Augmented Neural ODEs

Emilien Dupont
University of Oxford
dupont@stats.ox.ac.uk

Arnaud Doucet
University of Oxford
doucet@stats.ox.ac.uk

Yee Whye Teh
University of Oxford
y.w.teh@stats.ox.ac.uk

Abstract

We show that Neural Ordinary Differential Equations (ODEs) learn representations that preserve the topology of the input space and prove that this implies the existence of functions Neural ODEs cannot represent. To address these limitations, we introduce Augmented Neural ODEs which, in addition to being more expressive models, are empirically more stable, generalize better and have a lower computational cost than Neural ODEs.

1 Introduction

The relationship between neural networks and differential equations has been studied in several recent works (Weinan, 2017; Lu et al., 2017; Haber & Ruthotto, 2017; Ruthotto & Haber, 2018; Chen et al., 2018). In particular, it has been shown that Residual Networks (He et al., 2016) can be interpreted as discretized ODEs. Taking the discretization step to zero gives rise to a family of models called Neural ODEs (Chen et al., 2018). These models can be efficiently trained with backpropagation and have shown great promise on a number of tasks including modeling continuous time data and building normalizing flows with low computational cost (Chen et al., 2018).

In this work, we explore some of the consequences of taking this continuous limit and which restrictions this might create compared with regular neural nets. In particular, we show that there are simple classes of functions Neural ODEs (NODEs) cannot represent. While it is often possible for NODEs to approximate these functions in practice, the resulting flows are complex and lead to ODE problems that are computationally expensive to solve. To overcome these limitations, we introduce Augmented Neural ODEs (ANODEs) which are a simple extension of NODEs. ANODEs augment the space on which the ODE is solved, allowing the model to use the additional dimensions to learn more complex functions using simpler flows (see Fig. 1). In addition to being more expressive models, ANODEs significantly reduce the computational cost of both forward and backward passes of the model compared with NODEs. Our experiments also show that ANODEs generalize better, achieve lower losses with fewer parameters and are more stable to train.

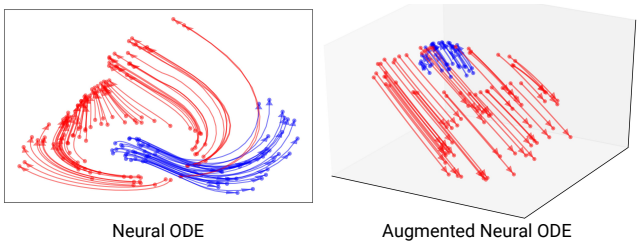


Figure 1: Learned flows for a Neural ODE and an Augmented Neural ODE. The flows (shown as lines with arrows) map input points to linearly separable features for binary classification. Augmented Neural ODEs learn simpler flows that are easier for the ODE solver to compute.

Preprint. Work in progress.

2 Neural ODEs

NODEs are a family of deep neural network models that can be interpreted as a continuous equivalent of Residual Networks (ResNets). To see this, consider the transformation of a hidden state from a layer t to $t+1$ in ResNets

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{f}_t(\mathbf{h}_t)$$

where $\mathbf{h}_t \in \mathbb{R}^d$ is the hidden state at layer t and $\mathbf{f}_t : \mathbb{R}^d \to \mathbb{R}^d$ is some differentiable function which preserves the dimension of $\mathbf{h}_t$ (typically a CNN or an MLP). We can rearrange this equation as

$$\frac{\mathbf{h}_{t+\Delta t} - \mathbf{h}_t}{\Delta t} = \mathbf{f}_t(\mathbf{h}_t)$$

where $\Delta t = 1$. Now letting $\Delta t \to 0$ we see that

$$\lim_{\Delta t \to 0} \frac{\mathbf{h}_{t+\Delta t} - \mathbf{h}_t}{\Delta t} = \frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{h}(t), t)$$

so the hidden state can be parameterized by an ODE. We can then map a data point $\mathbf{x}$ into a set of features $\phi(\mathbf{x})$ by solving the Initial Value Problem (IVP)

$$\begin{cases} \dfrac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{h}(t), t) \\ \mathbf{h}(0) = \mathbf{x} \end{cases}$$

to some time T. The hidden state at time T, i.e. $\mathbf{h}(T)$, corresponds to the features learned by the model. The analogy with ResNets can then be made more explicit. In ResNets, we map an input $\mathbf{x}$ to some output $\mathbf{y}$ by a forward pass of the neural network. We then adjust the weights of the network to match $\mathbf{y}$ with some $\mathbf{y}_{\text{true}}$. In NODEs, we map an input $\mathbf{x}$ to an output $\mathbf{y}$ by solving an ODE starting from $\mathbf{x}$. We then adjust the dynamics of the system (encoded by $\mathbf{f}$) such that the ODE transforms $\mathbf{x}$ to a $\mathbf{y}$ which is close to $\mathbf{y}_{\text{true}}$.

We note that $\mathbf{f}$ can be parameterized by any standard neural net architecture, including ones with activation functions that are not everywhere differentiable such as ReLU. Existence and uniqueness of solutions to the ODE are still guaranteed and all results in this paper hold under these conditions (see appendix for details).

ODE flows. We also define the flow associated to the vector field $\mathbf{f}(\mathbf{h}(t), t)$ of the ODE. The flow $\phi_t : \mathbb{R}^d \to \mathbb{R}^d$ is defined as the hidden state at time t, i.e. $\phi_t(\mathbf{x}) = \mathbf{h}(t)$, when solving the ODE from the initial condition $\mathbf{h}(0) = \mathbf{x}$. The flow measures how the states of the ODE at a given time t depend on the initial conditions $\mathbf{x}$. We define the features of the ODE as $\phi(\mathbf{x}) := \phi_T(\mathbf{x})$, i.e. the flow at the final time T to which we solve the ODE.

NODEs for regression and classification. We can use ODEs to map input data $\mathbf{x} \in \mathbb{R}^d$ to a set of features or representations $\phi(\mathbf{x}) \in \mathbb{R}^d$. However, we are often interested in learning functions from $\mathbb{R}^d$ to $\mathbb{R}$, e.g. for regression or classification. To define a model from $\mathbb{R}^d$ to $\mathbb{R}$, we follow the example given in Lin & Jegelka (2018) for ResNets. We define the NODE $g : \mathbb{R}^d \to \mathbb{R}$ as $g(\mathbf{x}) = \mathcal{L}(\phi(\mathbf{x}))$ where $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ is a linear map and $\phi : \mathbb{R}^d \to \mathbb{R}^d$ is the mapping from data to features. As shown in Fig. 2, this is a simple model architecture: an ODE layer, followed by a linear layer.

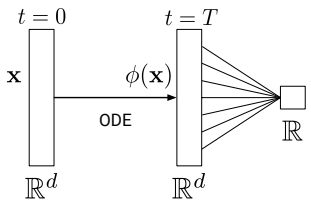


Figure 2: Diagram of Neural ODE architecture.

3 A simple example in 1d

In this section, we introduce a simple function which ODE flows cannot represent, motivating many of the examples we will see later. Let $g_{1\mathrm{d}} : \mathbb{R} \to \mathbb{R}$ be a function such that

$$\begin{cases} g_{1\mathrm{d}}(-1) = 1 \\ g_{1\mathrm{d}}(1) = -1 \end{cases}$$

Proposition 1. *The flow of an ODE cannot represent* $g_{1d}(x)$.

A detailed proof is given in the appendix, however, the intuition behind the proof is simple. Indeed, the trajectories mapping -1 to 1 and 1 to -1 must intersect each other (see Fig. 3a). However, ODE trajectories cannot cross each other, so the flow of an ODE cannot represent $g_{1\mathrm{d}}(x)$. This simple observation is at the core of all the examples provided in this paper and forms the basis for many of the limitations of NODEs.

Experiments. We verify this behavior experimentally by training an ODE flow on the identity mapping and on $g_{1\mathrm{d}}(x)$. The resulting flows are shown in Fig. 3c and 3d. As can be seen, the model easily learns the identity mapping but cannot represent $g_{1\mathrm{d}}(x)$. Indeed, since the trajectories cannot cross, the model maps all input points to zero to minimize the mean squared error.

ResNets vs NODEs. NODEs can be interpreted as continuous equivalents of ResNets, so it is interesting to consider why ResNets can represent $g_{1\mathrm{d}}(x)$ but NODEs cannot. The reason for this is exactly because ResNets are a discretization of the ODE, allowing the trajectories to make discrete jumps to cross each other (see Fig. 3b). Indeed, the error arising when taking discrete steps allows the ResNet trajectories to cross. In this sense, ResNets can be interpreted as ODE solutions with large errors, with these errors allowing them to represent more functions.

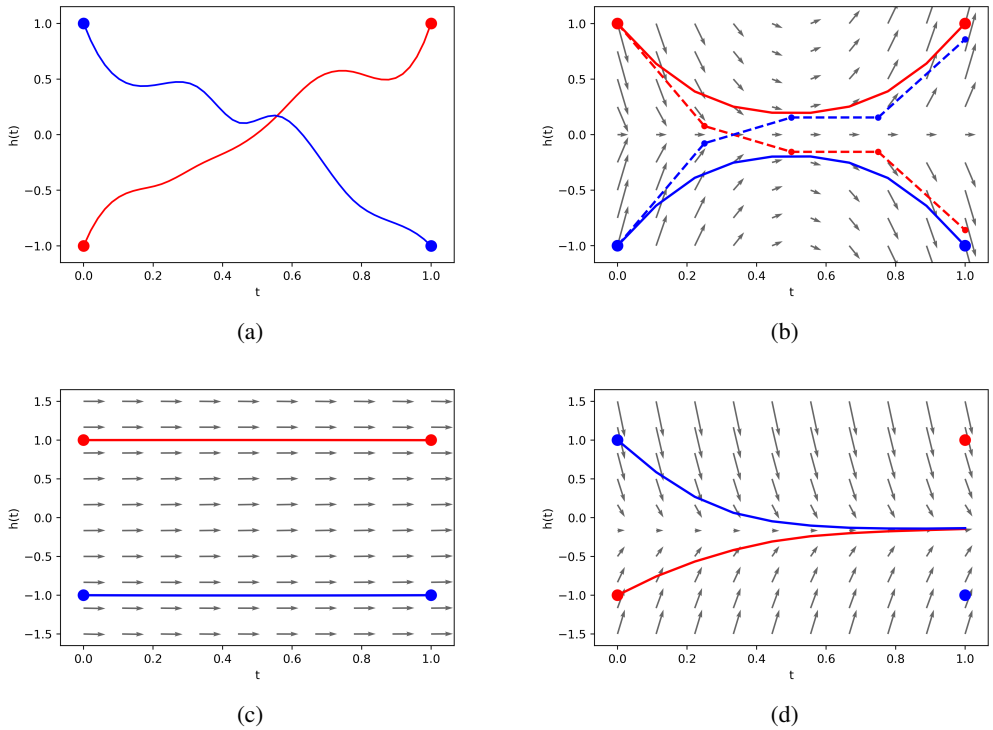


Figure 3: (a) Continuous trajectories mapping -1 to 1 (red) and 1 to -1 (blue) must intersect each other, which is not possible for an ODE. (b) Solutions of the ODE are shown in solid lines and solutions using the Euler method (which corresponds to ResNets) are shown in dashed lines. As can be seen, the discretization error allows the trajectories to cross. (c, d) Resulting vector fields and trajectories from training on the identity function (left) and $g_{1\mathrm{d}}(x)$ (right).

4 Functions Neural ODEs cannot represent

We now introduce classes of functions in arbitrary dimension d which NODEs cannot represent. Let $0 < r_1 < r_2 < r_3$ and let $g : \mathbb{R}^d \to \mathbb{R}$ be a function such that

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

where $\|\cdot\|$ is the Euclidean norm. An illustration of this function for $d = 2$ is shown in Fig. 4. The function maps all points inside the blue sphere to -1 and all points in the red annulus to 1.

Proposition 2. *Neural ODEs cannot represent* $g(\mathbf{x})$.

A proof is given in the appendix. While the proof requires tools from ODE theory and topology, the intuition behind it is simple. In order for the linear layer to map the blue and red points to -1 and 1 respectively, the features $\phi(\mathbf{x})$ for the blue and red points must be linearly separable. Since the blue region is enclosed by the red region, points in the blue region must cross over the red region to become linearly separable, requiring the trajectories to intersect, which is not possible. In fact, we can make more general statements about which features Neural ODEs can learn.

Proposition 3. *The feature mapping* $\phi(\mathbf{x})$ *is a homeomorphism, so the features of Neural ODEs preserve the topology of the input space.*

A proof is given in the appendix. This statement is a consequence of the flow of an ODE being a homeomorphism (Younes, 2010), i.e. a continuous bijection whose inverse is also continuous, implying that NODEs can only continuously deform the input space and cannot for example tear a connected region apart.

Discrete points and continuous regions. It is worthwhile to consider what these results mean in practice. Indeed, when optimizing NODEs we train on inputs which are sampled from the continuous regions of the annulus and the sphere (see Fig. 4). The flow could then squeeze through the gaps between sampled points making it possible for the NODE to learn a good approximation of the function. However, flows which need to stretch and squeeze the input space in such a way are likely to lead to ill posed ODE problems that are numerically difficult to solve. In order to explore this, we run a number of experiments.

4.1 Experiments

We first compare the performance of ResNets and NODEs on simple regression tasks. To provide a baseline, we not only train on $g(\mathbf{x})$ but also on data which can be made linearly separable without altering the topology of the space (implying that Neural ODEs should be able to easily learn this function). To ensure a fair comparison, we run large hyperparameter searches for each model and repeat each experiment 20 times to ensure results are meaningful across initializations. Full details can be found in the appendix.

We show results for experiments with $d = 1$ and $d = 2$ in Fig. 5. As can be seen, in $d = 1$ the ResNet easily fits the function, while the NODE can not approximate $g(\mathbf{x})$. For $d = 2$, the NODE eventually learns to approximate $g(\mathbf{x})$, but struggles compared to ResNets. This problem is less severe for the

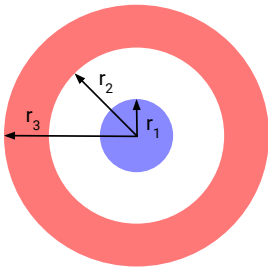
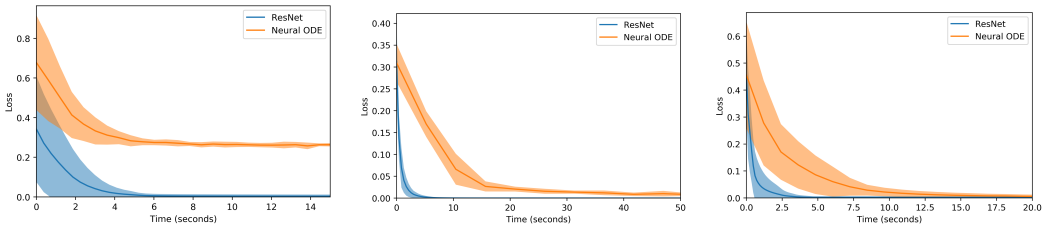


Figure 4: Diagram of $g(\mathbf{x})$ for $d = 2$.



(a) $g(\mathbf{x})$ in $d = 1$ (b) $g(\mathbf{x})$ in $d = 2$ (c) Separable function in $d = 2$

Figure 5: Comparison of training losses of NODEs and ResNets. Compared to ResNets, NODEs struggle to fit $g(\mathbf{x})$ both in $d = 1$ and $d = 2$. The difference between ResNets and NODEs is less pronounced for the separable function.

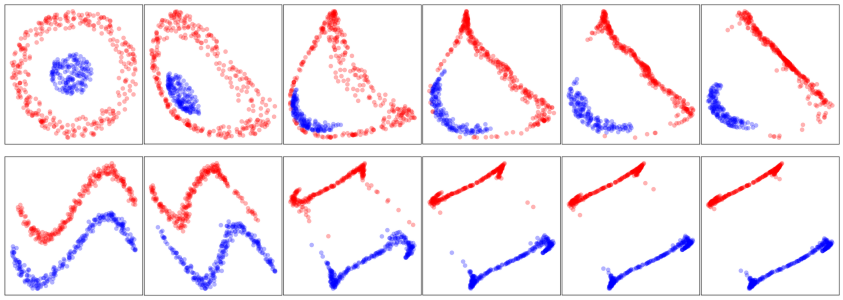


Figure 6: Evolution of the feature space as training progresses. The leftmost tile shows the feature space for a randomly initialized NODE and the rightmost tile shows the feature space after training. The top row shows a model trained on $g(\mathbf{x})$ and the bottow row a model trained on a separable function. As can be seen in the top row, the NODE struggles to push the inner sphere out of the annulus and requires a complicated flow to do so.

separable function, presumably because the flow does not need to break apart any regions to linearly separate them.

To understand what NODEs are learning, it is interesting to visualize how features evolve during training. Plots of the feature space as training progresses are shown in Fig. 6. For $g(\mathbf{x})$, NODEs initially try to move the inner sphere out of the annulus by pushing against and stretching the barrier. Eventually, since we are mapping discrete points and not a continuous region, the flow is able to break apart the annulus to let the flow through. On the other hand, when training on the separable dataset, the NODE easily transforms the input space.

4.2 Computational Cost and Number of Function Evaluations

One of the known limitations of NODEs is that, as training progresses and the flow gets increasingly complex, the number of steps required to solve the ODE increases (Chen et al., 2018; Grathwohl et al., 2018). As the ODE solver evaluates the function $\mathbf{f}$ at each step, this problem is often referred to as the increasing number of function evaluations (NFE). In this section, we explore how the stretching of the input space affects the NFEs and hence the computational cost of making a forward pass of the NODE. To analyse this, we measure how many function evaluations are required for each forward pass of the model as training progresses. Presumably, as the function $\mathbf{f}$ needs to stretch the input space in more complex ways, the number of function evaluations required to solve the ODE system will increase. As illustrated in Fig. 7, this is indeed the case. As training progresses, the NFEs increases implying that the ODE stretching the space to linearly separate the two regions becomes more difficult to solve, making the computation slower.

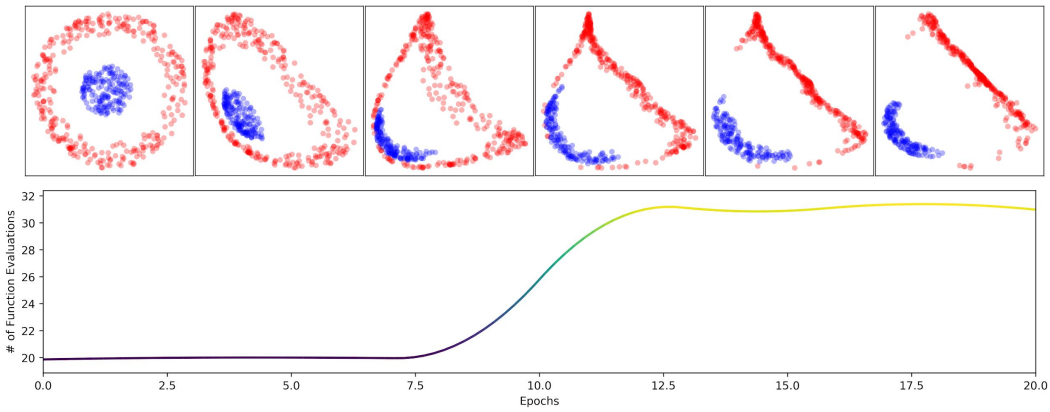


Figure 7: Evolution of the feature space as training progresses and the corresponding number of function evaluations required to solve the ODE. As the ODE needs to break apart the annulus, the number of function evaluations increases.

5 Augmented Neural ODEs

Motivated by our theory and experiments, we introduce Augmented Neural ODEs (ANODEs) which provide a simple solution to the problems we have discussed. We augment the space on which we learn and solve the ODE from $\mathbb{R}^d$ to $\mathbb{R}^{d+p}$, allowing the ODE flow to lift points into the additional dimensions to avoid trajectories intersecting each other. Letting $\mathbf{a}(t) \in \mathbb{R}^p$ denote a point in the augmented part of the space, we can formulate the augmented ODE problem as

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\mathbf{h}(t)\\ \mathbf{a}(t)\end{bmatrix} = \mathbf{f}(\begin{bmatrix}\mathbf{h}(t)\\ \mathbf{a}(t)\end{bmatrix}, t) \\ \begin{bmatrix}\mathbf{h}(0)\\ \mathbf{a}(0)\end{bmatrix} = \begin{bmatrix}\mathbf{x}\\ \mathbf{0}\end{bmatrix} \end{cases}$$

i.e. we concatenate every data point $\mathbf{x}$ with a vector of zeros and solve the ODE on this augmented space. We hypothesize that this will also make the learned (augmented) $\mathbf{f}$ smoother, giving rise to simpler flows that the ODE solver can compute in fewer steps.

In the following sections, we verify this behavior experimentally and show both on toy and image datasets that ANODEs achieve lower losses, better generalization and lower computation cost than regular NODEs.

5.1 Experiments

We first compare the performance of NODEs and ANODEs on toy datasets. As in previous experiments, we run large hyperparameter searches and select the best parameters for each model to ensure a fair comparison. As can be seen on Fig. 8, when trained on $g(\mathbf{x})$ in different dimensions, ANODEs are able to fit the functions NODEs are not and learn much faster than NODEs despite the increased dimension of the input. The corresponding flows learned by the model are shown in Fig. 9. As can be seen, in $d = 1$, the ANODE travels into a higher dimension to linearly separate the points, resulting in a simple, nearly linear flow. Similarly, in $d = 2$, the Neural ODE learns a complicated flow whereas ANODEs simply lift out the inner circle to separate the data.

We can also visualize how the learned features evolve during training. As can be seen in Fig. 10, the features are easily separated by ANODEs whereas NODEs struggle to stretch the outer annulus apart (see Fig. 6).

Computational cost and number of function evaluations. ANODEs strongly reduce the NFEs required to solve the ODE flows for several problems. Indeed, as ANODEs learn simpler flows, they would presumably require fewer iterations to compute. To test this, we measure the NFEs for NODEs and ANODEs when training on $g(\mathbf{x})$. As can be seen in Fig. 11, the NFEs required by ANODEs

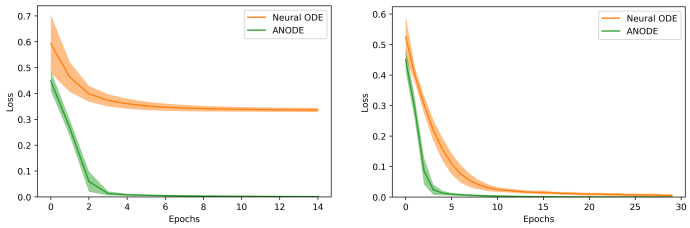


Figure 8: Loss plots for NODEs and ANODEs trained on $g(\mathbf{x})$ in $d = 1$ (left) and $d = 2$ (right). ANODEs easily approximate the functions and are consistently faster than NODEs.

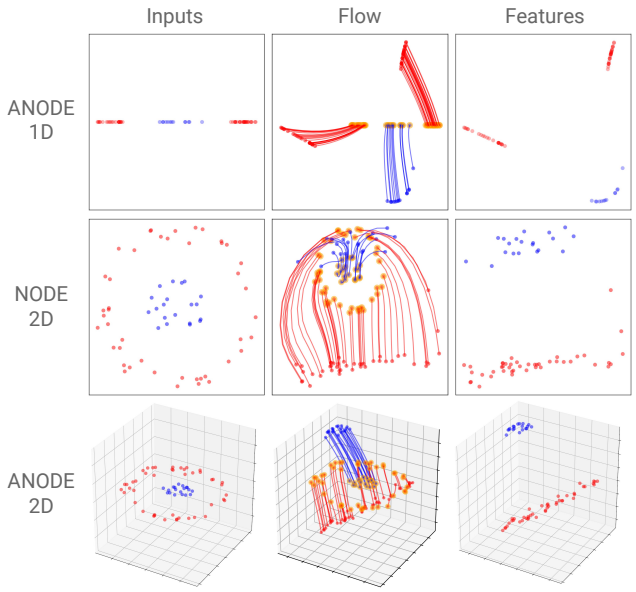


Figure 9: Flows learned by NODEs and ANODEs. ANODEs learn simple nearly linear flows while NODEs learn complex flows that are difficult for the ODE solver to compute.

hardly increases during training while it nearly doubles for NODEs. We obtain similar results when training NODEs and ANODEs on image datasets (see Section 5.2).

Generalization. As ANODEs learn simpler flows, we also hypothesize that they generalize better to unseen data than NODEs. To test this, we first visualize to which value each point in the input space gets mapped by a NODE and an ANODE that have been optimized to approximately zero training loss. As can be seen in Fig. 12a, since NODEs can only continuously deform the input space, the learned flow must squeeze the points in the inner circle through the annulus, leading to poor generalization. ANODEs, in contrast, map all points in the input space to reasonable values.

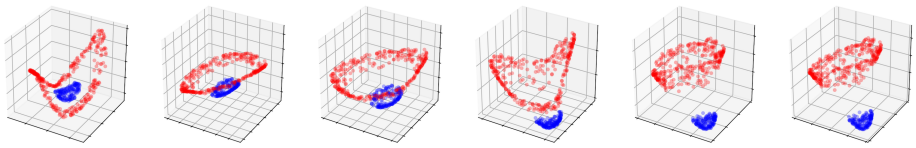


Figure 10: Evolution of features during training for an Augmented Neural ODE. The leftmost tile shows the feature space for a randomly initialized ANODE and the rightmost tile shows the features after training.

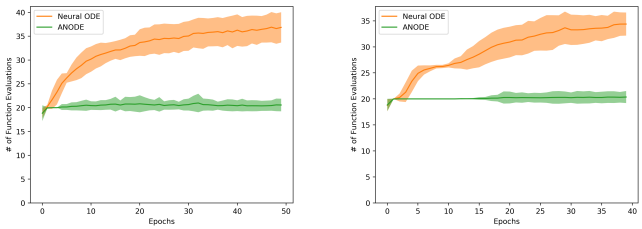
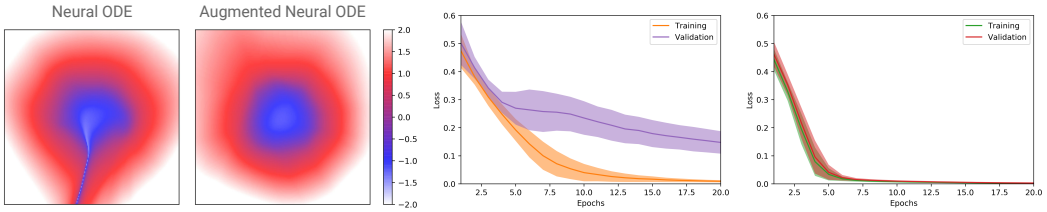


Figure 11: Evolution of the number of function evaluations during training for NODEs and ANODEs trained on $g(\mathbf{x})$ in $d = 1$ (left) and $d = 2$ (right).



(a) (b) (c)

Figure 12: (a) Plots of how NODEs and ANODEs map points in the input space to different outputs (both models achieve approximately the same zero training loss). As can be seen, the Augmented Neural ODE generalizes better. (b) Training and validation losses for NODE. (c) Training and validation losses for ANODE.

As a further test, we can also create a validation set by removing random slices of the input space (e.g. removing all points whose angle is in $[0, \frac{\pi}{5}]$) from the training set. We train both NODEs and ANODEs on the training set and plot the evolution of the validation loss during training in Fig. 12c. While there is a large generalization gap for NODEs, presumably because the flow moves through the gaps in the training set, ANODEs generalize much better and achieve near zero validation loss.

As we have shown, experimentally we obtain lower losses, simpler flows, better generalization and ODEs requiring fewer NFEs to solve when using ANODEs. We now test this behavior on image data by training models on MNIST and CIFAR10.

5.2 Image Experiments

We perform experiments on MNIST and CIFAR10 using convolutional architectures for $\mathbf{f}(\mathbf{h}(t), t)$. As the input $\mathbf{x}$ is an image, the hidden state $\mathbf{h}(t)$ is now in $\mathbb{R}^{c \times h \times w}$ where c is the number of channels and h and w are the height and width respectively. In the case where $\mathbf{h}(t) \in \mathbb{R}^d$ we augmented the space as $\mathbf{h}(t) \in \mathbb{R}^{d+p}$. For images we augment the space as $\mathbb{R}^{c \times h \times w} \to \mathbb{R}^{(c+p) \times h \times w}$, i.e. we add p channels of zeros to the input image. While there are other ways to augment the space, we found that increasing the number of channels works well in practice and use this method for all experiments. Full training and architecture details can be found in the appendix.

We also note that different architectures exist for training NODEs on images. Chen et al. (2018) for example, use a combination of downsampling with regular convolutions before applying a sequence of repeated ODE flows. While some of these architectures can be understood as implicitly augmenting the space (since downsampling convolutions increase the number of channels), for sake of comparison we here refer to NODEs as an ODE layer followed by a linear layer (as described in Section 2) and ANODEs as an augmented ODE layer followed by a linear layer.

Results for models trained with and without augmentation are shown in Fig. 13. As can be seen, ANODEs train faster and obtain lower losses at a smaller computational cost than NODEs. On MNIST for example, ANODEs with 10 augmented dimensions achieve the same loss in roughly 10 times fewer iterations (for CIFAR10, ANODEs are roughly 5 times faster).

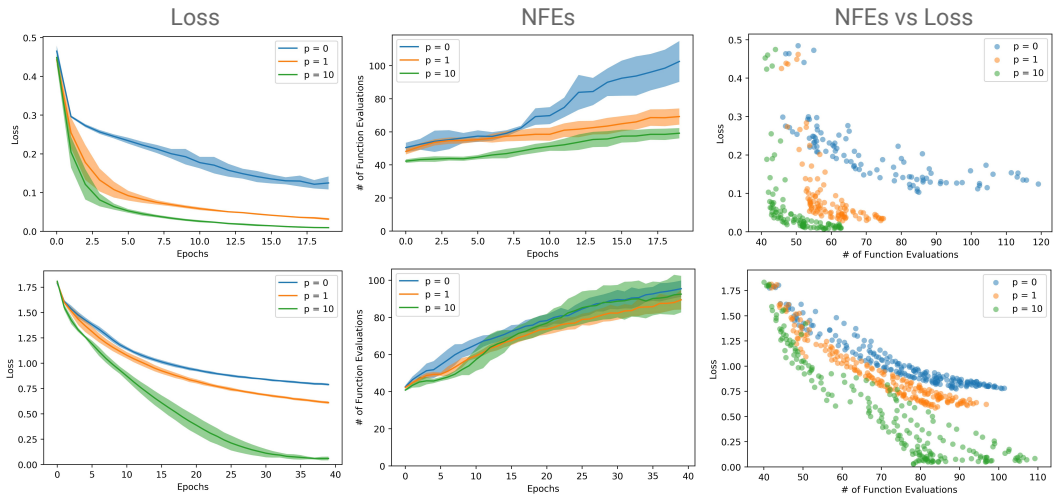


Figure 13: Losses, NFEs and NFEs vs Loss for various augmented models on MNIST (top row) and CIFAR10 (bottom row). Note that p indicates the size of the augmented dimension, so $p = 0$ corresponds to a regular NODE model.

Perhaps most interestingly, we can plot the NFEs against the loss to understand roughly how complex a flow (i.e. how many NFEs) are required to model a function that achieves a certain loss. For example, to compute a function which obtains a loss of 0.8 on CIFAR10, a NODE requires approximately 100 function evaluations whereas ANODEs only require 50. Similar observations can be made for MNIST, implying that ANODEs can model equally rich functions at half the computational cost of NODEs.

Parameter efficiency. As we augment the dimension of the ODEs, we also increase the number of parameters of the models, so it may be that the improved performance of ANODEs is due to the higher number of parameters. To test this, we train a NODE and an ANODE with the same number of parameters on both MNIST (84k weights) and CIFAR10 (172k weights). As can be seen in Fig. 14, the augmented model achieves lower losses with fewer NFEs than a NODE with the same number of parameters, suggesting that ANODEs use the parameters more efficiently than NODEs.

NFEs and weight decay. The increased computational cost as training progresses is a known issue with Neural ODEs and has previously been tackled by adding weight decay during training (Grathwohl et al., 2018). As ANODEs also achieve lower computational cost, we test models with various combinations of weight decay and augmentation (see Fig. 15). As can be seen, ANODEs outperform NODEs even when using weight decay. However, using both weight decay and augmentation achieves the lowest NFEs at the cost of a slightly higher loss. Combining augmentation with weight decay may therefore be a fruitful avenue for further scaling Neural ODE models.

Generalization for images. As noted in Section 5.1, ANODEs generalize better than NODEs on simple datasets, presumably because they learn simpler and smoother flows. We also test this behavior on CIFAR10 by training models with and without augmentation on the training set and calculating the loss on the test set. As can be seen on Fig. 16, both the NODE and ANODE overfit the training data, but ANODEs achieve lower validation loss than NODEs (1.18 vs 1.34). This suggests that ANODEs also achieve better generalization on image datasets.

Stability and scaling. While experimenting with NODEs we found that the NFEs could often become prohibitively large. For example, when overfitting a NODE on MNIST, the learned flow can become so ill posed the ODE solver requires timesteps that are smaller than machine precision resulting in underflow. Further, while we cap the NFEs to 1000 during training (which roughly corresponds to a ResNet with 1000 layers which should be sufficient for MNIST), the model regularly requires more than this to compute the flow. Further, this complex flow often leads to unstable training resulting in exploding or wildly varying losses. This unstable behavior is likely a function of many factors, such as the choice of architecture, activation function and optimizer and it may be that an appropriate choice of these could lead to more stable models. However, we observed this phenomenon

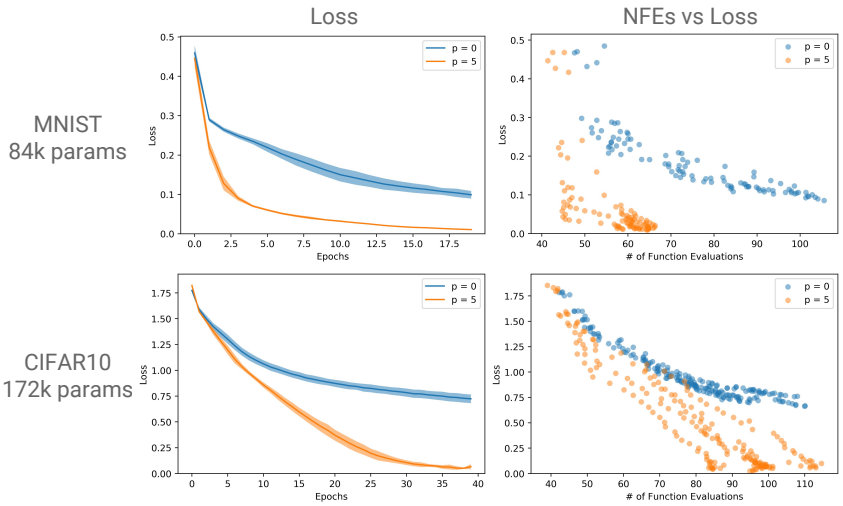


Figure 14: Losses, NFEs and NFEs vs Loss for various augmented models on MNIST and CIFAR10. Note that p indicates the size of the augmented dimension, so $p = 0$ corresponds to a regular NODE model.

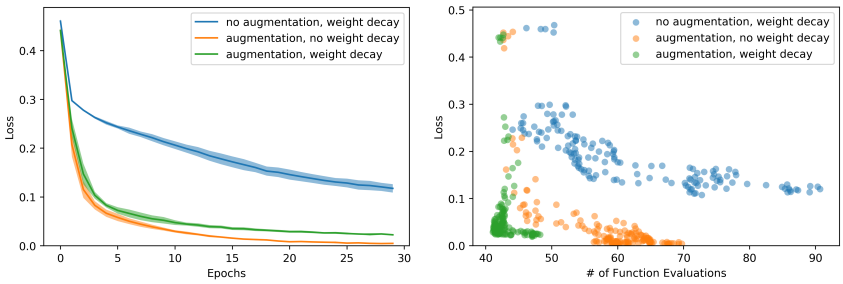


Figure 15: Losses and NFEs for models with and without weight decay. ANODEs perform better than NODEs with weight decay but adding weight decay to ANODEs also reduces their NFEs at the cost of a slightly higher loss.

across a variety of architectures and hyperparameters. We also found that augmentation consistently lead to more stable training and fewer function evaluations, even when overfitting datasets. Results for models (with the same number of parameters) trained on MNIST are shown in Fig. 17. As can be seen, the training of ANODEs is stable, leading to models with low losses and low NFEs. NODEs in contrast tend to become unstable when overfitting the data and often learn flows that require a prohibitively large number of steps to solve (several hundred function evaluations). We hope that ANODEs will facilitate the scaling of Neural ODE models and plan to explore this further in future research.

Augmentation for ResNets. Since ResNets can be interpreted as discretized equivalents of Neural ODEs, it is also interesting to consider how augmenting the space could affect the training of ResNets. Indeed, most ResNet architectures (He et al., 2016; Xie et al., 2017) already employ a form of augmentation by performing convolutions with a large number of filters before applying residual blocks. This effectively corresponds to augmenting the space by the number of filters minus the number of channels in the original image. Further, Behrmann et al. (2018) also augment the input with zeros to build invertible ResNets. Through the analogy between NODEs and ResNets, we hope that some of the ideas presented in this paper could be used to guide future research into ResNet architectures.

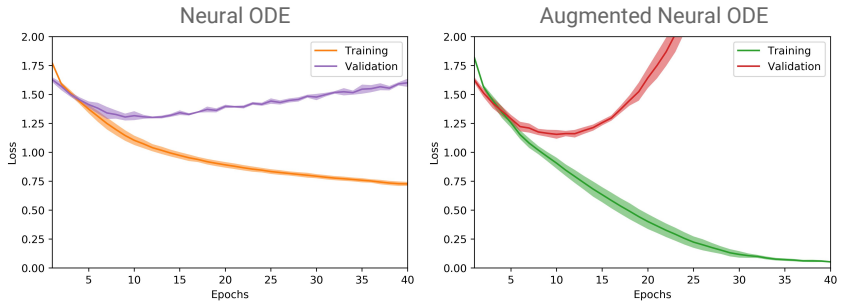


Figure 16: Training and validation losses on CIFAR10 for NODEs and ANODEs. Both models overfit the training data, but ANODEs achieve a lower minimum for the validation loss.

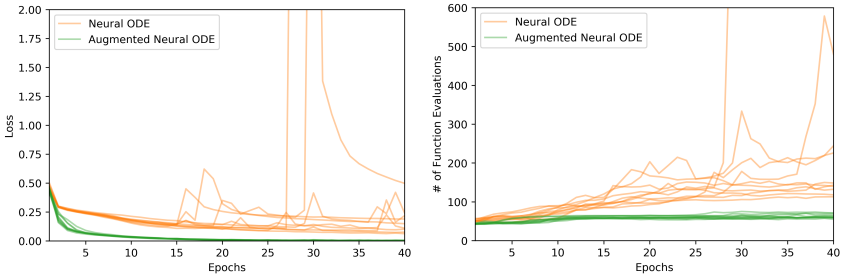


Figure 17: Instabilities in the loss (left) and NFEs (right) when fitting NODEs to MNIST. In the latter stages of training NODEs can become unstable and the loss and NFEs become erratic.

6 Scope and Future Work

In this section, we describe some limitations of ANODEs, outline potential ways they may be overcome and list ideas for future work. First, while ANODEs are faster than NODEs, they are still slower than ResNets. This stands in the way of scaling such models to very large datasets and we believe further research is needed in this area. Second, there may be different architectural choices that could have similar properties to those exhibited by ANODEs. Chen et al. (2018) for example, downsample MNIST twice with regular convolutions (and hence also increase the number of channels in a similar way to ANODEs) before applying a sequence of NODEs to train on MNIST. Finally, the augmented dimension can be seen as an extra hyperparameter to tune. While the model is robust for a large range of augmented dimensions, we observed that for excessively large augmented dimensions (e.g. adding a 100 channels to MNIST), the model tends to perform worse yielding higher loss and NFEs.

We believe the ideas presented in this paper could create other interesting avenues for future research, including:

Overcoming the limitations of Neural ODEs. In order to allow trajectories to travel across each other, we augmented the space on which the ODE is solved. However, there may be other ways to achieve this, such as learning an augmentation (as in ResNets) or adding noise (in a similar way to Wang et al. (2018)).

Augmentation for Normalizing Flows. The NFEs typically becomes prohibitively large when training continuous normalizing flow models (Grathwohl et al., 2018). Adding augmentation to continuous normalizing flows could likely mitigate this effect and we plan to explore this in future work.

Improving our understanding of augmentation. It would be interesting to provide more theoretical arguments for how and why augmentation improves the training of NODE models. For example it would be interesting to more precisely characterize what is meant by ANODEs learning simpler flows and to explore how this could guide our choice of architectures and optimizers for Neural ODEs.

7 Conclusion

In this paper, we highlighted and analysed some of the limitations of Neural ODEs. We proved that there are classes of functions Neural ODEs cannot represent and, in particular, that Neural ODEs only learn features that are homeomorphic to the input space. We showed through experiments that this lead to slower learning and increasingly complex flows which are expensive to compute. To mitigate these issues, we proposed Augmented Neural ODEs which learn the flow from input to features in an augmented space. Our experiments show that Augmented Neural ODEs can model more complex functions using simpler flows. In addition, they achieve lower losses, reduce computational cost, and improve stability and generalization. In future work we hope to extend these ideas to a more general setting, including for continuous normalizing flows.

Acknowledgements

We would like to thank Anthony Caterini, Daniel Paulin, Abraham Ng, Joost Van Amersfoort and Hyunjik Kim for helpful discussions and feedback. Emilien gratefully acknowledges his PhD funding from Google DeepMind. Arnaud Doucet acknowledges support of the UK Defence Science and Technology Laboratory (Dstl) and Engineering and Physical Research Council (EPSRC) under grant EP/R013616/1. This is part of the collaboration between US DOD, UK MOD and UK EPSRC under the Multidisciplinary University Research Initiative. Yee Whye Teh's research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 617071.

References

Shair Ahmad and Antonio Ambrosetti. *A Textbook on Ordinary Differential Equations*, volume 88. Springer, 2015.

Mark Anthony Armstrong. *Basic Topology*. Springer Science & Business Media, 2013.

Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2018.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *32nd Conference on Neural Information Processing Systems*, 2018.

Earl A Coddington and Norman Levinson. *Theory of Ordinary Differential Equations*. Tata McGraw-Hill Education, 1955.

Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

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

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*, pp. 770–778, 2016.

Ralph Howard. The Gronwall inequality. 1998. URL `http://people.math.sc.edu/howard/Notes/gronwall.pdf`.

Hongzhou Lin and Stefanie Jegelka. Resnet with one-neuron hidden layers is a universal approximator. In *32nd Conference on Neural Information Processing Systems*, 2018.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.

Bao Wang, Binjie Yuan, Zuoqiang Shi, and Stanley J Osher. Enresnet: Resnet ensemble via the Feynman-Kac formalism. *arXiv preprint arXiv:1811.10745*, 2018.

E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1492–1500, 2017.

Laurent Younes. *Shapes and Diffeomorphisms*, volume 171. Springer Science & Business Media, 2010.

A Proofs

Throughout this section, we refer to the following Initial Value Problem (IVP)

$$\begin{cases} \frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{h}(t), t) \\ \mathbf{h}(0) = \mathbf{x} \end{cases} \tag{1}$$

where $\mathbf{h}(t) \in \mathbb{R}^d$ and $\mathbf{f} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is continuous in t and globally Lipschitz continuous in $\mathbf{h}$, i.e. there is a constant $C \geq 0$ such that

$$\|\mathbf{f}(\mathbf{h}_2(t), t) - \mathbf{f}(\mathbf{h}_1(t), t)\| \leq C\|\mathbf{h}_2(t) - \mathbf{h}_1(t)\|$$

for all $t \in \mathbb{R}$. These conditions imply the solutions of the IVP exist and are unique for all t (see e.g. Theorem 2.4.5 in Ahmad & Ambrosetti (2015)).

We define the flow $\phi_t(\mathbf{x})$ associated to the vector field $\mathbf{f}(\mathbf{h}(t), t)$ as the solution at time t of the ODE starting from the initial condition $\mathbf{h}(0) = \mathbf{x}$. The flow measures how the solutions of the ODE depend on the initial conditions. Following the analogy between ResNets and Neural ODEs, we define the features $\phi(\mathbf{x})$ output by the ODE as the flow at the final time T to which we solve the ODE, i.e. $\phi(\mathbf{x}) = \phi_T(\mathbf{x})$. Finally, we define the Neural ODE model as the composition of the feature function $\phi : \mathbb{R}^d \to \mathbb{R}^d$ and a linear map $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.

For clarity and completeness, we include proofs of all statements. Whenever propositions or theorems are already known we include references to proofs.

A.1 ODE trajectories do not intersect

This result is well known and proofs can be found in standard ODE textbooks (e.g. Proposition C.6 in Younes (2010)).

Proposition. *Let $\mathbf{h}_1(t)$ and $\mathbf{h}_2(t)$ be two solutions of the ODE (1) with different initial conditions, i.e. $\mathbf{h}_1(0) \neq \mathbf{h}_2(0)$. Then, for all $t \in (0, T]$, $\mathbf{h}_1(t) \neq \mathbf{h}_2(t)$. Informally, this proposition states that ODE trajectories cannot intersect.*

Proof. Suppose there exists some $\tilde{t} \in (0, T]$ where $\mathbf{h}_1(\tilde{t}) = \mathbf{h}_2(\tilde{t})$. Define a new IVP with initial condition $\mathbf{h}(\tilde{t}) = \mathbf{h}_1(\tilde{t}) = \mathbf{h}_2(\tilde{t})$ and solve it backwards to time $t = 0$. As the backwards IVP also satisfies the existence and uniqueness conditions, its solution $\mathbf{h}(t)$ is unique implying that its value at $t = 0$ is unique. This contradicts the assumption that $\mathbf{h}_1(0) \neq \mathbf{h}_2(0)$ and so there is no $\tilde{t} \in (0, T]$ such that $\mathbf{h}_1(\tilde{t}) = \mathbf{h}_2(\tilde{t})$.

A.2 Gronwall's Lemma

We will make use of Gronwall's Lemma and state it here for completeness. We follow the statement as given in Howard (1998):

Theorem. *Let $U \subset \mathbb{R}^d$ be an open set. Let $\mathbf{f} : U \times [0, T] \to \mathbb{R}^d$ be a continuous function and let $\mathbf{h}_1, \mathbf{h}_2 : [0, T] \to U$ satisfy the IVPs:*

$$\frac{\mathrm{d}\mathbf{h}_1(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{h}_1(t), t), \quad \mathbf{h}_1(0) = \mathbf{x}_1$$

$$\frac{\mathrm{d}\mathbf{h}_2(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{h}_2(t), t), \quad \mathbf{h}_2(0) = \mathbf{x}_2$$

Assume there is a constant $C \geq 0$ such that

$$\|\mathbf{f}(\mathbf{h}_2(t), t) - \mathbf{f}(\mathbf{h}_1(t), t)\| \leq C\|\mathbf{h}_2(t) - \mathbf{h}_1(t)\|$$

Then for $t \in [0, T]$

$$\|\mathbf{h}_2(t) - \mathbf{h}_1(t)\| \leq e^{Ct}\|\mathbf{x}_2 - \mathbf{x}_1\|$$

Proof. See e.g. Howard (1998) or Theorem 3.8 in Younes (2010).

B Proof for 1d example

Let $g_{1\text{d}} : \mathbb{R} \to \mathbb{R}$ be a function such that

$$\begin{cases} g_{1\text{d}}(-1) = 1 \\ g_{1\text{d}}(1) = -1 \end{cases}$$

Proposition 1. *The flow of an ODE cannot represent $g_{1d}(x)$.*

Proof. The proof follows two steps:

(a) Continuous trajectories mapping -1 to 1 and 1 to -1 must cross each other.

(b) Trajectories of ODEs cannot cross each other.

which is a contradiction and implies the proposition. Part (b) was proved in Section A.1. All there is left to do is to prove part (a).

Suppose there exists an $\mathbf{f}$ such that there are trajectories $h_1(t)$ and $h_2(t)$ where

$$\begin{cases} h_1(0) = -1 \quad h_1(T) = 1 \\ h_2(0) = 1 \quad h_2(T) = -1 \end{cases}$$

As $h_1(t)$ and $h_2(t)$ are solutions of the IVP, they are continuous (Coddington & Levinson, 1955). Define the function $h(t) = h_2(t) - h_1(t)$. Since both $h_1(t)$ and $h_2(t)$ are continuous, so is $h(t)$. Now $h(0) = 2$ and $h(T) = -2$, so by the Intermediate Value Theorem there is some $\tilde{t} \in [0, T]$ where $h(\tilde{t}) = 0$, i.e. where $h_1(\tilde{t}) = h_2(\tilde{t})$. So $h_1(t)$ and $h_2(t)$ intersect.

C Proof that $\phi_t(\mathbf{x})$ is a homeomorphism

Since the following theorem plays a central part in the paper, we include a proof of it here for completeness. For a more general proof, see e.g. Theorem C.7 in Younes (2010).

Theorem. *For all $t \in [0, T]$, $\phi_t : \mathbb{R}^d \to \mathbb{R}^d$ is a homeomorphism.*

Proof. In order to prove that ϕ_t is a homemorphism, we need to show that

(a) ϕ_t is continuous

(b) ϕ_t is a bijection

(c) ϕ_t^{-1} is continuous

Part (a). Consider two initial conditions of the ODE system, $\mathbf{h}_1(0) = \mathbf{x}$ and $\mathbf{h}_2(0) = \mathbf{x} + \delta$ where δ is some perturbation. By Gronwall's Lemma, we have

$$\|\mathbf{h}_2(t) - \mathbf{h}_1(t)\| \leq e^{Ct} \|\mathbf{h}_1(0) - \mathbf{h}_2(0)\| = e^{Ct} \|\delta\|$$

Rewriting in terms of $\phi_t(\mathbf{x})$, we have

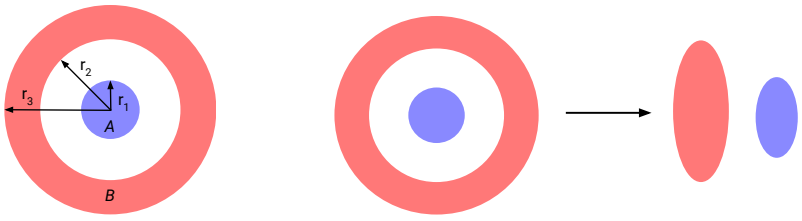
$$\|\phi_t(\mathbf{x} + \delta) - \phi_t(\mathbf{x})\| \leq e^{Ct} \|\delta\|$$

Letting $\delta \to 0$, this implies that $\phi_t(\mathbf{x})$ is continuous in $\mathbf{x}$ for all $t \in [0, T]$.

Part (b). Suppose there exists initial conditions $\mathbf{x}_1 \neq \mathbf{x}_2$ such that $\phi_t(\mathbf{x}_1) = \phi_t(\mathbf{x}_2)$. We define the IVP starting from $\phi_t(\mathbf{x}_1)$ and solve it backwards to time $t = 0$. The solution of the IVP is unique, so it cannot map $\phi_t(\mathbf{x}_1)$ back to both $\mathbf{x}_1$ and $\mathbf{x}_2$. So for each $\mathbf{x}_1 \neq \mathbf{x}_2$, we must have $\phi_t(\mathbf{x}_1) \neq \phi_t(\mathbf{x}_2)$, that is the map between $\mathbf{x}$ and $\phi_t(\mathbf{x})$ is one-to-one.

Part (c). To check that the inverse ϕ_t^{-1} is continuous, we note that we can set the initial condition to $\mathbf{h}(t) = \phi_t(\mathbf{x})$ and solve the IVP backwards in time (as it satisfies the existence and uniqueness conditions). The same reasoning as part (a) then applies.

Therefore ϕ_t is a continuous bijection and its inverse is continuous, i.e. it is a homeomorphism.



(a) (b)

Figure 18: (a) Diagram of $g(\mathbf{x})$ in 2d. (b) An example of the map $\phi(\mathbf{x})$ from input data to features necessary to represent $g(\mathbf{x})$ (which NODEs cannot learn).

Corollary. *Features of Neural ODEs preserve the topology of the input space.*

Proof. Since $\phi_t(\mathbf{x})$ is a homeomorphism, so is $\phi(\mathbf{x}) = \phi_T(\mathbf{x})$. Homeomorphims preserve topological properties, so Neural ODEs can only learn features which have the same topology as the input space.

This corollary implies for example that Neural ODEs cannot break apart or create holes in a connected region of the input space.

D Proof that there are classes functions Neural ODEs cannot represent

This section presents a proof of the main claim of the paper.

Let $0 < r_1 < r_2 < r_3$ and let $g : \mathbb{R}^d \to \mathbb{R}$ be a function such that

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

We denote the sphere where $g(\mathbf{x}) = -1$ by $A = \{\mathbf{x} : \|\mathbf{x}\| \leq r_1\}$ and the annulus where $g(\mathbf{x}) = 1$ by $B = \{\mathbf{x} : r_2 \leq \|\mathbf{x}\| \leq r_3\}$ (see Fig. 18). For a set S, we write $\phi(S) = \{\mathbf{y} : \mathbf{y} = \phi(\mathbf{x}), \mathbf{x} \in S\}$ to denote the feature transformation of the set.

Proposition 2. *Neural ODEs cannot represent* $g(\mathbf{x})$.

Proof. For a Neural ODE to map points in A to -1 and points in B to $+1$, the linear map $\mathcal{L}$ must map the features in $\phi(A)$ to -1 and the features in $\phi(B)$ to $+1$, which implies that $\phi(A)$ and $\phi(B)$ must be linearly separable. We now show that this is not possible if ϕ is a homeomorphism.

Define a disk $D \subset \mathbb{R}^d$ by $D = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| \leq r_2\}$ with boundary $\partial D = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = r_2\}$ and interior $\text{int}(D) = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| < r_2\}$. Now $A \subset \text{int}(D)$, $A \cap \partial D = \emptyset$ and $\partial D \subset B$, that is all points in ∂D should be mapped to $+1$ (i.e. they are in B) and a subset of points in $\text{int}(D)$ should be mapped to -1 (i.e. they are in A). So if $\phi(\text{int}(D))$ and $\phi(\partial D)$ are not linearly separable, then neither are $\phi(A)$ or $\phi(B)$.

The feature transformation ϕ is a homeomorphism, so $\phi(\text{int}(D)) = \text{int}(\phi(D))$ and $\phi(\partial D) = \partial(\phi(D))$, i.e. points on the boundary get mapped to points on the boundary and points in the interior to points in the interior (Armstrong, 2013). So it remains to show that $\text{int}(\phi(D))$ and $\partial(\phi(D))$ cannot be linearly separated. For notational convenience, we will write $D' = \phi(D)$.

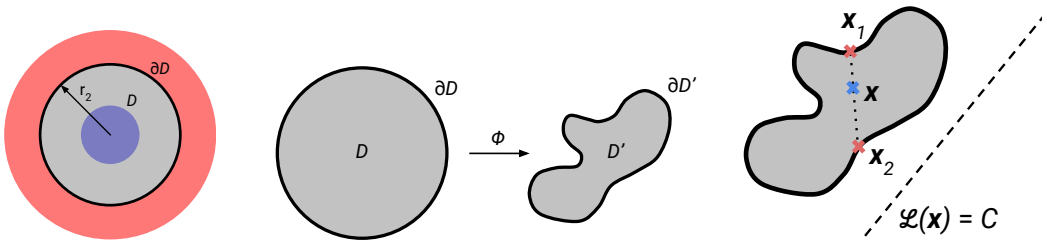
Suppose all points in $\partial D'$ lie above some hyperplane, i.e. suppose there exists a linear function $\mathcal{L}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ and a constant C such that $\mathcal{L}(\mathbf{x}) > C$ for all $\mathbf{x} \in \partial D'$. If $\text{int}(D')$ were linearly separable from $\partial D'$ then $\mathcal{L}(\mathbf{x}) < C$ for all $\mathbf{x} \in \text{int}(D')$. We now show that this is not the case. Since D' is a connected subset of $\mathbb{R}^d$ (since D is connected and ϕ is a homeomorphism), every point $\mathbf{x} \in \text{int}(D')$ can be written as a convex combination of points on the boundary $\partial D'$ (to see this consider a line passing through a point $\mathbf{x}$ in the interior and its intersection with the boundary). So if $\mathbf{x} \in \text{int}(D')$, then

$$\mathbf{x} = \lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$$

for some $\mathbf{x}_1, \mathbf{x}_2 \in \partial D'$ and $0 < \lambda < 1$. Now,

$$\begin{aligned}
\mathcal{L}(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} \\
&= \mathbf{w}^T(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \\
&= \lambda \mathbf{w}^T \mathbf{x}_1 + (1-\lambda)\mathbf{w}^T \mathbf{x}_2 \\
&\geq \lambda C + (1-\lambda)C \\
&= C
\end{aligned}$$

so all points in the interior are on the same side of the hyperplane as points on the boundary, that is the interior and the boundary are not linearly separable. This implies that the set of features $\phi(A)$ and $\phi(B)$ cannot be linearly separated and so that Neural ODEs cannot represent $g(\mathbf{x})$.



(a) (b) (c)

Figure 19: (a) Diagram of the disk D and its boundary. The boundary is equal to the inner boundary of B. (b) An example of how ϕ transforms the disk. (c) The boundary of the transformed set is above the hyperplane, which implies that all points on the interior must also be above the hyperplane.

E Experimental Details

We used the ODE solvers in the `torchdiffeq`[1] library for all experiments (Chen et al., 2018). We used the Runge-Kutta 4 solver with an absolute and relative error tolerance of 1e-3.

E.1 Architecture

Throughout all our experiments we used the ReLU activation function. We also experimented with softplus but found that this generally slowed down learning.

E.1.1 Toy datasets

We parameterized $\mathbf{f}$ by an MLP with the following structure and dimensions

$$d_{\text{input}} + 1 \to d_{\text{hidden}} \to \text{ReLU} \to d_{\text{hidden}} \to \text{ReLU} \to d_{\text{input}}$$

where the additional dimension on the input layer is because we append the time t as an input. Choices for d_{input} and d_{hidden} are given for each model in the following section.

E.1.2 Image datasets

We parameterized $\mathbf{f}$ by a convolutional block with the following structure and dimensions

- 1×1 conv, k filters, 0 padding.
- 3×3 conv, k filters, 1 padding.
- 1×1 conv, c filters, 0 padding.

[1] https://github.com/rtqichen/torchdiffeq

where k is specified for each architecture in the following sections and c is the number of channels (1 for MNIST and 3 for CIFAR10). We append the time t as an extra channel on the feature map before each convolution.

E.2 Hyperparameters

For the toy datasets, each experiment was repeated 20 times. The resulting plots show the mean and standard deviation for these runs.

E.2.1 Hyperparameter search

To ensure a fair comparison between models, we ran a large hyperparameter search for each model and chose the best hyperparameters to generate the loss plots in the paper. We used `skorch` and `scikit-learn` (Pedregosa et al., 2011) to run the hyperparameter searches and ran 3 cross validations for each setting.

For $d = 1$ and $d = 2$ we trained on $g(\mathbf{x})$ (i.e. on the dataset of concentric spheres), with 1000 points in the inner sphere and 2000 points in the outer annulus. We used $r_1 = 0.5$, $r_2 = 1.0$ and $r_3 = 1.5$ and trained for 50 epochs. The space of hyperparameters we searched were:

- Batch size: 64, 128
- Learning rate: 1e-3, 5-4, 1e-4
- Hidden dimension: 16, 32
- Number of layers (for ResNet): 2, 5, 10
- Number of augmented dimensions (for ANODE): 1, 2, 5

The best parameters for ResNets:

- $d = 1$: Batch size 64, learning rate 1e-3, hidden dimension 32, 5 layers
- $d = 2$: Batch size 64, learning rate 1e-3, hidden dimension 32, 5 layers

The best parameters for Neural ODEs:

- $d = 1$: Batch size 64, learning rate 1e-3, hidden dimension 32
- $d = 2$: Batch size 64, learning rate 1e-3, hidden dimension 32

The best parameters for Augmented Neural ODEs:

- $d = 1$: Batch size 64, learning rate 1e-3, hidden dimension 32, augmented dimension 5
- $d = 2$: Batch size 64, learning rate 1e-3, hidden dimension 32, augmented dimension 5

E.2.2 Image experiments

For both MNIST and CIFAR10, we used $k = 64$ filters and repeated each experiment 5 times. For models with the same number of parameters we used, for MNIST

- Neural ODE: 92 filters $\rightarrow$ 84395 parameters
- Augmented Neural ODE: 64 filters, augmented dimension 5 $\rightarrow$ 84816 parameters

and for CIFAR10

- Neural ODE: 125 filters $\rightarrow$ 172358 parameters
- Augmented Neural ODE: 64 filters, augmented dimension 10 $\rightarrow$ 171799 parameters

.