the activation function used. The evidence of this is first observed in Section 4.1 as well as in preliminary analysis of their performance. It compares the process of training and the way spiral dataset is "disentangled" during the training in both types of networks.

Lastly Section 4.3 endeavours to test the quality of both types of networks used in previous experiments in the form of their robustness to random noise as well as adversarial examples. The main goal is to investigate whether the different performance observed before in networks with different activation functions impacts the robustness and in what way.

4.1 Relation between the weight matrix and data complexity

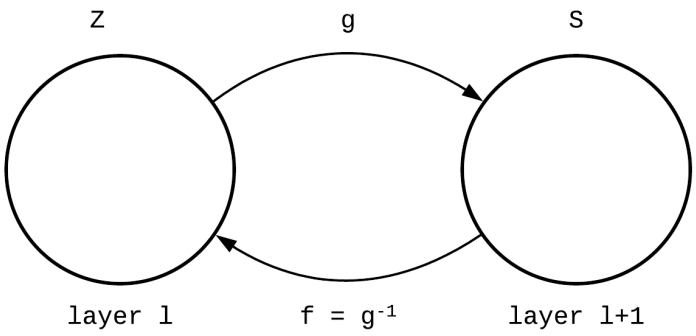
As mentioned before the weight matrix of the NN is the core of its decision process and therefore a good source to draw information from.

Making it more intuitive - the classification process in the NN can be looked at as a change of variables [50] - each layer seen as transforming the variables from previous layer to new ones. The goal of this is the discovery of such a variable transformation that the classes become linearly separable before the last layer - since that is the softmax layer which is a linear classifier.

Let Z and S be the data densities (or, in other words, the data from spiral dataset) of layers

l and $l+1$ correspondingly - then there is an invertible, differentiable map g (diffeomorphism - see Section 2.2.6) - which represents the transformation of the data distribution between layers in a NN (see Figure 4.1).

Figure 4.1: Diagram illustrating the transformation of the data distribution.



Knowing this, one can compute the change of shape the probability density (which is another popular way of describing NNs) as follows [50]:

$$p_S(s) = p_Z(f(s))|\det Dg(z)|^{-1} \tag{4.1}$$

with z, s as elements of Z, S respectively and $Dg(z) = \dfrac{\partial g}{\partial z}$ being the Jacobian matrix (for detailed definition see Appendix B) of g.

However, as defined in Section 2.4.3, g is known:

$$g = \psi(Wz + bias) \tag{4.2}$$

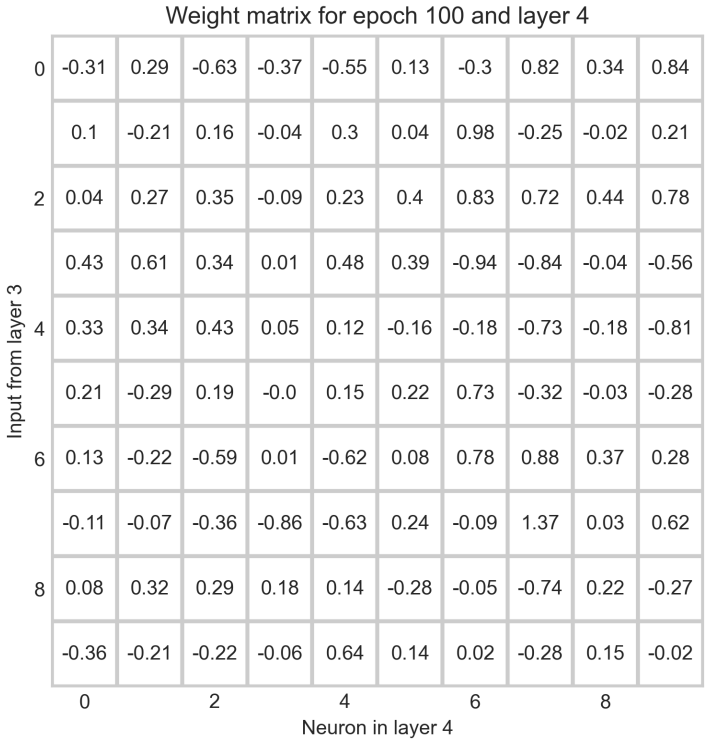
where ψ is an activation function of choice, W the weight matrix of a particular layer and z the input.

The core of g is Wz - the weights control how much impact an input will have on the output as it is the centre of the learning process.

However analysis of a weight matrix in its basic form, by simple inspection is quite difficult - as can be seen in Figure 4.2 even a weight matrix for a relatively small NN with only 10 neurons in each hidden layer does not give much information at first glance as shown in Figure 4.2.

This can of course be somewhat improved by, for example, visualising the matrix as a heatmap which makes it possible to see if there exist any significantly smaller or greater values

Figure 4.2: Weight matrix of neural network with 10 neurons in each hidden layer. Even for a relatively small matrix (the NNs used in most experiments involve 50×50 matrices) it is already too big to be able to draw conclusions from simple visualisation in its raw form.



in it (see Figure 4.3) however this still does not solve the issue of comparing various matrices.

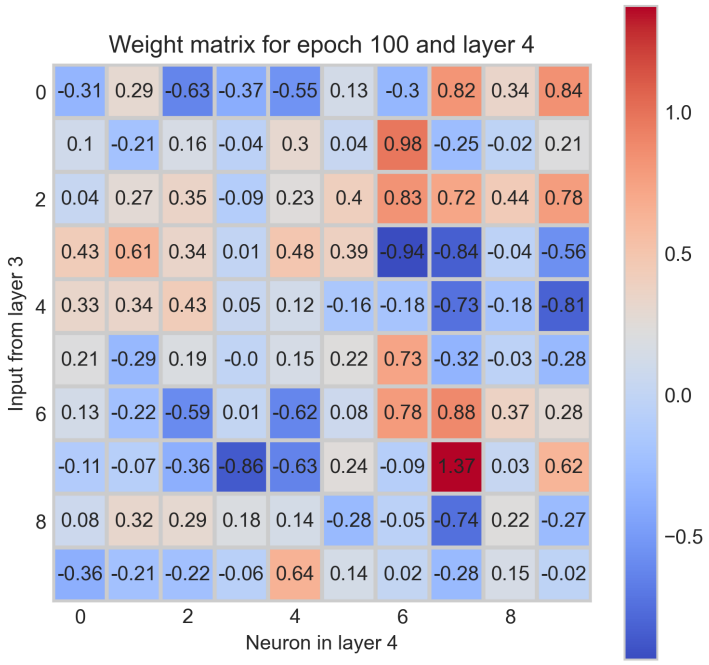
Hence the intuition behind the experiments was to choose different data that can be obtained from a matrix.

Determinant of the Jacobian matrix (which is often called a Jacobian as well), can be interpreted as change of density (of the data distribution). The volume - $\int p(x)dx = 1$ - is unchanged by definition of probability but the change of density may result in it being "thinly stretched" - that is with very small or close to zero values - in some areas and dense in others. It is why it would be undesirable for it to have a large value in a NN - the goal is to have the input data disentangled smoothly, without large changes in any direction. However analysis of Jacobian matrix or determinant is a computationally heavy task, especially for larger matrices (which for NNs they often are).

However the same logic leads to considering the eigenvalues - since their corresponding eigenvectors represent the points most "stretched" or "shrunk" by the matrix (see Section 2.2.2). As such they provide much of the same information as the Jacobian matrix would - a Jacobian of a linear map being another linear map on top of it.

Then, the simplest way to gain some measure of intuition about the weight matrix would

Figure 4.3: Weight matrix of neural network with 10 neurons in each hidden layer as a heatmap. While this can give some indication of presence of very large or small values it is generally of limit use when drawing conclusions about the matrix as a whole.



be to compute Frobenius norm (see Section 4.1.1). It has a similar meaning to eigenvalues, geometrically; it is much simpler to compute and compare with the disadvantage of giving much less information - therefore being a good starting point to the experiments.

Thanks to the design of the spiral dataset (See Section 2.3) increasing the entanglement of the spirals is an easy way to create a controlled rise in complexity of the dataset.

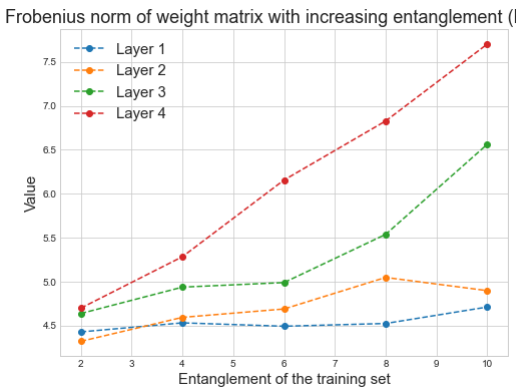
The experiments start with NNs of the same architecture; that is with identical amount of layers, neurons and activation functions but trained on spiral datasets with different entanglement value. This creates a way to observe the difference in the learned representation of the NN depending on the complexity of the data.

4.1.1 Hypothesis 1 - Value of Frobenius norm rises with the complexity of the training dataset and has higher value in final layers

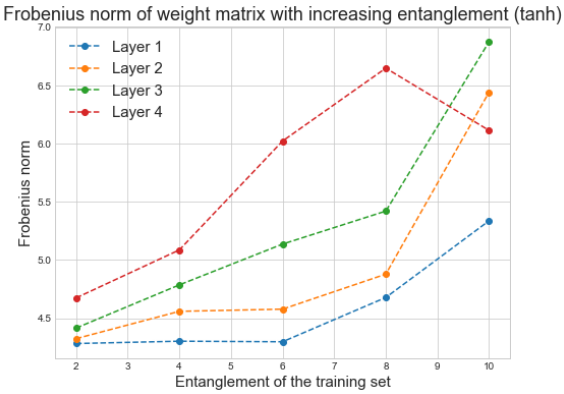
As mentioned above one of simple ways to analyse and compare some properties of matrices would be to look at their Frobenius norm (see Section 2.2.5).

The experiments were split into two sets - with different activation functions, namely ReLU and tanh (see Section 4.1.2), used for all the hidden layers (and softmax used in the output layer).

Figure 4.4: Frobenius norm of weight matrices in different layers depending on the entanglement of the training set. NNs using both activation functions both have a rising value of the norm for more complex datasets however there is some unpredictable behaviour present in NNs using tanh (Figure 4.4b)



(a) NNs with ReLU activation function



(b) NNs with tanh activation function

Looking at the results (see Figure 4.4) there is a clear tendency for the Frobenius-norm to have a higher value for the NNs trained on datasets with higher entanglement. There is also noticeable rise in value between the first and last hidden layers (in most cases).

However there is a difference between the results using different activation functions with tanh exhibiting a more irregular behaviour across the layers, especially for the last, most complex set. Meanwhile values for ReLU keep rising steadily with the complexity of training set for all layers, with the highest values being noted for final layers.

4.1.2 Hypothesis 2 - Norms of eigenvalues rise with complexity of the training dataset and have higher values in final layers

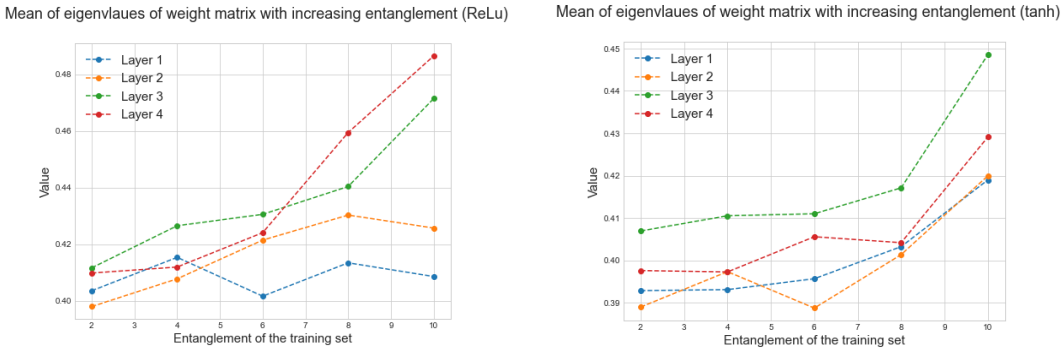
As means to further analyse the observations, since Frobenius does provide only some indication as to properties of the matrix, one can take a closer look at the eigenvalues themselves.

The experiments were run in the same way as detailed before in Section 4.1.1, with NNs using tanh or ReLU as activation functions for the hidden layers.

There are a few measures of interest when computing eigenvalues, as inferred from Section 4.1. For the results a norm of eigenvalues was taken - due to real entries being present in the matrices, complex eigenvalues have appeared in many cases. After some further testing there seems to be no significant change in the number of complex eigenvalues with either change of the activation function or increase of the entanglement in the training data set.

As mentioned the desirable outcome would be for all of the eigenvalues to remain relatively small as that would mean that the data is transformed smoothly and without irregularities in

Figure 4.5: Mean of eigenvalues of weight matrices in different layers after the end of the training period with varying complexity of the training dataset. With NNs using both ReLU and tanh there is a clear rise of values for later layers and higher complexity however there are differences in exact behaviour.



(a) NNs with ReLu activation function

(b) NNs with tanh activation function

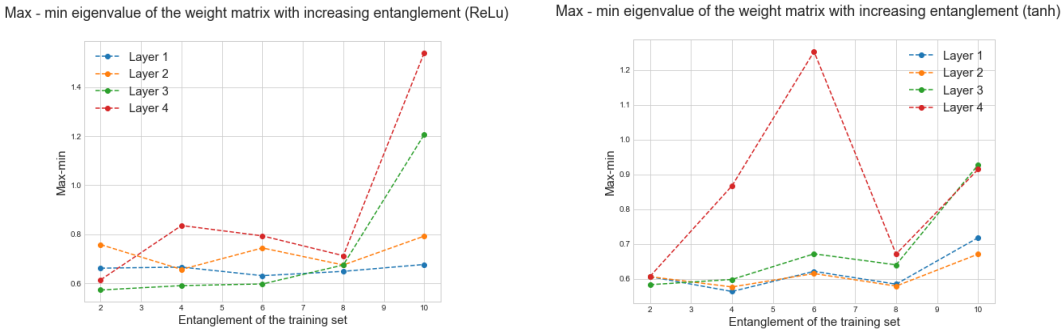
any certain directions.

The results showed increase of value for more complex dataset, however the difference in values is small - the general trend can be seen in Figure 4.5 although instead of all values a mean is used for the sake of clarity.

However the trend observed in Section 4.1.1 of highest values being observed for final layers does not necessarily hold true here - especially for tanh. While the final layer does seem to have fairly large or even largest value for some training tests it is not consistently greater for either of the network types.

Then $|\lambda_{max}| - |\lambda_{min}|$ (where λ denotes an eigenvalue) was analysed to compare the size of the range in which all the eigenvalues fall.

Figure 4.6: $|\lambda_{max}| - |\lambda_{min}|$ for weight matrices in different layers after the end of the training period with varying complexity of the training dataset. ReLU exhibits a clear rise in values for more complex training set (Figure 4.6a) while the behaviour of NNs using tanh is much more erratic yet still exhibiting higher values for final layers (Figure 4.6b)



(a) NNs with ReLu activation function

(b) NNs with tanh activation function

We can see that while for ReLU the range between $|\lambda_{max}|$ and $|\lambda_{min}|$ seems mostly consistent - rising with complexity of the training set, although with a spike on the last example - tanh is

very irregular for the final layer [1].

Similiar experiments were run for the conditional number of the weight matrix ($\frac{|\lambda_{max}|}{|\lambda_{min}|}$) however no clear trend was observed (see Appendix D).

While it seemed that majority of eigenvalues stayed within a similar range regardless of the layer and complexity a peculiar pattern was observed for both activation functions where the eigenvalues for the matrix of the final layer would have one significantly larger outlier eigenvalue. Evidence of this can be seen in Figure D.1 where the $|\lambda_{max}| - |\lambda_{min}|$ increases for the final layer.

From a practical point of view both of the values (the eigenvalues themselves as well as the condition numbers) should be minimised to achieve a smooth disentangling of the spirals.

Tanh vs ReLu - possible causes of different behaviour The choice of these two activation functions is not accidental.

Tanh (hyperbolic tangent) may not be the best performing function but from a mathematical point of view it has quite a few interesting properties mainly coming from it being a diffeomorphism (see Section 2.2.6). Colloquially one could say it means that the tanh function does not "rip" the data space during transformations instead classifying it smoothly. Since it is bijective the results can be "reversed" - furthermore it means that the map between layers of a NN is a diffeomorphism which allows for many more analytical tools, such as Equation 4.1.

From a mathematical point of view it is a much neater function then ReLU but its performance suffers.

ReLU - $f(x) = max(0, x)$ for input x- is one of the more commonly used functions in various NNs as it generally achieves good performance - as seen in previous results. However due to how to function is defined, effectively zeroing every value < 0, it can sometimes lead to a number of issues in its analysis (although the sparse nature of the representation does also have its advantages in some situations according to some sources [51]). It is also only a diffeomorphism if $x > 0$ which introduces further complication into mathematical analysis.

Furthermore ReLU is an activation function very often used in NNs for various purposes as it is known to achieve good performance [52]. For example while they introduce the vanishing gradient problem they do fix the issue of over saturation that functions such as tanh suffer from (since all values are mapped into interval between 0 and 1 all of the large values will become extremely close to 1 which can cause issues) [18].

[1] More experiments were run to make sure it was not a consequence of unfortunate randomisation and the results still showed irregular rise and significantly higher values for the last layer.

4.1.3 Summary

In line with both hypothesis presented in this section there is a notable rise in value for both eigenvalues and Frobenius norm for higher complexity training datasets in both types of networks (using ReLU and tanh). It implies that as the classification task grows more complex the NNs struggle to disentangle it in a "smooth" way. And although the behaviour is similar in many ways it is worth noting that in most experiments NNs using tanh seem to behave more erratically with some values being significantly higher or lower while NNs using ReLU present a more steady rise.

This section also unveiled evidence that the choice of activation function has a very large impact even in fairly simple NNs such as the ones used. While ReLU has been known to be an arguably better choice then tanh, as mentioned before, and its performance is often the subject of academic work there has not been a direct investigation into this type of comparison between activation functions and the exact effect they have on the classification process. It is the goal of other experiments of this thesis to further investigate the nature of this difference.

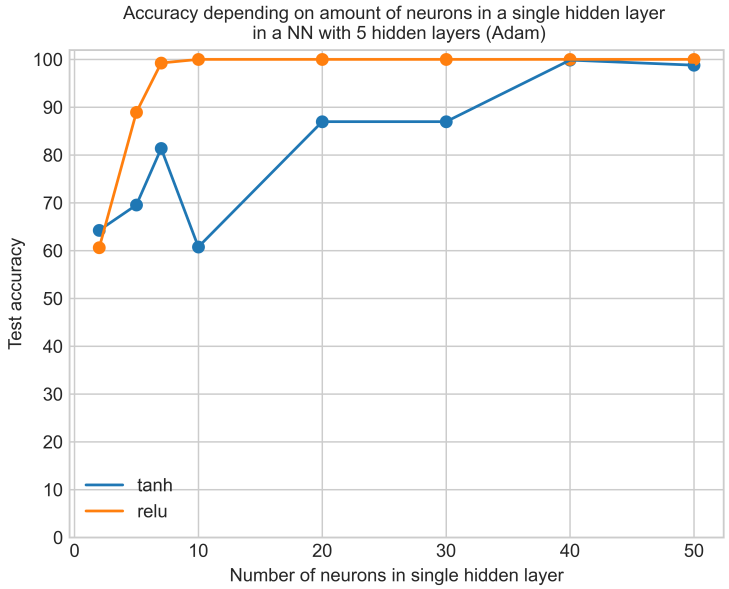
4.2 Difference in NNs behaviours depending on the activation function and class treatment

The results of the previous experiments have shown evidence of significant differences in results achieved by networks using different activation function (ReLU and tanh). While some of the differences are more obvious then others (see Section 4.1.2) this has prompted a more in depth investigation of their impact on the performance as well as possible causes.

Accuracy during training depending on size of hidden layers During the experiments with eigenvalues and Frobenius norms it was noticed that NNs using tanh as the activation function seemed to require a significantly larger number of neurons in the hidden layers (with all the other parameters kept the same) then the ones using ReLU. Hence an effort was made to analyse the change in accuracy when gradually increasing the number of hidden neurons in a layer - each NN had multiple hidden layers with the same number of neurons in each.

As can be seen in Figure 4.7 a NN using ReLU can be significantly smaller and still obtain good results (though for comparison's sake the same, larger network was used in both cases as to not introduce additional variables). Furthermore once again tanh seems to exhibit a more

Figure 4.7: Accuracy for NNs with tanh and ReLU activation functions depending on number of hidden neurons in a layer.



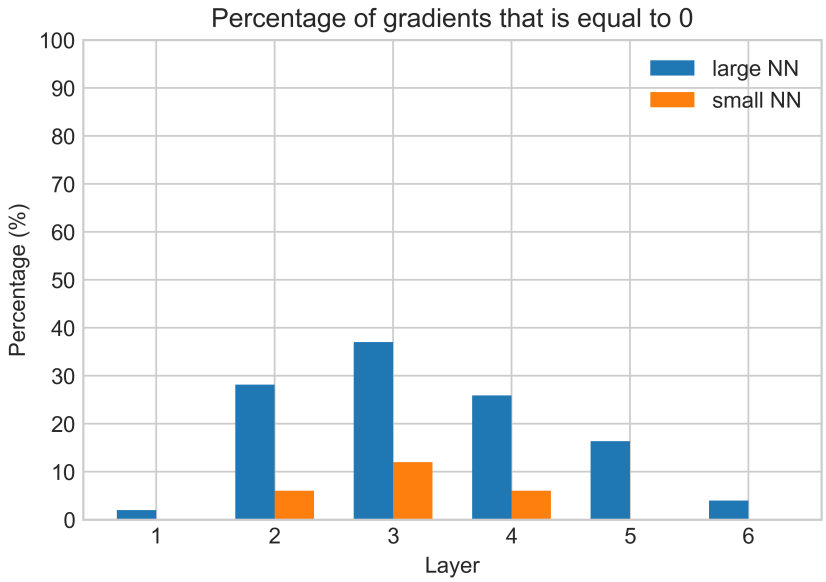
chaotic behaviour.

ReLu - vanishing gradients The vanishing gradients problem is an issue that can arise with backpropagation (see Section 2.4.4) based NNs (especially Recurrent NNs) when the gradients of loss with respect to input become increasingly small or equal to zero [53]. A consequence of such behaviour in the backpropagation algorithm is that certain weights stop being updated or become updated by number far too small which slows training progress.

Initial tests showed that the NN using ReLU had quite a big percentage of gradients vanish in every layer. However this behaviour could possibly be explained by the results seen when measuring accuracy - for ReLU a NN sufficient for this classification is much smaller than the one used for the experiments (which, as mentioned, has the same parameters for both ReLU and tanh and therefore needed to be larger to enable tanh to work efficiently). This could result in the gradient getting increasingly smaller if the weights have already reached the optimal value earlier in the training.

An experiment was run with both the larger network, as used in all previous experiments, with 5 hidden layers each with 50 neurons and with a smaller network with 5 hidden layers and 10 neurons in each (all other parameters left unchanged) which, as could be seen from Figure 4.7 is sufficient for classification. Looking at results in Figure 4.8 it can be seen that while ReLU clearly has a good number of zeroed gradients for the larger network, the problem was

Figure 4.8: (ReLU) Percentage of gradients value of which is equal to 0 in each layer. Both NNs have 5 hidden layers (and an additional softmax output layer): "small NN" with 10 neurons each, "large NN" with 50 neurons each



much lesser for the smaller NN - while vanishing gradients were still present, it was a not of the same magnitude.

Those results could indicate that the vanishing gradient issue could be caused by using a network much larger then needed. However as the issue is still present in the smaller network at least part of the problem is most likely caused by the nature of ReLu which returns zero for any input smaller then 0.

Tanh showed no vanishing gradients for any layer or size of network.

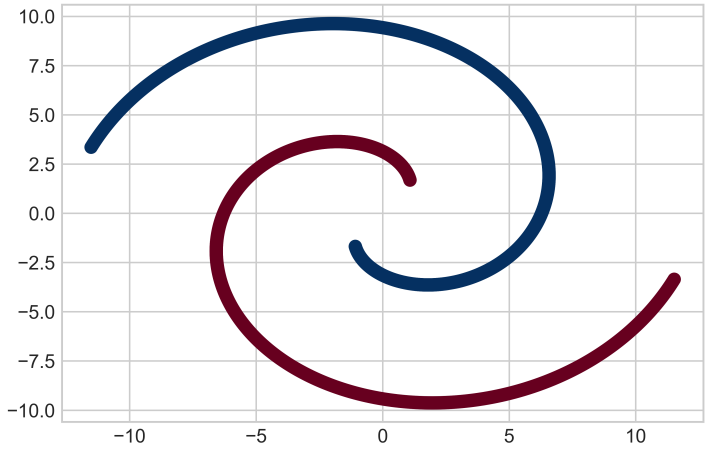
4.2.1 Hypothesis 3 - Classes are treated differently during training depending on activation function

In an endeavour to gain more insight into the process of training and "detangling" the spiral data was to, instead of a more efficient and complex one, use a small network with just one hidden layer with only two neurons. This way after each epoch of training a plot could be made which would show where the points from training data (a sample from spiral dataset) would be mapped by the network as it learns.

The spiral chosen for the training set (see Figure 4.9) had deliberately low entanglement as due to the simple architecture of the NN used it would otherwise struggle with more complex data.

For this experiment tanh and ReLU were used as activation function to compare the be-

Figure 4.9: Spiral from which points were sampled to form the training set for these experiments

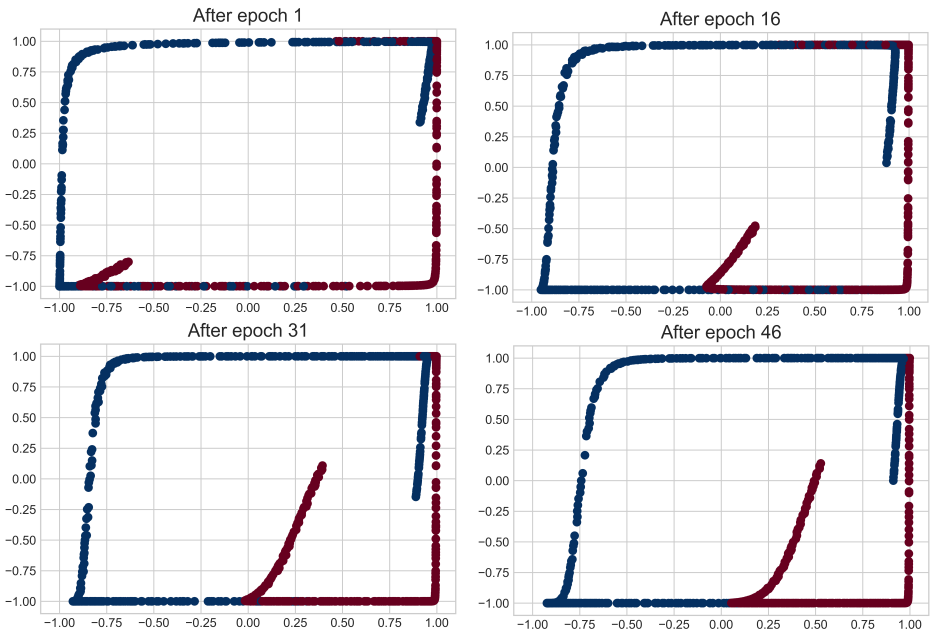


haviour of the network, continuing to investigate the trends that have started showing up in previous experiments.

Mapping over training time The first series of experiments included monitoring of the change in how the points are mapped as the training progresses with results shown every few epochs until there are no clear changes in shape for the sake of clarity.

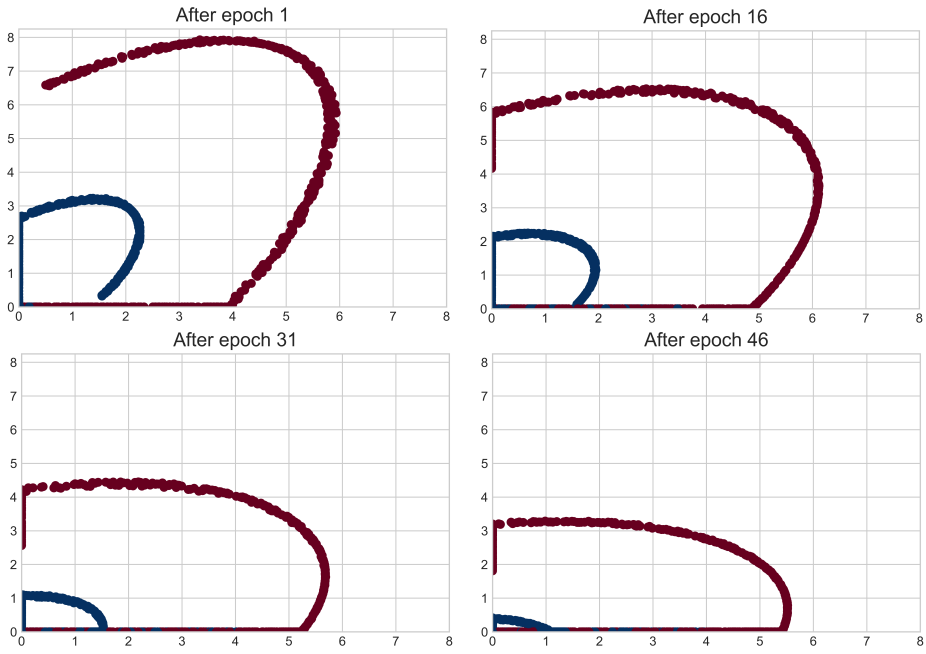
First the experiments were run on a NN with hidden layer using the tanh activation function (see Figure 4.10), then using ReLU instead (see Figure 4.11).

Figure 4.10: Mapping of points during training NN with tanh as the activation function of the hidden layer



Even at first glance there is a visible difference in how the two activation functions impact the way in which the spirals are disentangled into linearly separable.

Figure 4.11: Mapping of points during training NN with ReLu as the activation function of the hidden layer



In the case of tanh, considering that further epochs not shown here have not provided any drastic change, it is easy to notice how this would cause issues with the accuracy that have showed up before. While the shape is significantly simplified from the original spiral it does not seem to become fully disentangled.

ReLu seems to learn much quicker (again, as indicated before with accuracy measures) but as the training progresses one of the spirals begins to collapse into almost a point, with only a few outliers mixed in.

Mapping depending on batch size One of the constants throughout previous experiments was batch size but an interesting thing to observe was the impact of setting it either smaller or larger then generally recommended [54] considering the size of the training dataset. As before the set of experiments have been run on the networks with tanh and ReLU as the activation functions.

There was no noticeable difference in shape of the mapping depending on batch size with one spiral compressed so far that for the sake of comparison and visibility it had to be mapped separately (see Figure 4.14).

Figure 4.12: Mapping of points with tanh as the activation function with varying batch size of training data

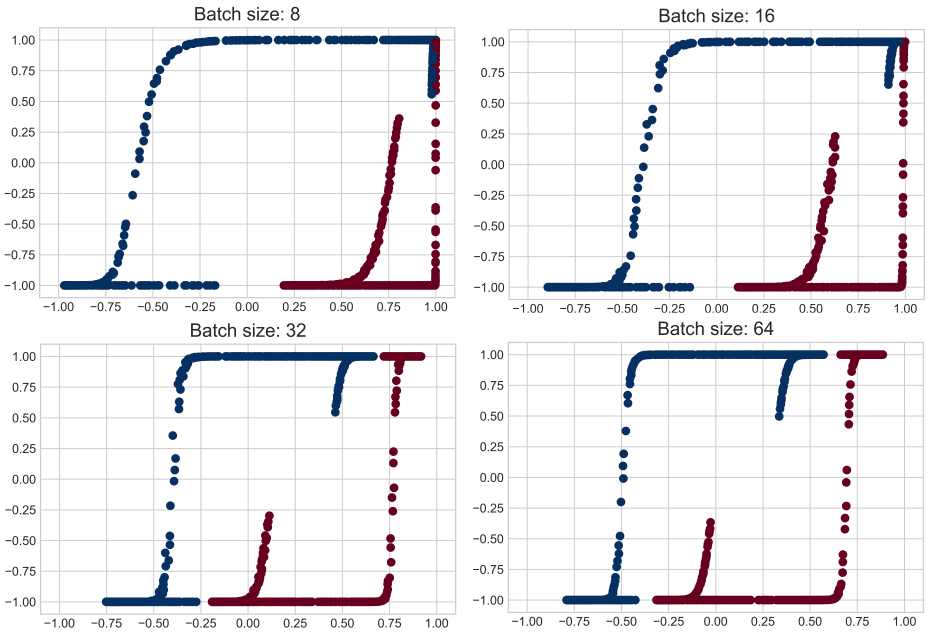
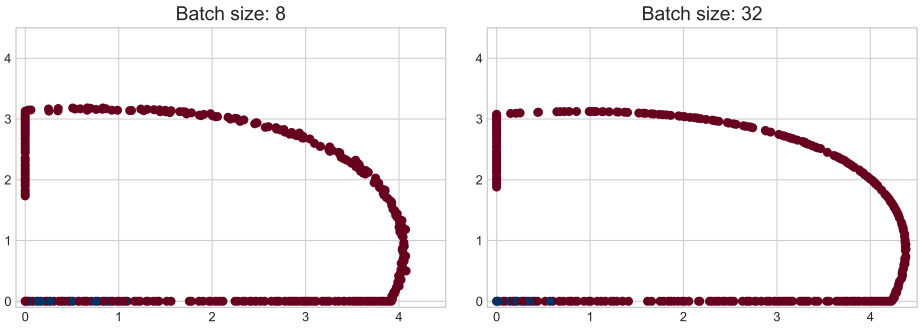


Figure 4.13: Mapping of points with ReLU as the activation function with varying batch size of training data



4.2.2 Analysis of treatment of different regions of the spiral and the disentanglement process

One thing that cannot be inferred from the previous results of how the networks maps the data in the process of learning it is how it affects different "regions" of the spiral. Due to the spiral dataset being based on Archimedean which have constant gradient at any point unlike, for example, logarithmic spirals (see Section 2.3.1), there are no obvious sensitive regions.

The initial hypothesis, intuitively, was that the points further away from the spiral would be "moved" further away from their original position then the ones closer the centre (centre being the $(0, 0)$ point).

With this in mind Euclidean distance was measured between the original points in the training dataset and their final mapping after training.

Figure 4.14: Mapping of points with ReLU as the activation function with varying batch size of training data shown on separate graph for each of the two spirals

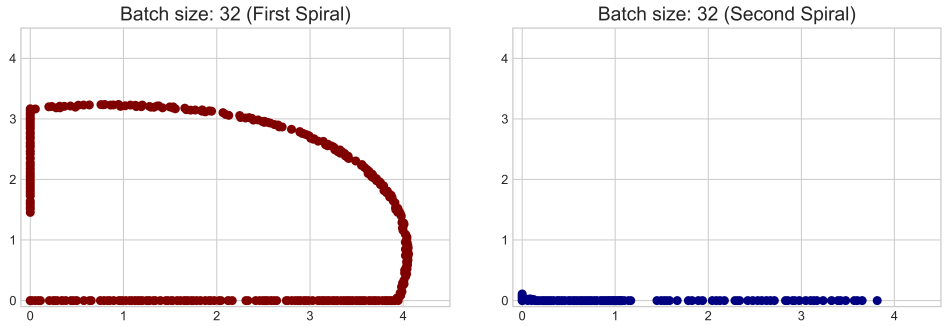
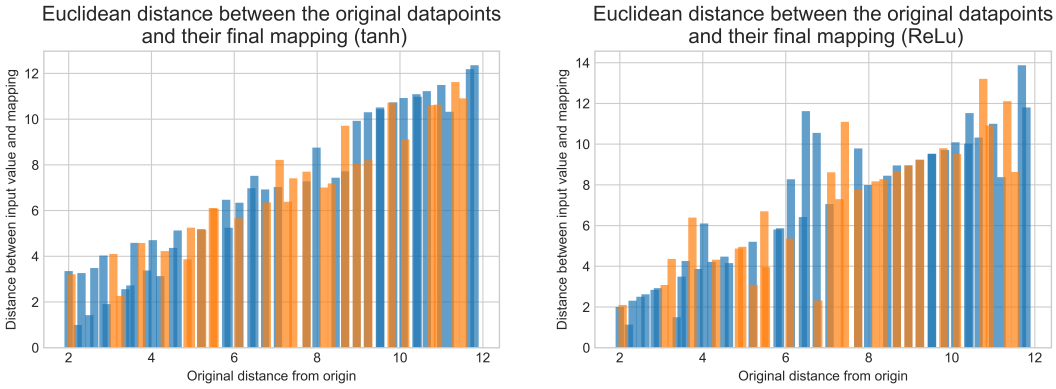


Figure 4.15: Euclidean distance between the original position of data points and their final mapped position (values captured for 10% of the training data) with orange and blue representing the two different spiral classes. While the plots are not identical values for both ReLU and tanh rise in a similar, linear fashion.



The results can be seen in Figure 4.15 with the two colours representing the two spirals which make the spiral dataset.

Despite the entirely different way the points are mapped, as seen in Sections 4.2.1 and 4.2.1, both NNs are similar in some regard in how the regions of the spiral are treated. It would also imply that the initial hypothesis was correct and the spiral is disentangled, in a sense, from the outer edges towards centre (that is the outer points are moved more and the centre remains less changed).

4.2.3 Summary

This section has investigated the nature of changes in classification depending on the choice of activation function. While both can achieve similar accuracy for sufficiently large networks the way points are moved to be linearly separable is significantly different. That said both types of NNs treat regions of the dataset similarly.

One of the more interesting question in the face of the observed behaviour is whether there

are noticeable downsides to such unequal treatment, as appearing in ReLU.

An area in which unequal treatment of two classes may impact the performance is the case of test data that varies slightly from the training dataset - robustness to random perturbations in data. Another question would be whether there is any difference in how the NNs will respond to adversarial attacks.

4.3 Robustness

Robustness in the context of NNs is a broad term meaning the program's ability to cope with with erroneous input. This definition however is very general and leads to many interpretation of how exactly robustness is measured in NNs hence it is important to specify the exact definition used whenever talking about NN robustness. Using the definitions presented in [55] which compares the different methods the one used in this work is Classification Robustness.

Definition 15 (Classification Robustness)**.** Let $\hat{x}$ be the input and x denote the input withing ϵ distance from it (ϵ-ball). Then $f(x)$ denotes the output and y the class label for original data for $\hat{x}$.

$$CR(\epsilon) \triangleq \forall x : ||x - \hat{x}|| \leq \epsilon \implies argmax f(x) = y \tag{4.3}$$

Intuitively classification robustness seeks to achieve a network that guarantees successful classification as long as the example is not further away then ϵ from the original data point.

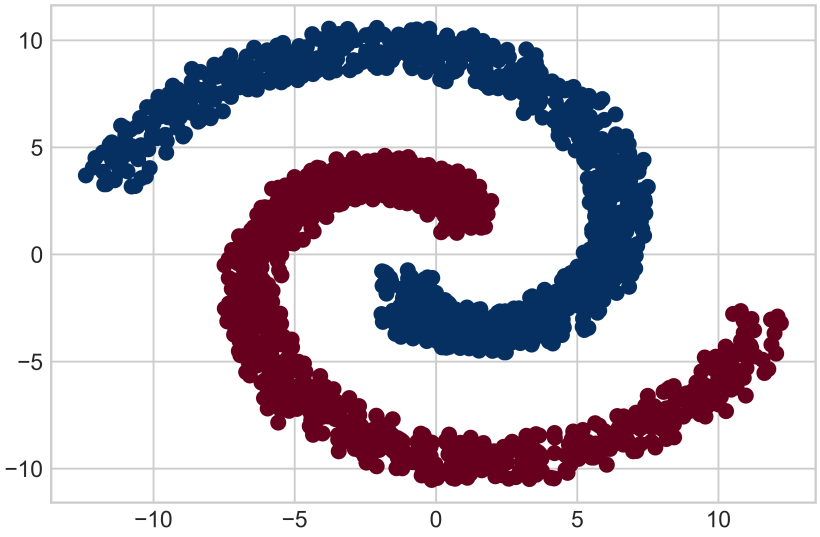
While it can be challenging to pick the right measure and care must be taken to specify the definition of robustness used in a particular experiment it is a good measure to compare the performance of NNs using tanh and ReLU. This will allow to verify whether the difference in their class treatment has any impact on the performance apart from accuracy (which was already considered before).

4.3.1 Hypothesis 4 - Part 1: The choice of activation function impacts robustness to random noise

In order to test the performance in such case a test set was designed where data points would all be slightly moved from the spirals themselves (by a small, randomised distance) while the training set remained unchanged.

Predictably a more varied test set does induce a small drop in accuracy. The variance of a set is controlled by setting a maximum value by which a single point can be moved along both

Figure 4.16: Modified spiral dataset with added noise



of the axis and as long as it was not significantly larger then the distance between the spirals (for definition see Section 2.3), which would make the test data lose shape of the spirals entirely, the network didn't perform significantly worse.

For smaller network from this Section's experiments accuracy stayed in the ranges $76.67 - 73.93\%$ for tanh and $81.97 - 75.30\%$for ReLU. The same experiments have been repeated for the larger NNs that were used in Section 4.1 for weight matrix oriented experiments which resulted in accuracy in $98.94 - 94.24\%$ for tanh and $100 - 98.79\%$ for ReLU.

Therefore while tanh seems to achieve slightly worse results overall the performance of networks using both activation functions is comparable.

4.3.2 Hypothesis 4 - Part 2: The choice of activation function impacts adversarial robustness

One of the more common measures of testing a NN is to test its performance when using adversarial examples (see Section 2.4.6) - examples design specifically to mislead the network.

There exist multiple ways to generate such examples, mostly grouped into white-box and black-box methods - the first ones requiring some degree of access to the network architecture, the second assuming one can only see the input and output [56]. For all the experiments white-box methods were used as the goal was determining the performance against adversarial examples - not the security against the outside attacks of someone who does not have access to the network itself.

Fast Gradient Sign Method Fast Gradient Sign Method (FGSM) is arguably the simplest method of obtaining adversarial examples. In essence it works by adding noise in the direction that would maximise the loss function:

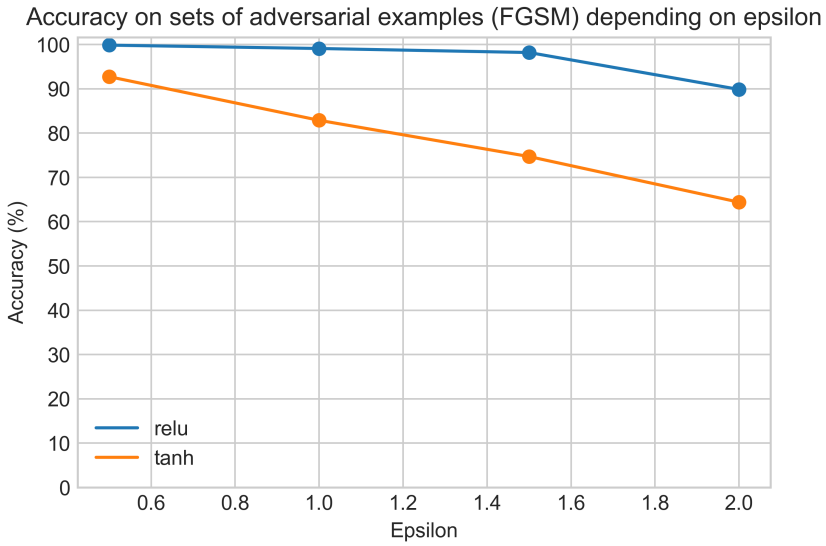
$$X_{adversarial} = X + \varepsilon sign(\bigtriangledown_X J(X, Y)) \tag{4.4}$$

where $\bigtriangledown_X J(X, Y)$ is the gradient of the loss function - its magnitude is not relevant for this method, only its sign and ε is a value that acts as a constraint.

As this method is most commonly used with image classification networks a sufficiently small ε would be used so that altering the original image would produce one that, to human eye, isn't visibly different (and yet is classified incorrectly by the network). However on the spiral dataset this is not a concern - due to the very mathematical nature of the dataset any change in position outside of the spiral would be easily spotted by a human so eliminating this is not a goal.

Reasonably in the spiral's case the ε should be smaller than the distance between spirals (as was the case for random noise) - otherwise a point could simply be mapped to the other spiral which would guarantee it being misclassified. In the case of these experiments the distance between spirals is 2 (see Figure 4.17) and each point of the test set was separately generated as an adversarial example.

Figure 4.17: Accuracy on networks with tanh and ReLU on sets of adversarial examples (FGSM) depending on value of epsilon



There is a much more noticeable drop in performance between tanh and ReLU here then

there was in previous experiments with random noise.

Basic Iterative Method Basic Iterative Method (BIM) is sometimes also called Iterative FGSM which is a good description of the method - since simple FGSM is quite simple and easy to defend against (although in the case of spiral dataset it is enough to cause significant issues, as shown before) - BIM runs an iterative optimisation of FGSM with a smaller epsilon [57]. It is quite a slow method but since the spiral dataset is relatively small this was not a major concern. It can be further improved with a method called Madry et. al's Attack [58] which has shown that the performance is better if the algorithm starts in a random point inside the ε norm ball - that is instead of starting at a point belonging to a spiral a random point no further away then ε in any direction is chosen.

To make sure that the final adversarial example stays within the ε norm ball a clipping function is introduced:

$$Clip_{X,\varepsilon}\{X'\} = min\{X + \varepsilon, max\{X - \varepsilon, X'\}\} \tag{4.5}$$

Then with X denoting the original input, N the number of iterations and $X_0^{adversarial} = X$ the adversarial example is computed as follows:

$$X_{N+1}^{adversarial} = Clip_{X,\varepsilon}\{X^{adversarial_N + \alpha sign(\bigtriangledown_X J(X_N^{adversarial}, Y))}\} \tag{4.6}$$

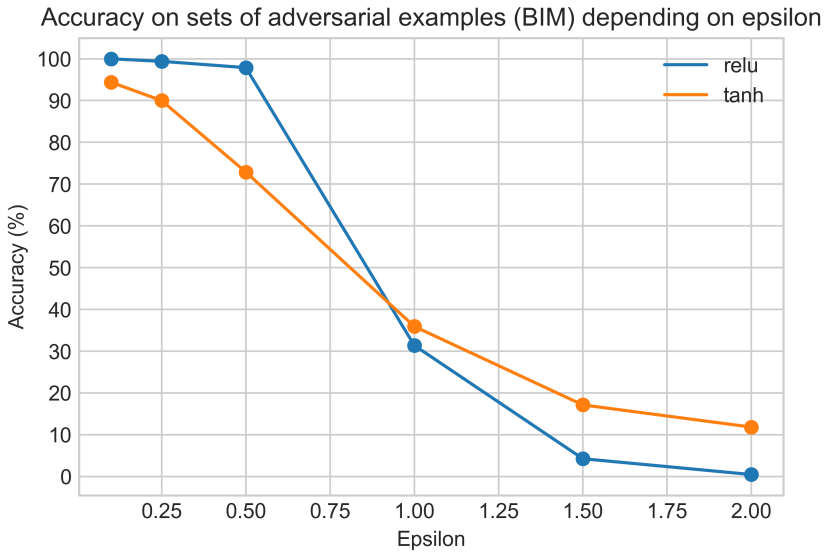
Where α is another parameter, smaller then ε but sufficiently large that with all iterations it could reach the edges of the ε ball - empirically its value was chosen as $0,5 * \varepsilon$.

It's easy to see that with better method for generating the adversarial attacks the performance of all tested NNs dropped dramatically.

The severe drop for $\varepsilon = 2$ is understandable as that allows for ideal adversarial example that can get mapped to a position exactly within the other spiral - and it can be noted that while it is possible BIM does not always return it. In usual circumstances an attack would strive to remain as close to original as possible to remain unnoticed. Including larger values of ε has however made it possible to compare the performance of BIM and FGSM - the simpler method was not enough to generate examples that would trip the trained networks.

However while BIM is much better at finding the adversarial examples (as expected) NNs - especially ReLU NNs - are still performing decently for small ε. Especially ReLU which sees only

Figure 4.18: Accuracy on networks with tanh and ReLu on sets of adversarial examples (BIM) depending on value of epsilon



a 3% accuracy drop for $\varepsilon \leq 0,5$ which means that NN is quite robust in small neighbourhoods.

4.3.3 Further analysis involving decision boundary and data complexity

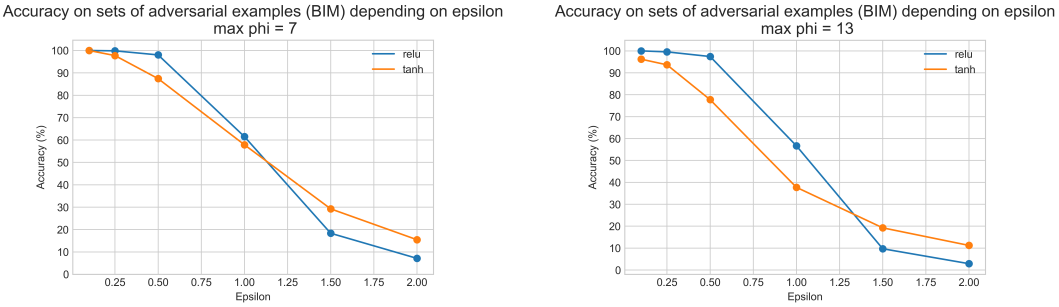
One relationship the previous experiments have not covered is what happens to robustness with changing complexity of the dataset. Intuitively the more complex the dataset the harder classification is - and so it would follow that it would suffer more when it comes to robustness.

This has been observed to be true for both tanh and ReLU based networks with worse accuracy being observed for NNs trained and tested on spirals with higher complexity (see Figure 4.19). The observed were relatively small but interestingly drop the most for small values of ϵ indicating that adversarial examples closer to the original entry in the dataset are more often misclassified in NNs using complex datasets. This is particularly important as one of the goals of a good adversarial example is for it to be as close to original data as possible to make it harder to detect.

However more experiments would need to be run to investigate this phenomenon in depth - in this case increasing the complexity of the dataset further results in the NNs failing to classify even the original data with dramatic fall of accuracy and so makes futher analysis complicated.

There is a known, although not fully understood yet, link between the decision boundary of a NN and its robustness. Some attempts have been made both to help analyse the decision boundaries and use them as a tool to gain information about the NN - this is made difficult by the fact that just generating examples and looking at the shape of the boundary is in itself a

Figure 4.19: Accuracy on networks with tanh and ReLu on sets of adversarial examples (BIM) depending on value of epsilon with varying complexity of the dataset



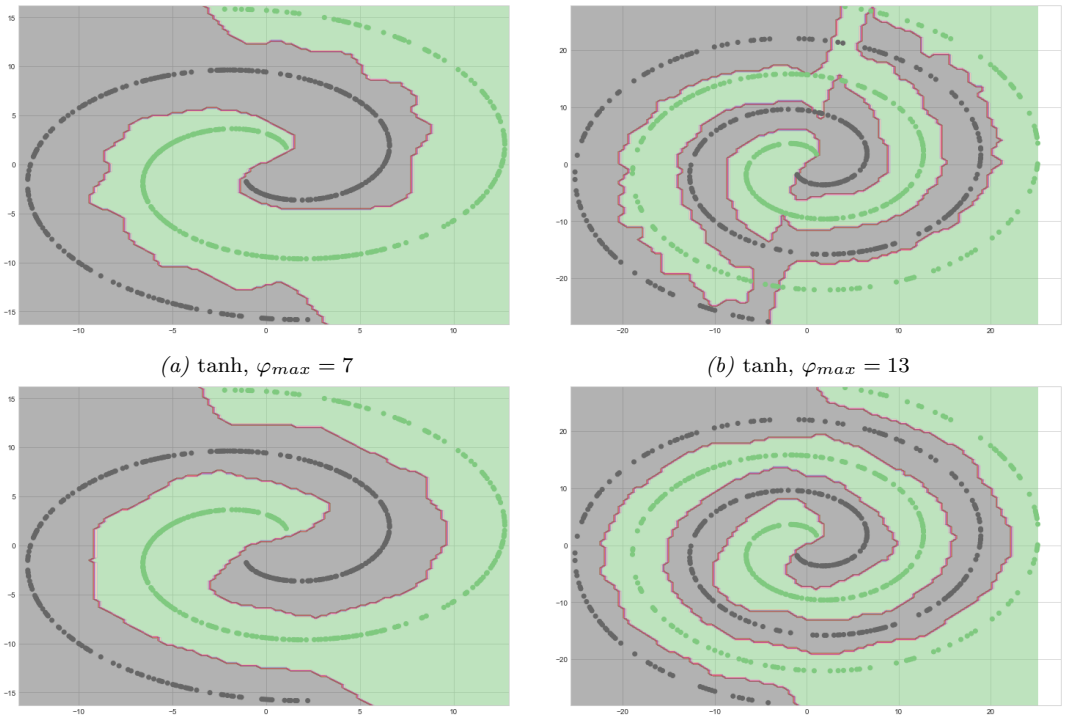
(a) NNs using dataset with $\varphi_{max} = 7$

(b) NNs using dataset with $\varphi_{max} = 13$

challenge for very complex NNs [59].

One of the already discovered properties is of particular interest here - that there is direct connection between the ability of the NN to generalise to new data (including perturbed data similar to the ones used in above experiments) and the complexity of its decision boundary [60]. The "simpler" it is - the less "jagged" the edges, the closer to the ideal straight line it is - the better the NN generalises.

Figure 4.20: Decision boundary for NNs using tanh and ReLU activation functions, trained and tested on datasets with different complexity (entanglement)



(a) tanh, $\varphi_{max} = 7$

(b) tanh, $\varphi_{max} = 13$

(c) ReLU, $\varphi_{max} = 7$

(d) ReLU, $\varphi_{max} = 13$

Looking at the decision boundary generated for NNs tested and trained on datasets with varying complexity (see Figure 4.20) there is some indication that the boundary does become

more complex for datasets with higher entanglement. Furthermore some evidence was found that adversarial examples using smaller ϵ (closer to the original entry in the dataset) seemed to be

However there can be no decisive conclusion reached in regards to this since there is no defined metric used to measure the complexity of the decision boundary that would be necessary for any further analysis, making this an interesting direction of further research.

4.3.4 Summary

It can be seen that both networks using ReLU and tanh deal fairly well with small random perturbations and at least some of the simpler adversarial attacks.

ReLU based networks achieve higher accuracy in most cases which was not an unexpected outcome considering how the classes were mapped in this type of networks (see Figures 4.11, 4.13) - thanks to one class being progressively squeezed into a line or point and how separate the classes get it is much harder to cause misclassification. However, as mentioned, this approach has its downsides and could prove more volatile with larger amount of classes.

NNs using tanh performed overall worse in the matters of accuracy however it did get a higher score for some of the examples using BIM method (see Figure 4.18) - it does not always perform worse then ReLU when it comes to choice of activation function.

The experiments disprove the Hypothesis 5 - "The choice of activation function impacts robustness", be it part 1 or 2 relating to random noise and adversarial examples - as there does not seem to be a clear relation between the choice of activation function and the performance of the network. While tanh seems to tend to perform worse the results were not conclusive.

Furthermore some connection in indicated between drop of robustness and the increase in complexity of the dataset the NN uses (see Section 4.3.3) but while possible cause has been suggested further research is needed to reach any definitive results.

Chapter 5

Design and Implementation

This chapter will go into more depth about the implementation used in the experiments and the design choices made in the process. All of the methods and classes developed for the purposes of this work will be released alongside it as an open-source tool (see Section 3.3) that is avilable on `https://github.com/ndslusarz/Slusarz_BSc_thesis` and so this section will also endeavour to describe all the provided functionalities.

All of the code was developed in Python 3.7.

5.1 Neural network implementation

At the core of this project is the neural network implementation used in all of the experiments.

As mentioned before (see Section 3.6) PyTorch was used as the main tool to implement the NNs. It was chosen because while there exist other machine learning frameworks, such as scikit-learn, PyTorch is the one most commonly used in other papers and it gives a bigger measure of control and access when it comes to inner parameters of NNs.

The network implemented is a multilayer, feed-forward NN (see Section 2.4.4) with user having control over the number and size of layers.

As the problem of this project is a binary classification problem (the spiral dataset has only two classes) a choice had to be made between functions softmax and sigmoid (for definitions see Appendix C) which are commonly used for tasks of this type. Mathematically sigmoid and softmax are equivalent for the case with two classes - in this sense sigmoid is a particular simplified case of softmax which can also be used for multi-class classification tasks. Although it should be noted that some experiments on neural networks built with those functions and

their respective loss functions - as implemented by PyTorch - cross-entropy loss and binary cross-entropy loss have yielded slightly different accuracies suggesting that the implementation is not actually fully equivalent. After some debate softmax was used since it can be later scaled to work for problems with more classes giving this implementation more versatility.

All important training parameters can be customised - the length of training (epochs), the batch-size, activation function for hidden layers (currently set up for tanh and ReLU but can be easily expanded for any other PyTorch function) and optimiser.

For the sake of convenience the network takes data in the form of numpy arrays and has internal methods to convert them to the tensor format PyTorch uses.

Class methods Most of the functions involving obtaining data are implemented within the class as to have full access to NN parameters.

The core of many of the experiments (see Section 4.1) is the method to obtain weight matrices for all layers as well as their eigenvalues for specified epochs of training (as it can be beneficial to compare them at various stages of the learning process) as well as a method to visualise it as a heatmap - while this has very limited functionality it can give some indication to the values in the network.

For the sake of quality control the tool provides a method to check is the network suffers from gradient vanishing problem (see Section 4.2)

Furthermore the NN class provides implementation of two adversarial example generation algorithms - FGSM and BIM, used in Section 4.3.2.

Small neural network for point mapping After some consideration a small, altered class was created for the purpose of the experiments involving point mapping (see Section 4.2.1). This network has a fixed size of just one hidden layer with two neurons.

It was separated from the main class as for the purposes of experiments from Section 4.2.2 it was vital to keep track of the order of singular data entries during training and testing (as the data is shuffled in both of those cases) as they involve considerations on particular areas of the original spiral.

Furthermore said experiments were possible only due to the size of the network and as such could potentially cause confusion or odd behaviour if places in the more versatile NN class.

The particular functions involved are `plot_mapping` which visualises how the data is mapped as a 2D graph and `plot_distance` which handles the Euclidean distance between the final

mapping of the data and its original position as well as visualises the results.

Other functions Other functions used for the remaining experiments are grouped in files by their general use - with separate collection of methods pertaining to weigh matrices, eigenvalues and norms and another for methods related to accuracy and performance comparison.

Accuracy of a NN is the first measure of comparison used in most cases, before any more in depth investigation is used. As this work was interested in comparison of NNs with different activation functions a variety of methods exist to visualise the difference in their accuracy depending on complexity of the dataset, size of the network or adversarial example used in testing.

The tool provides quite a few methods pertaining to weight matrix analysis. Firstly, after the weight matrix is obtained there is a choice of methods to extract its Frobenius norm, eigenvalues, their norms or number of complex values for various layers and epochs of training. This enables investigation of the networks used in many different states of training.

An overreaching function is provided that integrates most of the available functionalities, giving a comparison of Frobenius norm as well as eigenvalues for all layers at the end of training. This provides a good starting point that can be modified to facilitate other experiments.

5.2 Spiral dataset implementation

The spiral dataset (for definition of parameters see Section 2.3) is implemented as a class of its own. Due to time constraints projection into higher dimension was not investigated in experiments hence the codimension property is constant. Distance between spirals and entanglement, the two other properties of the dataset, can be passed as arguments to control the dataset.

The dataset also provides a method that allows for adding random noise.

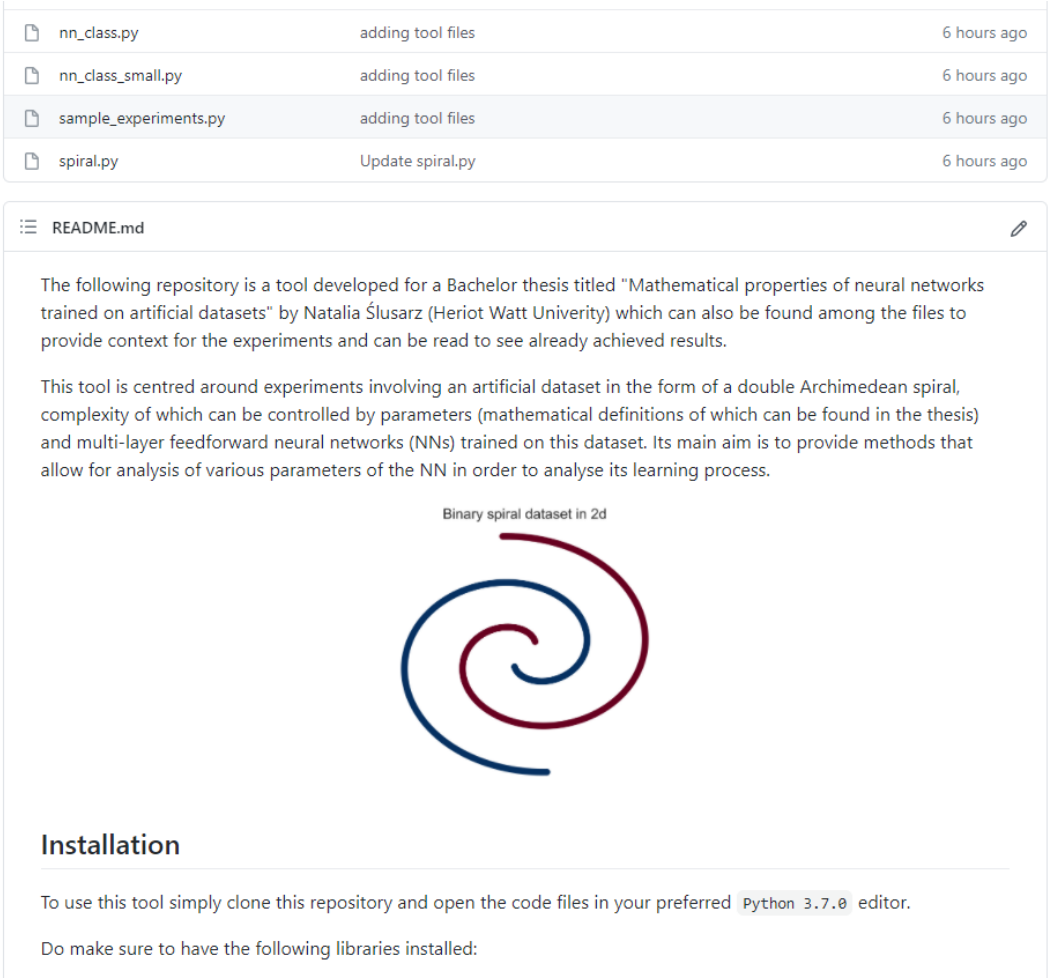
5.3 Tool

The final tool produced during this work is an open source repository containing all the functionalities mentioned above that is be made available online.

It is be fully available on GitHub as a public repository at `https://github.com/ndslusarz/Slusarz_BSc_thesis` with an open source license (Apache License 2.0) for use other projects.

The tool is be accompanied by a README file that will provide a characterisation of

Figure 5.1: Screenshot of the tool alongside part of the README file available at the GitHub repository linked before



general structure of the tool, including description of all available classes and methods, as well as instructions on the tool's installation and use. The code itself will be appropriately commented to explain the functionalities of all methods, including visualisation methods, to new users. Furthermore to aid this purpose a file with some sample experiments using various methods has been included.

Care was taken to separate the methods by the theme and purpose of experiments they can be used with (for example grouping all methods related to weight matrices, their eigenvalues or Frobenius norm with their corresponding visualisation functions in one file). This should provide an easy way of locating the functionalities of interest.

Chapter 6

Summary

This section will summarise the results of this work as well as discuss their limitations and future work that can be undertaken.

6.1 Conclusions

Starting with one of the first goals stated as motivation for this research - the improvement of theoretical understanding of neural networks - this project took care to include the theoretical basis for the experiments as well as investigate possible explanations for the achieved results. It was vital to thoroughly explain and understand the classification process of the NN - looking at it as a change of variables (see Section 4.1) - as it leads to weight matrix as the core of the decision process and is the motivation for its analysis.

Overall this project adds to the understating of the relationship between certain qualities of the weight matrix of a NN and its performance as well as the data distribution. Furthermore the results discover vast difference in the shape of data transformation during training that NNs perform depending on activation function - this indicates that there exist deeper differences between the functions and the way they could influence the NNs performance. A thorough search of the relevant literature did not yield works that have performed a similar in-depth comparison of particular activation functions.

Moving on to more specific conclusions, beginning with Hypothesis 1 and 2 (Sections 4.1.1, 4.1.2), this thesis investigated the connection between the complexity of the training data set and the weight matrix of the NN using Frobenius norm and eigenvalues. It was observed that the values for both of the variables keep rising for the networks using more complex datasets

which would suggest a less "smooth" detangling of the spiral dataset.

This set of experiments has additionally shown a difference in results and behaviour of the networks depending on the activation function used (see Figures 4.4, 4.5, D.1).

This difference between tanh and ReLU was further explored in Hypothesis 3 and 4 (Section 4.2) through more in-depth analysis of the way the spirals from the spiral dataset get disentangled by investigating how the input points are mapped in an attempt to achieve a linearly separable dataset. Experiments were designed to observe the tendencies at various intervals during training time and later varied the batch size for the training data.

As can be seen from Figures 4.10, 4.11, Hypothesis 3 was shown to be correct with tanh treating both classes equally during training while ReLU exhibiting a tendency to collapse one of the spirals into a point or line in its attempts to make the classes linearly separable.

However the investigation of the last hypothesis (Hypothesis 4) has shown that there exists some measure of similarity between both types of networks. Euclidean distance was measured between the original point and its final mapping in an attempt to check whether there were any sensitive regions in the input - however, as predicted due to the nature of the dataset which, being based on Archimedean spirals, has no obvious sensitive regions, both types of networks exhibited similar behaviour. The tendency was to map points further away from their original position the further they originally were from the centre (in a way "disentangling" the spiral from outer to inner edges).

Lastly in Section 4.3 this work has focused on determining the quality of the networks used in previous experiments by measuring their robustness to random noise as well as multiple adversarial attacks (generated by FGSM and BIM algorithms). Networks using both activation functions have proven decently robust against random noise. In case of adversarial examples ReLU has achieved better results (there are however still unexpected risks that come with one of the classes being deprecated during training investigation of which is left for future work).

6.2 Limitations

This work, however, is subject to several limitations. Firstly this area of study, due to its highly theoretical nature, is not yet very well developed resulting in lack of reference material for some of the experiments such as the investigation of class treatment and its consequences.

Due to specificity of the dataset used in this work and the time constraints which prevented

further experiments on different datasets the results achieved are not generalised. Hence there is limited application for the achieved results until more work can be done to relate those findings to real-world datasets.

6.3 Evaluation

As described in Section 3.8 the evaluation strategy is split into multiple stages to allow for better analysis of the different parts of this thesis. The evaluation of the success of this thesis will be done according to specified measures of success.

Implementation	see Chapter 5
T1	Yes
T2	Yes, along with additional methods
T3	Yes
T4	Yes
Experimentation	see Chapter 4
E1	Yes
E2	Yes
E3	Yes
E4	Yes
E5	Yes
Reasoning	see Sections 2.5, 6.1 and Chapter 4
R1	Yes
R2	Yes
R3	Yes

As can be seen from the table above all of the measures intended to evaluate this thesis against were fulfilled and so this project can be considered successful.

6.4 Future work

Possible directions of future expansions on this topic include:

Projection into higher dimensions Since all the experiments have used 2-dimensional data it would be beneficial to test if similar results can be reproduced on similar datasets projected

into higher dimensionalities. The spiral dataset has codimensionality as a defined parameter (see Section 2.3) however it has not been used in this work as due to time restrictions no experiments involving higher dimensionalities were performed.

Investigation of other activation functions The experiments performed have only taken into account ReLU and tanh as activation functions since they are both considered good choices for this type of a task (as mentioned already in Section 4.1.2). It would be interesting to verify whether similar patterns or entirely new changes can be observed when using other activation functions, especially when it comes to experiments considering the nature of point mapping (see Section 4.2).

Expanding the experiments to other datasets All of the experiments have been done with the use of artificial dataset defined for the purposes of this work (the spiral dataset defined in Section 2.3). Future work could involve reproducing the experiments on more complex artificial datasets with the final goal being the usage of real-world datasets (such as MNIST [61] or GTRSB [7])

Defining different properties of NNs The words "properties" is used in a very wide context in NN analysis - from weight matrices of the NN to robustness as a property itself. While some were analysed in the course of this work other interesting properties could be defined and analysed.

Influence of activation functions in NN verification While this thesis has investigated the robustness of the NNs for spiral dataset with regards to activation function (see Section 4.3) there remains the matter of NN verification. Further work in this direction could uncover more consequences of the differences in the classification process.

Use of property driven training There exist multiple definitions of robustness, each involving different properties of the NN. A NN can be trained with a specific robustness property in mind (for example techniques such as data augmentation [62] or adversarial training [32]). There has been work done to compare those different definitions [55] and experiments could be run on the spiral dataset to extend and support the results from this work.

Bibliography

[1] W. McCulloch and W. Pitts, "Mcculloch, w.s., pitts, w. a logical calculus of the ideas immanent in nervous activity.," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115—-133, 1943.

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 5, pp. 359–366, 1989.

[3] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[4] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," vol. 21 of *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, (Galleria 2, 6928 Manno-Lugano, Switzerland), International Joint Conference on Artificial Intelligence, 2011.

[5] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, JMLR Workshop and Conference Proceedings, 13–15 May 2010.

[6] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The german traffic sign recognition benchmark: A multi-class classification competition," in *The 2011 International Joint Conference on Neural Networks*, pp. 1453–1460, 2011. `http://benchmark.ini.rub.de/`.

[7] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Computer Aided Verification*, vol. 10426 of *Lecture Notes in Computer Science*, pp. 3–29, Cham: Springer International Publishing, 2017.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[10] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, *et al.*, "Knowledge discovery and data mining: Towards a unifying framework.," in *KDD*, vol. 96, pp. 82–88, 1996.

[11] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, pp. 75–78. Burlington: Morgan Kaufmann, 2011.

[12] J. N. Lester, T. Muskett, and M. O'Reilly, "Naturally occurring data versus researcher-generated data," in *A Practical Guide to Social Interaction Research in Autism Spectrum Disorders* (J. N. Lester, T. Muskett, and M. O'Reilly, eds.), ch. 3, pp. 87–116, Palgrave Macmillan, 2017.

[13] R. D. Reed and R. J. Marks, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, p. 269. MIT Press, 1999.

[14] H. Yanai, *Projection Matrices, Generalized Inverse Matrices, and Singular Value Decomposition by Haruo Yanai, Kei Takeuchi, Yoshio Takane.*, pp. 125–150. Statistics for Social and Behavioral Sciences, New York, NY: Springer New York, 1st ed. 2011. ed., 2011.

[15] Mei Tian, Si-Wei Luo, and Ling-Zhi Liao, "An investigation into using singular value decomposition as a method of image compression," in *2005 International Conference on Machine Learning and Cybernetics*, vol. 8, pp. 5200–5204 Vol. 8, 2005.

[16] J. M. Lee, *Introduction to Riemannian Manifolds by John M. Lee.*, pp. 9–13. Graduate Texts in Mathematics, 176, 2nd ed. 2018. ed., 2018.

[17] C. Fefferman, S. Mitter, and H. Narayanan, "Testing the manifold hypothesis," *Journal of the American Mathematical Society*, vol. 29, 10 2013.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, pp. 195, 290, 326. MIT Press, 2016. `http://www.deeplearningbook.org`.

[19] *Matrix computations / by Gene H. Golub and Charles F. Van Loan.*, p. 55. North Oxford Academic, 1983.

[20] Z. Jiang, J. Zhou, and H. Huang, "Relationship between manifold smoothness and adversarial vulnerability in deep learning with local errors," *Chinese Physics B*, 2020.

[21] AdiJapan, "Archimedean spiral," Oct 2009. `https://commons.wikimedia.org/wiki/File:Archimedean_spiral.svg`.

[22] V. Govorov and A. Kharshiladze, "Codimension. encyclopedia of mathematics." `http://encyclopediaofmath.org/index.php?title=Codimension&oldid=43514`, November 2018.

[23] D. Olive, *Linear regression*, pp. 1–9. 04 2017.

[24] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.

[25] G. Zhang, "What is the kernel trick? why is it important?." `https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d`, November 2018.

[26] M. Hein and O. Bousquet, "Hilbertian metrics and positive definite kernels on probability measures," in *AISTATS*, 2005.

[27] L. Le, J. Hao, Y. Xie, and J. Priestley, "Deep kernel: Learning kernel function from data using deep neural network," in *2016 IEEE/ACM 3rd International Conference on Big Data Computing Applications and Technologies (BDCAT)*, pp. 1–7, 2016.

[28] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[29] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[30] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, (Berlin, Heidelberg), p. 9–50, Springer-Verlag, 1998.

[31] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, pp. 1–98, 06 2017.

[32] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

[33] A. Athalye, N. Carlini, and D. A. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *CoRR*, vol. abs/1802.00420, 2018.

[34] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks," *ArXiv*, vol. abs/1803.06567, 2018.

[35] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow, "The relationship between high-dimensional geometry and adversarial examples.".

[36] N. Narodytska, "Formal verification of deep neural networks," in *2018 Formal Methods in Computer Aided Design (FMCAD)*, pp. 1–1, 2018.

[37] H. Anton and C. Rorres, *Elementary Linear Algebra: Applications Version*.

[38] C. H. Martin, "Rank collapse in deep learning," Oct 2018.

[39] L. Schott, J. Rauber, W. Brendel, and M. Bethge, "Robust perception through analysis by synthesis," *CoRR*, vol. abs/1805.09190, 2018.

[40] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[41] F. Leisch and E. Dimitriadou, *mlbench: Machine Learning Benchmark Problems*, 2021. R package version 2.1-3.

[42] T. W. Rinker, *wakefield: Generate Random Data*. Buffalo, New York, 2018. version 0.3.3.

[43] T. Wiatowski and H. Bölcskei, "A mathematical theory of deep convolutional neural networks for feature extraction," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1845–1866, 2018.

[44] S. Mallat, "Group invariant scattering," 2012.

[45] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks," *CoRR*, vol. abs/1803.06567, 2018.

[46] D. Guidotti, "Enhancing neural networks through formal verification.," in *DDC@ AI* IA*, pp. 107–112, 2019.

[47] F. Ponulak and A. J. Kasinski, "Generalization properties of spiking neurons trained with resume method.," in *ESANN*, pp. 629–634, Citeseer, 2006.

[48] F. Ponulak, "Analysis of the resume learning process for spiking neural networks.," *International Journal of Applied Mathematics & Computer Science*, vol. 18, no. 2, 2008.

[49] "End user license agreement - anaconda® individual edition - anaconda." `https://docs.anaconda.com/anaconda/eula/`.

[50] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, p. 1–1, 2020.

[51] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr 2011.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[53] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions.," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[54] "Effect of batch size on training dynamics." `https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e`.

[55] M. Casadio, M. Daggitt, E. Komendantskaya, W. Kokke, D. Kienitz, and R. Stewart, "Property-driven training: All you (n)ever wanted to know about," 2021.

[56] A. Kurakin, I. J. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie, J. Wang, Z. Zhang, Z. Ren, A. L. Yuille, S. Huang, Y. Zhao, Y. Zhao, Z. Han, J. Long, Y. Berdibekov, T. Akiba, S. Tokui, and M. Abe, "Adversarial attacks and defences competition," *CoRR*, vol. abs/1804.00097, 2018.

[57] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *CoRR*, vol. abs/1607.02533, 2016.

[58] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[59] H. Karimi, T. Derr, and J. Tang, "Characterizing the decision boundary of deep neural networks," *CoRR*, vol. abs/1912.11460, 2019.

[60] S. Guan and M. Loew, "Analysis of generalizability of deep neural networks based on the complexity of decision boundary," *arXiv preprint arXiv:2009.07974*, 2020.

[61] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. `http://yann.lecun.com/exdb/mnist/`.

[62] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, pp. 1–48, 2019.

[63] V. I. Bogachev, *Measure Theory*, pp. 3–9. Measure Theory ; 1, 1st ed. 2007. ed., 2007.

[64] E. Suli and B. S. Jovanovic, *Analysis of Finite Difference Schemes: For Linear Partial Differential Equations with Generalized Solutions*, vol. 46 of *Springer series in computational mathematics*. London: Springer London, Limited, 2013.

[65] A. A. Kodiyan, "An overview of ethical issues in using ai systems in hiring with a case study of amazon's ai based hiring tool," 11 2019.

[66] P. Grother, M. Ngan, K. Hanaoka, N. I. of Standards, and T. (U.S.), *Face Recognition Vendor Test (FVRT): Part 3, Demographic Effects*, pp. 1–12. NISTIR; NIST IR; NIST interagency report; NIST internal report, National Institute of Standards and Technology, 2019.

Appendix A

Support

To define support (or topological support) both measure and $\sigma - algebra$ need to be explained.

Measure of a set X can be informally described as a function that assigns a non-negative number to each subset of X. The motivation behind it is a more precise notion of "size" of each of the subsets [63].

Definition 16 ($\sigma - algebra$)**.** $\sigma - algebra$ of X is a collection Σ of subsets of X that includes X itself, and is closed under complement (if A is in Σ then X
A is in Σ) as well as under countable unions (if A_1, A_2 are in Σ then $A_1 \cup A_2$ is in Σ).

Then a measurable space (X, Σ) referrs to a set and its subsets that form a $\sigma - algebra$ where a measure can be defined [63].

Finally support can be defined [64]:

Definition 17 (Support)**.** Let $\mu : \Sigma \to [0, +\infty]$ be a measure on a measurable space (X, Σ). Then the support of μ is a subset of the $\sigma - algebra$:

$$supp(\mu) = \overline{\{A \in \Sigma \| \mu(A) > 0\}} \tag{A.1}$$

where the overbar denotes set closure (closure of a set X is the smallest set containing X).

Appendix B

Jacobian

A Jacobian matrix (sometimes called a Jacobian) is a matrix of first order partial derivatives defined as follows:

Definition 18 (Jacobian matrix)**.** Let $f : \mathbb{R}^n \to \mathbb{R}^m$ such that its first order derivatives exist on $\mathbb{R}^n$. Then the Jacobian matrix is an $m \times n$ matrix of the form:

$$J = [\frac{\partial f}{\partial x_1} ... \frac{\partial f}{\partial x_n}] \tag{B.1}$$

The determinant of the Jacobian matrix is also sometimes referred to as a Jacobian therefore to avoid confusion both will be used with their full names.

Appendix C

Softmax and Sigmoid

Softmax is a function commonly used in machine learning classification problems with multiple classes - especially useful since it normalises output to a probability distribution over predicted output classes.

Definition 19 (Softmax)**.** Let $x_1, ..., x_n$ be inputs. Then the softmax function for i-th element is defined as:

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))} \tag{C.1}$$

Sigmoid is an activation function commonly used for binary classification problems since it normalises output to be between 0 and 1 making it simple to use for just two classes.

Definition 20 (Sigmoid)**.**

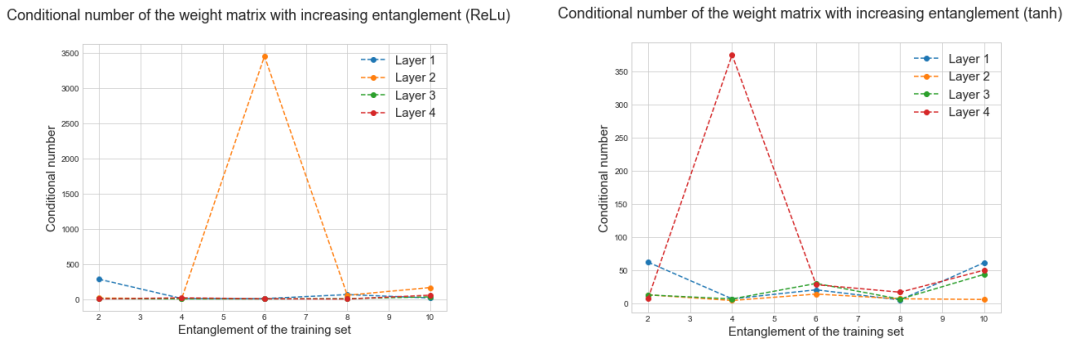
$$sigmoid(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{C.2}$$

Appendix D

Conditional Number

As mentioned in Section 4.1.2 similar experiment as the ones presented there, investigating the relationship between the conditional number of the weight matrix ($\frac{|\lambda_{max}|}{|\lambda_{min}|}$, with λ denoting an eigenvalue) and the entanglement of the training dataset for NNs using tanh and ReLU was run.

Figure D.1: Conditional number of weight matrices in different layers after the end of the training period with varying complexity of the training dataset. As can be seen in the plots below the conditional number rises extremely for some values.



(a) NNs with ReLu activation function

(b) NNs with tanh activation function

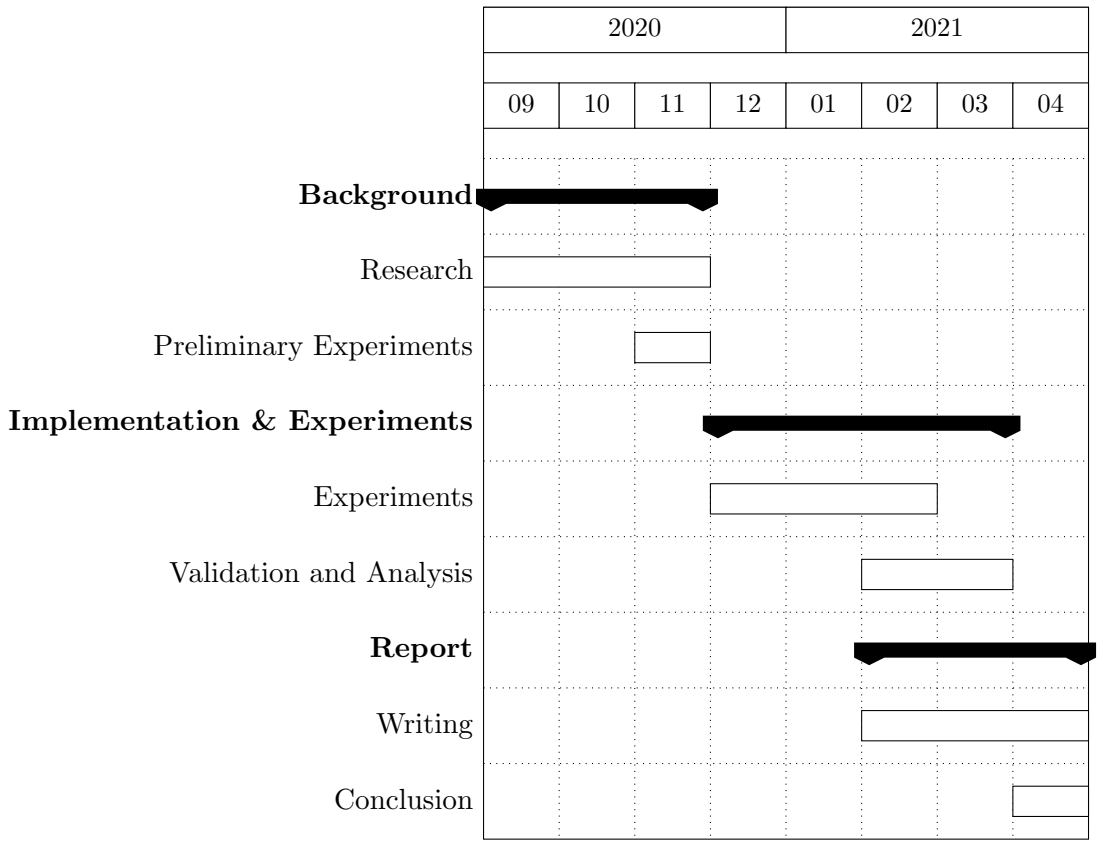
As can be seen in Figure D.1 there seems to be no clear trend with the conditional number rising drastically for some layers in some NNs. More experiments were run with slightly different datasets still achieving similar behaviour.

A more in depth investigation into the cause however reveals that the most likely cause is that for some weight matrices the value of $|\lambda_{min}|$ will be significantly smaller, effectively causing unproportional rise in conditional number for this entry. As such it was impossible to draw any clear conclusion about the state of the network based on this metric.

Appendix E

Project Management

E.1 Schedule



E.2 Risk Analysis

There are several issues that can arise and pose a risk to the project schedule unless recognised and properly mitigated if the need be. Below find the recognised risks as well as mitigation

strategies.

Risk	**Impact**	**Mitigation Strategy**
Hardware failure	Low	Ensure acess to backup hardware
Loss of data	High	Ensure regular back-ups to cloud
Issues with experiment design	Low	Go back to literature, ask people knowledgeble in the area
Result of experiment non-pertinent	Medium	Change parameters, go back to redesign the experiment
Delays	Medium	Have weekly meetings, allocate suitable time margin in all stages
Results difficult to conceptualise and explain	Medium	Consult literature or people knowledgeable in the area

E.3 Professional, Legal, Ethical and Social Issues

E.3.1 Professional and Legal Issues

All software used in this project is open-source. Individual Edition of Anaconda is used which is an open source distribution for Python and R under 3-clause BSD License (as well as all used libraries) to be found at [49]. License of Anaconda Individual Edition as well as all other software used in this project will be respected.

This project will follow ORBIT principles for responsible research and innovation (`https://www.orbit-rri.org/`). Furthermore all the code will be made available on GitHub as *open source* under the Apache 2.0 License once the project is complete.

The experiments will be well documented to ensure they can be easily replicated.

E.3.2 Ethical and Social Issues

The is no involvement of human subjects or any identifiable personal data in this project and therefore no risks involving handling sensitive information.

In some of the experiments real world data will be used; this can potentially lead to biased results as the data itself may be biased or incomplete. It is important to recognise that machine learning as a discipline with many real-world applications can be prone to perpetuating disadvantages or prejudices as was the case with for example the Amazon's AI based hiring tool [65] which in testing showed bias against women or NIST report showing that some facial recognition software (although the performance varied between different tested algorithms, some showing almost no bias) performs worse for certain demographics (e.g. More false positives for all women, people from West and East Africa or East Asia) [66]. The hope is that the increased understanding of AI and NNs will help eliminate such problems.