# Overcoming Limitations of Neural Ordinary Differential Equations

**AJ**

## Abstract

Neural Ordinary Differential Equations (Neural ODEs) have recently been introduced as a new class of continuous-depth neural nets in the spirit of the limits of ResNets. Subsequent work has shown that reliance on ODEs can prove problematic in terms of expressivity. In this work, we seek to highlight these issues and how they have been addressed. Notably, we discuss data augmentation and a generalized formulation of Neural ODEs. Finally, we use a toy dataset to display the issues at hand.

## 1 Introduction

In 2018, Chen et al. [1] popularized the concept of neural ordinary differential equations as a limiting case of Residual Networks [2]. Formally, starting from an input layer $\boldsymbol{x} = \boldsymbol{y}(0) \in \mathbb{R}^{n_x}$, the output $\boldsymbol{y}_T \in \mathbb{R}^{n_x}$ of a ResNet is given by successively applying

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + f(\theta_t, \boldsymbol{y}_t).$$

Interpreting this as the Euler discretization of a continuous transformation, in the limit where time steps go to zero, the output becomes the solution to the initial value problem (IVP) at time $T$

$$\frac{\mathrm{d}\boldsymbol{y}(t)}{\mathrm{d}t} = f_\theta(t, \boldsymbol{y}(t)), \quad \boldsymbol{y}(0) = \boldsymbol{x} \tag{1}$$

where $\theta \in \mathbb{R}^{n_\theta}$ is a vector of parameters of $f_\theta$, a neural network architecture with $f_\theta$ Lipschitz. The initial motivation of Chen et al. [1] was partially to develop continuous time-series models in a memory efficient way. However, some months later, Dupont et al. [3] displayed some issues with Neural ODEs, namely that the feature mapping $\boldsymbol{y}$ solving 1 is a homeomorphism for each $t \in \mathcal{T} = [0, T]$, so that the topology of the input space is preserved. This implies the impossibility of training some functions such as the reflection function $\phi(x) = -x$ in one dimension. The natural question that arises is can we get around this theoretical limitation, and if so, how? In the same paper, Dupont et al. [3] introduce a new type of net, ANODEs (Augmented Neural Ordinary Differential Equations) in order to increase expressivity. Zhang et al. [4] prove that for a sufficiently sized augmentation, ANODEs give rise to a universal approximation theorem for homeomorphisms. In contrast, Massaroli et al. [5] propose a more general version of 1, introducing depth-dependence[1] in the net, generalizing augmentation and giving alternatives to it.

**Our contributions** In this paper, we seek to discuss ways of ameliorating the expressivity of neural ODEs. In the following section, we outline the theory of data augmentation and its universal approximation theorem. In Section 3, we show how a generalized IVP brings about alternative solutions to the homeomorphism issue. Moreover, in Section 4, we use a dataset akin to the concentric annuli dataset [3] to visualize the generalization effects of different NODE types and analyse their performance.

[1] By dependence on depth, we mean dependence on $t \in \mathcal{T}$, which is the continuous analogue of depth in ResNets and discrete neural nets in general. Time and depth are used interchangeably.

**Related Work** The paper by Chen et al. [1] generated a lot of traction in the community. Finlay et al. [6] introduce stability regularizations as a way of training generative based models, building on the work in [7]. On the other hand, differential equation nets have since evolved to other settings, such as controlled differential equations [8] as a limit of recurrent nets, or stochastic differential equations [9, 10]. Neural ODEs have been used to model multi-body dynamics using concepts from physics such as Lagrangian NODEs [11] and Hamiltonian NODEs [12]. Kidger recently published his PhD thesis [13] compiling results on neural differential equations.

## 2 Data Augmentation

A basic property of ODEs is that their solutions cannot intersect. By association, this means that neural ODEs cannot approximate certain functions. In their paper, Dupont et al. [3] give the example of functions on concentric annuli in $\mathbb{R}^d$. For $0 \leq r_1 < r_2 < r_3 < r_4$, define $g : \mathbb{R}^d \to \mathbb{R}$ to be any function such that

$$\begin{cases} g(\boldsymbol{x}) = -1 & \text{if } r_1 \leq \|\boldsymbol{x}\|_2 \leq r_2 \\ g(\boldsymbol{x}) = 1 & \text{if } r_3 \leq \|\boldsymbol{x}\|_2 \leq r_4 \end{cases}$$

Then an ODE net cannot represent $g$. In practice, when training such a net, the number of data points used is finite, technically making the data separable. But this comes at the expense of computation and generalization. Indeed, the net will create a flow that opens up the outer annulus, and separate the data. This will not generalize well: there's a region of space that will be misclassified. Moreover, when calling an ODE solver at each step, it determines automatically, based on the complexity of the ODE, the number of discretization steps to find the solution. The flow required to open up the space gets increasingly complicated to ensure the classification is done correctly, therefore the number of function evaluations (NFE) increases. One can find other functions that cannot be approximated, such as the reflection and its generalization as stated in [4, Theorem 1]. This leads to computational issue when training as the net will learn a complicated flow to approximate $g$ by heavily warping input space.

### 2.1 Augmented Neural ODEs

To circumvent this theoretical and computational impasse, Dupont et al. [3] have the idea of lifting the ODE flow in a higher dimension by embedding the vector field in a higher dimension and padding the input vector with zeros. The idea is that allowing movement in higher dimensions would prevent trajectories crossing due to the extra freedom in direction, and thus the learned trajectories would be simpler. The computational benefit would come from the fact that a simple flow implies few NFEs, and therefore less training time. In this setting, the IVP 1 becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix}\boldsymbol{y}(t)\\ \boldsymbol{h}(t)\end{pmatrix} = f_\theta\left(\begin{pmatrix}\boldsymbol{y}(t)\\ \boldsymbol{h}(t)\end{pmatrix}, t\right), \qquad \begin{pmatrix}\boldsymbol{y}(0)\\ \boldsymbol{h}(0)\end{pmatrix} = \begin{pmatrix}\boldsymbol{x}\\ \mathbf{0}\end{pmatrix} \tag{2}$$

where $\boldsymbol{h}(t), \mathbf{0} \in \mathbb{R}^p$ and $p$ is the augmentation dimension. We call such a net a $\mathbf{0}_p$-ODE-net, or simply an ANODE. The experiments reported in the paper show a significant improvement over vanilla neural ODEs in terms of generalization, NFEs and flow learned.

### 2.2 Universal Approximation Theorem

The natural question to ask is does augmentation actually improve the generalization capabilities of Neural ODEs. In [4], Zhang et al. prove that for a homeomorphism $h : U \to U$ where $U \subseteq \mathbb{R}^p$, there exists a $\mathbf{0}_{2p}$-ODE-net approximating $h$. Moreover, by adding a linear layer after the ODE net, we have the following universal approximation theorem [4, Theorem 7]

**Theorem 1** *Let $F : \mathbb{R}^p \to \mathbb{R}^r$ be a neural network approximating $f : U \to \mathbb{R}^r$, where $U \subseteq \mathbb{R}^p$ is compact and each $f_i$, $i = 1, \ldots, r$ is integrable. Then there exists a $\mathbf{0}_{p+r}$-ODE-net followed by a linear layer that can approximate $F$.*

Although this theorem does not provide a recipe to build ODE nets, it provides the guarantee that they can approximate vector fields with universal approximation properties. This begs the question, can we extend this idea to more general vector fields? The answer is yes, as discussed below.

### 2.3 Augmentation With a Net

The augmentation outlined until now only involved padding the input vector with zeros. We can extend this idea and introduce a more general augmentation strategy. Massaroli et al. [5] propose a generalized version of 1 (see Section 3.1), which includes introducing an input map $\boldsymbol{h}_x : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$ and an output map $\boldsymbol{h}_y : \mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$. We call such nets IL-NODEs (for input layer) where $\boldsymbol{h}_x$ and $\boldsymbol{h}_y$ are learned nets. In practice, linear layers are used to preserve some structure in the data and to prevent excessive distortion of the learned vector field. In fact, one can even show a universal approximation theorem assuming only continuity of $f_\theta$ when using affine[2] $\boldsymbol{h}_x$ and $\boldsymbol{h}_y$, see [13, Theorem 2.13]. Therefore, augmentation strategies essentially alleviate NODEs of their expressivity limitations.

## 3 Alternatives to Augmentation

### 3.1 A General IVP

In their paper, Massaroli et al. [5] generalize the IVP 1 by modifying three components. First, they generalize the augmentation strategy in [3] by adding maps $\boldsymbol{h}_x, \boldsymbol{h}_y$. Second, they introduce a direct dependence of the parameters on depth. Finally, they add a dependence on the input in the dynamics, which was not present in [1]. These modifications lead to the generalized IVP

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{z}(t)}{\mathrm{d}t} = f_{\theta(t)}(t, \boldsymbol{x}, \boldsymbol{z}(t)) \\ \boldsymbol{z}(0) = \boldsymbol{h}_x(\boldsymbol{x}) \qquad\qquad t \in \mathcal{T} \\ \boldsymbol{y}(t) = \boldsymbol{h}_y(\boldsymbol{z}(t)) \end{cases} \tag{3}$$

We note to have a well-posed problem, we assume $f_{\theta(t)}$ is Lipschitz for all $t$. Dependence on the input (or Data Control) makes the neural ODE learn a collection of vector fields instead of only one. The idea is for the net to adapt the dynamics based on the position of the input in state space. In particular, this can make neural ODEs learn the reflection function in one dimension, which even augmented neural ODEs couldn't (without linear layers). This can be seen from the following proposition [5, Proposition 1]

**Proposition 1** *For every $\varepsilon > 0$, there exists $\theta > 0$ such that $|\phi(x) - z(1)| < \varepsilon$ where $z(1)$ is the solution of the IVP*

$$\begin{cases} \frac{\mathrm{d}z(t)}{\mathrm{d}t} = -\theta(z(t) + x) \qquad t \in [0, 1] \\ z(0) = x \end{cases}$$

The idea of conditioning the vector field on the input is particularly useful in situations where crossing trajectories is a known issue, such as continuous normalizing flows ([1, 6]). However, when training on large datasets, adding dependence on each data point might be inefficient as the number of parameters will increase significantly. Another generalization introduced in the IVP 3 is dependence on depth, which makes the formalism of Neural ODEs as the limit of ResNets rigorous. Indeed, in a ResNet, the parameters are theoretically different at each layer, and intuitively so should their limit. There is a subtle difference between $f_\theta$ in 1 and $f_{\theta(t)}$ in 3. While it is true that $f_\theta$ depends on time, the weights in the neural net don't because the time dependency is fed to the net by concatenating $\boldsymbol{y}(t)$ and $t$. The ODE is therefore rendered non-autonomous, but in a higher dimension ($\mathbb{R}^{n_x} + 1$). Meanwhile, the time dependence is directly incorporated in the weights of $f_{\theta(t)}$. But by doing so, the weights now live in an infinite-dimensional space, which is a problem for training. Let the choice of loss function for 3 be

$$\mathcal{L} \coloneqq L(\boldsymbol{z}(T)) + \int_{\mathcal{T}} l(\tau, \boldsymbol{z}(\tau))\, \mathrm{d}\tau \tag{4}$$

where $L$ is a terminal loss function and $l$ is a regularizing function. We then have [5, Theorem 1]

[2]To avoid confusion, we recall that a linear layer applies an affine transformation to the input and not a linear one.

**Theorem 2** *Given 4 and $\theta : \mathcal{T} \to \mathbb{R}^{n_\theta}$ square integrable, the functional derivative of the loss is*

$$\frac{\delta l}{\delta \theta(t)} = \boldsymbol{a}^\intercal \frac{\partial f_{\theta(t)}}{\partial \theta(t)} \text{ where } \boldsymbol{a}(t) \text{ satisfies } \begin{cases} \dot{\boldsymbol{a}}^\intercal(t) = -\boldsymbol{a}^\intercal \frac{\partial f_{\theta(t)}}{\partial \boldsymbol{z}} - \frac{\partial l}{\partial \boldsymbol{z}} \\ \boldsymbol{a}^\intercal(T) = \frac{\partial L}{\partial \boldsymbol{z}(T)} \end{cases} \tag{5}$$

Since this is a derivative in infinite dimensional space, we have to choose an approximation. Massaroli et al. [5] propose two ways of doing so. The first one is to approximate $\theta(t)$ by using a truncated projection on a basis of $\mathbb{L}_2(\mathcal{T} \to \mathbb{R}^{n_\theta})$ such as Chebychev polynomials or Fourier Series (spectral discretization). The second one is to assume $\theta(t)$ to be piecewise constant, equivalent to stacking neural nets. This type of architecture closes the gap between intuition and rigour regarding the connection between neural ODEs and ResNets; however, it remains unclear whether it provides practical benefits over augmentation.

### 3.2 Adaptive Depth

The main problem of NODEs highlighted in this paper is the theoretical lack of expressivity. On one hand, it is true that ODE solutions cannot cross. On the other, for each input, there is no reason, a priori, to find the solution at some predetermined time (uniform among the inputs). By having a different evaluation time for each input, the net can learn trajectories to correctly approximate functions without needing to cross trajectories. Formally, the output of the ODE net would be

$$\hat{\boldsymbol{y}} = \boldsymbol{h}_y \left( \boldsymbol{h}_x(\boldsymbol{x}) + \int_0^{\boldsymbol{g}_\omega(\boldsymbol{x})} f_{\theta(\tau)}(\tau, \boldsymbol{x}, \boldsymbol{z}(\tau)) \, \mathrm{d}\tau \right)$$

where $\boldsymbol{g}_\omega$ is a learned net with parameters $\omega$. Using this idea, Massaroli et al. [5] manage to learn the reflection map in one dimension, which was impossible for vanilla NODEs as well as Augmented NODEs.

## 4 Experimental Results

In this section, we test the methods presented to overcome the limitations of Neural ODEs on a classification toy dataset. We inspire ourselves from the "concentric spheres" problem, central in [3], and make it somewhat harder. We embed two different spheres in a larger sphere, thus having three non-linearly separable classes (that is, after applying any homeomorphism). Our code is largely adapted from [3] and [5]. We use the `TorchDyn` package [14] to implement Data Control, Adaptive Depth and Augmentation strategies as used in [5]. Moreover, we modify part of the code of [3] to build the data set and plot the results. The baseline model used is a net with a hidden layer of width 32 on a dataset of 1024 points in batches of 96. We train each neural net using `PyTorch` [15] with an Adam optimizer (learning rate of $10^{-3}$) for 5 epochs. The nets use a Dormand-Price method to solve ODEs (implemented in `TorchDyn`) and calculates gradients using 'autograd'. Setting the hidden layer's width to 32 for all experiments hinders the comparison as the number of trainable parameters might differ between models. Therefore, we choose the hidden layer's width so that the number of parameters is approximately equal between models as can be seen in Table 1. As an aside, we tested for many activation functions and found that ReLU was the favorite for all but the depth-variant NODEs which were trained with $\tanh$. We also tried calculating gradients using the adjoint method as in [1] and [3] but found a substantial slowdown in training. We include a ResNet in our results for comparison purposes. We find that although this is a simple dataset, it showcases the topics discussed in this paper. First, by looking at the decision boundaries in Figure 2, it is evident that vanilla NODEs and depth-dependent NODEs do not generalize. Moreover, we find

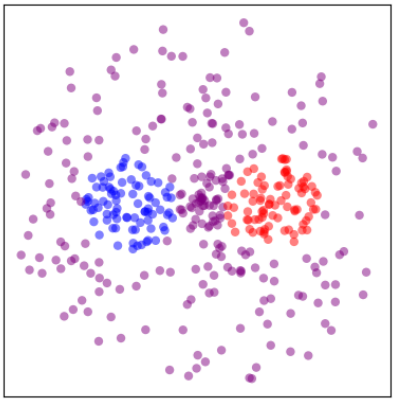

Figure 1: Three Spheres dataset

Table 1: Result summary for different nets.

| Net type | HL Width | # param. | Train. Acc. | Test Acc. | Time/epoch |
|---|---|---|---|---|---|
| ResNet | 33 | 1.3k | 98.7% | 99.0% | $\sim 10s$. |
| Vanilla NODE | 32 | 1.3k | 97.9% | 98.4% | $\sim 27s$. |
| ANODE | 32 | 1.3k | 98.8% | 99.3% | $\sim 21s$. |
| IL-NODE | 32 | 1.3k | 99.2% | 99.3% | $\sim 22s$. |
| Data Controlled | 32 | 1.3k | 99.2% | 99.3% | $\sim 20s$. |
| Chebychev | 15 | 1.3k | 97.8% | 97.4% | $\sim 42s$. |
| Fourier | 11 | 1.4k | 97.6% | 97.7% | $\sim 67s$. |
| Stacked | 14 | 1.4k | 97.9% | 98.0% | $\sim 80s$. |

their training time to be higher than that of ANODEs and IL-NODEs. This suggests that ANODES and IL-NODES resolve the expressivity issues of NODES with the additional benefit of being more efficient. Moreover, we conclude that even if depth variant NODEs provide good theoretical insight on NODEs as limits of ResNets, it can be tricky to implement them effectively. The way the tails align in a structured way in the Fourier NODE decision boundary implies that the method might require a certain class of data to work properly rather than in any scenario. This lack of generalization ability for vanilla NODEs and depth variant NODEs is apparent on the decision boundary and by extension in the training time because of the complicated flows learned. In contrast, IL-NODE prove to generalize well, which is to be expected given that they are extensions of ANODEs. Moreover, Data Controlled NODEs appear to offer an effective alternative to augmentation in terms of generalization and training efficiency. Indeed, they perform as well as IL-NODEs and ANODEs on the test set and take less time to train with a well-defined decision boundary. However, it is worth mentioning that ResNets are clearly faster at training than any NODE, with a slight disadvantage in test accuracy compared to ANODEs, IL-NODEs and Data Controlled NODEs.

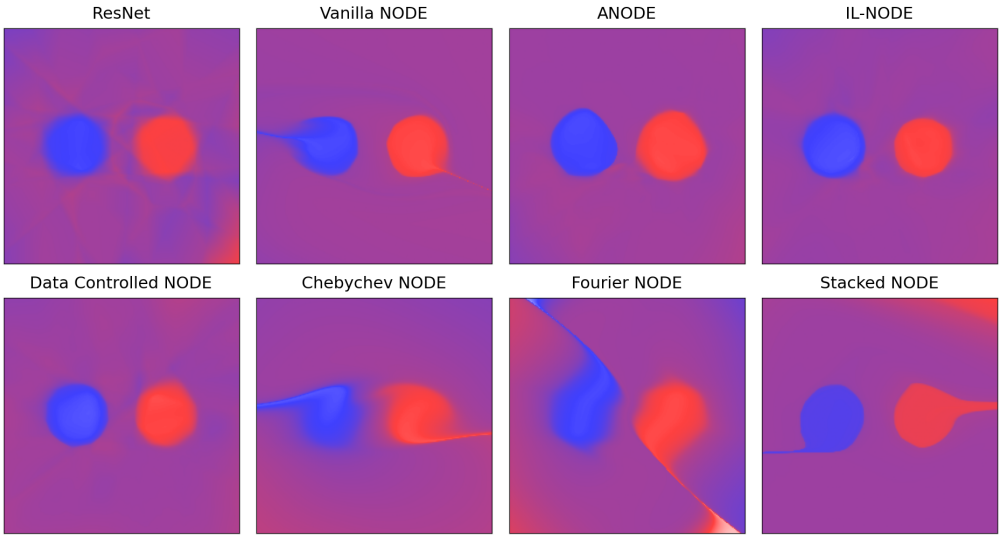


Figure 2: Decision boundaries for different ODE nets and ResNet on the Three Spheres dataset.

## 5 Conclusion

In this paper, we sought to address some initial concerns about the expressivity of neural ODEs. We discussed some theoretical results guaranteeing a universal approximation theorem via augmentation. On the other hand, we found that placing neural ODEs in a more general setting opened the door to new ways of sidestepping the topology preserving property of ODEs and of thinking of neural ODEs as deep limits of ResNets. Having established that neural ODEs clearly have great expressivity potential, it remains to be seen how we can implement them effectively to compete with more traditional nets.

## References


[1] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. 2018.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, 2015.

[3] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs, 2019.

[4] Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation Capabilities of Neural ODEs and Invertible Residual Networks. In *International Conference on Machine Learning*, 2020.

[5] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting Neural ODEs. 2020.

[6] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to Train Your Neural ODE: The World of Jacobian and Kinetic Regularization, 2020.

[7] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models, 2018.

[8] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural Controlled Differential Equations for Irregular Time Series, 2020.

[9] Patrick Kidger, James Foster, Xuechen Li, and Terry Lyons. Efficient and Accurate Gradients for Neural SDEs, 2021.

[10] Patrick Kidger, James Foster, Xuechen Li, Harald Oberhauser, and Terry Lyons. Neural SDEs as Infinite-Dimensional GANs, 2021.

[11] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian Neural Networks, 2020.

[12] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks, 2019.

[13] Patrick Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

[14] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. TorchDyn: A Neural Differential Equations Library, 2020.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.