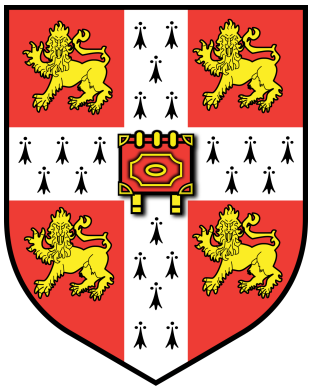
On Hessian Analysis in Neural Ordinary Differential Equations

Candidate Number: 8202W
Supervisor: Prof Pietro Liò

May 17, 2021



Department of Physics

University of Cambridge

Abstract

Neural Ordinary Differential Equations (Neural ODEs) are a class of deep learning architecture that, instead of specifying a sequence of hidden layers, model the derivative of a hidden state with a neural network. These are particularly suitable for learning the dynamics of physical systems. In addition, they typically possess fewer parameters than standard neural networks (10^3-10^4 vs 10^6-10^7). This means an analysis of the second-order information of the loss surface through full calculation of the Hessian matrix is computationally feasible. In this project, I first evaluated the possible approaches to Hessian calculation. Then, I developed multiple tools for this task, and obtained evidence for their accuracy and reliability. Finally, I used these to examine the Hessian of Neural ODEs and the effects of augmentation on the eigenspectrum. Hessian structure was seen to be consistent with previous findings made in regular neural networks, but was less stable during training. Augmented Neural ODEs found flatter minima and gave a smoother Hessian evolution than Neural ODEs. To my best knowledge, this is the first work on analysing the loss curvature of Neural ODE systems, and the techniques developed offer a promising method to examine a range of other architectural choices.

Acknowledgements

I would like to thank my project supervisor, Prof Pietro Liò, for his kind help and support. I am also sincerely grateful to my day-to-day supervisors Ben Day, Cris Bodnar and Alex Norcliffe, with whom I met regularly for the duration of this project. It has been a pleasure to work with them, and their insights have proved invaluable.

Plagiarism Declaration

Except where specific reference is made to the work of others, this work is original and has not been already submitted either wholly or in part to satisfy any degree requirement at this or any other university.

Contents

Graphical Abstract

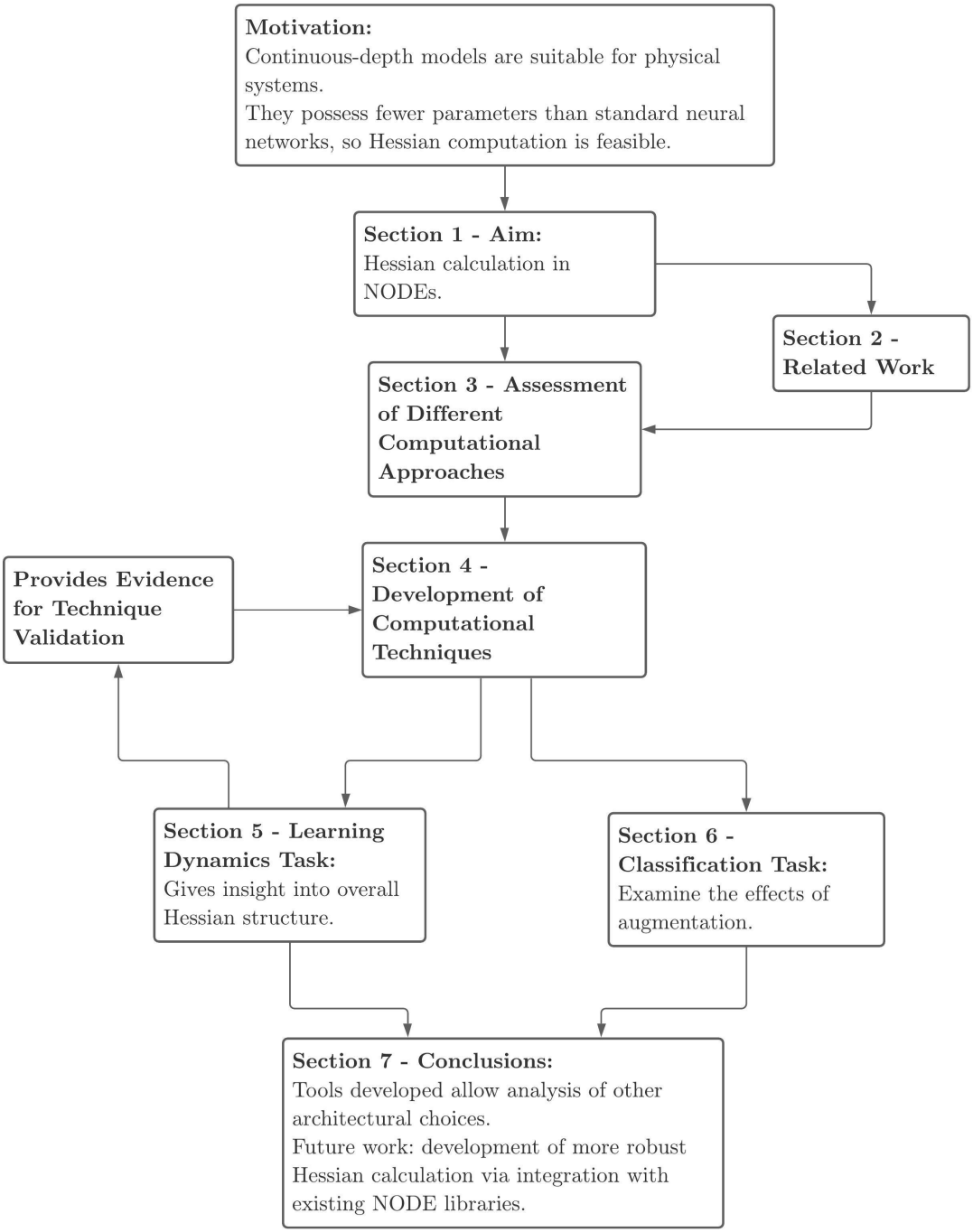


Figure 1: Report structure.

1 Background and Introduction

This section explains the concepts needed to understand the research conducted. It then describes the motivation and aim of this work.

1.1 Neural Networks

Neural networks have achieved success in a range of fields, including physics. For example, they have been used to compute Parton Distribution Functions of the proton from data obtained at the Large Hadron Collider [3, 8, 18].

In this project, I analyse the loss surface of supervised learning tasks by examining the second order information. To introduce this, suppose there exists a dataset $\mathcal{D} = \{(\boldsymbol{X}_i, \boldsymbol{Y}_i)\}_{i=1}^N$ where $\boldsymbol{X}_i \in \mathbb{R}^a$, $\boldsymbol{Y}_i \in \mathbb{R}^b$ are the input and label vectors respectively. Neural networks are function approximation algorithms that estimate the function $f : \mathbb{R}^a \to \mathbb{R}^b$ such that $f(\boldsymbol{X}_i, \boldsymbol{\theta}) \approx \boldsymbol{Y}_i$, where $\boldsymbol{\theta} \in \mathbb{R}^p$ is a parameter vector on which the network depends (and p is the number of model parameters). To quantify the difference between the network output and the desired value, a scalar *loss* function $L = \sum_{i=1}^N L(\boldsymbol{X}_i, \boldsymbol{Y}_i, \boldsymbol{\theta})$ is defined. For clarity, the dependence on $\boldsymbol{X}_i$ and $\boldsymbol{Y}_i$ is omitted in the remainder of this report. $L(\boldsymbol{\theta})$ can be thought of as defining a p-dimensional scalar field in parameter space, referred to as the *loss surface*. A commonly used example is the Mean Squared Error (MSE):

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \| f(\boldsymbol{X}_i, \boldsymbol{\theta}) - \boldsymbol{Y}_i \|^2 \tag{1}$$

The simplest type of neural network is the 'feedforward network', in which a discrete set of transformations act on the input vector:

$$\boldsymbol{x}_{n+1} = f(\boldsymbol{x}_n, \boldsymbol{\theta}_n), \qquad f(\boldsymbol{x}_n, \boldsymbol{\theta}_n) = \sigma(w_n \boldsymbol{x}_n + b_n), \qquad \boldsymbol{x}_0 = \boldsymbol{X}_i \tag{2}$$

where w_n is a matrix of *weights*, b_n is the *bias* vector and σ is a non-linear *activation function* that acts element-wise. Examples include the *sigmoid* function, $\sigma(x) = (1+e^{-x})^{-1}$, and the *softplus* function, $\sigma(x) = ln(1 + e^x)$. The vector of parameters $\boldsymbol{\theta}_n$ belongs to the n^{th} layer of the network, and is thus a subset of the elements of $\boldsymbol{\theta}$. In this context, $\boldsymbol{x}_n$ is called the *state vector*.

A key strength of neural networks is that they obey the universal approximation theorem: for an arbitrarily wide hidden layer, a feedforward network can estimate any continuous function to any desired accuracy, provided that a non-linear activation function is used. This was first proven for a sigmoid activation function [4], but has since been extended to more general cases [15, 34]. This feature, when combined with an optimisation algorithm known as *gradient descent* (GD), gives neural networks a versatility enabling their application to a variety of tasks. During GD, the network's parameters are updated according to:

$$\theta_i \to \theta_i - \eta \frac{\partial L}{\partial \theta_i} \tag{3}$$

where η is a constant parameter called the *learning rate* and $\frac{dL}{d\theta_i}$ is the local gradient of the loss surface. Provided η is sufficiently small, this process decreases the loss at each iteration,

thus improving performance. This is referred to as model *training*. The gradients are computed using an algorithm called *backpropagation* [30]. GD is an example of a broader class of algorithms called 'gradient methods', which use the local gradient to define a search direction. It can be shown (under some assumptions) that gradient methods are guaranteed to reach a minimum in the loss surface. Since supervised learning problems are non-convex, this is likely to be a local minimum.

An important variation of GD is *Stochastic Gradient Descent* (SGD). This performs parameter updates using an approximation of the gradient obtained from a randomly-selected subset of $\mathcal{D}$, called a *batch*. This reduces computational cost in exchange for a lower convergence rate. It has also been observed that SGD finds flatter minima in the loss surface, which are thought to give better generalisation [14, 17].

Optimisation need not use only first-order information of the loss surface; if the second order information can be obtained, this too can be exploited. An example is Newton's method, for which parameter updates are given by:

$$\boldsymbol{\theta} \to \boldsymbol{\theta} - H^{-1}\frac{\partial L}{\partial \boldsymbol{\theta}} \tag{4}$$

where H is a symmetric square matrix of second-order partial derivatives called the *Hessian*:

$$H_{ij} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j} \tag{5}$$

Calculating the Hessian can provide information on the geometry of the loss surface, and therefore how to modify the optimisation algorithm for increased time-performance and to find better minima. However, the computational complexity of this task is $O(p^2)$, meaning it is intractable in state-of-the-art (SoTA) neural networks (which typically have 10^6-10^7 parameters). Section 2.1 describes the work undertaken in this area.

1.2 Neural Ordinary Differential Equations

A more advanced type of neural network is the Residual Network (ResNet) [13], which has 'skip connections' between layers:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + f(\boldsymbol{x}_n, \boldsymbol{\theta}_n), \qquad f(\boldsymbol{x}_n, \boldsymbol{\theta}_n) = \sigma(w_n \boldsymbol{x}_n + b_n), \qquad \boldsymbol{x}_0 = \boldsymbol{X}_i \tag{6}$$

This equation can be seen as an Euler discretisation of a continuous transformation [12, 20, 31]. In the limit of adding more layers with smaller steps, the dynamics of the hidden state are specified by an ordinary differential equation (ODE) parametrised by a neural network [2]:

$$\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x}, \boldsymbol{\theta}, t), \qquad \boldsymbol{x}(t_0) = \boldsymbol{X}_0 \tag{7}$$

This family of models is called Neural Ordinary Differential Equations. Instead of specifying a discrete set of hidden layers, a Neural ODE is a continuous-depth architecture. Note that in a ResNet, a different set of parameters is used in each layer, whereas in the most basic Neural ODE, a single set of parameters is used to specify the derivative of the hidden state at every point. In this sense, Neural ODEs are not the true limit of ResNets but

the introduction of time-dependence in Equation 7 guarantees that Neural ODEs obey the universal approximation theorem [5]. Depth-varying parameters have been proposed in other works [21].

Neural ODEs can be trained using GD, as in neural networks. Evaluating the loss requires solving an ODE which, in practice, is done numerically:

$$L(\boldsymbol{x}(t_1)) = L\Big(\boldsymbol{x}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{x}, \boldsymbol{\theta}, t)\, dt\Big) \approx L(\text{ODESolve}(\boldsymbol{x}(t_0), t_0, t_1, f, \boldsymbol{\theta})) \tag{8}$$

Example ODE solvers include the Euler, Runge-Kutta and DOPRI [6] methods. ODE solvers are constructed from algebraic operations, each of which are individually differentiable. Hence ODE solvers are differentiable, making it possible to compute gradients through backpropagation. However, doing so requires storing all intermediate ODE solutions, which can incur a high memory cost [9]. There are also numerical instabilities associated with some solvers [2, 29]. An alternative approach to gradient calculation is the *adjoint sensitivity method* (adjoint method). This evaluates gradients by solving an augmented ODE backwards in time (from the output). A detailed explanation can be found in [2, 23], but is not required for the purposes of this project.

1.3 Project Aim

Neural ODEs combine two complementary techniques for describing non-linear behaviour: ODEs and neural networks. As such, they are especially useful in learning dynamics of physical systems [23].

In practice, Neural ODE models (which typically contain 10^3-10^4 parameters) are smaller than regular neural networks (10^6-10^7). This is because the model is applied at every function evaluation as the ODE solver calculates the solution to Equation 7. This means that, unlike in neural networks, explicit Hessian calculation is feasible. The aim of this project is two-fold:

1. Develop a computational approach to Hessian calculation in Neural ODEs;
2. Use this to examine the second-order information of the loss surface.

As in standard neural networks, this could lead to an improved understanding of the loss surface, ultimately informing on how to train Neural ODE models with improved time-performance and accuracy.

In the next section, I describe the previous work in this area. Section 3 outlines the possible approaches to Hessian calculation and justifies the choice of method. The computational details of this are explained in Section 4, which also informs on the efforts made to validate these techniques. Sections 5 and 6 explain the experiments run, as well as their results and discussion. Finally, Section 7 summarises this project's findings and directions for future work.

The structure of this work is summarised by the graphical abstract shown in Figure 1.

2 Related Work

2.1 Hessian Analysis

The Hessian of small neural networks was first analysed in [32, 33]. Eigenvalues were calculated exactly with a method for evaluating the Hessian vector product first introduced by Pearlmutter [28]. It was found that the Hessian eigenvalue density spectrum (eigenspectrum) possesses a low-rank structure with a bulk of eigenvalues close to zero and a smaller number of larger positive outliers. For classification tasks, the number of such outliers was seen to be approximately $c-1$, where c is the number of classes. Wu et al. [36] work towards mathematically justifying some of these results, in particular verifying that the number of so-called 'top eigenvalues' is approximately $c-1$ for random 2-layer networks.

More recent work has focused on approximating the Hessian for larger models using tools from advanced numerical linear algebra. These techniques are approximate but formed with a rigorous mathematical background. Multiple papers make use of the Lanczos algorithm [19], which computes the spectrum of a symmetric matrix by reducing it to tridiagonal form. Papyan [26] confirmed the results described above on SoTA neural networks with $O(10^7)$ parameters. Ghorbani et al. [10] studied the evolution of the Hessian eigenspectrum throughout the entire optimisation process, demonstrating that the outlier eigenvalues slow optimisation and that batch normalisation suppresses them, therefore speeding up training. It was also shown that, during training, the gradient converges to the top eigenspace of the Hessian; this was also shown in [11].

Yao et al. [37] used this technique to calculate the Hessian eigenspectrum, and other linear algebra tools to compute the trace. They also investigated the effects of batch normalisation, demonstrating that it does not necessarily make the loss landscape smoother (in contrast to consensus of the machine learning community), particularly for shallower networks.

Finally, Parker-Holder et al. [27] developed an optimisation algorithm that follows the eigenvectors of the Hessian instead of the local gradient, enabling the discovery of different local optima (which is important in some contexts). However, the techniques used are limited since they cannot efficiently compute Hessian information far from extreme points of the spectrum.

2.2 Augmented Neural ODEs

Various modifications of Neural ODEs have been proposed. Of these, this project is focused on Augmented Neural ODEs (ANODEs). Dupont et al. [7] demonstrated that mappings learnt by Neural ODEs are limited to homeomorphisms which must preserve the topology of the input space. This implies the existence of functions which they cannot represent. As a solution to this, ANODEs augment the input state, adding extra dimensionality to the space on which the ODE is learned:

$$\frac{d}{dt}\begin{bmatrix}\boldsymbol{x}(t)\\ \boldsymbol{a}(t)\end{bmatrix} = f\left(\begin{bmatrix}\boldsymbol{x}(t)\\ \boldsymbol{a}(t)\end{bmatrix}, \boldsymbol{\theta}, t\right), \qquad \begin{bmatrix}\boldsymbol{x}(t_0)\\ \boldsymbol{a}(t_0)\end{bmatrix} = \begin{bmatrix}\boldsymbol{X}_0\\ \mathbf{0}\end{bmatrix} \tag{9}$$

It was seen that these extra dimensions allow ANODEs to learn a smoother solution (f) that the ODE solver can compute with fewer steps. This resulted in models which are more stable and generalise better than typical Neural ODEs.

3 Approaches to Hessian Calculation in Neural ODEs

The first stage in this project was to establish an approach to Hessian calculation. This section summarises the techniques considered and justifies the choice made.

An analytical approach to Hessian calculation for Neural ODEs does not exist[1], meaning computational techniques must be used. Adjoint-based computation of second derivatives has been considered in other fields [16, 25]. A detailed mathematical analysis is provided by Caplan [1], Nandi and Singh [22]. To summarise, second-order derivatives can be obtained both through direct backpropagation of the ODE solver and the adjoint method. The availability of two techniques for both stages of differentiation yields 4 ways in which the Hessian can be obtained:

1. Direct-Direct
2. Direct-Adjoint
3. Adjoint-Direct
4. Adjoint-Adjoint

where the first term refers to the method of first-order derivative calculation, and likewise for the second. As with gradient calculation, use of the adjoint method avoids direct backpropagation through the ODE solver. However, an implementation of techniques 2-4 does not yet exist and developing this was likely to require more time than was available during this project, since it would require adapting advanced Neural ODE libraries such as `torchdiffeq` [2].

In spite of the associated memory cost and potential numerical instabilities, the Direct-Direct method was identified as the most appropriate. The small scale of Neural ODEs ensures that the method is computationally feasible and the ODE solver was chosen to minimise the possibility of unstable behaviour. These considerations mean that technique validation was especially important to this project, and is described below.

4 Implementation

This section outlines the details of the computational techniques used and the efforts made to validate them. All code was written in a `Python3` Jupyter Notebook using the `PyTorch` package, and can be found in the supplementary material. Hessian calculation was performed using the 2.3GHz CPU runtime available with Google Colab.

4.1 Computational Techniques

Having multiple techniques available for Hessian calculation is beneficial. Different approaches may have their own strengths (making them more suitable in specific contexts) and a comparison of results obtained can increase overall confidence in the method. This is especially important since an analytical comparison was not possible. As such, three separate tools were developed:

[1] A closed-form expression for the derivative of Equation 7 has not been found.

1. Advanced Auto-differentiation (AA);
2. Explicit Auto-differentiation (EA);
3. Finite Differences (FD).

Technique 1 uses advanced methods that are available within the `PyTorch` automatic differentiation package. Gradient calculation is performed once per row of the Hessian. It performs accurately, is the most efficient and all results obtained were reproducible.

Technique 2 uses a similar approach, but breaks down the calculation to make it more explicit. Gradient calculation is performed once per element. While less efficient than AA, it was seen to be equally accurate (since both operate via direct backpropagation), whilst also being more interpretable and straightforward to implement in new systems. As such, it offers a useful tool to validate AA.

For techniques 1 and 2, `DOPRI5` was chosen as the ODE solver since it is thought to be stable as reported on the `torchdiffeq` Github page (the official repository for the Neural ODE paper by Chen et al. [2]).

Technique 3 makes a basic numerical approximation of the curvature by finite differences. For a loss function $L(\boldsymbol{\theta})$, this gives:

$$\frac{\partial^2 L}{\partial\theta_i\partial\theta_j} \approx \frac{L(\theta_i+h,\theta_j+h)-L(\theta_i+h,\theta_j-h)-L(\theta_i-h,\theta_j+h)+L(\theta_i-h,\theta_j-h)}{4h^2} \tag{10}$$

where $h \in \mathbb{R}$ is a small perturbation to the parameter vector. The main practical limitation is the computational demand in calculating 4 forward passes for each element, meaning that full Hessian calculation can only be performed for small models (found to be ~ 100 parameters at most). In addition, the requirement that h is small can create difficulties with machine precision. 64-bit floating point arithmetic was used to help mitigate this. It was also found that the accuracy of the technique was sensitive to h in an unpredictable manner (see Section 5.1). Nonetheless, FD can still be used to calculate a subset of Hessian elements in larger systems, thereby providing a measure of agreement to other approaches. It is independent of techniques 1 and 2 and is likely to be correct (within the bounds of machine precision) since it relies only on loss evaluation. This means that it provides a valuable tool for validation of the more sophisticated approaches.

The relative strengths and weaknesses of each approach are summarised in Table 1. Evidence for these behavioural features is provided in Sections 4.2 and 5.

4.2 Validation

The aim of technique validation was to assess each technique's strengths and limitations. This was done at all stages in the project, and the results for more sophisticated experiments are detailed in Sections 5 and 6. This subsection outlines the techniques used and the results of preliminary experiments.

Being able to draw comparisons between results is crucial to the validation process. To do this, two complementary measures were used: comparison of eigenspectra[2] and the

[2]During this project, these were shown using eigenvalue histograms.

average fractional absolute difference between the calculated elements. These were chosen since they were straightforward to interpret whilst ensuring differences are identified where appropriate.

Table 1: Ranked comparison of different approaches for Hessian calculation.

Name	Repeatability	Accuracy	Time-performance	Other Comments
AA	= 1	= 1	1	Once validated, this is the optimal approach.
EA	= 1	= 1	2	More interpretable than AA and is easily applied to new systems.
FD	= 1	3	3	Independent of other techniques but sensitive to perturbation parameter.

4.2.1 Agreement with Analytical Results in Neural Networks

The calculation tools were first tested in neural networks, for which an analytical expression for the Hessian could be obtained. The networks used were small (containing only 6 parameters) but were varied such that a range of loss and activation functions were tested. The results shown in Table 2 demonstrate an excellent level of agreement between the analytical and computational techniques.

Table 2: Comparison of computational and analytical tools for Hessian calculation in 4 different feedforward networks. The fractional difference between the sum of matrix elements calculated by computational and analytical approaches is shown. In all cases, the differences are small relative to the size of matrix elements, although the error in the FD approach is larger.

Loss	Activation	AA Error	EA Error	FD Error
Absolute Error	tanh	3.35×10^{-8}	3.35×10^{-8}	1.92×10^{-2}
Absolute Error	sigmoid	3.72×10^{-7}	3.72×10^{-7}	4.03×10^{-3}
MSE	tanh	2.13×10^{-7}	2.13×10^{-7}	2.34×10^{-2}
MSE	sigmoid	4.56×10^{-7}	4.56×10^{-7}	5.62×10^{-3}

4.2.2 Extension to Neural ODEs

The strong agreement seen in the context of neural networks does not guarantee similar results for Neural ODEs since, as explained in Section 3, backpropagation through the ODE solver could yield numerical instabilities.

To assess performance in this case, all three Hessian tools were applied to several Neural ODE architectures of varying size. The results are found in Sections 5 and 6.

5 Learning Dynamics

The experimental part of this project consisted of two stages. Firstly, insight into the general traits of the Hessian was obtained through an analysis of Neural ODEs designed to model simple low-dimensional ODEs. An advantage of these systems was that their small size[3] reduced the time required for Hessian computation.

The aim of this task is to learn the dynamics of a system described by an underlying ODE:

$$\frac{d\boldsymbol{x}}{dt} = g(\boldsymbol{x}, t) \qquad \boldsymbol{x}(t_0) = \boldsymbol{X}_0 \tag{11}$$

A Neural ODE is trained to learn the system behaviour as accurately as possible, such that $f(\boldsymbol{x}, \boldsymbol{\theta}, t) \approx g(\boldsymbol{x}, t)$. In particular, models were trained on two different systems:

1. A 1D exponential ODE;
2. Two coupled sinusoidal ODEs.

Unless otherwise stated, this was done using gradient descent. Hessian calculation was performed throughout the optimisation process. Additional results to those found in this section are presented in Appendix B.1.

5.1 Accuracy of Computational Approaches

Overall, the results generated using the three calculation techniques were seen to be consistent, providing strong evidence of their accuracy and reliability.

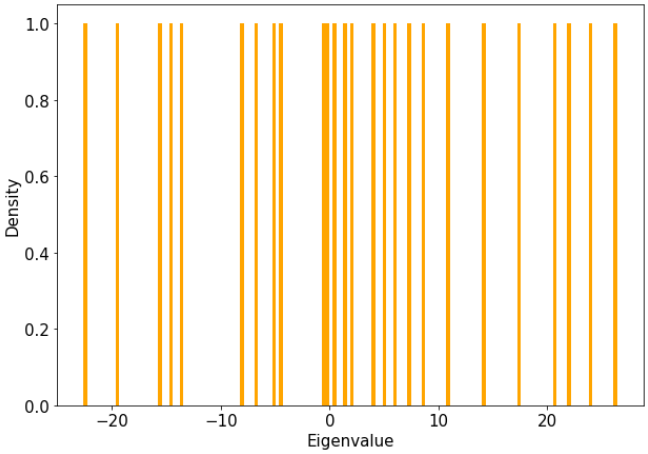
For example, the bulk of the eigenspectra for a 27-parameter Neural ODE trained to model two coupled sinusoidal ODEs using an absolute-difference loss and a *tanh* activation are shown in Figure 2. The agreement seen is supported by a relative difference between matrix elements that is measured as 0.0 (perfect agreement within machine precision) between AA and EA, and 0.0023 between techniques AA and FD. The outlier eigenvalues were also seen to be consistent.

This behaviour was typical of the majority of tests performed. In all cases during validation and in the remainder of the project, the AA and EA approaches were consistent.

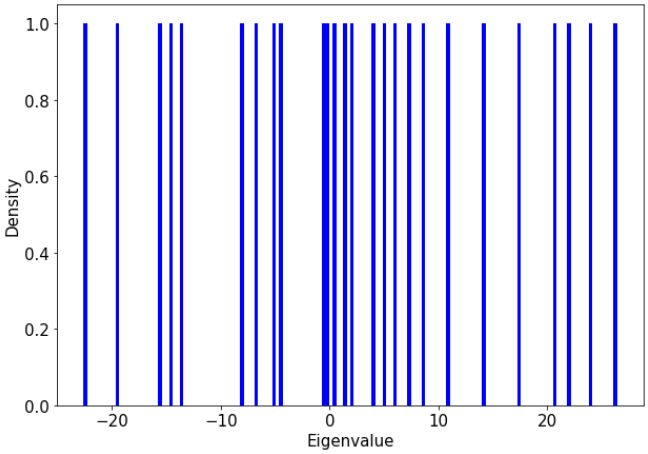
However, limitations were seen in the FD approach. It is sensitive to the size of h in a way that is unpredictable across different models. Figure 4 shows the fractional difference between the Hessian obtained via AA and FD as a function of h for the model described above. The FD approximation is only valid for small h, but machine precision compromises the accuracy for values of h below a certain threshold[4]. This means accuracy is only acceptable for a limited range. As a result, h needed to be chosen on a case-by-case basis.

Another limitation is that, in some instances, the FD method did not validate the other approaches. Figure 3 shows the calculated eigenspectra for the above model at a more advanced stage during training. The value of h used during FD was chosen to give the most similar results to those of AA and EA. Despite this, they are not in agreement.

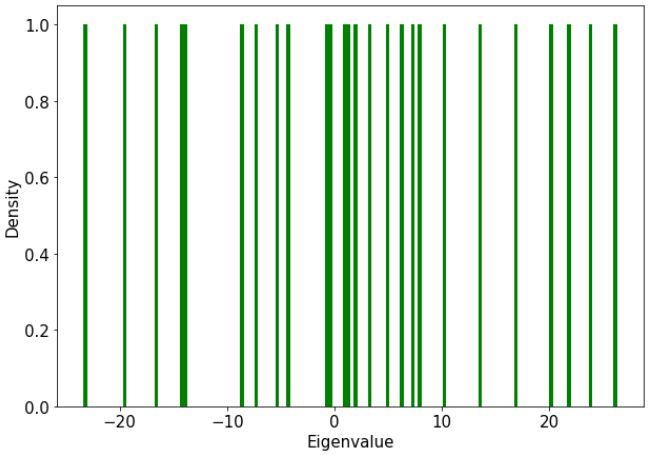
[3] Typically, only 50-100 parameters were necessary.
[4] In practice, seen to be $O(10^{-8})$.



(a) AA approach.

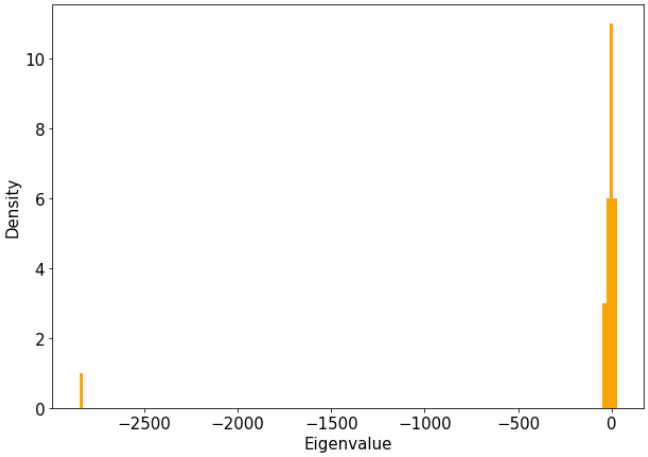


(b) EA approach.

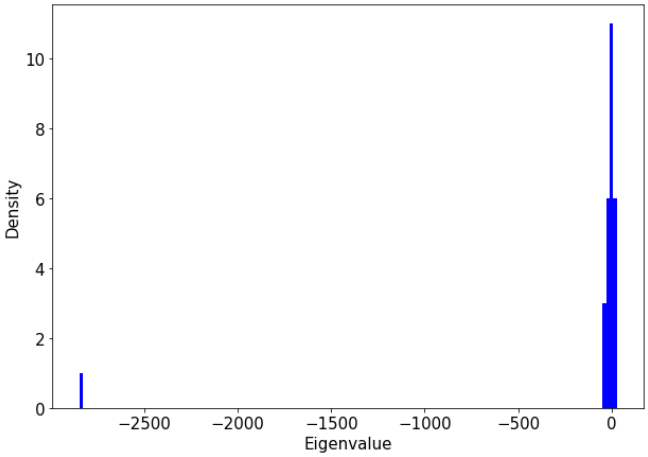


(c) FD approach.

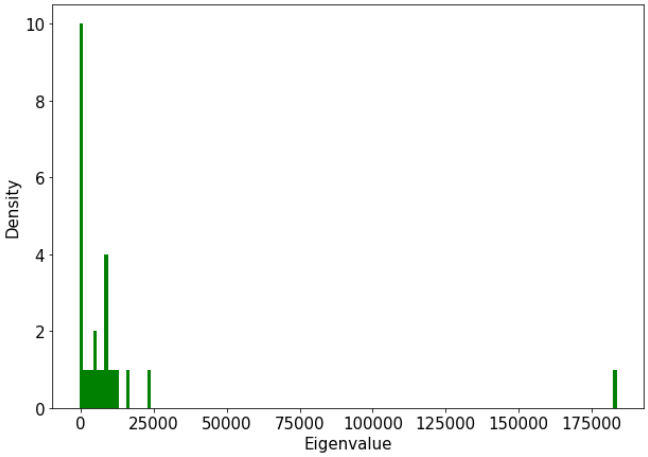
Figure 2: Detailed comparison of the bulks of the eigenspectra for a 27-parameter Neural ODE trained with an absolute-difference loss and a *tanh* activation function. Whilst difficult to assess visually, some differences can be seen between AA/EA and FD. For instance, at an eigenvalue of -15, there is a larger bulk for FD than AA and EA. The average fractional difference across all bulk eigenvalues was 0.0 between AA and EA, and 0.0028 between AA and FD.



(a) AA approach.



(b) EA approach.



(c) FD approach.

Figure 3: Comparison of eigenspectra produced for a model trained using SGD. FD compares poorly. Further tests suggest this is due to shortcomings in the FD approach.

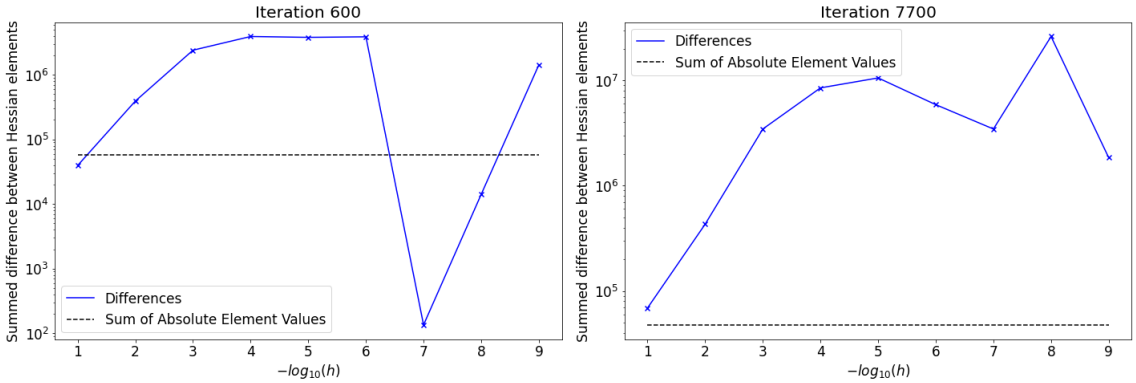


Figure 4: The fractional difference between the Hessian obtained via AA, and that from FD for varying h. The dashed line indicates the summed magnitudes of all Hessian elements from AA. The left panel demonstrates the results at iteration 600. The differences are smaller than the Hessian elements for a narrow range of h. The right panel shows the results at iteration 7700 - the differences between matrix elements are always larger than the elements themselves.

Further tests provide insight into this behaviour. Comparison between the FD and AA methods shown in Figure 5 demonstrates that there is no consistency between results obtained by FD across multiple values of h. In addition, the extremal eigenvalue seen in Figure 3 is different to that seen in the rest of training by several orders of magnitude. Both of these observations suggest the results obtained were due to shortcomings in the FD approach, the cause of which is likely to be the limitations discussed above.

Although the limitations of FD are significant, instances in which they prevent a reasonable calculation[5] of the Hessian were rarely encountered during training.

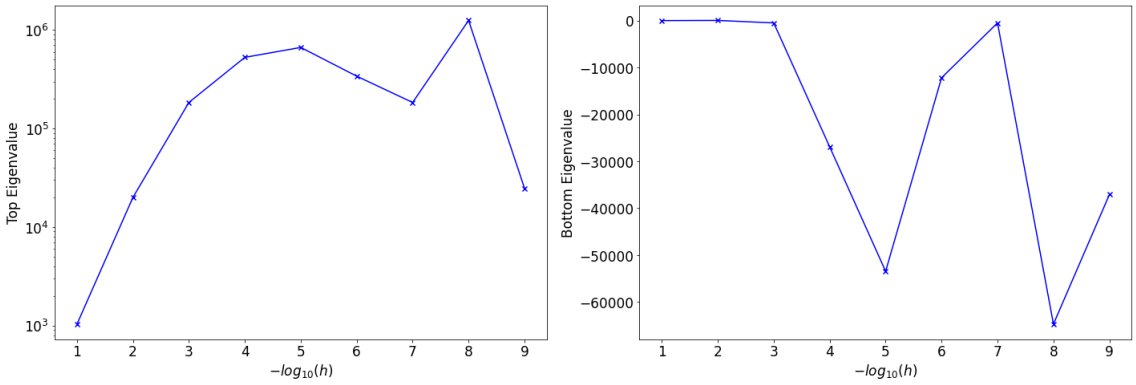


Figure 5: Extremal eigenvalues calculated via FD, for varying h. There is little consistency between the eigenspectra, indicating that it is inaccurate for this model.

5.2 Scalability

The scalability of the developed tools was assessed by measuring the time required for Hessian calculation in various models. The results are shown in Figure 6. The AA approach performs most efficiently, and can be applied to systems with $O(10^4)$ parameters within hours.

[5]I.e. one which is comparable to that obtained through AA and EA.

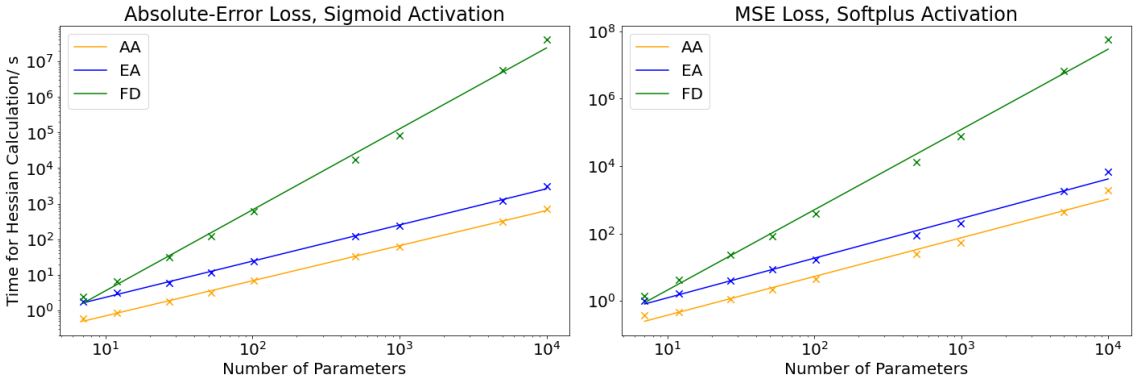


Figure 6: Time required for Hessian computation of randomly-initialised Neural ODEs with a single hidden layer, trained to learn the dynamics of two coupled sinusoidal ODEs. The left panel uses an absolute-error loss and sigmoid activation. The right panel uses MSE loss and a softplus activation. The AA approach has best time-performance and can compute Hessians for $O(10^4)$ parameters in $O(10^3)$ seconds. The 4 longest-time measurements for FD were not measured exactly, but were instead estimated by extrapolating the time required for a subset of matrix elements.

5.3 Similarity between Neural ODE and Neural Network Eigenspectra

The notable observed features of the Hessian are as follows:

Bulk-and-outlier structure: This was seen in all experiments performed during this project. Examples can be seen in Figures 3 and 8.

Non-trivial evolution of extremal eigenvalues: Figures 7 and 8 show the evolution of the top eigenvalue in a system containing 6 parameters during training to fit a 1D exponential. As was seen by [11], the extremal eigenvalues evolved non-trivially.

SGD destabilises Hessian evolution: The evolution of the top eigenvalue was complicated further by using SGD. The extremal eigenvalue was sometimes seen to be negative, indicating a local maximum in the loss surface. This effect is more apparent with decreasing batch size, as seen in Figure 7.

To determine if this behaviour is unique to Neural ODEs, a regular neural network was trained on the same task, with varying batch sizes. The results shown in Figure 9 demonstrate that this was not the case.

5.4 Discussion

The consistency shown by all three tools across a range of models suggests that they are accurate in this task. This supports the observation that the Hessian structure seen for these small Neural ODE systems is similar to that seen in previous work on standard neural networks. This is not unexpected since Neural ODEs, like neural networks, are highly overparametrised systems and are trained using the same optimisation techniques. Furthermore, recent work has highlighted the similarities between ResNets and trained Neural ODE models [24]. All of this increases confidence in the performance of the tools developed.

Likewise, the effects of SGD on the Hessian have been examined previously [26, 35]. However, the destabilising effects shown above have not been observed in other works. A possible interpretation of this behaviour is that the stochastic nature of the gradient used to update

the model parameters can mean that local maxima are encountered during optimisation. That this behaviour is also seen in small-scale neural networks suggests it is not unique to Neural ODEs, but may be a consequence of small model size.

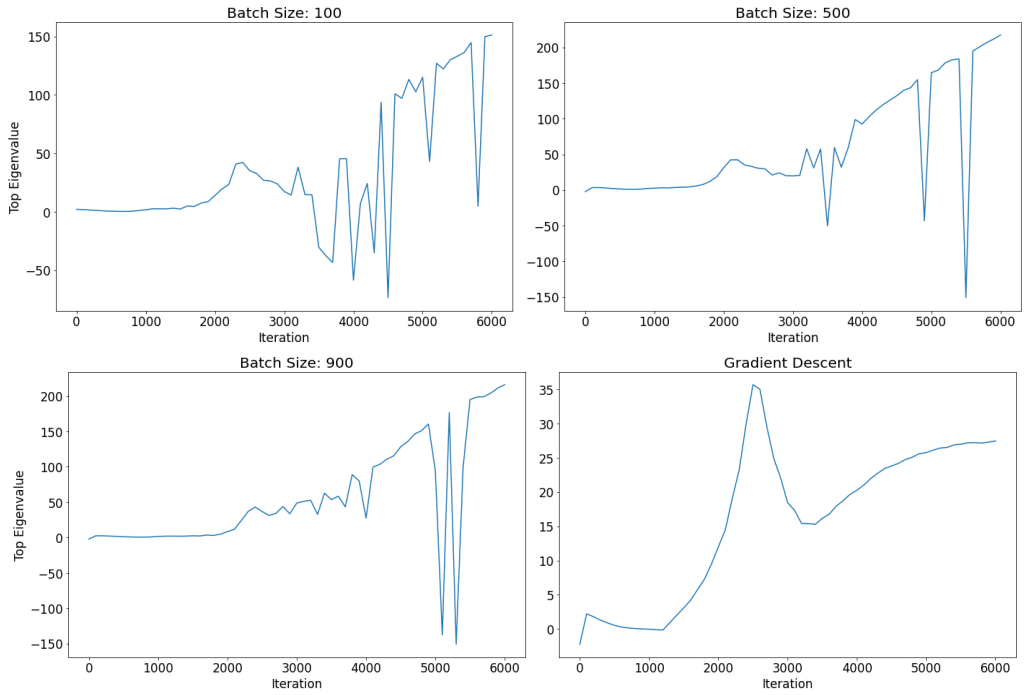


Figure 7: Non-trivial evolution of the top eigenvalue throughout training to fit a 1D exponential. The model has 4 parameters and is trained with varying batch sizes (as shown). The bottom-right panel uses full-batch gradient descent. SGD destabilises the top eigenvalue evolution.

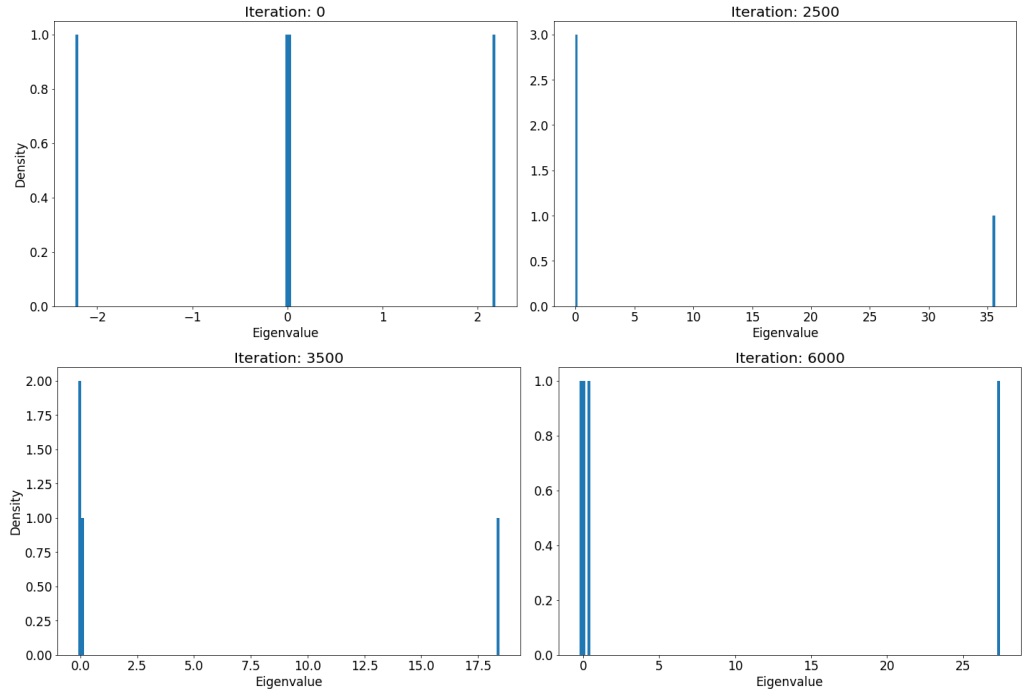


Figure 8: Eigenspectra for the system described in Figure 7 at several stages during training with GD. The eigenspectrum evolves non-trivially.

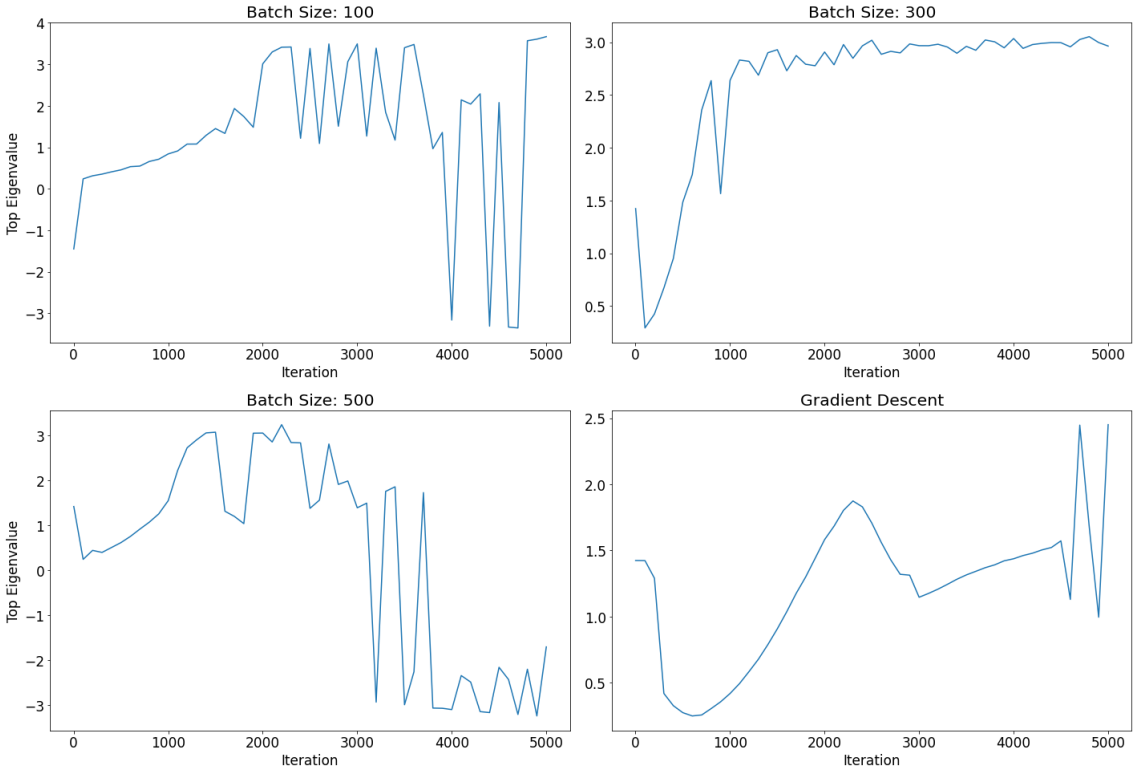


Figure 9: Non-trivial evolution of the top eigenvalue for a regular neural network with 31 parameters and varying batch sizes (as shown). The bottom-right panel uses GD. SGD destabilises the top eigenvalue evolution, just as for Neural ODEs.

Overall, these results provide evidence that the tools designed can be suitable for Neural ODE systems, whilst providing some initial insight into Hessian behaviour. However, they were limited by the small scale of the systems examined and were only trained on one type of task. Both of these were addressed by the experiments detailed in Section 6.

6 Classification Task

ANODEs perform significantly better than Neural ODEs in many tasks and are prevalent in the Neural ODE literature [21, 23], making them an interesting and relevant avenue of research. This was done in the context of the Nested Spheres experiment.

Nested Spheres is a term used by Massaroli et al. [21] to describe a set of functions introduced by Dupont et al. [7], who considered the mapping $g : \mathbb{R}^d \to \mathbb{R}$ such that:

$$\begin{cases} g(\boldsymbol{x}) = -1 & \text{if } \|\boldsymbol{x}\| \leq r_1 \\ g(\boldsymbol{x}) = 1 & \text{if } r_2 \leq \|\boldsymbol{x}\| \leq r_3 \end{cases} \tag{12}$$

where $r_1 < r_2 < r_3$. Since Neural ODE transformations are a homeomorphism, they preserve the topology of the input space and are therefore unable to learn $g(\boldsymbol{x})$. In practice, the Neural ODE is trained on inputs that are a discrete sample of this continuous distribution. This results in gaps for the flow to fit through, meaning it can learn an accurate representation - but the complex flow results in ODEs that are expensive to solve.

Instead, ANODEs use their extra dimension to learn $g(\boldsymbol{x})$ with a simpler flow, yielding a smaller computational cost during training and better generalisation.

This behaviour is illustrated in Figure 10. Appendix B.2 supplements the results shown in the remainder of this section.

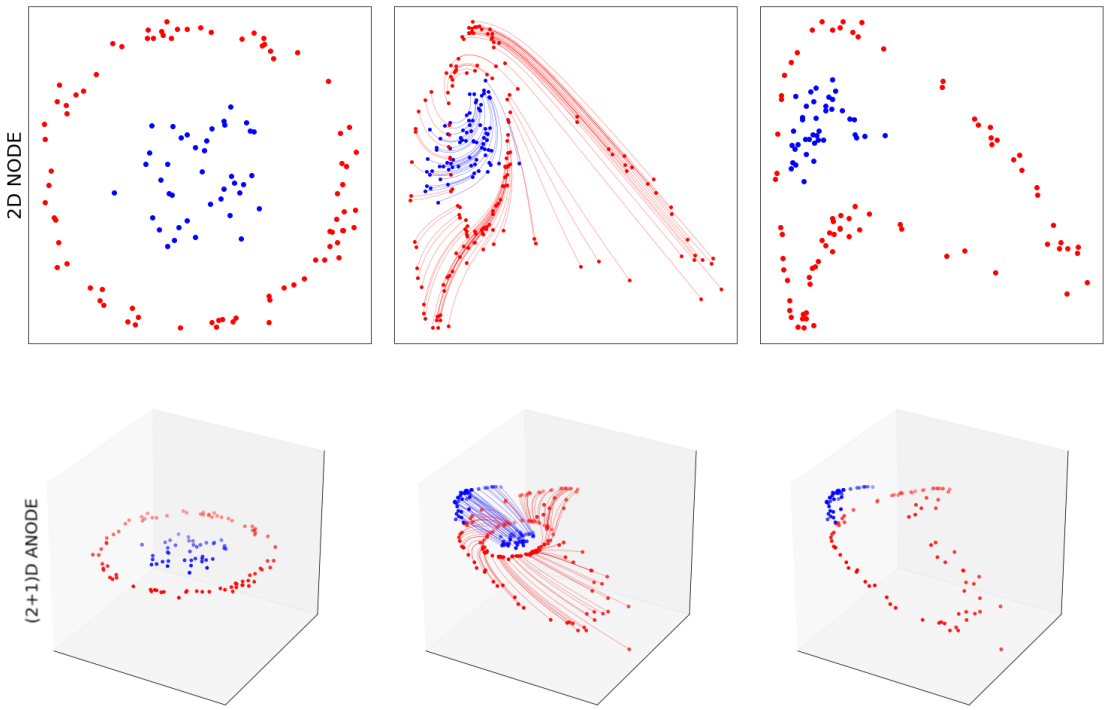


Figure 10: Illustration of a 2D version of the Nested Spheres experiment. The models are trained on a discrete sample of the continuous distribution given by Equation 12. ANODEs use their extra dimensionality to learn the mapping with a simpler flow. This gives rise to smaller computational cost during training and better generalisation.

6.1 More Evidence for Accuracy of the Computational Approaches

Hessian analysis was performed for Neural ODEs and ANODEs in 2, 3 and 4 dimensions. The models used were too large for FD to be performed fully throughout training; this was done only in select cases. Otherwise, a smaller set of Hessian elements were calculated. Performance was seen to be consistent with Section 5.1, although there were no instances in which FD did not yield a Hessian matrix similar to those obtained via the other techniques.

6.2 Augmentation Alters the Hessian

Whilst Hessian evolution and structure was similar to that seen in Section 5.3 - thus corroborating these results - multiple differences were seen between the Hessians of Neural ODEs and ANODEs:

ANODEs find flatter minima: After an initial period of growth, the top eigenvalues were seen to decay as training progressed. The final values reached were smaller for ANODEs - as seen in Figure 11 - indicating they find flatter local minima.

A more stable evolution of eigenvalues: Figure 11 also shows that ANODE eigenvalues develop more smoothly than their Neural ODE counterparts at later stages in training.

ANODE Hessians possess smaller negative eigenvalues: By the end of training, the negative eigenvalues found by ANODEs were smaller in magnitude than for Neural ODEs. This too is shown in Figure 11.

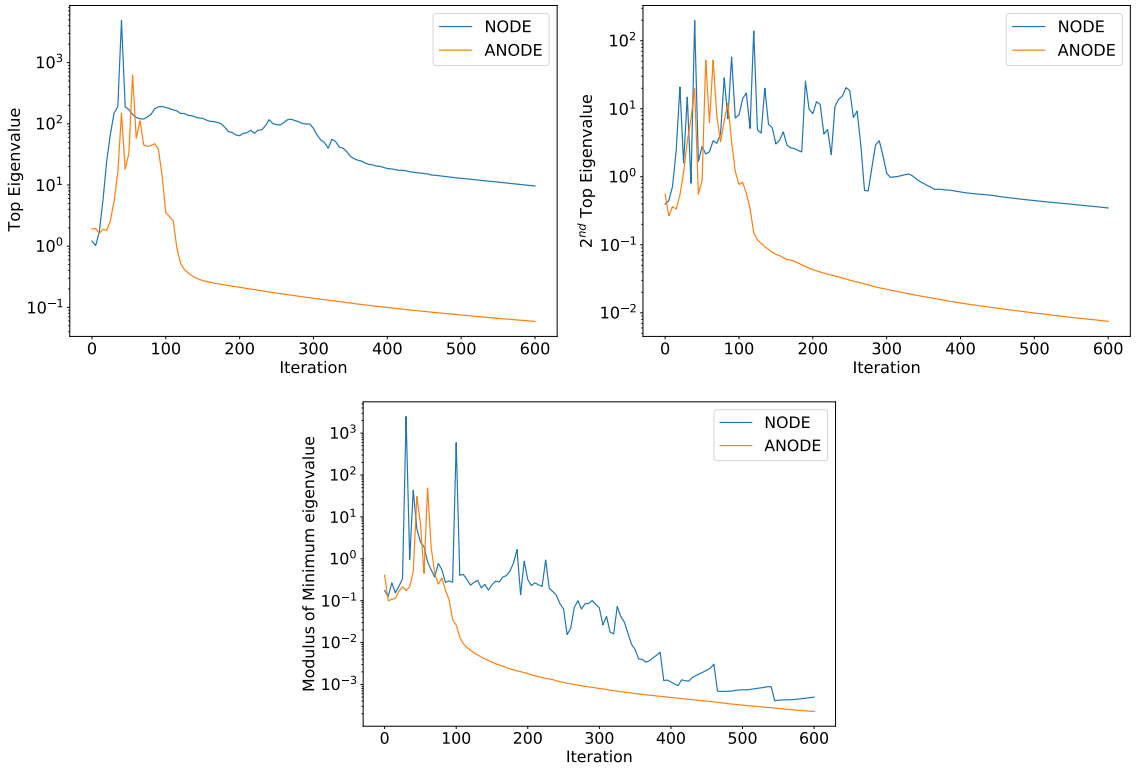


Figure 11: Differences in the evolution of top, 2nd top and bottom eigenvalues (top-left, top-right and bottom respectively) for ANODEs and Neural ODEs, trained on 2D input data. It is seen that the ANODE top eigenvalues are smaller than for Neural ODEs, indicating they find flatter minima. ANODE eigenvalues also evolve more smoothly, and the minimum eigenvalue is smaller in magnitude.

6.3 Discussion

These features of the Hessian can be related to ANODE performance. As explained in Section 1.1, it has previously been observed that flatter minima in neural network loss landscapes tend to generalise better. The above results show this claim is also true in the context of ANODEs; they discover flatter minima at the end of training and achieve better generalisation in this task. It has been seen empirically that ANODEs are more stable to train [7], which is presumably linked to the smooth behaviour seen in the evolution of Hessian eigenvalues. Finally, the smaller negative eigenvalues encountered by ANODEs indicates that system is closer to a true local minimum.

These results demonstrate a clear relationship between augmentation and the geometry of the loss surface.

7 Conclusion

After assessing possible approaches, this project presented multiple tools to calculate the Hessian for Neural ODEs and validated their effectiveness in a range of tasks. The AA technique was seen to be most accurate and time-efficient. I believe these are of value to the research community, since they can inform on the geometry of the loss surface.

These tools were then used to analyse the second-order information of the loss. The results corroborate the bulk-and-outlier structure and non-trivial eigenvalue evolution seen in ResNets. However, SGD lead to a top eigenvalue evolution that is unseen in previous work and may suggest that local maxima are encountered during optimisation. This behaviour was also observed in small-scale neural networks. The effects of augmentation were then analysed; it was observed that ANODEs find flatter minima than Neural ODEs, lead to a smoother top eigenvalue evolution and possess smaller-magnitude negative eigenvalues at the end of training.

7.1 Limitations and Future Work

Notwithstanding the achievements of this project, there are two key areas in which improvements could be made:

Model size: The systems examined are large enough to solve small-scale problems and therefore gain several insights into the Hessian. However, Neural ODEs used in larger experiments can have 10^3-10^4 parameters and it would be important to test whether these findings hold for such systems.

Numerical instabilities: The potential for unstable behaviour affecting results cannot be ruled out in any instance, but is more likely when using a different ODE solver to `DOPRI5`.

Numerical instabilities could be overcome with an implementation of the generalised adjoint method for second order derivatives. This is an important direction of future work. Increased model size could be explored using the current techniques - doing so would simply require more time. Indeed, this work provides a method to explore a wide range of architectural features of Neural ODEs such as activation function[6], Second Order Neural ODEs [23] and other regularisations. Moreover, the access to eigenvectors provided by these methods was not exploited during this project. Using these to perform an analysis similar to that of Parker-Holder et al. [27] could be of interest in some contexts and the issue of computational efficiency encountered by the authors would be less significant. Lastly, the relationship between ResNets and Neural ODEs illustrated by Ott et al. [24] could be investigated in more detail by examining how step size of the ODE solver effects the Hessian.

I believe this work to be the first to analyse the Hessian of Neural ODEs. By doing so, I hope that Neural ODEs can be used in a broader range of practical applications, thus improving our understanding of the physical world.

[6] Currently, this is chosen empirically.

References

[1] Philip Caplan. Numerical computation of second derivatives with applications to optimization problems. *Unpublished academic report, MIT*, 2011.

[2] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[3] Juan Cruz-Martinez, Stefano Forte, and Emanuele R Nocera. Future tests of parton distributions. *arXiv preprint arXiv:2103.08606*, 2021.

[4] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[5] Jared Quincy Davis, Krzysztof Choromanski, Jake Varley, Honglak Lee, Jean-Jacques Slotine, Valerii Likhosterov, Adrian Weller, Ameesh Makadia, and Vikas Sindhwani. Time dependence in non-autonomous neural odes. *arXiv preprint arXiv:2005.01906*, 2020.

[6] John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[7] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *arXiv preprint arXiv:1904.01681*, 2019.

[8] Stefano Forte and Stefano Carrazza. Parton distribution functions. *arXiv preprint arXiv:2008.12305*, 2020.

[9] Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

[10] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pages 2232–2241. PMLR, 2019.

[11] Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

[12] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1): 1–42, 1997.

[15] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[16] Shin-ichi Ito, Takeru Matsuda, and Yuto Miyatake. Adjoint-based exact hessian computation. *BIT Numerical Mathematics*, pages 1–20, 2021.

[17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[18] Rabah Abdul Khalek, Jacob J Ethier, and Juan Rojo. Nuclear parton distributions from lepton-nucleus scattering and the impact of an electron-ion collider. *The European Physical Journal C*, 79(6):1–35, 2019.

[19] Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

[20] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018.

[21] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *arXiv preprint arXiv:2002.08071*, 2020.

[22] Souransu Nandi and Tarunraj Singh. Adjoint based hessians for optimization problems in system identification. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 626–631. IEEE, 2017.

[23] Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Liò. On second order behaviour in augmented neural odes. *arXiv preprint arXiv:2006.07220*, 2020.

[24] Katharina Ott, Prateek Katiyar, Philipp Hennig, and Michael Tiemann. Resnet after all: Neural {ode}s and their numerical solution. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=HxzSxSxLOJZ`.

[25] DI Papadimitriou and KC Giannakoglou. Direct, adjoint and mixed approaches for the computation of hessian in airfoil design problems. *International journal for numerical methods in fluids*, 56(10):1929–1943, 2008.

[26] Vardan Papyan. The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv preprint arXiv:1811.07062*, 2018.

[27] Jack Parker-Holder, Luke Metz, Cinjon Resnick, Hengyuan Hu, Adam Lerer, Alistair Letcher, Alex Peysakhovich, Aldo Pacchiano, and Jakob Foerster. Ridge rider: Finding diverse solutions by following eigenvectors of the hessian. *arXiv preprint arXiv:2011.06505*, 2020.

[28] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6 (1):147–160, 1994.

[29] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. Snode: Spectral discretization of neural odes for system identification. *arXiv preprint arXiv:1906.07038*, 2019.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[31] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[32] Levent Sagun, Léon Bottou, and Yann LeCun. Singularity of the hessian in deep learning. *arXiv preprint arXiv:1611.07476*, 2016.

[33] Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

[34] Felix Voigtlaender. The universal approximation theorem for complex-valued neural networks. *arXiv preprint arXiv:2012.03351*, 2020.

[35] Mingwei Wei and David J Schwab. How noise affects the hessian spectrum in overparameterized neural networks. *arXiv preprint arXiv:1910.00195*, 2019.

[36] Yikai Wu, Xingyu Zhu, Chenwei Wu, Annie Wang, and Rong Ge. Dissecting hessian: Understanding common structure of hessian in neural networks. *arXiv preprint arXiv:2010.04261*, 2020.

[37] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 581–590. IEEE, 2020.

A Impacts of COVID-19

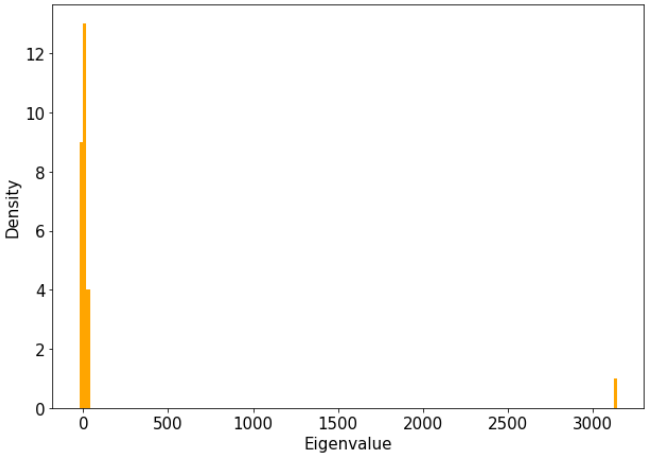
The COVID-19 pandemic has forced significant changes upon all areas of the research community. However, the computational nature of this project has meant that my ability to work was not affected to a significant extent.

B Additional Results

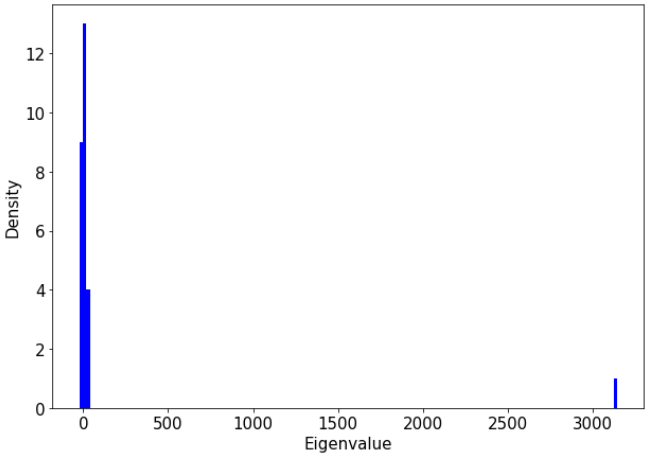
The extensive experimental analysis performed during this project is too substantial to present in full. This appendix presents some additional results to complement those described in the main text.

B.1 Learning Dynamics

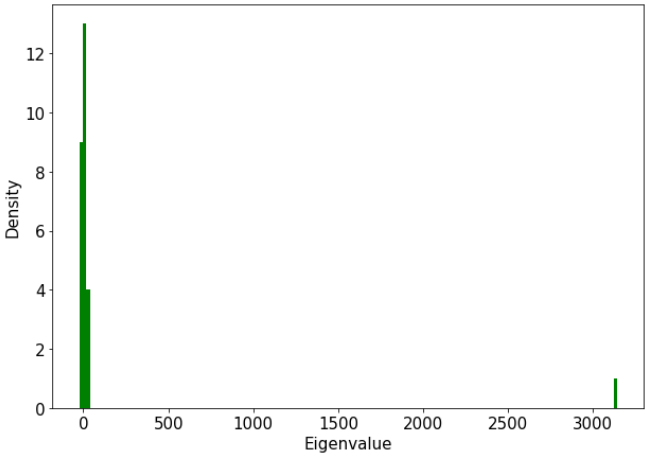
Two examples of full eigenspectra calculated for Neural ODEs during training to learn the dynamics of two coupled sinusoidal ODEs can be seen in Figures 12 and 13 (overleaf). These results demonstrate the behaviour that was typically seen during this task.



(a) AA approach.

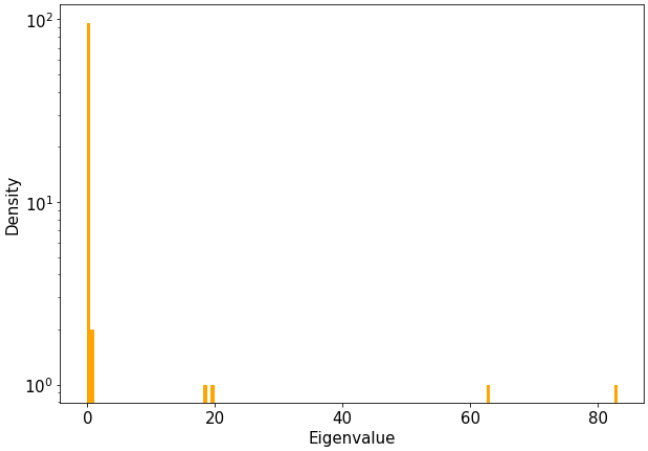


(b) EA approach.

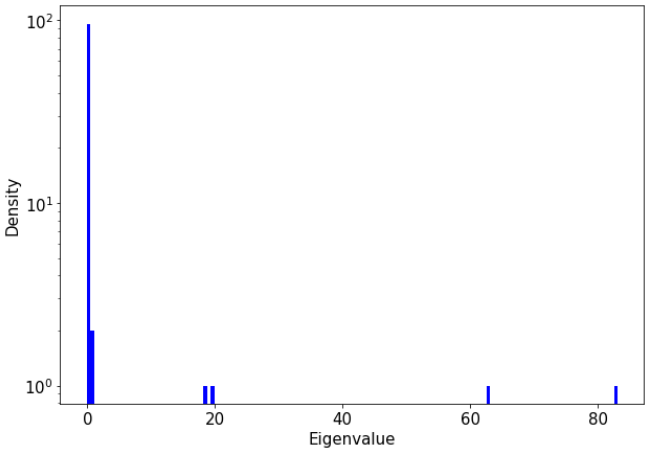


(c) FD approach.

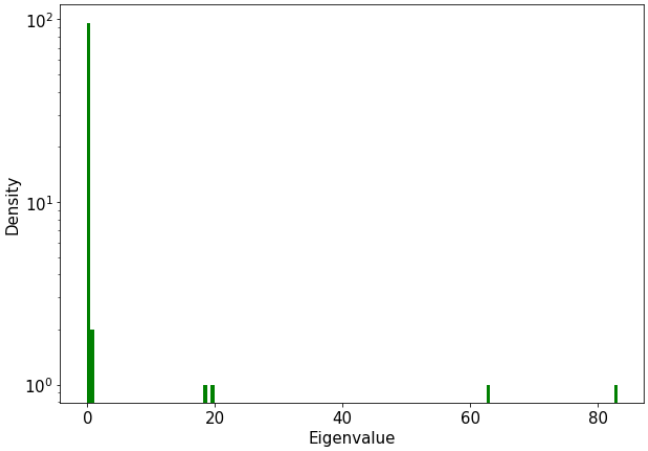
Figure 12: Comparison of full eigenspectra for a 27-parameter Neural ODE trained with an absolute-difference loss and a tanh activation function, as in Figure 2. The three Hessian calculation tools show clear agreement.



(a) AA approach.



(b) EA approach.



(c) FD approach.

Figure 13: Comparison of full eigenspectra for a 102-parameter Neural ODE trained with an MSE loss and a softplus activation. Again, the Hessian calculation tools show clear agreement.

.

B.2 Classification Task

Figures 14 and 15 show two examples of eigenspectrum evolution throughout training on the Nested Spheres. In spite of the larger systems size, the results observed are highly similar to those seen in the Learning Dynamics task.

Figures 16 and 17 give an example of comparisons between Hessian evolution of Neural ODEs and ANODEs in 3 and 4 dimensions. The behaviour seen is similar to that described in Section 6.

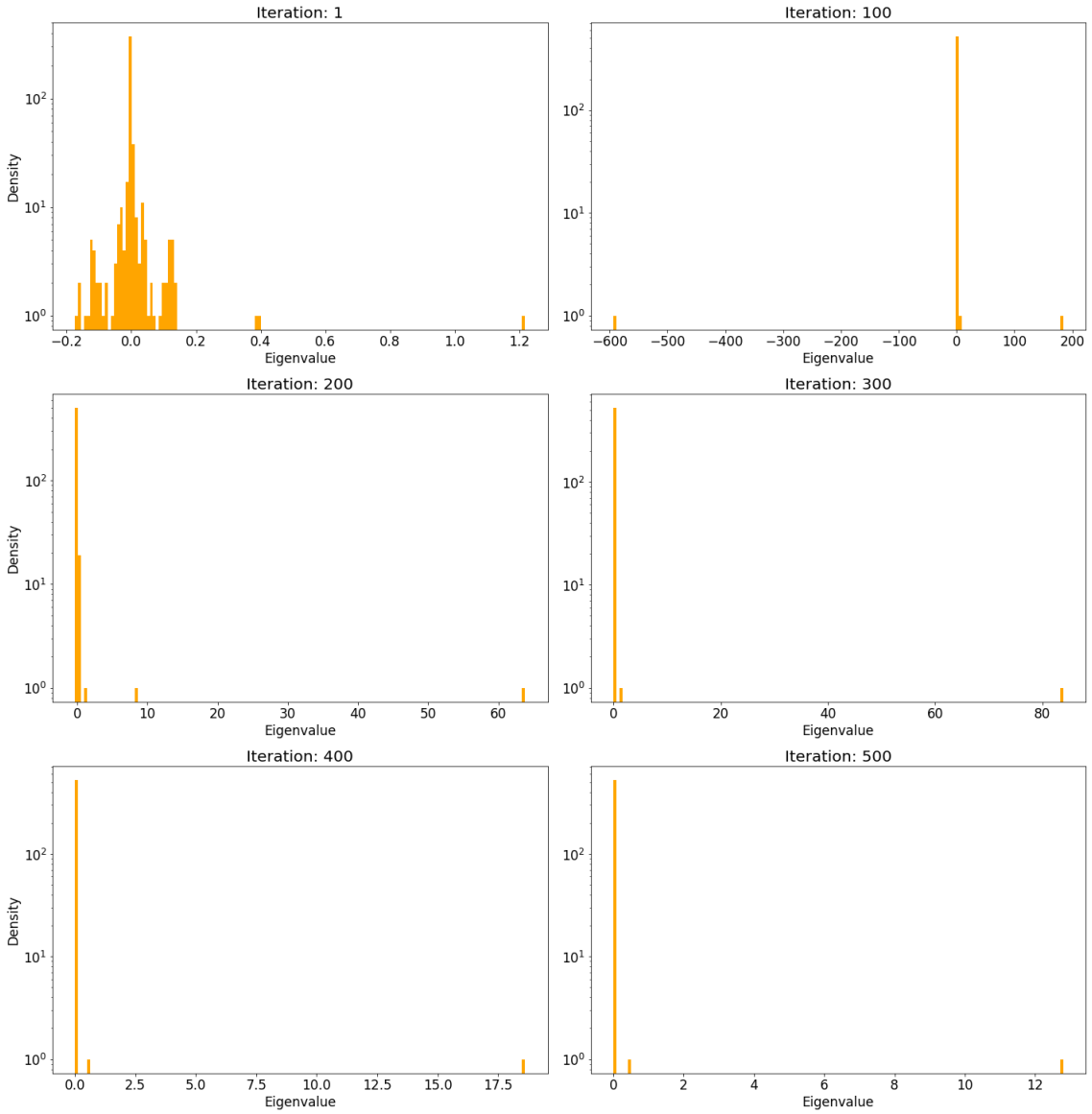


Figure 14: Evolution of the Hessian eigenspectrum during training to solve the Nested Spheres task in 2D. The model used is a 528-parameter Neural ODE. All Hessians were calculated using the AA approach. Unstable behaviour can be seen at iteration 100.

.

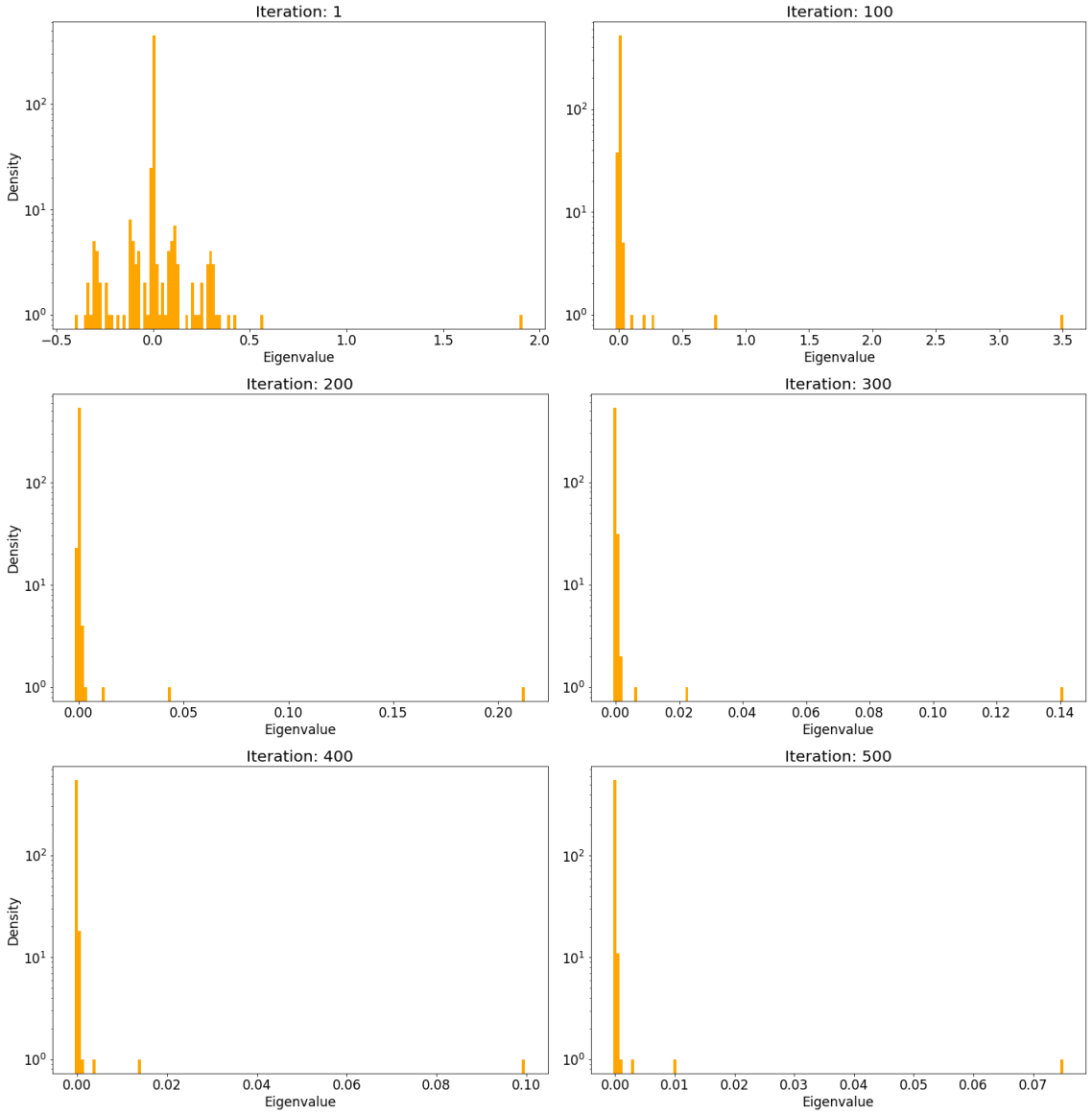


Figure 15: Evolution of the Hessian eigenspectrum during training to solve the Nested Spheres task in 2D. The model used is a 571-parameter ANODE. All Hessians were calculated using the AA approach. No unstable behaviour is seen, and the outlier eigenvalues at the end of training are smaller than in Figure 14.

.

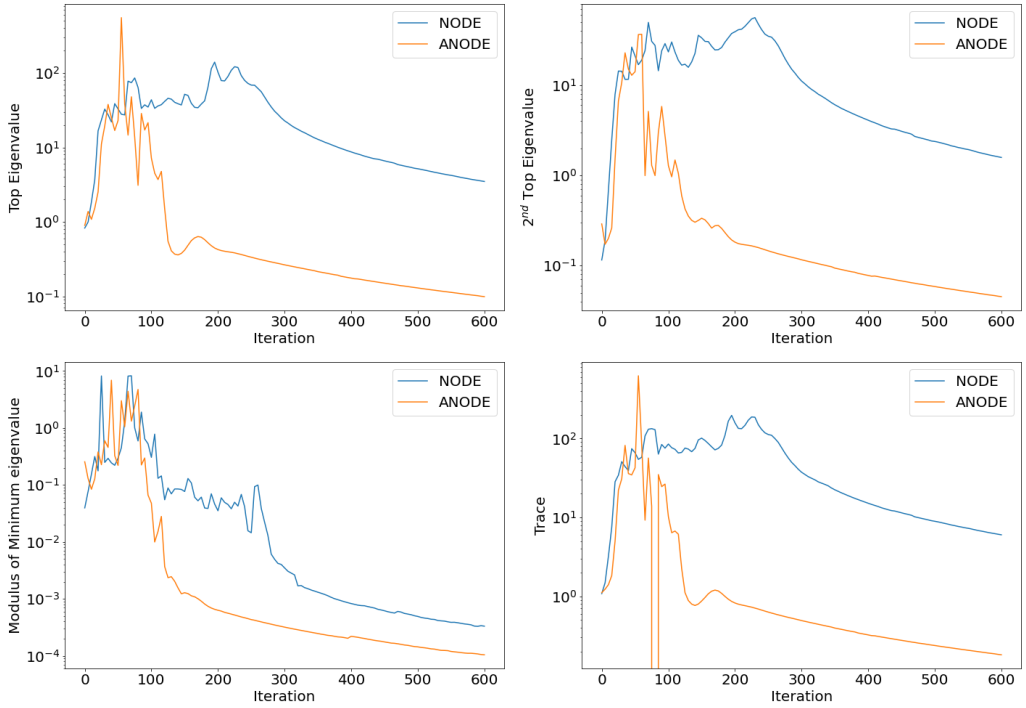


Figure 16: Evolution of top, 2nd top and bottom eigenvalues, and trace (top-left, top-right, bottom-left and bottom-right respectively) for ANODEs and Neural ODEs. These have been trained on the Nested Spheres task using 3D input data.

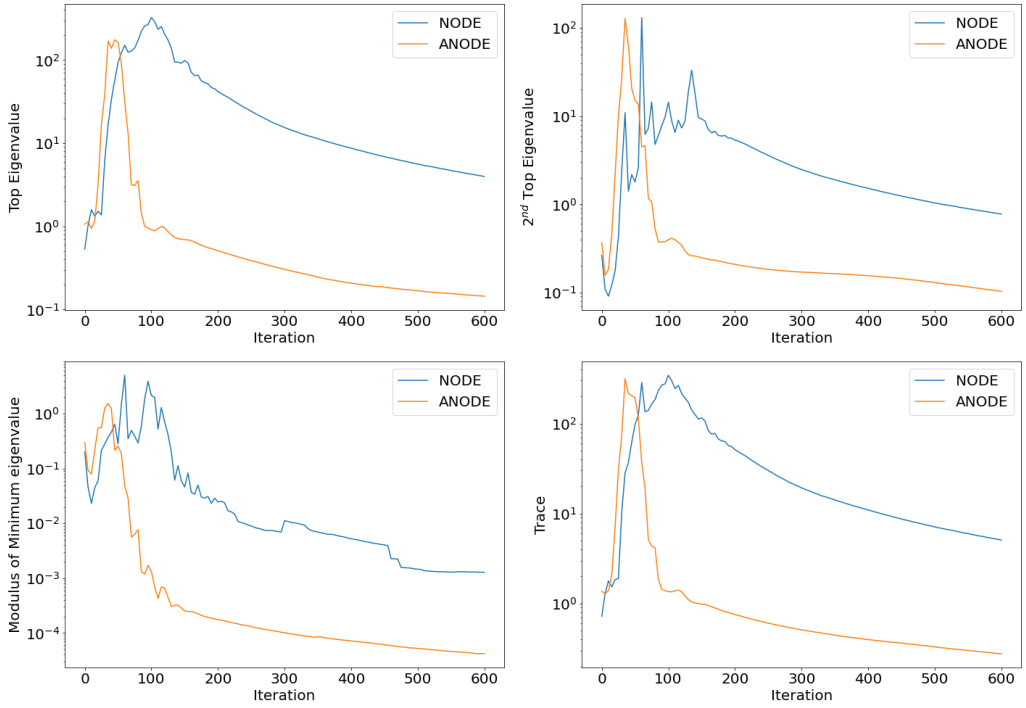


Figure 17: Same as above, except 4D input data has been used.