Dissecting Neural ODEs

Stefano Massaroli[*]
The University of Tokyo
massaroli@robot.t.u-tokyo.ac.jp

Michael Poli[*]
KAIST
poli_m@kaist.ac.kr

Jinkyoo Park
KAIST
jinkyoo.park@kaist.ac.kr

Atsushi Yamashita
The University of Tokyo
yamashita@robot.t.u-tokyo.ac.jp

Hajime Asama
The University of Tokyo
asama@robot.t.u-tokyo.ac.jp

Abstract

Continuous deep learning architectures have recently re–emerged as variants of *Neural Ordinary Differential Equations* (Neural ODEs). The infinite–depth approach offered by these models theoretically bridges the gap between deep learning and dynamical systems; however, deciphering their inner working is still an open challenge and most of their applications are currently limited to the inclusion as generic *black–box* modules. In this work, we "open the box" and offer a system–theoretic perspective with the aim of clarifying the influence of several design choices on the underlying dynamics. We also introduce novel architectures: among them, a Galërkin–inspired depth–varying parameter model and Neural ODEs with data–controlled vector fields.

1 Introduction

Neural ODEs (Chen et al., 2018) represent the latest instance of continuous deep learning models, first developed in the context of continuous recurrent networks (Cohen and Grossberg, 1983). Since their introduction, research on Neural ODEs variants (Tzen and Raginsky, 2019; Jia and Benson, 2019; Zhang et al., 2019b; Yıldız et al., 2019; Poli et al., 2019) has progressed at a rapid pace. However, the search for concise explanations and experimental evaluations of novel architectures has left many fundamental questions unanswered.

In this work, we establish a general system–theoretic Neural ODE formulation (1) and dissect it into its core components; we analyze each of them separately, shining light on peculiar phenomena unique to the continuous deep learning paradigm. In particular, augmentation strategies are generalized beyond ANODEs (Dupont et al., 2019), and the novel concepts of *data–control* and *adaptive–depth* enriching (1) are showcased as effective approaches to learn maps such as *reflections* or *concentric annuli* without augmentation.

While explicit dependence on the depth–variable has been considered in the original formulation (Chen et al., 2018), a parameter depth–variance in continuous models has been overlooked. We provide a treatment in infinite–dimensional space required by the true *deep limit* of ResNets, the solution of which leads to a Neural ODE variant based on a spectral discretization.

Neural Ordinary Differential Equation

$$\begin{cases} \dot{\mathbf{z}} = f_{\theta(s)}(s, \mathbf{x}, \mathbf{z}(s)) \\ \mathbf{z}(0) = h_x(\mathbf{x}) \\ \hat{\mathbf{y}}(s) = h_y(\mathbf{z}(s)) \end{cases} \quad s \in \mathcal{S} \quad (1)$$

Input	$\mathbf{x}$	$\mathbb{R}^{n_x}$
Output	$\hat{\mathbf{y}}$	$\mathbb{R}^{n_y}$
(Hidden) State	$\mathbf{z}$	$\mathbb{R}^{n_z}$
Parameters	$\theta(s)$	$\mathbb{R}^{n_\theta}$
Neural Vector Field	$f_{\theta(s)}$	$\mathbb{R}^{n_z}$
Input Network	h_x	$\mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$
Output Network	h_y	$\mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$

---

[*]Equal contribution

● **Depth–variance**   Vanilla Neural ODEs (Chen et al., 2018) cannot be considered the deep limit of ResNets. We discuss the subtleties involved, uncovering a formal optimization problem in functional space as the price to pay for true depth–variance. Obtaining its solution leads to to two novel variants of Neural ODEs: a Galërkin–inspired spectral discretization (GalNODE) and a piecewise–constant model. GalNODEs are showcased on a task involving a loss distributed on the depth–domain, requiring the introduction of a *generalized* version of the adjoint in (Chen et al., 2018).

● **Augmentation strategies**   The augmentation idea of ANODEs (Dupont et al., 2019) is taken further and generalized to novel dynamical system–inspired and parameter efficient alternatives, relying on different choices of h_x in (1). These approaches, which include *input–layer* and *higher–order* augmentation, are verified to be more effective than existing methods in terms of performance and parameter efficiency.

● **Beyond augmentation: data–control and adaptive–depth**   We unveil that although important, augmentation is not always necessary in challenging tasks such as learning *reflections* or *concentric annuli* (Dupont et al., 2019). To start, we demonstrate that depth–varying vector fields alone are sufficient in dimensions greater than one. We then provide theoretical and empirical results motivating two novel Neural ODE paradigms: *adaptive–depth*, where the integration bound is itself determined by an auxiliary neural network, and *data–controlled*, where $f_{\theta(s)}$ is conditioned by the input data, allowing the ODE to learn a *family* of vector fields instead of a single one. Finally, we warn against input networks h_x of the multilayer, nonlinear type, as these architectures can lead to *superfluous* Neural ODE flows.

2   Continuous–Depth Models

A general formulation   In the context of Neural ODEs we suppose to be given a stream of input–output data $\{(\mathbf{x}_k, \mathbf{y}_k)\}_{k \in \mathcal{K}}$ (where $\mathcal{K}$ is a linearly–ordered finite subset of $\mathbb{N}$). The inference of Neural ODEs is carried out by solving the *inital value problem* (IVP) (1), i.e.

$$\hat{\mathbf{y}}(S) = h_y \left(h_x(\mathbf{x}_k) + \int_{\mathcal{S}} f_{\theta(\tau)}(\mathbf{x}_k, \tau, \mathbf{z}(\tau)) \mathrm{d}\tau \right)$$

Well–posedness   If $f_{\theta(s)}$ is Lipschitz, for each $\mathbf{x}_k$ the initial value problem in (1) admits a unique solution z defined in the whole $\mathcal{S}$. If this is the case, there is a mapping ϕ from $\mathbb{R}^{n_\theta} \times \mathbb{R}^{n_x}$ to the space of absolutely continuous functions $\mathcal{S} \mapsto \mathbb{R}^{n_z}$ such that $\mathbf{z}_k := \phi(\theta, \mathbf{x}_k)$ satisfies the ODE in (1). This in turn implies that, for all $k \in \mathcal{K}$, the map $(\theta, \mathbf{x}_k, s) \mapsto \gamma(\theta, \mathbf{x}_k, s) := h_y\big(\phi(\theta, \mathbf{x}_k)(s)\big)$ satisfies $\hat{\mathbf{y}} = \gamma(\theta, \mathbf{x}_k, s)$. For compactness, for any $s \in \mathcal{S}$, we denote $\phi(\theta, \mathbf{x}_k)(s)$ by $\phi_s(\theta, \mathbf{x}_k)$.

Training: optimal control   (Chen et al., 2018) treated the training of constant–parameters Neural ODE (i.e. $\theta \in \mathbb{R}^{n_\theta}$) considering only *terminal* loss functions depending on the terminal state $\mathbf{z}(S)$. However, in the framework of Neural ODEs, the latent state evolves through a continuum of layers steering the model output $\hat{\mathbf{y}}(s)$ towards the label. It thus makes sense to introduce a loss function also distributed on the whole depth domain $\mathcal{S}$. In this paper, we consider scalar loss functions

$$\ell := L(\mathbf{z}(S)) + \int_{\mathcal{S}} l(\tau, \mathbf{z}(\tau)) \mathrm{d}\tau \tag{2}$$

In this more general formulation the training can be cast into the *optimal control* (Pontryagin et al., 1962) problem

$$
\begin{aligned}
\min_{\theta} &\frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \ell_k \\
\text{subject to } &\dot{\mathbf{z}} = f_{\theta(s)}\left(s, \mathbf{x}_k, \mathbf{z}(s)\right) \quad s \in \mathcal{S} \\
&\mathbf{z}(0) = h_x(\mathbf{x}_k), \quad \hat{\mathbf{y}}(s) = h_y(\mathbf{z}(s))
\end{aligned}
\quad, \ \forall k \in \mathcal{K} \tag{3}
$$

solved by gradient descent. Here, if θ is constant, the gradients can be computed with $\mathcal{O}(1)$ memory efficiency by generalizing the adjoint sensitivity method in (Chen et al., 2018).

Theorem 1 (Generalized Adjoint Method). *Consider the loss function* (2). *Then,*

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \int_{\mathcal{S}} \mathbf{a}^\top(\tau) \frac{\partial f_\theta}{\partial \theta} \mathrm{d}\tau \ \text{ where } \mathbf{a}(s) \text{ satisfies } \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top \frac{\partial f_\theta}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \end{cases}$$

Appendix B contains additional insights on the choice of activation, training regularizers and approximation capabilities of Neural ODEs, along with a detailed derivation of the above result.

3 Depth-Variance: Infinite Dimensions for Infinite Layers

Bring residual networks to the deep limit   *Vanilla* Neural ODEs, as they appear in the original paper (Chen et al., 2018), cannot be considered the deep limit of ResNets. In fact, while each residual block is characterized by its own parameters vector θ_s, the authors consider model $\dot{\mathbf{z}} = f_\theta(s, \mathbf{z}(s))$ where the depth variable s enters in the dynamics *per se*[2] rather than in the map $s \mapsto \theta(s)$. The first attempt to pursue the true deep limit of ResNets is the *hypernetwork* approach of (Zhang et al., 2019b) where another neural network parametrizes the dynamics of $\theta(s)$.

However, this approach is not backed by any theoretical argument and it exhibits a considerable parameter inefficiency, as it generally scales polynomially in n_θ. We adopt a different approach, setting out to tackle the problem theoretically in the general formulation. Here, we uncover an optimization problem in functional space, solved by a direct application of the adjoint sensitivity method in infinite-dimensions. We then introduce two parameter efficient depth–variant Neural ODE architectures based on the solution of such problem: *Galërkin* Neural ODEs and *Stacked* Neural ODEs.

Gradient descent in functional space ●   When the model parameters are depth–varying, $\theta : \mathcal{S} \to \mathbb{R}^{n_\theta}$, the nonlinear optimization problem (3) should be in principle solved by iterating a gradient descent algorithm in a functional space (Smyrlis and Zisis, 2004), e.g. $\theta_{k+1}(s) = \theta_k(s) - \eta \delta \ell_k / \delta \theta(s)$ once the Gateaux derivative $\delta \ell_k / \delta \theta(s)$ is computed. Let $\mathbb{L}_2^\theta$ be the space of square–integrable functions $\mathcal{S} \to \mathbb{R}^{n_\theta}$. Hereafter, we show that if $\theta(s) \in \mathbb{L}_2^\theta$, then the loss sensitivity to $\theta(s)$ can be computed through the adjoint method.

Theorem 2 (Infinite–Dimensional Gradients). *Consider the loss function* (2) *and let* $\theta(s) \in \mathbb{L}_2$. *Then, sensitivity of ℓ with respect to $\theta(s)$ (i.e. directional derivative in functional space) is*

$$\frac{\delta \ell}{\delta \theta(s)} = \mathbf{a}^\top(s) \frac{\partial f_{\theta(s)}}{\partial \theta(s)} \ \ \text{where } \mathbf{a}(s) \text{ satisfies} \ \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s) \frac{\partial f_{\theta(s)}}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \end{cases}$$

Note that, although Th. 2 provides a constructive method to gradient the loss gradient in the *infinite–dimensional* setting, the implementation requires some numerical method to *discretize* the problem. We offer two solutions: a spectral discretization approach relying on reformulating the problem on some functional bases and a depth discretization approach.

Spectral discretization: *Galërkin Neural ODEs* ●   The idea is to expand $\theta(s)$ on a complete orthogonal basis of a predetermined subspace of $\mathbb{L}_2^\theta$ and truncate the series to the m-th term:

$$\theta(s) = \sum_{j=1}^m \alpha_j \odot \psi_j(s)$$

In this way, the problem is turned into final dimension and the training will aim to optimize the parameters $\alpha = (\alpha_1, \ldots, \alpha_m) \in \mathbb{R}^{mn_\theta}$ whose gradients can be computed as follows

Corollary 1 (Spectral Gradients). *Under the assumptions of Theorem 2, if $\theta(s) = \sum_{j=1}^m \alpha_j \odot \psi_j(s)$,*

$$\frac{\mathrm{d}\ell}{\mathrm{d}\alpha} = \int_\mathcal{S} \mathbf{a}^\top(\tau) \frac{\partial f_{\theta(s)}}{\partial \theta(s)} \psi(\tau) \mathrm{d}\tau, \quad \psi = (\psi_1, \ldots, \psi_m)$$

Depth discretization: *Stacked Neural ODEs* ●   An alternative approach to parametrize $\theta(s)$ is to assume it piecewise constant in $\mathcal{S}$, i.e. $\theta(s) = \theta_i \ \forall s \in [s_i, s_{i+1}]$ and $\mathcal{S} = \bigcup_{i=0}^{p-1} [s_i, s_{i+1}]$. It easy to see how evaluating this model is equivalent to *stacking* p Neural ODEs with constant parameters,

$$\mathbf{z}(S) = h_x(\mathbf{x}) + \sum_{i=0}^{p-1} \int_{s_1}^{s_{i+1}} f_{\theta_i}(\tau, \mathbf{x}, \mathbf{z}(\tau)) \mathrm{d}\tau$$

Here, the training is carried out optimizing the resulting pn_θ parameters using the following:

Corollary 2 (Stacked Gradients). *Under the assumptions of Theorem 2, if $\theta(s) = \theta_i \ \forall s \in [s_i, s_{i+1}]$,*

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta_i} = -\int_{s_{i+1}}^{s_i} \mathbf{a}^\top(\tau) \frac{\partial f_{\theta_i}}{\partial \theta_i} \mathrm{d}\tau \ \ \text{where } \mathbf{a}(s) \text{ satisfies} \ \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s) \frac{\partial f_{\theta_i}}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} & s \in [s_i, s_{i+1}] \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \end{cases}$$

---

[2]In practice, s is concatenated to $\mathbf{x}$ and fed to f_θ

The two approaches offer different perspectives on the problem of parametrizing the evolution of $\theta(s)$; while the spectral method imposes a stronger prior to the model class, based on the chosen bases (e.g. Fourier series, Chebyshev polynomials, etc.) the depth–discretization method allows for more freedom. Further details on proofs, derivation and implementation of the two models are given in Appendix B.
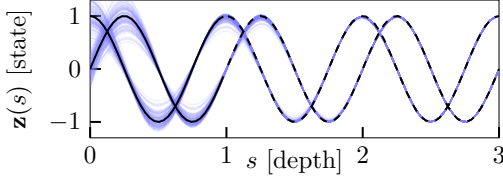
Periodic Tracking with Integral Loss

Figure 1: Galërkin Neural ODEs trained with integral losses accurately recover periodic signals.

Tracking signals via depth–variance Consider the problem of tracking a periodic signal $\beta(s)$. We show how this can be achieved without introducing additional inductive biases such as (Greydanus et al., 2019) through a synergistic combination of a two–layer Galërkin Neural ODEs and the generalized adjoint with integral loss $l(s) := \|\beta(s) - \mathbf{z}(s)\|_2^2$. The models, trained in $s \in [0, 1]$ generalize accurately in extrapolation, recovering the dynamics. Fig.2 showcases the depth–dynamics of $\theta(s)$ for Galërkin and Stacked variants trained to solve a simple binary classification problem. Additional insights and details are reported in Appendix.

4 Augmenting Neural ODEs

Augmented Neural ODEs (ANODEs) (Dupont et al., 2019) propose solving the *initial value problem* (IVP) in a higher dimensional space to limit the complexity of learned flows, i.e. having $n_z > n_x$. The proposed approach of the seminal paper relies on initializing to zero the $n_a := n_z - n_x$ augmented dimensions: $\mathbf{z}(0) = [x, \mathbb{0}]$. We will henceforth refer to this augmentation strategy as 0–*augmentation*. In this section we discuss alternative augmentation strategies for Neural ODEs that match or improve on 0–augmentation in terms of performance or parameter efficiency.

Input–layer augmentation ● Following the standard deep learning approach of increasing layer width to achieve improved model capacity, 0–augmentation can be generalized by introducing an input network $h_x : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$ to compute $\mathbf{z}(0)$:

$$\mathbf{z}(0) = h_x(\mathbf{x}) \tag{4}$$

leading to the general formulation of (1). This approach (4) gives the model more freedom in determining the initial condition for the IVP instead of constraining it to a concatenation of $\mathbf{z}(0)$ and $\mathbb{0}$, at a small parameter cost if h_x is, e.g., a linear layer. We refer to this type of augmentation as *input layer* (IL) augmentation and to the model as *IL–Neural ODE* (IL–NODE).
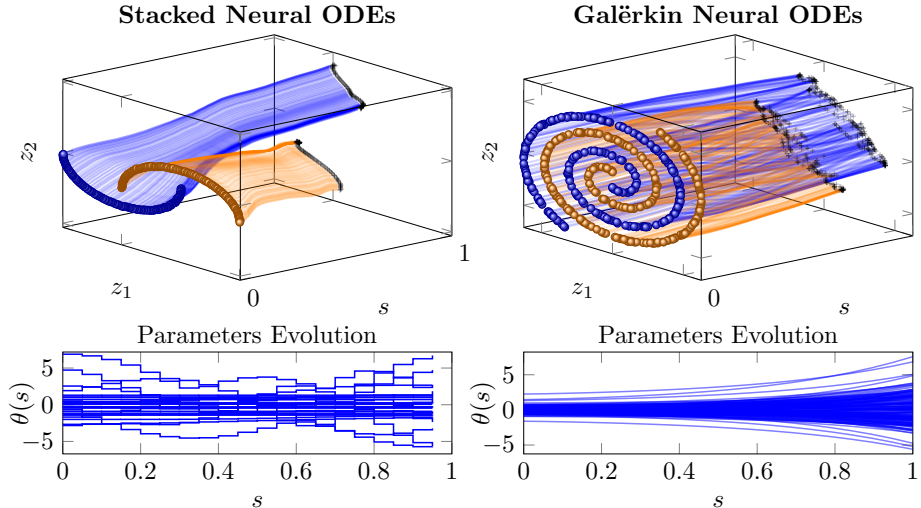

Figure 2: Galërkin and Stacked parameter-varying Neural ODE variants. Depth flows (Above) and evolution of the parameters (Below).

4

Note that 0-augmentation is compatible with the general IL formulation, as it corresponds to
$$h_x(\mathbf{x}) = [\mathbb{I}_{n_x} \quad \mathbb{O}_{n_x}] \mathbf{x}$$
In applications where maintaining the structure of the first n_x dimensions is important, e.g. approximation of dynamical systems, a parameter efficient alternative of (4) can be obtained by modifying the input network h_x to only affect the additional n_a dimensions, i.e. $h_x := [\mathbf{x}, \tilde{h}(\mathbf{x})]$

Higher–order Neural ODEs ● Further parameter efficiency can be achieved by lifting the Neural ODEs to higher orders. For example, let $\mathbf{z}(s) = [\mathbf{z}_q(s), \mathbf{z}_p(s)]$ a second–order Neural ODE of the form:

$$\ddot{\mathbf{z}}_q = f_{\theta(s)}(s, \mathbf{z}(s)). \tag{5}$$

equivalent to the first–order system

$$\begin{aligned}
\dot{\mathbf{z}}_q &= \mathbf{z}_p(s) \\
\dot{\mathbf{z}}_p &= f_{\theta(s)}(s, \mathbf{z}_q(s), \mathbf{z}_p(s))
\end{aligned} \tag{6}$$

The above can be extended to higher–order Neural ODEs as

$$\frac{\mathrm{d}^n \mathbf{z}^1}{\mathrm{d}s^n} = f_{\theta(s)}\left(s, \mathbf{z}, \frac{\mathrm{d}\mathbf{z}^1}{\mathrm{d}s}, \cdots, \frac{\mathrm{d}^{n-1}\mathbf{z}^1}{\mathrm{d}s^{n-1}}\right), \quad \mathbf{z} = [\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^n], \ \mathbf{z}^i \in \mathbb{R}^{n_z/n} \tag{7}$$

Note that the parameter efficiency of this method arises from the fact that $f_{\theta(s)} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_z/n}$ instead of $\mathbb{R}^{n_z} \to \mathbb{R}^{n_z}$. A limitation of system (6) is that a naive extension to second–order requires a number of augmented dimensions $n_a = n_x$. To allow for flexible augmentations of few dimensions $n_a < n_x$, the formulation of second–order Neural ODEs can be modified to *select* only a few dimensions to have higher order dynamics. We include formulation and additional details of *selective* higher–order augmentation in the supplementary material. Finally, higher–order augmentation can itself be compatible with input–layer augmentation.

Revisiting results for augmented Neural ODEs In higher dimensional state spaces, such as those of image classification settings, the benefits of augmentation become subtle and manifest as performance improvements and a lower *number of function evaluations* (NFEs) (Chen et al., 2018). We revisit the MNIST experiments of (Dupont et al., 2019) and evaluate four classes of depth–invariant Neural ODEs: namely, vanilla (no augmentation), ANODE (0–augmentation), IL-NODE (input–layer augmentation), and second–order (input–layer augmentation). For a fair comparison, the CNN $f_{\theta(s)}$ parametrizing the vector field for all augmented Neural ODEs take as input six–channel images: in the case of ANODEs this is achieved by an augmentation with $n_a = 5$, whereas for IL–NODEs and second order NODEs a linear convolutional layer is used to lift the channel dimension from one to six. All CNN architectures, aside from vanilla Neural ODEs, as well as hyperparameter setups, match that of (Dupont et al., 2019). The input network h_x is composed of a single, linear layer. The results for five experiments are reported in Table 4. IL–NODEs consistently achieve lower NFEs and higher performance than other variants, whereas second–order Neural ODEs offer best relative parameter efficiency.

It should be noted that prepending an input multi–layer neural network to the Neural ODE was the approach chosen in the experimental evaluations of the original Neural ODE paper (Chen et al., 2018) and that (Dupont et al., 2019) opted for a comparison between no input layer and 0–augmentation. However, a significant difference exists between architectures depending on the depth and expressivity of h_x. Indeed, utilizing non–linear and multi–layer input networks can be detrimental, as discussed in Sec. 5. Empirical evaluations on MNIST show that an augmentation through a single linear layer is sufficient to relieve vanilla Neural ODEs of their limitations and achieves the best performance.

	NODE	ANODE	IL-NODE	Second–Order
Test Acc.	96.5 ± 0.26	98.0 ± 0.12	98.2 ± 0.09	97.8 ± 0.38
NFE	56	45	33	54
Param.	$44,971$	$45,616$	$45,676$	$45,481$

Table 1: Test results across 5 seeded runs on MNIST (mean and standard deviation). We report the mean NFE at convergence.
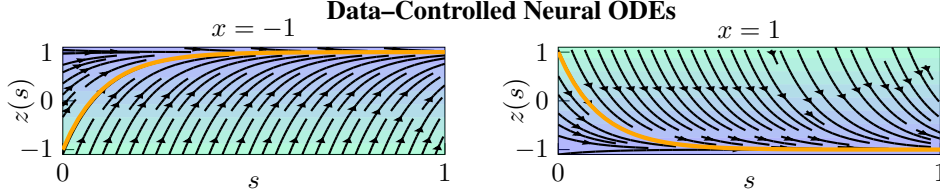
Figure 3: Depth trajectories over vector field of the *data–controlled* neural ODEs (9) for $x = 1$, $x = -1$. The model learns a family of vector fields conditioned by the input x to approximate $\phi(x)$.

5 Beyond Augmentation: Data–Control and Depth–Adaptation

Augmentation strategies are not always necessary for Neural ODEs to solve challenging tasks such as *concentric annuli* (Dupont et al., 2019). While it is indeed true that two distinct trajectories can never intersect in the state–space in the one–dimensional case, this does necessarily hold in general. In fact, dynamics in the first two spatial dimensions are substantially different e.g no chaotic behaviors are possible (Khalil and Grizzle, 2002). In the two–dimensions of $\mathbb{R}^2$ (and so in $\mathbb{R}^n$), infinitely wider than $\mathbb{R}$, distinct trajectories of a time–varying process can well intersect in the state–space, provided that they do not pass through the same point at the same time (Khalil and Grizzle, 2002). This implies that, in turn, depth–varying models such as Galërkin Neural ODEs can solve these tasks in all dimensions but $\mathbb{R}$.

The above described phenomenon is deeply linked to the *curse of dimensionality*, which refers to the exponential increase in complexity known to stem from a transition to higher–dimensional spaces (Friedman, 1997). In the case of Neural ODEs the curse turns into a blessing, as the flows have access to infinitely many additional directions for each increase in dimensions.

Starting from the one–dimensional case, we propose new classes of models allowing Neural ODEs to perform challenging tasks such as approximating *reflections* (Dupont et al., 2019) without the need of any augmentation.

5.1 Data–controlled Neural ODEs ⊙

We hereby derive a new class of models, namely *data–controlled Neural ODEs*.

To introduce the proposed approach, we start with an analytical result regarding the approximation of reflection maps such as $\varphi(x) = -x$. The proof provides a design recipe for a simple handcrafted ODE capable of approximating φ with arbitrary accuracy by leveraging input data x. We denote the conditioning of the vector field with x necessary to achieve the desired result as *data–control*.

This result highlights that, through data–control, Neural ODEs can approximate φ without augmentation, providing a novel perspective on existing results about expressivity limitations of continuous models (Dupont et al., 2019). The result is the following:

Theorem 3. *For all $\epsilon > 0$, $x \in \mathbb{R}$ there exists a parameter $\theta > 0$ such that*

$$|\varphi(x) - z(1)| < \epsilon, \tag{8}$$

where $h(1)$ is the solution of the Neural ODE

$$\begin{cases} \dot{z}(s) = -\theta(z(s) + x) \\ z(0) = x \end{cases}, \quad s \in [0, 1] \ . \tag{9}$$

The proof is reported in the Appendix. Fig. (3) shows a version of model (9) where θ is trained with standard backpropagation. This model is indeed able to closely approximate $\varphi(x)$ without augmentation, confirming the theoretical result. From this preliminary example, we then define the general *data–controlled* Neural ODE as

$$\dot{\mathbf{z}} = f_{\theta(s)}(s, \mathbf{x}, \mathbf{z}(s)), \qquad \mathbf{z}(0) = \mathbf{x}. \tag{10}$$

Model (10) incorporates input data $\mathbf{x}$ into the vector field, effectively allowing the ODE to learn a *family* of vector fields instead of a single one. Explicit dependence on $\mathbf{x}$ further constrains the ODE to be smooth with respect to the initial condition, acting as a regularizer. Indeed, in the experimental evaluation at the end of Sec. 5, data–controlled models recover an accurate decision boundary. Further experimental results with the latter general model on the representation of φ are reported in the Appendix.

6

Computationally, **x** can be passed to $f_{\theta(s)}$ in different ways, such as through concatenation in each call of the `ODE function`. In Appendix C we provide a PyTorch code snippet detailing the general implementation.

5.2 Adaptive–Depth Neural ODEs ◉

Let us come back to the approximation of $\varphi(x)$. Indeed, without incorporating input data into $f_{\theta(s)}$, it is not possible to realize a mapping $x \mapsto \phi_s(x)$ mimicking φ due to the topology preserving property of the flows. Nevertheless, a Neural ODE can be employed to approximate $\varphi(x)$ without the need of any *crossing trajectory*. In fact, if each input is integrated for in a different depth domain, $\mathcal{S}(x) = [0, s_x^*]$, it is possible to learn φ without crossing flows as shown in Fig. 4. In general, we can use a hypernetwork

Adaptive Integration Depth
Inputs trajectories through network depth

Figure 4: Depth trajectories over vector field of the *adaptive–depth* Neural ODEs. The reflection map can be learned by the proposed model. The key is to assign different integration times to the inputs, thus not requiring the intersection of trajectories.

g trained to learn the integration depth of each sample. In this setting, we define the general *adaptive depth* class as Neural ODEs performing the mapping $\mathbf{x} \mapsto \phi_{g_\omega(\mathbf{x})}(\mathbf{x})$, i.e.

$$\hat{\mathbf{y}} = \mathbf{z} + \int_0^{g_\omega(\mathbf{x})} f_{\theta(s)}(\tau, \mathbf{x}, \mathbf{z}(\tau)) d\tau, \tag{11}$$

where $g_\omega : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\omega} \to \mathbb{R}$ is a neural network with trainable parameters ω. Appendix B contains details on differentiation under the integral sign, required to backpropagate the loss gradients into ω.

Experiments of non–augmented models We inspect the performance of different Neural ODE variants: depth–invariant, depth–variant, Galërkin Neural ODEs and data–controlled. The concentric annuli (Dupont et al., 2019) dataset is utilized, and the models are qualitatively evaluated based on the complexity of the learned flows and on how accurately they extrapolate to unseen points, i.e. the learned decision boundaries. For Galërkin Neural ODEs, we choose a Fourier series with $m = 5$ harmonics as the eigenfunctions ψ_k, $k = 1, \ldots, 5$ to compute the parameters $\theta(s)$ of the GalNODE as described in Sec. 3. Figure 6 shows how data–controlled Neural ODEs accurately extrapolate outside of the training data with best performance in terms of NFEs, thus with the simplest learned flows.
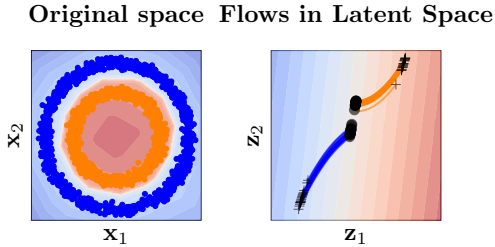
Original space Flows in Latent Space

Figure 5: Solving concentric annuli without augmentation by prepending a nonlinear transformation performed by a two–layer fully–connected network.

Mind your input networks An alternative approach to learning maps that prove to be challenging to approximate for vanilla Neural ODEs involves solving the ODE in a latent state space. Fig. 5 shows that with no augmentation, a network composed by a two fully–connected layers with non–linear activation followed by a Neural ODE can solve the concentric annuli problem. However, the flows learned by the Neural ODEs are superfluous: indeed, the clusters were already linearly separable after the first non–linear transformation. This example warns against superficial evaluations of Neural ODE architectures preceded or followed by several layers of non–linear input and output transformations. In these scenarios, the learned flows risk performing unnecessary transformations and in pathological cases can collapse into a simple identity map. To sidestep these issues, we propose visually inspecting trajectories or performing an ablation experiment on the Neural ODE block.
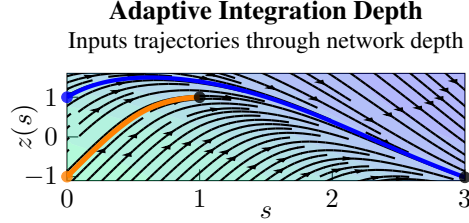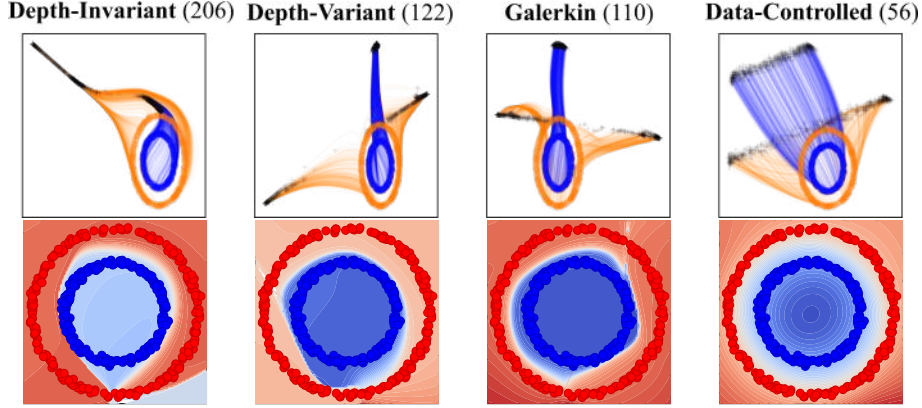
Figure 6: [Above] Depth-flows of the data in the state–space. [Below] Resulting decision boundaries.

6 Related Work

We include a brief history of classical approaches to continuous and dynamical system–inspired deep learning.

A brief historical note on continuous deep learning   Continuous neural networks have a long history that goes back to continuous time variants of recurrent networks (Cohen and Grossberg, 1983). Since then, several works explored the connection between dynamical systems, control theory and continuous recurrent networks (Zhang et al., 2014), providing stability analyses and introducing delays (Marcus and Westervelt, 1989). Many of these concepts have yet to resurface in the context of Neural ODEs. Haber and Ruthotto (2017) provides an analysis of ResNet dynamics and linked stability with robustness. Injecting stability into discrete neural networks has inspired the design of a series of architectures (Chang et al., 2019; Haber et al., 2019; Bai et al., 2019). Hauser et al. (2019) explored the algebraic structure of neural networks governed by finite difference equations, further linking discretizations of ODEs and ResNets in (Hauser et al., 2019) through the lens of geometry.

Approximating ODEs with neural networks has been discussed in (Wang and Lin, 1998; Filici, 2008). On the optimization front, several works leverage dynamical system formalism in continuous time (Wibisono et al., 2016; Maddison et al., 2018; Massaroli et al., 2019).

Neural ODEs   This work concerns Neural ODEs (Chen et al., 2018) and a system–theoretic discussion of their dynamical behavior. The main focus is on Neural ODEs and not the extensions to other classes of differential equations (Tzen and Raginsky, 2019; Jia and Benson, 2019), though the insights developed here can be broadly applied to continuous–depth models. More recently, Finlay et al. (2020) introduced regularization strategies to alleviate the heavy computational training overheads of Neural ODEs. These terms are propagated during the forward pass of the model, requiring state–augmentation techniques akin to those used in *continuous normalizing flows* (Chen et al., 2018) (CNFs). Leveraging our *generalized adjoint* formulation provides an method to approach integral regularization terms without augmentation and memory overheads.

7 Conclusion

In this work, we establish a general system–theoretic framework for Neural ODEs and dissect it into its core components. With the aim of shining light on fundamental questions regarding depth–variance, we formulate and solve the *infinite–dimensional* problem linked to the true *deep limit* formulation of Neural ODE. We provide numerical approximations to the infinite–dimensional problem, leading to novel model variants, such as Galërkin and piecewise–constant Neural ODEs. Augmentation is developed beyond existing approaches (Dupont et al., 2019) to include *input–layer* and *higher–order* augmentation strategies showcased to be more performant and parameter efficient. Finally, the novel paradigms of data–control and depth–adaptation are introduced to perform challenging tasks such as learning *reflections* without augmentation.

Broader Impact

As continuous deep learning sees increased utilization across fields such as healthcare (Rubanova et al., 2019; Yıldız et al., 2019), it is of utmost importance that we develop appropriate tools to further our understanding of neural differential equations. The search for robustness in traditional deep learning has only recently seen a surge in ideas and proposed solutions; this work aims at providing exploratory first steps necessary to extend the discussion to this novel paradigm. The *leitmotif* of this work is injecting system–theoretic concepts into the framework of continuous models. These ideas are of foundational importance in tangential fields such control and forecasting of dynamical systems, and are routinely used to develop robust algorithms with theoretical and practical guarantees.

References

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. 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 688–699. 2019.

B. Chang, M. Chen, E. Haber, and E. H. Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.

T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

M. A. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE transactions on systems, man, and cybernetics*, (5): 815–826, 1983.

E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pages 3134–3144, 2019.

C. Filici. On a neural approximator to odes. *IEEE transactions on neural networks*, 19(3):539–543, 2008.

C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman. How to train your neural ode. *arXiv preprint arXiv:2002.02798*, 2020.

J. H. Friedman. On bias, variance, 0/1?loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, 1(1):55–77, 1997.

S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15353–15363, 2019.

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

E. Haber, K. Lensink, E. Triester, and L. Ruthotto. Imexnet: A forward stable deep neural network. *arXiv preprint arXiv:1903.02639*, 2019.

Y. Hanshu, D. Jiawei, T. Vincent, and F. Jiashi. On robustness of neural ordinary differential equations. In *International Conference on Learning Representations*, 2019.

M. Hauser, S. Gunn, S. Saab Jr, and A. Ray. State-space representations of deep neural networks. *Neural computation*, 31(3):538–554, 2019.

J. Jia and A. R. Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 9843–9854, 2019.

H. K. Khalil and J. W. Grizzle. *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ, 2002.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Q. Li, T. Lin, and Z. Shen. Deep learning via dynamical systems: An approximation perspective. *arXiv preprint arXiv:1912.10382*, 2019.

C. J. Maddison, D. Paulin, Y. W. Teh, B. O'Donoghue, and A. Doucet. Hamiltonian descent methods. *arXiv preprint arXiv:1809.05042*, 2018.

C. Marcus and R. Westervelt. Stability of analog neural networks with delay. *Physical Review A*, 39 (1):347, 1989.

S. Massaroli, M. Poli, F. Califano, A. Faragasso, J. Park, A. Yamashita, and H. Asama. Port-hamiltonian approach to neural network training. *arXiv preprint arXiv:1909.02702*, 2019.

S. Massaroli, M. Poli, M. Bin, J. Park, A. Yamashita, and H. Asama. Stable neural flows. *arXiv preprint arXiv:2003.08063*, 2020.

M. Poli, S. Massaroli, J. Park, A. Yamashita, H. Asama, and J. Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

L. S. Pontryagin, E. Mishchenko, V. Boltyanskii, and R. Gamkrelidze. The mathematical theory of optimal processes. 1962.

P. J. Prince and J. R. Dormand. High order embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 7(1):67–75, 1981.

Y. Rubanova, T. Q. Chen, and D. K. Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5321–5331, 2019.

G. Smyrlis and V. Zisis. Local convergence of the steepest descent method in hilbert spaces. *Journal of mathematical analysis and applications*, 300(2):436–453, 2004.

B. Tzen and M. Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

Y.-J. Wang and C.-T. Lin. Runge-kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks*, 9(2):294–307, 1998.

A. Wibisono, A. C. Wilson, and M. I. Jordan. A variational perspective on accelerated methods in optimization. *proceedings of the National Academy of Sciences*, 113(47):E7351–E7358, 2016.

Ç. Yıldız, M. Heinonen, and H. Lähdesmäki. Ode [2] vae: Deep generative second order odes with bayesian neural networks. *arXiv preprint arXiv:1905.10994*, 2019.

H. Zhang, Z. Wang, and D. Liu. A comprehensive review of stability analysis of continuous-time recurrent neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 25(7): 1229–1262, 2014.

H. Zhang, X. Gao, J. Unterman, and T. Arodz. Approximation capabilities of neural ordinary differential equations. *arXiv preprint arXiv:1907.12998*, 2019a.

T. Zhang, Z. Yao, A. Gholami, K. Keutzer, J. Gonzalez, G. Biros, and M. Mahoney. Anodev2: A coupled neural ode evolution framework. *arXiv preprint arXiv:1906.04596*, 2019b.

H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li. Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4. IEEE, 2015.

Dissecting Neural ODEs
Supplementary Material

Table of Contents

A Proofs and Additional Theoretical Results

A.1 Proof of Theorem 1

Theorem 1 (Generalized Adjoint Method). *Consider the loss function* (2). *Then,*

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \int_{\mathcal{S}} \mathbf{a}^\top(\tau)\frac{\partial f_\theta}{\partial \theta}\mathrm{d}\tau \ \ \text{where } \mathbf{a}(s) \text{ satisfies } \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top\frac{\partial f_\theta}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \end{cases}$$

Proof. The proof follows from Massaroli et al. (2020). Let us define a *Lagrange multiplier* or *adjoint state* $\mathbf{a}$, dual to $\mathbf{z}$. As $\mathbb{R}^{n_z}$ is self–adjoint, $\mathbf{a} \in \mathbb{R}^{n_z}$. Moreover, let $\mathcal{L}$ be a perturbed loss function of the form

$$\mathcal{L} := \ell - \int_0^S \mathbf{a}^\top(\tau)\left[\dot{\mathbf{z}}(\tau) - f_\theta(s, \mathbf{x}_t, \mathbf{z}(\tau))\right]\mathrm{d}\tau$$

Since $\dot{\mathbf{z}} - f_\theta(s, \mathbf{x}, \mathbf{z}) = 0$ by construction, the integral term in $\mathcal{L}$ is always null and, thus, $\mathbf{a}(s)$ can be freely assigned while $\mathrm{d}\mathcal{L}/\mathrm{d}\theta = \mathrm{d}\ell/\mathrm{d}\theta$. For the sake of compactness we do not explicitly write the dependence on variables of the considered functions unless strictly necessary. Note that,

$$\int_0^S \mathbf{a}^\top\dot{\mathbf{z}}\mathrm{d}\tau = \mathbf{a}^\top(\tau)\mathbf{z}(\tau)\big|_0^S - \int_0^S \dot{\mathbf{a}}^\top\mathbf{z}\mathrm{d}\tau$$

obtained via integration by parts. Hence,

$$\mathcal{L} = \ell - \mathbf{a}^\top(\tau)\mathbf{z}(\tau)\big|_0^S + \int_0^S \left(\dot{\mathbf{a}}^\top \mathbf{z} + \mathbf{a}^\top f_\theta\right)\mathrm{d}\tau$$

$$= L(\mathbf{z}(S)) - \mathbf{a}^\top(\tau)\mathbf{z}(\tau)\big|_0^S + \int_0^S \left(\dot{\mathbf{a}}^\top \mathbf{z} + \mathbf{a}^\top f_\theta + l\right)\mathrm{d}\tau \tag{12}$$

We can compute the gradient of ℓ with respect to θ as

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\theta} = \frac{\partial L(\mathbf{z}(S))}{\partial \mathbf{z}(S)}\frac{\mathrm{d}\mathbf{z}(S)}{\mathrm{d}\theta} - \mathbf{a}^\top(S)\frac{\mathrm{d}\mathbf{z}(S)}{\mathrm{d}\theta} - \mathbf{a}^\top(0)\frac{\mathrm{d}\mathbf{z}(\cancel{0})}{\cancel{\mathrm{d}\theta}}$$

$$+ \int_0^S \left[\dot{\mathbf{a}}^\top\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\theta} + \mathbf{a}^\top\left(\frac{\partial f_\theta}{\partial \theta} + \frac{\partial f_\theta}{\partial \mathbf{z}}\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\theta} + \frac{\partial f_\theta}{\partial \mathbf{x}}\frac{\mathrm{d}\cancel{\mathbf{x}}}{\cancel{\mathrm{d}\theta}} + \frac{\partial f_\theta}{\partial \tau}\frac{\mathrm{d}\cancel{\tau}}{\cancel{\mathrm{d}\theta}}\right) + \frac{\partial l}{\partial \mathbf{z}}\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\theta} + \frac{\partial l}{\partial \tau}\frac{\mathrm{d}\cancel{\tau}}{\cancel{\mathrm{d}\theta}}\right]\mathrm{d}\tau$$

which, by reorganizing the terms, yields to

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \left[\frac{\partial L}{\partial \mathbf{z}(S)} - \mathbf{a}^\top(S)\right]\frac{\mathrm{d}\mathbf{z}(S)}{\mathrm{d}\theta} +$$

$$+ \int_0^S \left(\dot{\mathbf{a}}^\top + \mathbf{a}^\top\frac{\partial f_\theta}{\partial \mathbf{z}} + \frac{\partial l}{\partial \mathbf{z}}\right)\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\theta}\mathrm{d}\tau \tag{13}$$

$$+ \int_0^S \mathbf{a}^\top\frac{\partial f_\theta}{\partial \theta}\mathrm{d}\tau$$

Now, if $\mathbf{a}(s)$ satisfies the *final* value problem

$$\dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s)\frac{\partial f_\theta}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}}, \quad \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \tag{14}$$

to be solved backward in $[0, S]$; then (13) reduces to

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \int_0^S \mathbf{a}^\top\frac{\partial f_\theta}{\partial \theta}\mathrm{d}\tau \tag{15}$$

proving the result. $\qquad\square$

Remark 1 (Implementation of the generalized adjoint method). *Note that, similarly to Chen et al. (2018), the gradient (15) is practically computed by defining the parameters adjoint state $\mathbf{a}_\theta$ and solving backward the system of ODEs*

$$\dot{\mathbf{a}}^\top = -\mathbf{a}^\top\frac{\partial f_\theta}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}}, \quad \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)}$$

$$\dot{\mathbf{a}}_\theta^\top = -\mathbf{a}^\top\frac{\partial f_\theta}{\partial \theta}, \qquad \mathbf{a}_\theta(S) = \mathbb{0}_{n_\theta} \tag{16}$$

Then,

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \mathbf{a}_\theta(0).$$

A.2 Proof of Theorem 2

Theorem 2 (Infinite–Dimensional Gradients). *Consider the loss function (2) and let $\theta(s) \in \mathbb{L}_2$. Then, sensitivity of ℓ with respect to $\theta(s)$ (i.e. directional derivative in functional space) is*

$$\frac{\delta\ell}{\delta\theta(s)} = \mathbf{a}^\top(s)\frac{\partial f_{\theta(s)}}{\partial\theta(s)} \ \text{ where } \mathbf{a}(s) \text{ satisfies } \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s)\frac{\partial f_{\theta(s)}}{\partial\mathbf{z}} - \frac{\partial l}{\partial\mathbf{z}} \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial\mathbf{z}(S)} \end{cases}$$

Proof. The proof follows the same steps of the one of Theorem 1 up to (12). However, here $\theta(s) \in \mathbb{L}_2$ and the loss sensitivity to $\theta(s)$ corresponds to the directional (Gateaux) derivative $\delta\ell/\delta\theta(s)$ in $\mathbb{L}_2$ derived as follows. We start by computing the total variation of ℓ:

$$\delta\ell = \frac{\partial L}{\partial\mathbf{z}(S)}\delta\mathbf{z}(S) - \mathbf{a}^\top(s)(\delta\mathbf{z}(S) - \delta\mathbf{z}(0))$$

$$+ \int_0^S \left[\dot{\mathbf{a}}^\top(\tau)\delta\mathbf{z}(\tau) + \mathbf{a}^\top(\tau)\left(\frac{\partial f_{\theta(\tau)}}{\partial\mathbf{z}(\tau)}\delta\mathbf{z}(\tau) + \frac{\partial f_{\theta(\tau)}}{\partial\theta(\tau)}\delta\theta(\tau)\right) + \frac{\partial l}{\partial\mathbf{z}(\tau)}\delta\mathbf{z}(\tau)\right]\mathrm{d}\tau$$

Thus,

$$\frac{\delta\ell}{\delta\theta(s)} = \left[\frac{\partial L}{\partial\mathbf{z}(S)} - \mathbf{a}^\top(s)\right]\frac{\delta\mathbf{z}(S)}{\delta\theta(s)} + \frac{\delta\mathbf{z}(0)}{\delta\theta(s)}$$
$$+ \int_0^S \left[\dot{\mathbf{a}}^\top(\tau)\frac{\delta\mathbf{z}(\tau)}{\delta\theta(s)} + \mathbf{a}^\top(\tau)\left(\frac{\partial f_{\theta(\tau)}}{\partial\mathbf{z}(\tau)}\frac{\delta\mathbf{z}(\tau)}{\delta\theta(s)} + \frac{\partial f_{\theta(\tau)}}{\partial\theta(\tau)}\frac{\delta\theta(\tau)}{\delta\theta(s)}\right) + \frac{\partial l}{\partial\mathbf{z}(\tau)}\frac{\delta\mathbf{z}(\tau)}{\delta\theta(s)}\right]\mathrm{d}\tau$$

Since it must hold

$$\int \frac{\delta\theta(\tau)}{\delta\theta(s)}\mathrm{d}\tau = 1,$$

then, model class choice $\theta(s) \in \mathbb{L}_2$ implies

$$\frac{\delta\theta(\tau)}{\delta\theta(s)} = \delta(\tau - s)$$

where $\delta(\tau - s)$ is the Dirac's delta. Therefore, it holds

$$\frac{\delta\ell}{\delta\theta(s)} = \left[\frac{\partial L}{\partial\mathbf{z}(S)} - \mathbf{a}^\top(s)\right]\frac{\delta\mathbf{z}(S)}{\delta\theta(s)} + \frac{\delta\mathbf{z}(0)}{\delta\theta(s)}$$
$$+ \int_0^S \left[\dot{\mathbf{a}}^\top(\tau)\frac{\delta\mathbf{z}(\tau)}{\delta\theta(s)} + \mathbf{a}^\top(\tau)\left(\frac{\partial f_{\theta(\tau)}}{\partial\mathbf{z}(\tau)}\frac{\delta\mathbf{z}(\tau)}{\delta\theta(s)} + \frac{\partial f_{\theta(\tau)}}{\partial\theta(\tau)}\delta(\tau - s)\right) + \frac{\partial l}{\partial\mathbf{z}(\tau)}\frac{\delta\mathbf{z}(\tau)}{\delta\theta(s)}\right]\mathrm{d}\tau$$

and, finally

$$\frac{\delta\ell}{\delta\theta(s)} = \left[\frac{\partial L}{\partial\mathbf{z}(S)} - \mathbf{a}^\top(s)\right]\frac{\delta\mathbf{z}(S)}{\delta\theta(s)} + \frac{\delta\mathbf{z}(0)}{\delta\theta(s)}$$
$$+ \int_0^S \left(\dot{\mathbf{a}}^\top(\tau) + \mathbf{a}^\top(\tau)\frac{\partial f_{\theta(\tau)}}{\partial\mathbf{z}(\tau)} + \frac{\partial l}{\partial\mathbf{z}(\tau)}\right)\frac{\delta\mathbf{z}(\tau)}{\delta\theta(s)}\mathrm{d}\tau$$
$$+ \mathbf{a}^\top(s)\frac{\partial f_{\theta(s)}}{\partial\theta(s)}$$

Hence, if for any $s \in \mathcal{S}$ the adjoint state $\mathbf{a}(s)$ satisfies

$$\dot{\mathbf{a}}^\top = -\mathbf{a}^\top\frac{\partial f_{\theta(s)}}{\partial\mathbf{z}} - \frac{\partial l}{\partial\mathbf{z}}, \quad \mathbf{a}^\top(S) = \frac{\partial L}{\partial\mathbf{z}(S)}$$

we have

$$\frac{\delta\ell}{\delta\theta(s)} = \mathbf{a}^\top(s)\frac{\partial f_{\theta(s)}}{\partial\theta(s)}$$

$\square$

A.3 Proof of Corollary 1

Corollary 1 (Spectral Gradients). *Under the assumptions of Theorem 2, if $\theta(s) = \sum_{j=1}^m \alpha_j \odot \psi_j(s)$,*

$$\frac{\mathrm{d}\ell}{\mathrm{d}\alpha} = \int_\mathcal{S} \mathbf{a}^\top(\tau)\frac{\partial f_{\theta(s)}}{\partial\theta(s)}\psi(\tau)\mathrm{d}\tau, \quad \psi = (\psi_1, \ldots, \psi_m)$$

Proof. The proof follows naturally from Theorem 2 by noticing that if $\theta(s)$ has some parametrization $\theta = \theta(s, \mu)$ with parameters $\mu \in \mathbb{R}^{n_\mu}$, then,

$$\frac{\mathrm{d}\ell}{\mathrm{d}\mu} = \int_0^S \mathbf{a}^\top(\tau)\frac{\partial f_\theta}{\partial\theta}\frac{\partial\theta}{\partial\mu}\mathrm{d}\tau \qquad (17)$$

Therefore, if

$$\theta(s) = \sum_{j=1}^m \alpha_j \odot \psi_j(s),$$

the loss gradient with respect to the parameters $\alpha := (\alpha_1, \ldots, \alpha_m) \in \mathbb{R}^{mn_\theta}$ is computed as

$$\frac{\mathrm{d}\ell}{\mathrm{d}\alpha} = \int_0^S \mathbf{a}^\top(\tau)\frac{\partial f_{\theta(\tau)}}{\partial\theta(\tau)}\frac{\partial\theta(s)}{\partial\alpha}\mathrm{d}\tau$$
$$= \int_0^S \mathbf{a}^\top(\tau)\frac{\partial f_{\theta(\tau)}}{\partial\theta(\tau)}\psi\mathrm{d}\tau$$

being $\psi := (\psi_1, \ldots, \psi_m)$. $\square$

Remark 2 (Choose your parametrization). *A further insight from this result, which paves the way to future developments, is that we can easily compute the loss gradients with respect to any parametrization of $\theta(s)$ through* (17)

A.4 Proof of Corollary 2

Corollary 2 (Stacked Gradients). *Under the assumptions of Theorem 2, if $\theta(s) = \theta_i \ \forall s \in [s_i, s_{i+1}]$,*

$$\frac{d\ell}{d\theta_i} = -\int_{s_{i+1}}^{s_i} \mathbf{a}^\top(\tau)\frac{\partial f_{\theta_i}}{\partial \theta_i}d\tau \ \text{ where } \mathbf{a}(s) \text{ satisfies } \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s)\frac{\partial f_{\theta_i}}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} & s \in [s_i, s_{i+1}] \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \end{cases}$$

Proof. The proof follows from the one of Theorems 1 and 2 by recalling the solution of the stacked neural ODEs:

$$\mathbf{z}(S) = h_x(\mathbf{x}) + \sum_{i=0}^{p-1}\int_{s_1}^{s_{i+1}} f_{\theta_i}(\tau, \mathbf{x}, \mathbf{z}(\tau))d\tau$$

We can recover a relation similar to (13)

$$\frac{d\ell}{d\theta_i} = \left[\frac{\partial L}{\partial \mathbf{z}(S)} - \mathbf{a}^\top(S)\right]\frac{d\mathbf{z}(S)}{d\theta_i} +$$

$$+ \sum_{j=0}^{p-1}\int_{s_j}^{s_{j+1}}\left(\dot{\mathbf{a}}^\top + \mathbf{a}^\top\frac{\partial f_{\theta_j}}{\partial \mathbf{z}} + \frac{\partial l}{\partial \mathbf{z}}\right)\frac{d\mathbf{z}}{d\theta_i}d\tau$$

$$+ \sum_{j=0}^{p-1}\int_{s_j}^{s_{j+1}}\mathbf{a}^\top\frac{\partial f_{\theta_j}}{\partial \theta_i}d\tau$$

Since

$$\forall j = 0, \ldots, p-1 \quad \frac{\partial f_{\theta_j}}{\partial \theta_i} \neq \mathbb{0} \Leftrightarrow j = i,$$

we have

$$\sum_{j=0}^{p-1}\int_{s_j}^{s_{j+1}}\mathbf{a}^\top\frac{\partial f_{\theta_j}}{\partial \theta_i}d\tau = \int_{s_i}^{s_{i+1}}\mathbf{a}^\top\frac{\partial f_{\theta_i}}{\partial \theta_i}d\tau = -\int_{s_{i+1}}^{s_i}\mathbf{a}^\top\frac{\partial f_{\theta_i}}{\partial \theta_i}d\tau$$

which leads to the result by assuming $\mathbf{a}(\tau)$ to satisfy

$$\dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s)\frac{\partial f_{\theta_i}}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} \quad s \in [s_i, s_{i+1}]$$

$$\mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)}$$

$\square$

A.5 Proof of Theorem 3

Theorem 3. *For all $\epsilon > 0$, $x \in \mathbb{R}$ there exists a parameter $\theta > 0$ such that*

$$|\varphi(x) - z(1)| < \epsilon, \tag{8}$$

where $h(1)$ is the solution of the Neural ODE

$$\begin{cases} \dot{z}(s) = -\theta(z(s) + x) \\ z(0) = x \end{cases}, \ s \in [0, 1] \ . \tag{9}$$

Proof. The general solution of (9) is

$$z(s) = x(2e^{-\theta s} - 1)$$

Thus,

$$e = \varphi(x) - z(1) = x + x(2e^{-\theta} - 1) = 2xe^{-\theta}$$

$$\Leftrightarrow |e| = 2|x|e^{-\theta}$$

It follows that

$$2|x|e^{-\theta} < \epsilon$$
$$\Leftrightarrow e^{-\theta} < \frac{\epsilon}{2|x|}$$
$$\Leftrightarrow \theta > -\ln\left(\frac{\epsilon}{2|x|}\right)$$

$\square$

A.6 Additional Theoretical Results

A.6.1 Explicit Parameter Dependence of the Loss

Note that, in both the seminal paper from Chen et al. (2018) and Theorem 1 the loss function was consider without explicit dependence on the parameters. However, in practical applications (see, e.g. (Finlay et al., 2020)) the loss has this explicit dependence:

$$\ell = L(\mathbf{z}(S), \theta) + \int_{\mathcal{S}} l(s, \mathbf{z}(\tau), \theta) d\tau, \tag{18}$$

In this case we need to modify the adjoint gradeints accordingly

Theorem 4 (Generalized Adjoint Method with Parameter–Dependent Loss). *Consider the loss function* (18). *Then,*

$$\frac{d\ell}{d\theta} = \frac{\partial L}{\partial \theta} + \int_{\mathcal{S}} \left(\mathbf{a}^\top(\tau) \frac{\partial f_\theta}{\partial \theta} + \frac{\partial l}{\partial \theta} \right) d\tau$$

where $\mathbf{a}(s)$ *satifies* (14).

Proof. The proof follows immediately from Theorem 1 by noticing that, with the explicit dependence on θ of ℓ, (13) would become

$$\frac{d\ell}{d\theta} = \frac{\partial L}{\partial \theta}$$
$$+ \left[\frac{\partial L}{\partial \mathbf{z}(S)} - \mathbf{a}^\top(S) \right] \frac{d\mathbf{z}(S)}{d\theta} +$$
$$+ \int_0^S \left(\dot{\mathbf{a}}^\top + \mathbf{a}^\top \frac{\partial f_\theta}{\partial \mathbf{z}} + \frac{\partial l}{\partial \mathbf{z}} \right) \frac{d\mathbf{z}}{d\theta} d\tau$$
$$+ \int_0^S \left(\mathbf{a}^\top \frac{\partial f_\theta}{\partial \theta} + \frac{\partial l}{\partial \theta} \right) d\tau$$

leading to the result. $\square$

In the depth–variant case where we might consider a loss function of type

$$\ell = L(\mathbf{z}(S), \theta(S)) + \int_{\mathcal{S}} l(\mathbf{z}(\tau), \theta(\tau)) d\tau \tag{19}$$

a similar result can be obtained for the infinite–dimensional adjoint.

A.6.2 Integration Bound Gradients

It is also possible to obtain the loss gradient with respect to the integration bound S.

Theorem 5 (Integration Bound Gradient). *Consider a loss function* 2. *Then,*

$$\frac{d\ell}{dS} = \frac{\partial L}{\partial \mathbf{z}(S)} f_{\theta(S)}(S, \mathbf{x}, \mathbf{z}(S)) + l(\mathbf{z}(S))$$

15

Proof.

$$\frac{\mathrm{d}\ell}{\mathrm{d}S} = \frac{\partial L}{\partial \mathbf{z}(S)} \frac{\mathrm{d}\mathbf{z}(S)}{\mathrm{d}S} + \frac{\mathrm{d}}{\mathrm{d}S} \int_0^S l(\mathbf{z}(\tau)) \mathrm{d}\tau$$

$$= \frac{\partial L}{\partial \mathbf{z}(S)} \frac{\mathrm{d}}{\mathrm{d}S} \left(h_x(\mathbf{x}) + \int_0^S f_{\theta(\tau)}(\tau, \mathbf{x}, \mathbf{z}(\tau)) \right) + \frac{\mathrm{d}}{\mathrm{d}S} \int_0^S l(\mathbf{z}(\tau)) \mathrm{d}\tau$$

Therefore, by applying the Leibniz integral rule we obtain

$$\frac{\mathrm{d}\ell}{\mathrm{d}S} = \frac{\partial L}{\partial \mathbf{z}(S)} f_{\theta(S)}(S, \mathbf{x}, \mathbf{z}(S)) + l(\mathbf{z}(S))$$

$\square$

B Practical Insights for Neural ODEs

B.1 Augmentation

Augmenting convolution and graph based architectures In the case of *convolutional neural network* (CNN) or *graph neural network* (GNN) architectures, augmentation can be performed along different dimensions i.e. *channel*, heigth, width or similarly node features or number of nodes. The most physically consistent approach, employed in (Dupont et al., 2019) for CNNs, is augmenting along the *channel* dimension, equivalent to providing each pixel in the image additional states. By viewing an image as a lattice graph, the generalization to GNN–based Neural ODEs (Poli et al., 2019) operating on arbitrary graphs can be achieved by augmenting each node feature with n_a additional states. While augmentation along the channel or node feature dimension has been the standard in the modern literature, it is unlikely to be the only high–performance option, leaving space for future work exploring augmentation in width, height or number of nodes.

Selective higher–order A limitation of system (6) is that a naive extension to second–order requires a number of augmented dimensions $n_a = n_z/2$. To allow for flexible augmentations of few dimensions $n_a < n_z/2$, the formulation of second–order Neural ODEs can be modified as follows. Let $\mathbf{z} := (\mathbf{z}_q, \mathbf{z}_p, \bar{\mathbf{z}}), \mathbf{z}_q, \mathbf{z}_p \in \mathbb{R}^{n_a}, \bar{\mathbf{z}} \in \mathbb{R}^{n_z - n_a}$. We can decide to give second order dynamics only to the first n_a states while the dynamics of other $n_z - n_a$ states is free. Therefore, while in This approach yields

$$\begin{bmatrix} \dot{\mathbf{z}}_q \\ \dot{\mathbf{z}}_p \\ \dot{\bar{\mathbf{z}}} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_p \\ f^p_{\theta(s)}(s, \mathbf{z}) \\ \bar{f}_{\theta(s)}(s, \mathbf{z}) \end{bmatrix} \quad , \tag{20}$$

A similar argument could be applied to orders higher than two. Note that a simple choice of n_z would be $n_z = n_x + n_a$. *Selective higher–order* Neural ODEs are compatible with input layer augmentation.

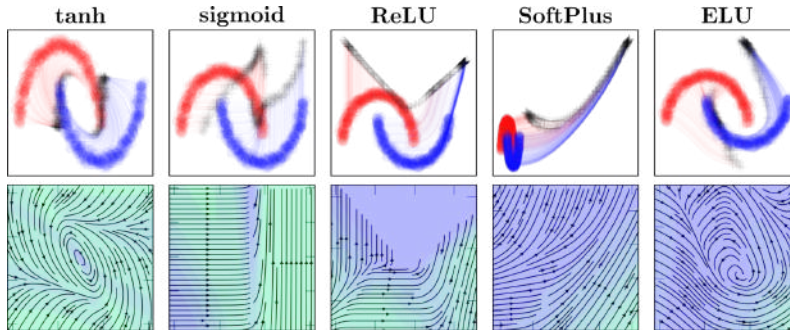

Figure 7: Depth trajectories of the hidden state and relative vector fields $f_\theta(\mathbf{z})$ for different activation functions in a nonlinear classification task. It can be noticed how the models with tanh and ELU outperform the others, as f_θ is able to steer $\mathbf{z}$ along negative directions.

B.2 Activations

Mind your activation   When using a Neural ODE within structured architectures we have to be especially careful with the choice of activation function in the last layer of f_θ. In fact, the chosen nonlinearity will strongly affect the "shape" of the vector field and, as a consequence, the flows learnable by the model. Therefore, while designing f_θ as a multi–layer neural network, it is generally advisable to append a linear layer to maximize the expressiveness of the underlying vector field. In some applications, conditioning the vector field (and thus the flows) with a specific nonlinearities can be desirable, e.g., when there exist priors on the desired transformation.

Effects of activations   In order to compare the effect of different activation functions in the last layer of f_θ, we set up a nonlinear classification task with the half–moons dataset. For the sake of completeness, we selected activations of different types, i.e., The dataset is comprised of 2^{13} data

Activation	Type
hyperbolic tangent (**tanh**)	bounded
sigmoid	bounded, non–negative output
ReLU	unbounded, non–negative output
SoftPlus	unbounded, non–negative output
ELU	lower–bounded

points. We utilize the entire dataset for training and evaluation since the experiment has the aim of delivering a qualitative description of the learned vector fields. f_θ has been selected as a multilayer perceptron with two hidden layers of 16 neurons each. The training has been carried out using Adam (Kingma and Ba, 2014) optimizer with learning rate 10^{-3} and weight decay set to 10^{-4}.

In Figure 7 we see how different activation functions in the last layer of f_θ condition the vector fields and the depth evolution of the hidden state in the classification of nonlinearly separable data. It is worth to be noticed that the models with better performance are the ones with *hyperbolic tangent* (tanh) and ELU (Clevert et al., 2015) as the vector field can assume both positive and negative values and, thus, can "force" the hidden state in different directions. On the other hand, with sigmoid, ReLU or softplus (Zheng et al., 2015), the vector field is nonnegative in all directions and thus has limited freedom. Further, Figure 8 shows how different activation functions shape the vector field and as a result the decision boundary.

B.3   Regularization for Stability

The concepts of stability can be used to regularize Neural ODEs through a variety of additional terms or different formulations (Hanshu et al., 2019; Massaroli et al., 2020). (Hanshu et al., 2019) proposes minimizing a loss term:

$$\ell_{\texttt{reg}} = \left\| \int_S^{2S} |f_{\theta(\tau)}(\tau, \mathbf{x}, \mathbf{z}(\tau))| \mathrm{d}\tau \right\|_2, \tag{21}$$

to achieve stability. However, (21) requires integration up to $2S$. A simple alternative stabilizing regularization term can be considered at no significant additional computational cost:

$$\ell_{\texttt{reg}} = \left\| f_{\theta(S)}(S, \mathbf{x}, \mathbf{z}(S)) \right\|_2, \tag{22}$$

which penalizes non–convergence to some fixed point of f_θ at $s = S$. The above can also be seen as a cheaper alternative to the kinetic energy regularization proposed in (Finlay et al., 2020).

B.4   Approximation Capabilities

Vanilla Neural ODEs are not, in general, universal function approximators (UFAs) (Zhang et al., 2019a). Besides some recent works on the topic (Zhang et al., 2019a; Li et al., 2019) this apparent limitation is still not well–understood in the context of continuous–depth models. When Neural ODEs are employed as general–purpose black–box modules, some assurances on the approximation

capabilities of the model are necessary. Let $n_z := n_x + 1$ and let $\mathbf{z} := (\mathbf{z}_x, z_a)$ ($\mathbf{z}_x \in \mathbb{R}^{n_x}$, $z_a \in \mathbb{R}$). (Zhang et al., 2019a) noticed that a depth–invariant augmented Neural ODE

$$\begin{bmatrix} \dot{\mathbf{z}}_x \\ \dot{z}_a \end{bmatrix} = \begin{bmatrix} \mathbb{0}_{n_x} \\ f_\theta(\mathbf{z}_x) \end{bmatrix}, \quad \begin{bmatrix} \mathbf{z}_x(0) \\ z_a(0) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}, \quad s \in [0, 1] \tag{23}$$

where the output is picked as $\hat{y} := z_a(1)$, can approximate any function $\Psi : \mathbb{R}^{n_x} \to \mathbb{R}$ provided that the neural network $f_\theta(\mathbf{x})$ is an approximator of Ψ, since $z_a(1) = f_\theta(\mathbf{x})$, mimicking the mapping $\mathbf{x} \mapsto f_\theta(\mathbf{x})$. Although this simple result is not sufficient to provide a constructive blueprint to the design of Neural ODE models, it suggests the following (open) questions:

- Why should we use a Neural ODE if its vector field can solve the approximation problem as a standalone neural network?

- Can Neural ODEs be UFAs with non-UFA vector fields?

On the other hand, if Neural ODEs are used for model discovery or observation of dynamical systems, requiring an UFA neural network to parametrize the model provides it with the ability to approximate arbitrary dynamical systems.

B.5 Software Implementation of Data–Control

We report here a short PyTorch code snippet detailing the implementation of a *data–controlled Neural ODE*, accompanied, for further accessibility, by a brief text description.

```python
class DC_DEFunc(nn.Module):
    """PyTorch implementation of data--controlled $f_\theta$"""
    def __init__(self, f):
        super().__init__()
        self.f = f

    def forward(self, s, z):
    """Forward is called by the ODE solver repeatedly"""
        self.nfe += 1
        # data-control step:
        # alternatives include multiplicative interactions `x * z`
        # or addition `x + z`
        z = torch.cat([z, self.x], 1)
        dz = self.f(z)
        return dz
```

where the initial condition $\mathbf{x}$ is passed to the model at the start of the integration at $s = 0$. The information contained is thus passed repeatedly to the function f_θ, conditioning the dynamics. It should be noted that even in the case of concatenation of $\mathbf{x}$ and $\mathbf{z}(s)$, the above is not a form of augmentation, since the state itself is not given additional dimensions during forward propagation. In fact, the dynamics take the form of a function $f_\theta : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \to \mathbb{R}^{n_z}$ instead of $f_\theta : \mathbb{R}^{n_a} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_a} \times \mathbb{R}^{n_x}$ as is the case for general first–order augmentation with $n_z = n_x + n_a$.

C   Experimental Details

Computational resources   The experiments were carried out on a cluster of 2 12GB NVIDIA® Titan Xp GPUs and CUDA 10.1. All Neural ODEs were trained on GPU using torchdiffeq (Chen et al., 2018) PyTorch package.

General experimental setup   We report here information and hyperparameters that are shared across several experiments. For the nested n–spheres problem we use $2^{13} = 8192$ data points evenly split between the two classes. In general, the models are trained for 2000 epochs using *Adam* (Kingma and Ba, 2014) with learning rate $lr = 10^{-3}$ and no scheduling. The adjoint sensitivity method is used
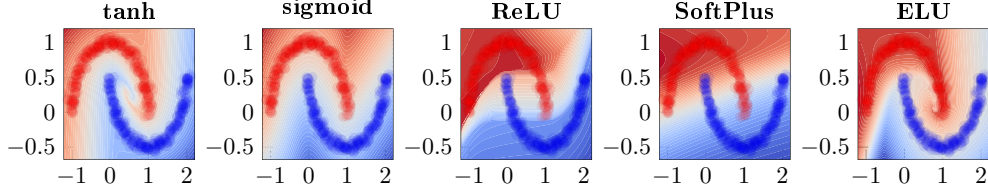
18

Figure 8: Decision boundaries learned by the vector field of a Neural ODE are directly conditioned by the choice of activation function.
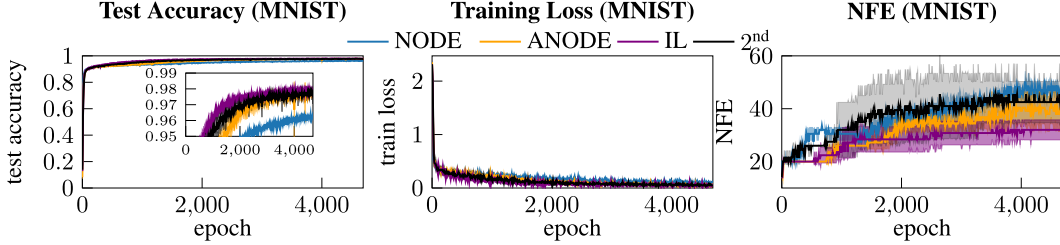

Figure 9: Training curves on MNIST for the different augmentation variants of NODEs. Shaded area indicates the 1 standard deviation interval.

only in MNIST experiments, whereas we choose backpropagation through the numerical solver for all the other empirical evaluations. This choice is motivated by the need for smaller numerical errors on the gradient, necessary to learn more complex and dynamic flows often required by depth–invariant Neural ODE variants. All Neural ODEs are solved numerically via the *Dormand–Prince* method (Prince and Dormand, 1981) with both relative and absolute tolerances set to 10^{-3} for MNIST classification and 10^{-6} in all other cases.

We refer to *concat* as the depth–variant Neural ODE variants where the depth–variable s is concatenated to $\mathbf{z}(s)$ as done in (Chen et al., 2018). Furthermore, we denote Galërkin Neural ODEs as *GalNODE* for convenience.

Benchmark problems   Throughout the paper we extensively utilize the *concentric annuli* benchmark task introduced in (Dupont et al., 2019) is used extensively. Namely, given $r > 0$ define $\varphi : \mathbb{R}^n \to \mathbb{Z}$

$$\varphi(\mathbf{x}) = \begin{cases} -1 & \|\mathbf{x}\|_2 < r \\ 1 & \|\mathbf{x}\|_2 \geq r \end{cases} . \tag{24}$$

We consider learning the map $\varphi(\mathbf{x})$ with Neural ODEs prepending a linear layer $\mathbb{R}^n \to \mathbb{R}$. Notice that φ has been slightly modified with respect to (Dupont et al., 2019), to be *well–defined* in its domain. For the one–dimensional case, we will often instead refer to the map $\varphi(x) = -x$ as the *crossing trajectories* problem. The optimization is carried out by minimizing *mean squared error* (MSE) losses of model outputs and mapping φ.

	Vanilla		ANODE		IL–NODE		2nd		
Layer	Input dim.	Output dim.	Input dim.	Output dim.	Input dim.	Output dim.	Input dim.	Output dim.	Activation
in–layer	–	–	–	–	1	6	1	6	None
f_θ–1	1	64	6	64	6	64	6	64	ReLU
f_θ–2	64	64	64	64	64	64	64	64	ReLU
f_θ–3	64	1	64	6	64	6	64	3	None

Table 2: Channel dimensions across the architectures used for MNIST experiments. The output is passed to a single linear layer to obtain classification probabilities for the 10 digit classes. For *2nd*, f_θ is only tasked with computing the velocity vector field and has thus smaller output dimensions.
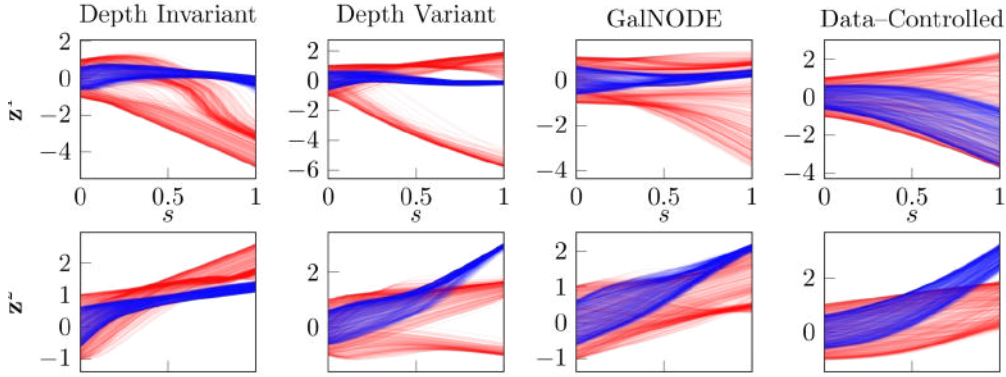
19

Figure 10: Evolution of the input data through the depth of the Neural ODEs

C.1   Experiments of Section 3

Trajectory tracking   Consider the problem of tracking a periodic signal $\beta(s)$. We show how this can be achieved without introducing additional inductive biases such as (Greydanus et al., 2019) through a synergistic combination of a two–layer Galërkin Neural ODEs and the generalized adjoint with integral loss $l(s) := \|\beta(s) - \mathbf{z}(s)\|_2^2$. In particular, we construct a two–layer Galërkin Neural ODE with Fourier series and $m = 2$ harmonics as the eigenfunctions. The training is carried out for 1000 epochs with learning rate 10^{-3}. The practical implementation of the generalized adjoint necessary to distribute the loss across the depth domain is discussed in Appendix A.

The models, trained in $s \in [0, 1]$ generalize accurately when tasked to perform long trajectory extrapolation of several seconds.

Depth–varying classification   We showcase how different discretization options of the functional optimization problem discussed in Sec. 3 affect the final dynamics of $\theta(s)$. Namely, we consider a simple binary classification on the *nested spirals* problem, training all models for 300 epochs and learning rate $5 \cdot 10^{-3}$. Galërking Neural ODEs are equipped with a polynomial basis with $m = 10$. The Fig.s in Sec 3 reveal the different nature of $\theta(s)$ depending on model choice: depth–discretization of *Stacked* yields a flexible, though lower resolution form of $\theta(s)$, whereas spectral discretizations limit the functional form of $\theta(s)$ to the span of a chosen eigenbasis.

Further experiments on non–augmented variants   Further experimental results on performance comparisons of different Neural ODEs dealing with the *concentric annuli* task are here reported. In particular, Fig. 10 shows the depth evolution of the data for the different architectures and Fig. 11 displays how the input data are transported by the vector fields through the Neural ODEs' depth towards linearly separable manifolds.

Mind your input network experiments   We tackle the *concentric annuli* task with a Neural ODE preceded by a simple two–layer neural network with 16 units and ReLU activation. The second layer is linear.

C.2   Experiments of Section 4

MNIST experiments   We closely follow the experimental setup of (Dupont et al., 2019). We use *Adam* with learning rate 10^{-3} and batch size 256. We use 3–layer depth–invariant CNNs for parametrizing the vector fields f_θ. The choice of depth–invariance is motivated by the discussion carried out in Section 5: both augmentation and depth–variance can relieve approximation limitations of vanilla, depth–invariant Neural ODEs. As a result, including both renders the ablation study for augmentation strategies less accurate.

For input layer augmented Neural ODE models, namely IL–NODE and 2nd order, we prepend to the Neural ODE a single, linear CNN layer. The hidden channel dimension of the CNN parametrizing
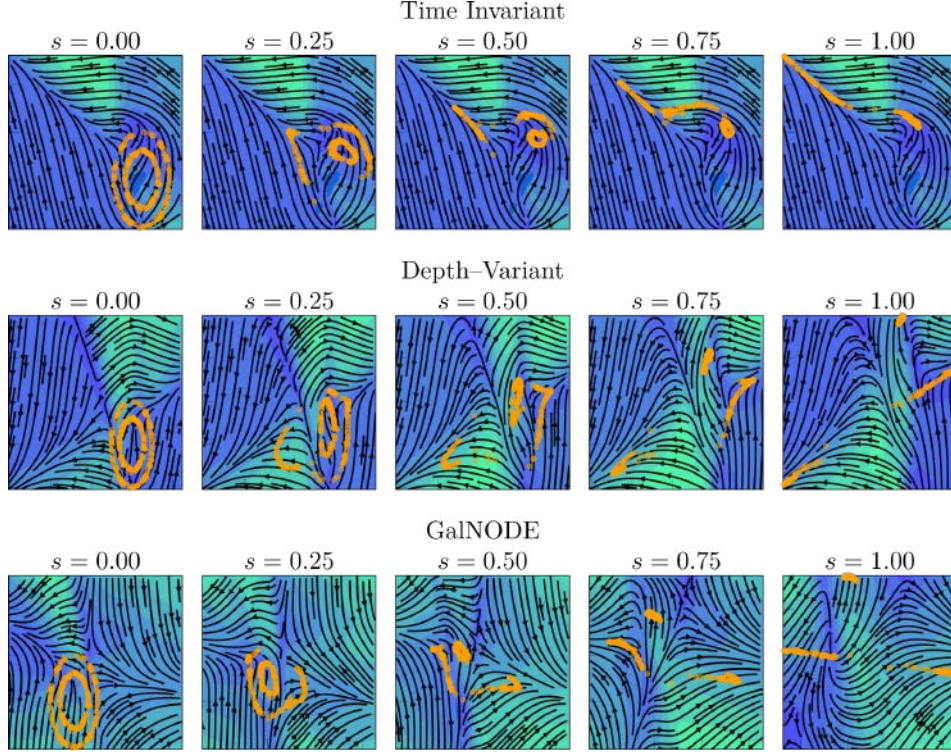
Figure 11: Depth evolution of the learned vector fields

f_θ is set to 64. Second order Neural ODEs, *2nd*, use f_θ to compute the vector field of velocities: therefore, the output of f_θ is $n_x/2$–dimensional, and the remaining $n_x/2$ outputs to concatenate (vector field of *positions*) are obtained as the last $n_x/2$ elements of **z**. Architectures are shown in detail in Table 2.

Figure 9 highlights how vanilla Neural ODEs are capable of convergence without any spikes in loss or NFEs. We speculate the numerical issues encountered in (Dupont et al., 2019) to be a consequence of the specific neural network architecture used to parametrize the vector field f_θ, which employed an excessive number of channels inside f_θ, i.e 92.

Augmentation experiments on nested 2-spheres   We qualitatively compare ANODE with second order Neural ODEs (*2nd*) on the nested 2-spheres problem to verify whether the proposed second order model is capable of learning simple, non–stiff flows in a more parameter efficient manner. Both models have f_θ parametrized by a neural network with 2 hidden layers of 16 neurons. Moreover, for both models, only the first two "non–augmented" hidden states are retained and fed to the linear classification layer following the Neural ODE. The training is carried out by backpropagating directly through the solver. Figure 13 shows the flows of states and augmented states for both models.

C.3   Experiments of Section 5

Experiments on crossing trajectories   We trained both current state–of–the–art as well as proposed models to learn the map $\varphi(x) = -x$. We created a training dataset sampling x equally spaced in $[-1, 1]$. The models have been trained to minimize L1 losses using Adam (Kingma and Ba, 2014) with learning rate $lr = 10^{-3}$ and weight decay 10^{-5} for 1000 epochs using the whole batch.

- **Vanilla Neural ODEs** We trained vanilla Neural ODEs, i.e. both depth–invariant and depth variant models ("concat" and GalNODE). As expected, these models cannot approximate φ. Both depth–invariant and *concat* have been selected with two hidden layers of 16 and 32 neurons each, respectively and *tanh* activation. The GalNODE have been designed with one hidden layer of 32 neurons whose depth–varying weights were parametrized by a Fourier

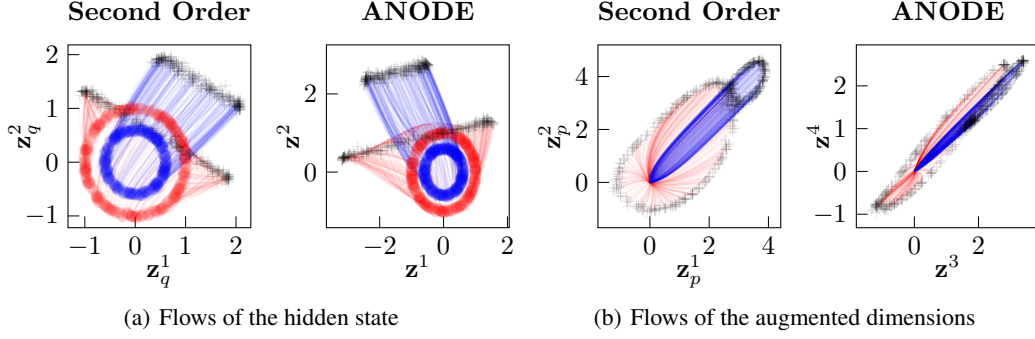(a) Flows of the hidden state   (b) Flows of the augmented dimensions

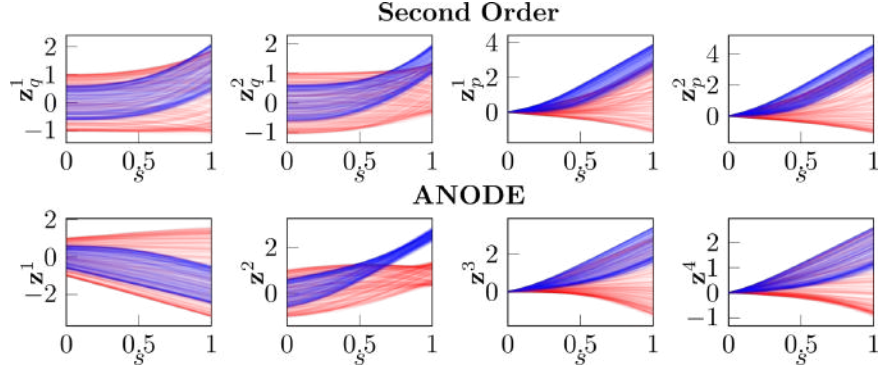Figure 12: Second Order versus ANODE. Trajectories in the respective hidden spaces of the data.

Figure 13: Depth evolution of the hidden and augmented dimensions.

series of five modes. The resulting trajectories over the learned vector fields are shown in Fig. 14.

- **Data–controlled Neural ODEs** We evaluate both the handcrafted linear depth–invariant model (9) and the general formulation of data–controlled models (10), realized with two hidden layers of 32 neurons each and *tanh* activation in all layers but the output. Note that the loss of the handcrafted model results to be convex and continuously differentiable. Moreover, proof A.5 provides analytically a lower bound on the model parameter to ensure the loss to be upper–bounded by a desired ϵ, making its training superfluous. Nevertheless, we provide results with a trained version to show that the benefits of data–controlled Neural ODEs are compatible with gradient–based learning.

The results are shown in Fig.s 14 and 15. The input data information embedded into the vector field allows the Neural ODE to steer the hidden state towards the desired label through its continuous depth. Data–controlled Neural ODEs can be used to learn challenging maps (Dupont et al., 2019) without augmentation.
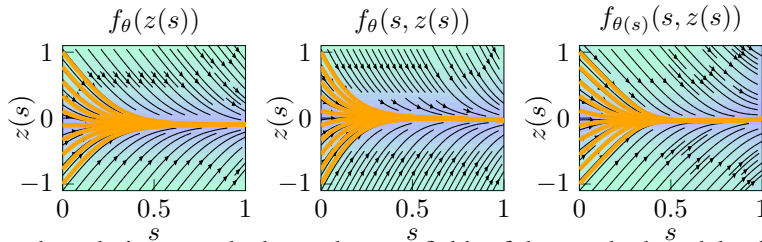

Figure 14: Depth evolution over the learned vector fields of the standard models: depth–invariant and depth–variant ("concat" $f_\theta(s, z(s))$ and GalNODE $f_\theta(s, z(s))$). As expected the Neural ODE cannot approximate the map $\varphi(x) = -x$.
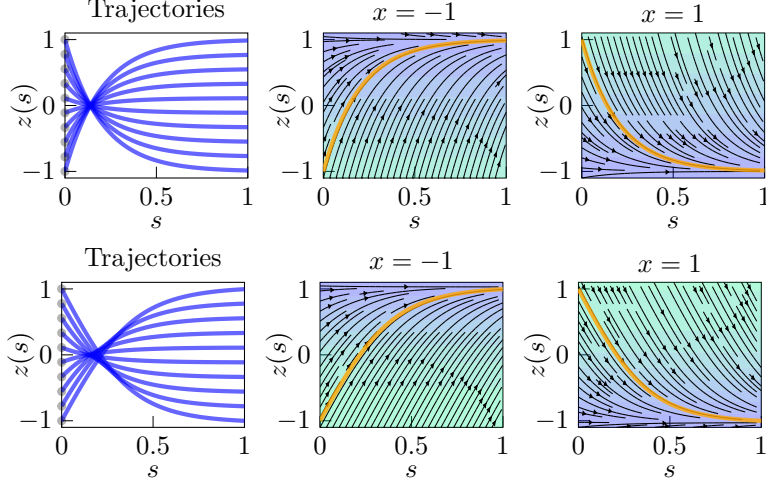
Figure 15: Depth evolution over the learned vector fields of the standard models: depth–invariant and depth–variant ("concat" $f_\theta(s, z)$ and GalNODE $f_{\theta(s)}(s, z)$). As expected the Neural ODE cannot approximate the map $\varphi(x) = -x$.
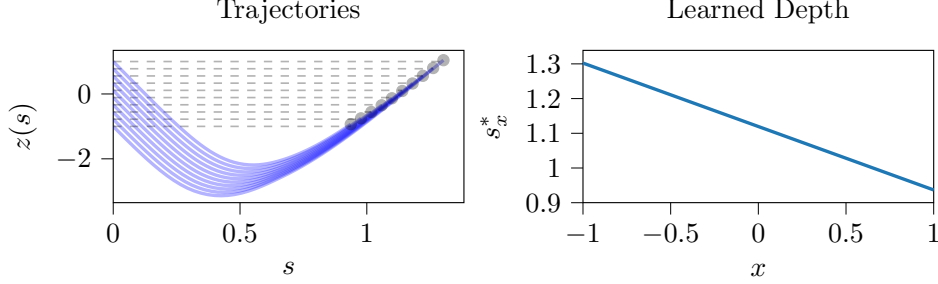

Figure 16: Evolution of the input data through the depth of the Neural ODEs

- **Adaptive depth Neural ODEs** The experiments have been carried out with a depth–variant Neural ODE in "concat" style where f was parametrized by a neural network with two hidden layers of 8 units and tanh activation. Moreover, the function $g_\omega(\mathbf{x})$ computing the data–adaptive depth of the Neural ODE was composed by a neural network with one hidden layer (8 neurons and ReLU activation) whose output is summed to one and then taken in absolute value,

$$g(\mathbf{x}) = \left| 1 + \mathbf{w}_o^\top \sigma(\mathbf{w}_i \mathbf{x} + \mathbf{b}_i) + b_o \right|$$

where σ is the ReLU activation, $\mathbf{w}_o$, $\mathbf{w}_i$, $\mathbf{b}_i \in \mathbb{R}^8$ and $\omega = (\mathbf{w}_o, b_o, \mathbf{w}_i, \mathbf{b}_i)$. In particular, the summation to one has been employed to help the network "sparsify" the learned integration depths and avoid highly stiff vector fields, while the absolute value is needed to avoid infeasible integration intervals. The training results can be visualized in Fig. 16. It is worth to be noticed how this early results should be intended as a new research direction for Neural ODEs rather than a definitive evaluation of the proposed method, out of the scope of this paper and postponed to future work. Furthermore, for the sake of simplicity, the result of Fig. 4 showed in the main text has been obtained by training the model only on $x \in \{-1, 1\}$ and setting by hand $s_{-1}^* = 1$, $s_1^* = 3$.