

# Dissecting Neural ODEs

Stefano Massaroli\*

The University of Tokyo, DiffEqML  
massaroli@robot.t.u-tokyo.ac.jp

Michael Poli\*

KAIST, DiffEqML  
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  
c The University of Tokyo  
asama@robot.t.u-tokyo.ac.jp

## Abstract

Continuous deep learning architectures have recently re-emerged as *Neural Ordinary Differential Equations* (Neural ODEs). This infinite-depth approach theoretically bridges the gap between deep learning and dynamical systems, offering a novel perspective. However, deciphering the inner working of these models is still an open challenge, as most applications apply them as generic *black-box* modules. In this work we “open the box”, further developing the continuous-depth formulation with the aim of clarifying the influence of several design choices on the underlying dynamics.

## 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

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

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	$h_x$	$\mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_z}$
Output Network	$h_y$	$\mathbb{R}^{n_z} \rightarrow \mathbb{R}^{n_y}$

\*Equal contribution. Author order was decided by flipping a coin.

● **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 two novel variants of Neural ODEs: a Galerkin-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  $\mathbf{x}$ , 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 can make Neural ODE flows *superfluous*.

## 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 *initial value problem* (IVP) (1), i.e.

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

Our degree of freedom, other than  $h_x$  and  $h_y$ , in the Neural ODE model is the choice of the parameter  $\theta$  inside a given pre-specified class  $\mathcal{W}$  of functions  $\mathcal{S} \rightarrow \mathbb{R}^{n_\theta}$ .

**Well-posedness** If  $f_{\theta(s)}$  is Lipschitz, for each  $\mathbf{x}_k$  the initial value problem in (1) admits a unique solution  $\mathbf{z}$  defined in the whole  $\mathcal{S}$ . If this is the case, there is a mapping  $\phi$  from  $\mathcal{W} \times \mathbb{R}^{n_x}$  to the space of absolutely continuous functions  $\mathcal{S} \mapsto \mathbb{R}^{n_z}$  such that  $\mathbf{z}_k := \phi(\mathbf{x}_k, \theta)$  satisfies the ODE in (1). This in turn implies that, for all  $k \in \mathcal{K}$ , the map  $(\theta, \mathbf{x}_k, s) \mapsto \gamma(s, \mathbf{x}_k, \theta) := h_y(\phi(\theta, \mathbf{x}_k)(s))$  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.  $\mathcal{W}$  is the space of constant functions) 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  $\mathcal{S}$ , e.g.

$$\ell := L(\mathbf{z}(S)) + \int_S l(\tau, \mathbf{z}(\tau)) d\tau \quad (2)$$

The training can be then cast into the *optimal control* (Pontryagin et al., 1962) problem

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

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

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

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

Supplementary material 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 fully considered the *deep limit* of ResNets. In fact, while each residual block is characterized by its own parameters vector  $\theta_s$ , the authors consider model  $\dot{\mathbf{z}} = f_\theta(s, \mathbf{z}(s))$  where the depth variable  $s$  enters in the dynamics *per se*<sup>2</sup> 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  $n_\theta$ . 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: *Galérkin Neural ODEs* and *Stacked Neural ODEs*.

**Gradient descent in functional space** • When the model parameters are depth-varying,  $\theta : \mathcal{S} \rightarrow \mathbb{R}^{n_\theta}$ , 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  $\mathbb{L}_2(\mathcal{S} \rightarrow \mathbb{R}^{n_\theta})$  be the space of square-integrable functions  $\mathcal{S} \rightarrow \mathbb{R}^{n_\theta}$ . Hereafter, we show that if  $\theta(s) \in \mathcal{W} := \mathbb{L}_2(\mathcal{S} \rightarrow \mathbb{R}^{n_\theta})$ , then the loss sensitivity to  $\theta(s)$  can be computed through the adjoint method.

**Theorem 1** (Infinite-Dimensional Gradients). *Consider the loss function (2) and let  $\theta(s) \in \mathbb{L}_2(\mathcal{S} \rightarrow \mathbb{R}^{n_\theta})$ . Then, sensitivity of  $\ell$  with respect to  $\theta(s)$  (i.e. directional derivative in functional space) is*

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

Note that, although Theorem 1 provides a constructive method to compute the loss gradient in the *infinite-dimensional* setting, its implementation requires choosing a finite dimensional approximation of the solution. We offer two alternatives: a *spectral discretization* approach relying on reformulating the problem on some functional bases and a *depth discretization* approach.

**Spectral discretization: Galérkin Neural ODEs** • The idea is to expand  $\theta(s)$  on a complete orthogonal basis of a predetermined subspace of  $\mathbb{L}_2(\mathcal{S} \rightarrow \mathbb{R}^{n_\theta})$  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 finite dimension and the training will aim to optimize the parameters  $\alpha = (\alpha_1, \dots, \alpha_m) \in \mathbb{R}^{mn_\theta}$  whose gradients can be computed as follows

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

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

**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 is easy to see how evaluating this model is equivalent to *stacking*  $p$  Neural ODEs with constant parameters,

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

Here, the training is carried out optimizing the resulting  $pn_\theta$  parameters using the following:

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

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

<sup>2</sup>In practice,  $s$  is often concatenated to  $\mathbf{z}$  and fed to  $f_\theta$ .

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 the Appendix.

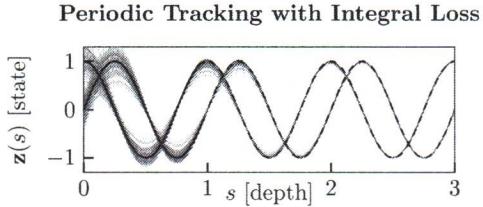


Figure 1: Galérkin Neural ODEs trained with integral losses accurately recover periodic signals. Blue curves correspond to different initial conditions and converge asymptotically to the reference desired trajectory.

**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 Galérkin Neural ODEs and the generalized adjoint with integral loss  $l(s) := \|\beta(s) - z(s)\|_2^2$ . 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 Galérkin and Stacked variants trained to solve a simple binary classification problem. Additional insights and details are reported in Appendix.

Depth-variance brings Neural ODEs closer to the ideal continuum of neural network layers with untied weights, enhancing their expressivity.

#### 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:  $\mathbf{z}(0) = [\mathbf{x}, \mathbf{0}]$ . 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

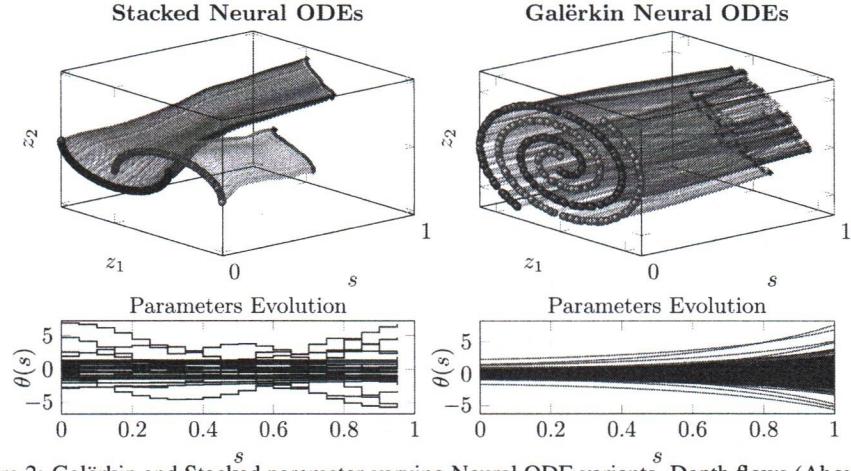


Figure 2: Galérkin and Stacked parameter-varying Neural ODE variants. Depth flows (Above) and evolution of the parameters (Below).

input network  $h_x : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_z}$  to compute  $\mathbf{z}(0)$ :

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

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  $\mathbf{x}$  and 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).

Note that 0-augmentation is compatible with the general IL formulation, as it corresponds to

$$\mathbf{x} \mapsto (\mathbf{x}, 0) := h_x(\mathbf{x})$$

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.  $h_x := [\mathbf{x}, \xi(\mathbf{x})]$ ,  $\xi : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_a}$ .

**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:

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

equivalent to the first-order system

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

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

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

or, equivalently,  $\dot{\mathbf{z}}^i(s) = \mathbf{z}^{i+1}(s)$ ,  $\dot{\mathbf{z}}^n(s) = f_{\theta(s)}(s, \mathbf{z}(s))$ . Note that the parameter efficiency of this method arises from the fact that  $f_{\theta(s)} : \mathbb{R}^{n_z} \rightarrow \mathbb{R}^{n_z/n}$  instead of  $\mathbb{R}^{n_z} \rightarrow \mathbb{R}^{n_z}$ . 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 image classification 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. The input network  $h_x$  is composed of a single, linear layer. Main objective of these experiments is to rank the efficiency of different augmentation strategies; for this reason, the setup does not involve hybrid or composite Neural ODE architectures and data augmentation.

The results for five experiments are reported in Table 4. IL–NODEs consistently preserve lower NFEs than other variants, whereas second–order Neural ODEs offer a parameter efficient alternative. The performance gap widens on CIFAR10, where the disadvantage of fixed 0 initial conditions forces 0-augmented Neural ODEs into performing a high number of function evaluations.

	NODE		ANODE		IL-NODE		2nd-Ord.	
	MNIST	CIFAR	MNIST	CIFAR	MNIST	CIFAR	MNIST	CIFAR
Test Acc.	96.8	58.9	98.9	70.8	99.1	<b>73.4</b>	<b>99.2</b>	72.8
NFE	98	93	71	169	44	65	<b>43</b>	<b>59</b>
Param.[K]	21.4	37.1	20.4	35.0	20.7	36.1	<b>20.0</b>	<b>34.6</b>

Table 1: Mean test results across 10 runs on MNIST and CIFAR. We report the mean NFE at convergence. Input layer and higher order augmentation improve task performance and preserve low NFEs at convergence.

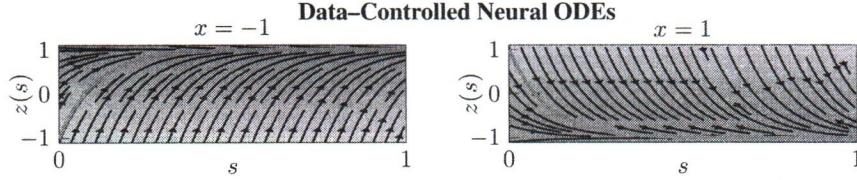


Figure 3: Depth trajectories over vector field of the *data-controlled* neural ODEs (9) for  $x = 1$ ,  $x = -1$ . The model learns a family of vector fields conditioned by the input  $x$  to approximate  $\varphi(x)$ .

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.

Augmentation relieves Neural ODEs of their expressivity limitations. Learning initial conditions improves on 0-augmentation in terms of performance and NFEs.

## 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 not 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 Galérkin Neural ODEs can solve these tasks in all dimensions but  $\mathbb{R}$ .

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  $\varphi$  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 arbitrarily approximate  $\varphi$  without augmentation, providing a novel perspective on existing results about expressivity limitations of continuous models (Dupont et al., 2019). The result is the following:

**Proposition 2.** *For all  $\epsilon > 0$ ,  $x \in \mathbb{R}$  there exists a parameter  $\theta > 0$  such that*

$$|\varphi(x) - z(1)| < \epsilon, \quad (8)$$

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

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

The proof is reported in the Appendix. Fig. (3) shows a version of model (9) where  $\theta$  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

$$\begin{aligned}\dot{\mathbf{z}}(s) &= f_{\theta(s)}(s, \mathbf{x}, \mathbf{z}(s)) \\ \mathbf{z}(0) &= h_x(\mathbf{x})\end{aligned}. \quad (10)$$

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. Direct 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  $\varphi$  are reported in the Appendix.

It should be noted that (10) does not require explicit dependence of the vector field on  $\mathbf{x}$ . Computationally,  $\mathbf{x}$  can be passed to  $f_{\theta(s)}$  in different ways, such as through an additional embedding step. In this setting, data-control offers a natural extension to conditional Neural ODEs.

#### Conditional Continuous Normalizing Flows

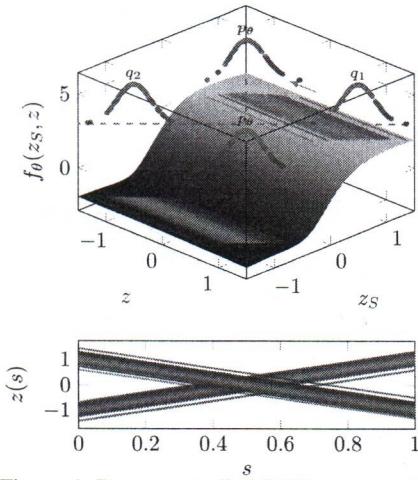


Figure 4: Data-controlled CNFs can morph prior distributions into distinct posteriors to produce conditional samples. This task often requires crossing trajectories and is not possible with vanilla CNFs. sampling the known distribution  $\mathbf{z}_S \sim q(\mathbf{z}_S)$  and evolve  $\mathbf{z}_S$  backward in the depth domain:

$$\mathbf{z}(0) = \mathbf{z}_S + \int_S^0 f_{\theta}(s)(s, \mathbf{z}(s))ds$$

In this context, introducing data-control into  $f_{\theta}$  allows the CNF to be conditioned with data or task information. Data-controlled CNFs can thus be used in multi-objective generative tasks e.g using a single model to sample from  $N$  different distribution  $p_{\theta}$  by warping  $N$  predetermined known distributions  $q_i$ . We train one-dimensional, data-controlled CNFs to approximate two different data distributions  $p_1, p_2$  by sampling from two distinct priors  $q_1, q_2$  and conditioning the vector field with the samples  $z_S$  of the prior distributions, i.e.

$$\dot{z}(s) = f_{\theta}(z_S, z(s)), \quad z_S \sim q_1 \text{ or } z_S \sim q_2$$

Fig 4 shows how data-controlled CNFs are capable of conditionally sampling from two normal target data distributions. In this case we selected  $p_1, p_2$  as univariate normal distributions with mean  $-1$  and  $1$ , respectively and  $q_1 \equiv p_2, q_2 \equiv p_1$ . The resulting learned vector field strongly depends on the value of the prior sample  $z_S$  and it is almost constant in  $z$ , meaning that the prior distributions are just shifted almost rigidly along the flow in a direction determined by the initial condition. This task is inaccessible to standard CNFs as it requires crossing flows in  $z$ . Indeed, the proposed benchmark represents a density estimation analogue to the crossing trajectories problem.

## 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  $\varphi$  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  $\varphi$  without crossing flows as shown in Fig. 5. In general, we can use a hypernetwork  $g$  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  $x \mapsto \phi_{g_\omega(x)}(x)$ , i.e. leading to

$$\hat{y} = h_y \left( h_x(x) + \int_0^{g_\omega(x)} f_{\theta(s)}(\tau, x, z(\tau)) d\tau \right),$$

where  $g_\omega : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\omega} \rightarrow \mathbb{R}$  is a neural network with trainable parameters  $\omega$ . Supplementary material contains details on differentiation under the integral sign, required to back-propagate the loss gradients into  $\omega$ .

## 5.3 Additional Results

**Experiments of non-augmented models** We inspect the performance of different Neural ODE variants: depth-invariant, depth-variant with  $s$  concatenated to  $z$  and passed to the vector field, Galerkin 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 Galerkin Neural ODEs, we choose a Fourier series with  $m = 5$  harmonics as the eigenfunctions  $\psi_k$ ,  $k = 1, \dots, 5$  to compute the parameters  $\theta(s)$ , as described in Sec. 3.

Data-control allows Neural ODEs to learn a family of vector fields, conditioning on input data information. Depth-adaptation sidesteps known expressivity limitations of continuous-depth models.

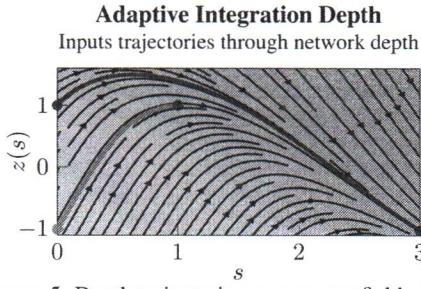


Figure 5: Depth trajectories over vector field of the *adaptive–depth* Neural ODEs. The reflection map can be learned by the proposed model. The key is to assign different integration times to the inputs, thus not requiring the intersection of trajectories.

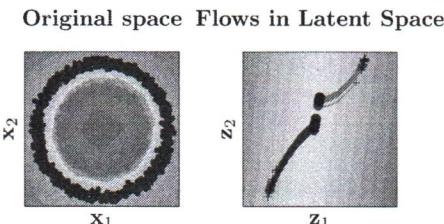


Figure 6: Solving concentric annuli without augmentation by prepending a nonlinear transformation performed by a two-layer fully-connected network.

architectures preceded or followed by several layers. 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.

**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. 6 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

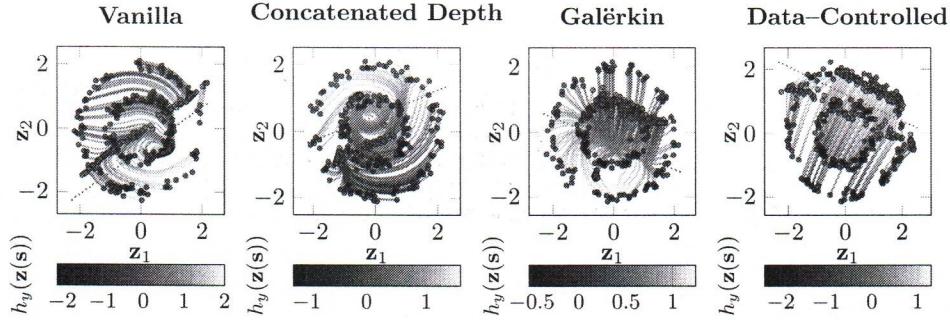


Figure 7: Depth-flows of the data in the state–space. The resulting decision boundaries of output linear layer  $h_y$  are indicated by the dotted orange line.

## 6 Related Work

We include a brief history of classical approaches to 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 machine learning (Zhang et al., 2014; Li et al., 2017; Lu et al., 2017; Weinan, 2017). (Marcus and Westervelt, 1989) provides stability analyses and introduces delays. Many of these concepts have yet to resurface in the context of Neural ODEs. Haber and Ruthotto (2017) analyzes ResNet dynamics and links stability with robustness. Injecting stability into neural networks has inspired the design of a series of architectures (Chang et al., 2019; Haber et al., 2019; Bai et al., 2019; Massaroli et al., 2020). 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).

Approximating ODEs with neural networks has been discussed in (Wang and Lin, 1998; Filici, 2008). (Poli et al., 2020a) explores the interplay between Neural ODEs and their solver. 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 (Li et al., 2020; 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 and thus require state–augmentation. Leveraging our *generalized adjoint* formulation provides an approach to 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 Galérkin 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. The code to reproduce all the experiments present in the paper is built on TorchDyn (Poli et al., 2020b) and PyTorch-Lightning (Falcon et al., 2019) and can be found in the following repo: <https://github.com/DiffEqML/diffeqml-research/tree/master/dissecting-neural-odes>.

## 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.

## Acknowledgment

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, 2018R1D1A1B07050443.

## 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.
- W. Falcon et al. Pytorch lightning. *GitHub. Note: https://github.com/williamFalcon/pytorch-lightning Cited by*, 3, 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.
- W. Grathwohl, R. T. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.
- 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.
- 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, L. Chen, C. Tai, and E. Weinan. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research*, 18(1):5998–6026, 2017.
- Q. Li, T. Lin, and Z. Shen. Deep learning via dynamical systems: An approximation perspective. *arXiv preprint arXiv:1912.10382*, 2019.
- X. Li, T.-K. L. Wong, R. T. Q. Chen, and D. Duvenaud. Scalable gradients for stochastic differential equations. volume 108 of *Proceedings of Machine Learning Research*, pages 3870–3882, Online, 26–28 Aug 2020. PMLR. URL <http://proceedings.mlr.press/v108/li20i.html>.
- Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems*, pages 6231–6239, 2017.
- 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.
- M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park. Hypersolvers: Toward fast continuous-depth models. *arXiv preprint arXiv:2007.09601*, 2020a.
- M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park. Torchdyn: A neural differential equations library. *arXiv preprint arXiv:2009.09346*, 2020b.
- 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.
- C. Villani. *Topics in optimal transportation*. Number 58. American Mathematical Society, 2003.
- 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.
- E. Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.
- 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<sup>2</sup> 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

<b>A Proofs and Additional Theoretical Results</b>	<b>13</b>
A.1 Proof of Theorem 1 . . . . .	13
A.2 Proof of Theorem 1 . . . . .	14
A.3 Proof of Corollary 1 . . . . .	15
A.4 Proof of Corollary 2 . . . . .	16
A.5 Proof of Theorem 2 . . . . .	16
A.6 Additional Theoretical Results . . . . .	17
<b>B Practical Insights for Neural ODEs</b>	<b>18</b>
B.1 Augmentation . . . . .	18
B.2 Activations . . . . .	18
B.3 Regularization for Stability . . . . .	19
B.4 Approximation Capabilities . . . . .	19
B.5 Example Implementation of Data–Control . . . . .	20
<b>C Experimental Details</b>	<b>20</b>
C.1 Experiments of Section 3 . . . . .	21
C.2 Experiments of Section 4 . . . . .	21
C.3 Experiments of Section 5 . . . . .	22

---

## A Proofs and Additional Theoretical Results

### A.1 Proof of Theorem 1

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

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

*Proof.* Let us define a *Lagrange multiplier* or *adjoint state*  $\mathbf{a}$ , dual to  $\mathbf{z}$ . As the dual of  $\mathbb{R}^{n_z}$  is  $\mathbb{R}^{n_z}$  itself,  $\mathbf{a} \in \mathbb{R}^{n_z}$ . Moreover, let  $\mathcal{L}$  be a perturbed loss function of the form

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

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  $d\mathcal{L}/d\theta = d\ell/d\theta$ . For the sake of compactness we do not explicitly write the

dependence on variables of the considered functions unless strictly necessary. Note that,

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

obtained via integration by parts. Hence,

$$\begin{aligned} \mathcal{L} &= \ell - \mathbf{a}^\top(\tau) \mathbf{z}(\tau)|_0^S + \int_0^S (\dot{\mathbf{a}}^\top \mathbf{z} + \mathbf{a}^\top f_\theta) d\tau \\ &= L(\mathbf{z}(S)) - \mathbf{a}^\top(\tau) \mathbf{z}(\tau)|_0^S + \int_0^S (\dot{\mathbf{a}}^\top \mathbf{z} + \mathbf{a}^\top f_\theta + l) d\tau \end{aligned} \quad (11)$$

We can compute the gradient of  $\ell$  with respect to  $\theta$  as

$$\begin{aligned} \frac{d\ell}{d\theta} &= \frac{d\mathcal{L}}{d\theta} = \frac{\partial L(\mathbf{z}(S))}{\partial \mathbf{z}(S)} \frac{d\mathbf{z}(S)}{d\theta} - \mathbf{a}^\top(S) \frac{d\mathbf{z}(S)}{d\theta} - \mathbf{a}^\top(0) \cancel{\frac{d\mathbf{z}(0)}{d\theta}} \\ &\quad + \int_0^S \left[ \dot{\mathbf{a}}^\top \frac{d\mathbf{z}}{d\theta} + \mathbf{a}^\top \left( \frac{\partial f_\theta}{\partial \theta} + \frac{\partial f_\theta}{\partial \mathbf{z}} \frac{d\mathbf{z}}{d\theta} + \frac{\partial f_\theta}{\partial \mathbf{x}} \cancel{\frac{d\mathbf{x}}{d\theta}} + \frac{\partial f_\theta}{\partial \tau} \cancel{\frac{d\tau}{d\theta}} \right) + \frac{\partial l}{\partial \mathbf{z}} \frac{d\mathbf{z}}{d\theta} + \frac{\partial l}{\partial \tau} \cancel{\frac{d\tau}{d\theta}} \right] d\tau \end{aligned}$$

which, by reorganizing the terms, yields to

$$\begin{aligned} \frac{d\ell}{d\theta} &= \left[ \frac{\partial L}{\partial \mathbf{z}(S)} - \mathbf{a}^\top(S) \right] \frac{d\mathbf{z}(S)}{d\theta} + \\ &\quad + \int_0^S \left( \dot{\mathbf{a}}^\top + \mathbf{a}^\top \frac{\partial f_\theta}{\partial \mathbf{z}} + \frac{\partial l}{\partial \mathbf{z}} \right) \frac{d\mathbf{z}}{d\theta} d\tau \\ &\quad + \int_0^S \mathbf{a}^\top \frac{\partial f_\theta}{\partial \theta} d\tau \end{aligned} \quad (12)$$

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

$$\dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s) \frac{\partial f_\theta}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}}, \quad \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \quad (13)$$

to be solved backward in  $[0, S]$ ; then (12) reduces to

$$\frac{d\ell}{d\theta} = \int_0^S \mathbf{a}^\top \frac{\partial f_\theta}{\partial \theta} d\tau \quad (14)$$

proving the result.  $\square$

**Remark 1** (Implementation of the generalized adjoint method). *Note that, similarly to (Chen et al., 2018), the gradient (14) is practically computed by defining the parameters adjoint state  $\mathbf{a}_\theta$  and solving backward the system of ODEs*

$$\begin{aligned} \dot{\mathbf{a}}^\top &= -\mathbf{a}^\top \frac{\partial f_\theta}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}}, \quad \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \\ \dot{\mathbf{a}}_\theta^\top &= -\mathbf{a}^\top \frac{\partial f_\theta}{\partial \theta}, \quad \mathbf{a}_\theta(S) = \mathbf{0}_{n_\theta} \end{aligned} \quad (15)$$

Then,

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

## A.2 Proof of Theorem 1

**Theorem 1** (Infinite-Dimensional Gradients). *Consider the loss function (2) and let  $\theta(s) \in \mathbb{L}_2(\mathcal{S} \rightarrow \mathbb{R}^{n_\theta})$ . Then, sensitivity of  $\ell$  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 (11). 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  $\ell$ :

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

Thus,

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

Since it must hold

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

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

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

and, finally

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

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

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

we have

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

□

### A.3 Proof of Corollary 1

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

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

*Proof.* The proof follows naturally from Theorem 1 by noticing that if  $\theta(s)$  has some parametrization  $\theta = \theta(s, \mu)$  with parameters  $\mu \in \mathbb{R}^{n_\mu}$ , then,

$$\frac{d\ell}{d\mu} = \int_0^S \mathbf{a}^\top(\tau) \frac{\partial f_{\theta(\tau)}}{\partial \theta} \frac{\partial \theta}{\partial \mu} d\tau \tag{16}$$

Therefore, if

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

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

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

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

**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 (16)*

#### A.4 Proof of Corollary 2

**Corollary 2** (Stacked Gradients). *Under the assumptions of Theorem 1, 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 1 by recalling the solution of the stacked neural ODEs:

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

We can recover a relation similar to (12)

$$\begin{aligned}\frac{d\ell}{d\theta_i} &= \left[ \frac{\partial L}{\partial \mathbf{z}(S)} - \mathbf{a}^\top(S) \right] \frac{d\mathbf{z}(S)}{d\theta_i} + \\ &\quad + \sum_{j=0}^{p-1} \int_{s_j}^{s_{j+1}} \left( \dot{\mathbf{a}}^\top + \mathbf{a}^\top \frac{\partial f_{\theta_j}}{\partial \mathbf{z}} + \frac{\partial l}{\partial \mathbf{z}} \right) \frac{d\mathbf{z}}{d\theta_i} d\tau \\ &\quad + \sum_{j=0}^{p-1} \int_{s_j}^{s_{j+1}} \mathbf{a}^\top \frac{\partial f_{\theta_j}}{\partial \theta_i} d\tau\end{aligned}$$

Since

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

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

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

$\square$

#### A.5 Proof of Theorem 2

**Proposition 2.** *For all  $\epsilon > 0, x \in \mathbb{R}$  there exists a parameter  $\theta > 0$  such that*

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

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

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

*Proof.* The general solution of (9) is

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

Thus,

$$\begin{aligned} e &= \varphi(x) - z(1) = x + x(2e^{-\theta} - 1) = 2xe^{-\theta} \\ \Leftrightarrow |e| &= 2|x|e^{-\theta} \end{aligned}$$

It follows that

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

□

## 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:

$$\ell = L(\mathbf{z}(S), \theta) + \int_S l(s, \mathbf{z}(\tau), \theta) d\tau, \quad (17)$$

In this case we need to modify the adjoint gradients accordingly

**Theorem 2** (Generalized Adjoint Method with Parameter–Dependent Loss). *Consider the loss function (17). Then,*

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

where  $\mathbf{a}(s)$  satisfies (13).

*Proof.* The proof follows immediately from Theorem 1 by noticing that, with the explicit dependence on  $\theta$  of  $\ell$ , (12) would become

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

leading to the result. □

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

$$\ell = L(\mathbf{z}(S), \theta(S)) + \int_S l(\mathbf{z}(\tau), \theta(\tau)) d\tau \quad (18)$$

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

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

*Proof.*

$$\begin{aligned}\frac{d\ell}{dS} &= \frac{\partial L}{\partial \mathbf{z}(S)} \frac{d\mathbf{z}(S)}{dS} + \frac{d}{dS} \int_0^S l(\mathbf{z}(\tau)) d\tau \\ &= \frac{\partial L}{\partial \mathbf{z}(S)} \frac{d}{dS} \left( h_x(\mathbf{x}) + \int_0^S f_{\theta(\tau)}(\tau, \mathbf{x}, \mathbf{z}(\tau)) d\tau \right) + \frac{d}{dS} \int_0^S l(\mathbf{z}(\tau)) d\tau\end{aligned}$$

Therefore, by applying the Leibniz integral rule we obtain

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

□

## 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*, height, 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.

**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  $\mathbf{z} := (\mathbf{z}_q, \mathbf{z}_p, \bar{\mathbf{z}})$ ,  $\mathbf{z}_q, \mathbf{z}_p \in \mathbb{R}^{n_a/2}$ ,  $\bar{\mathbf{z}} \in \mathbb{R}^{n_z - n_a}$ . 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, this approach yields

$$\begin{bmatrix} \dot{\mathbf{z}}_q \\ \dot{\mathbf{z}}_p \\ \bar{\mathbf{z}} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_p \\ f_{\theta(s)}^p(s, \mathbf{z}) \\ \bar{f}_{\theta(s)}(s, \mathbf{z}) \end{bmatrix}, \quad (19)$$

A similar argument could be applied to orders higher than two. *Selective higher-order* Neural ODEs are compatible with input layer augmentation.

### B.2 Activations

**Mind your activation** We investigate the effects of appending an activation function to the last layer of  $f_\theta$ . The chosen nonlinearity will strongly affect the “shape” of the vector field and, as a

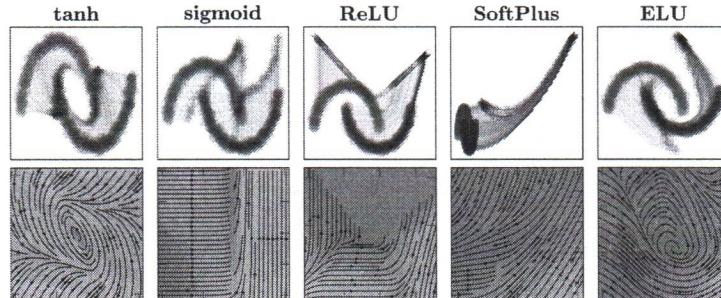


Figure 8: Depth trajectories of the hidden state and relative vector fields  $f_\theta(\mathbf{z})$  for different activation functions in a nonlinear classification task. It can be noticed how the models with tanh and ELU outperform the others, as  $f_\theta$  is able to steer  $\mathbf{z}$  along negative directions.

consequence, the flows learnable by the model. Therefore, while designing  $f_\theta$  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, such as boundedness of the vector field.

**Effects of activations** In order to compare the effect of different activation functions in the last layer of  $f_\theta$ , 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 ( <b>tanh</b> )	bounded
<b>Sigmoid</b>	bounded, non-negative output
<b>ReLU</b>	unbounded, non-negative output
<b>Softplus</b>	unbounded, non-negative output
<b>ELU</b>	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.  $f_\theta$  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 8 we see how different activation functions in the last layer of  $f_\theta$  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 9 shows how different activation functions shape the vector field and as a result the decision boundary.

### B.3 Regularization for Stability

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

$$\ell_{\text{reg}} = \int_S \|f_{\theta(\tau)}(\tau, \mathbf{x}, \mathbf{z}(\tau))\|_2 d\tau, \quad (20)$$

to achieve stability. A simple alternative stabilizing regularization term can be considered at no significant additional computational cost:

$$\ell_{\text{reg}} = \|f_{\theta(S)}(S, \mathbf{x}, \mathbf{z}(S))\|_2, \quad (21)$$

which penalizes non-convergence to some fixed point of  $f_\theta$  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

$$\begin{bmatrix} \dot{\mathbf{z}}_x \\ \dot{z}_a \end{bmatrix} = \begin{bmatrix} 0_{n_x} \\ f_\theta(\mathbf{z}_x) \end{bmatrix}, \quad \begin{bmatrix} \mathbf{z}_x(0) \\ z_a(0) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}, \quad s \in [0, 1] \quad (22)$$

where the output is picked as  $\hat{y} := z_a(1)$ , can approximate any function  $\Psi : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$  provided that the neural network  $f_\theta(\mathbf{x})$  is an approximator of  $\Psi$ , 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 Example Implementation of Data–Control

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

```
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 embeddings of input data `x` i.e. g(x)
        # 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  $f_\theta$ , 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  $f_\theta : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \rightarrow \mathbb{R}^{n_z}$  instead of  $f_\theta : \mathbb{R}^{n_a} \times \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_a} \times \mathbb{R}^{n_x}$  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 two NVIDIA® TITAN RTX GPUs with CUDA 10.1 and INTEL® i9 10980XE CPU. All Neural ODEs were trained on GPU. The code was built upon Pytorch’s `torchdyn` library for neural differential equations (Poli et al., 2020b).

**General experimental setup** We report here general information about the experiments. All Neural ODEs are solved numerically via the *Dormand–Prince* method (Prince and Dormand, 1981). 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 Galerkin 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  $\varphi : \mathbb{R}^n \rightarrow \mathbb{Z}$

$$\varphi(\mathbf{x}) = \begin{cases} -1 & \|\mathbf{x}\|_2 < r \\ 1 & \|\mathbf{x}\|_2 \geq r \end{cases}. \quad (23)$$

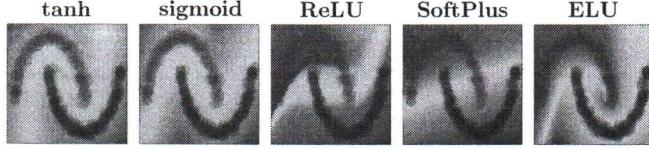


Figure 9: Decision boundaries learned by the vector field of a Neural ODE are directly conditioned by the choice of activation function.

We consider learning the map  $\varphi(\mathbf{x})$  with Neural ODEs prepending a linear layer  $\mathbb{R}^n \rightarrow \mathbb{R}$ . Notice that  $\varphi$  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  $\varphi$ .

### 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 Galérkin 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 Galérkin 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  $5 \cdot 10^{-3}$ . Galérkin 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.

**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

**Image classification** We use *AdamW* with learning rate  $10^{-3}$ , batch size 64, weight decay  $5 \cdot 10^{-4}$  and a learning rate step schedule with multiplicative factor  $\gamma = 0.9$  every 5 epochs. We train each model for 20 epochs. The vector fields  $f_\theta$  are parametrized by 3-layer depth-invariant CNNs, with each layer followed by an instance normalization layer. 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. We note that the results of this ablation analysis do not utilize any form of data augmentation; data augmentation can indeed be introduced to further improve performance.

For input layer augmented Neural ODE models, namely IL-NODE and 2nd order, we prepend to the Neural ODE a single, linear CNN layer. In the case of 2nd order models, we use input layer augmentation for the positions and initialize the velocities at 0. The hidden channel dimension of the CNN parametrizing  $f_\theta$  in augmented models is set to 32 on MNIST and 42 on CIFAR; vanilla Neural ODEs, on the other hand, are equipped with dimensions 42 and 62 for a fair comparison. The output class probabilities are then computed by mapping the output of the Neural ODE through average pooling followed by a linear layer. Second order Neural ODEs, 2nd, use  $f_\theta$  to compute the vector

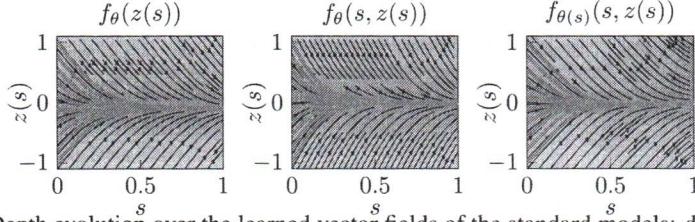


Figure 10: Depth evolution over the learned vector fields of the standard models: depth-invariant and depth-variant (“concat”  $f_\theta(s, z(s))$  and GalNODE  $f_\theta(s, z(s))$ ). As expected the Neural ODE cannot approximate the map  $\varphi(x) = -x$ .

field of velocities: therefore, the output of  $f_\theta$  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  $\mathbf{z}$ .

We note that 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  $f_\theta$ , which employed an excessive number of channels inside  $f_\theta$ , i.e 92.

### 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. We trained vanilla Neural ODEs, i.e. both depth-invariant and depth variant models (“concat” and GalNODE). As expected, these models cannot approximate  $\varphi$ . 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 series of five modes. The resulting trajectories over the learned vector fields are shown in Fig. 10.

**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  $\epsilon$ , 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 10 and 11. 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.

**Concentric annuli with non-augmented variants** We train each model for 1024 iterations using AdamW with learning rate  $10^{-3}$ , weight decay  $10^{-6}$  and batch size 1024. All models have a single hidden layer of dimension 32. The GalNODE layer is parametrized by a Fourier series of five modes.

**Conditional continuous normalizing flows** We train data-controlled continuous normalizing flows for 2000 iterations with samples of size  $2^{14}$ . We use AdamW with learning rate  $10^{-3}$  and weight decay  $10^{-7}$ . Absolute and relative tolerances of the chosen solver, *dopri5* are set to  $10^{-8}$ . The CNF network have 2 hidden layers of dimension 128 with softplus nonlinearities.

**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

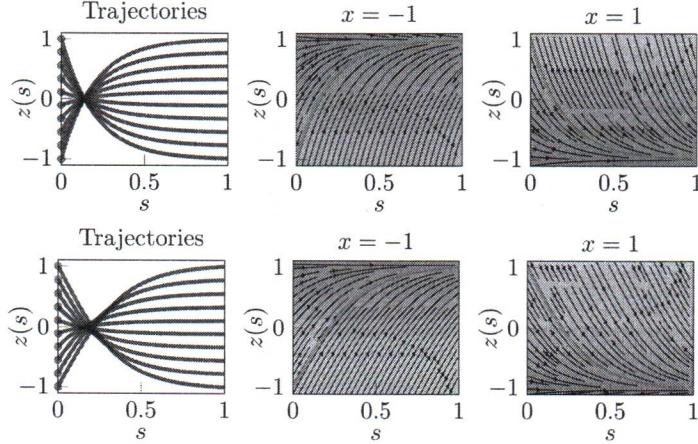


Figure 11: Depth evolution over the learned vector fields of (9) and a data-controlled Neural ODE. As discussed in Sec.5 introducing data-control allows the model to approximate the map  $\varphi(x) = -x$ .

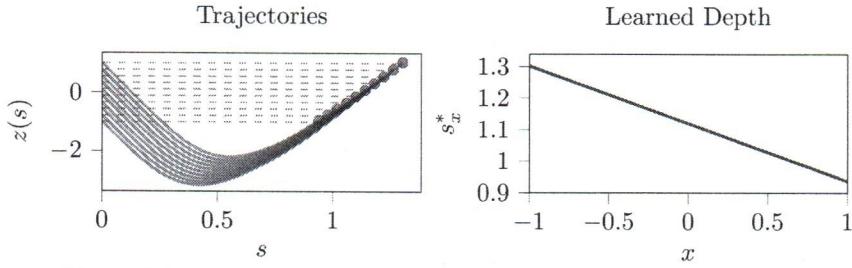


Figure 12: Evolution of the input data through the depth of the Neural ODEs

activation) whose output is summed to one and then taken in absolute value,

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

where  $\sigma$  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. 12. This early result should be intended as a proof of concept rather than a definitive evaluation of the depth adaptation methods, which we reserve for future work. We note that the result of Fig. 5 showed in the main text has been obtained by training the model only on  $x \in \{-1, 1\}$  and manually setting  $s_{-1}^* = 1, s_1^* = 3$ .