# A Shooting Formulation of Deep Learning

François-Xavier Vialard* Roland Kwitt† Susan Wei‡ Marc Niethammer§

June 19, 2020

**Abstract**

Continuous-depth neural networks can be viewed as deep limits of discrete neural networks whose dynamics resemble a discretization of an ordinary differential equation (ODE). Although important steps have been taken to realize the advantages of such continuous formulations, most current techniques are not truly continuous-depth as they assume *identical* layers. Indeed, existing works throw into relief the myriad difficulties presented by an infinite-dimensional parameter space in learning a continuous-depth neural ODE. To this end, we introduce a shooting formulation which shifts the perspective from parameterizing a network layer-by-layer to parameterizing over *optimal* networks described only by a set of *initial conditions*. For scalability, we propose a novel particle-ensemble parametrization which fully specifies the optimal weight trajectory of the continuous-depth neural network. Our experiments show that our particle-ensemble shooting formulation can achieve competitive performance, especially on long-range forecasting tasks. Finally, though the current work is inspired by continuous-depth neural networks, the particle-ensemble shooting formulation also applies to discrete-time networks and may lead to a new fertile area of research in deep learning parametrization.

## 1 Introduction

Deep neural networks (DNN) are closely related to optimal control (OC), a well-established field with both theoretical and practical insights on offer [25, 24, 20]. The sought-for control variable of DNNs corresponds to the parameters of neural network layers. To be able to talk about an *optimal* control requires the definition of a control cost, i.e., a norm on the control variable. We explore the ramifications of such a control cost in the context of DNN parametrization. For simplicity, we focus on continuous formulations in the spirit of neural ODEs [14]. However, both discrete and continuous OC formulations exist [13, 5, 38] and our approach could also be developed for the former.

Initial work on continuous DNN formulations was motivated by the realization that a `ResNet` [21, 22] resembles Euler forward time-integration [20, 24]. Specifically, the forward pass of some input vector $\tilde{\mathbf{x}}$ through a network with $L$ layers specified as, $\mathbf{x}(0) = \tilde{\mathbf{x}}$ and $\mathbf{x}(j+1) = \mathbf{x}(j) + f(\mathbf{x}(j), \theta(j))$, $j = 0, 1, \ldots, L$, closely relates to an explicit Euler [37] discretization of the ODE

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \theta(t)), \quad \mathbf{x}(0) = \tilde{\mathbf{x}}, \quad 0 \leq t \leq T \ . \tag{1.1}$$

In the ODE-inspired DNN setting, we seek an optimal $\theta$ such that the terminal prediction given by $\mathbf{x}(T)$, i.e., the solution to Eq. (1.1) at time $T$, minimizes $\ell(\mathbf{x}(T))$ for a task-specific loss function $\ell$.

*LIGM, UPEM, `francois-xavier.vialard@u-pem.fr`
†Department of Computer Science University of Salzburg; `Roland.Kwitt@sbg.ac.at`
‡School of Mathematics and Statistics University of Melbourne; `susan.wei@unimelb.edu.au`
§Department of Computer Science University of North Carolina at Chapel Hill; `mn@cs.unc.edu`

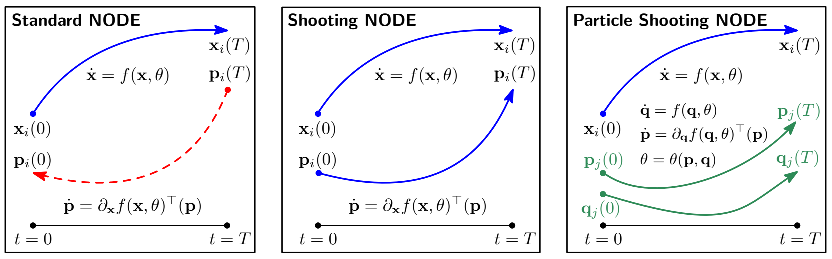

Standard NODE
$\mathbf{x}_i(T)$
$\dot{\mathbf{x}} = f(\mathbf{x}, \theta)$
$\mathbf{p}_i(T)$
$\mathbf{x}_i(0)$
$\mathbf{p}_i(0)$
$\dot{\mathbf{p}} = \partial_{\mathbf{x}} f(\mathbf{x}, \theta)^\top(\mathbf{p})$
$t = 0$
$t = T$
Shooting NODE
$\mathbf{x}_i(T)$
$\dot{\mathbf{x}} = f(\mathbf{x}, \theta)$
$\mathbf{p}_i(T)$
$\mathbf{x}_i(0)$
$\mathbf{p}_i(0)$
$\dot{\mathbf{p}} = \partial_{\mathbf{x}} f(\mathbf{x}, \theta)^\top(\mathbf{p})$
$t = 0$
$t = T$
Particle Shooting NODE
$\mathbf{x}_i(T)$
$\dot{\mathbf{x}} = f(\mathbf{x}, \theta)$
$\dot{\mathbf{q}} = f(\mathbf{q}, \theta)$
$\mathbf{p}_j(T)$
$\mathbf{x}_i(0)$
$\dot{\mathbf{p}} = \partial_{\mathbf{q}} f(\mathbf{q}, \theta)^\top(\mathbf{p})$
$\mathbf{p}_j(0)$
$\theta = \theta(\mathbf{p}, \mathbf{q})$
$\mathbf{q}_j(T)$
$\mathbf{q}_j(0)$
$t = 0$
$t = T$

**Figure 1:** Optimization in the neural ODE (NODE) framework of [14] (left) amounts to a forward pass with the gradient computed via backpropagation. Optimization under the shooting principle (middle) turns the forward-backward system into a forward *second-order* system, where we essentially run the backpropagation equation forward. We use a Hamiltonian particle ensemble (right) consisting of (position, momentum) pairs $(\mathbf{q}_j, \mathbf{p}_j)$ to make the shooting efficient. In shooting $\theta$ is time-dependent whereas in NODE $\theta(t) = \theta\ \forall t$.

Although Eq. (1.1) with time-varying parameter $\theta(t)$ can be considered as a neural network with an infinite number of layers, current implementations of ODE-inspired networks largely assume parameters $\theta$ that are fixed in time [14, 15], i.e., $\forall t : \theta(t) = \theta$, or that follow a designed, prescribed dynamics [44]. Instead, we explore time-varying $\theta(t)$. We employ regularization (i.e., a control cost) to render estimating $\theta(t)$ well-posed and to assure regularity of the resulting flow. Hence, we explore the regularized loss

$$\mathcal{E}(\theta) = \int_0^T R(\theta(t))\, \mathrm{d}t + \lambda\, \ell(\mathbf{x}(T)), \quad \lambda \in \mathbb{R}^+, \quad \text{subject to Eq. (1.1)}\ , \tag{1.2}$$

where $R : L^2([0,T], \mathbb{R}^d) \to \mathbb{R}$ is a complexity measure of $\theta$ corresponding to the control cost.

Instead of *directly* optimizing over the set of time-dependent $\theta(t)$ as in standard `ResNets`, *we restrict the optimization set to those $\theta$ which are critical points of $\mathcal{E}(\theta)$, thereby dramatically reducing the number of parameters.* In doing so, one can describe the optimization task as an *initial value problem.* Namely, we show that we can rewrite the loss in Eq. (1.2) solely in terms of the input $\mathbf{x}(0)$ and a corresponding finite-dimensional momentum variable, $\mathbf{p}(0)$. Such an approach, just like optimizing the initial speed of a mass particle to reach a given point, is called a *shooting method* in numerical analysis [31] and control [11], giving its name to our new formulation.

The first two panels of Fig. 1 illustrate the difference between the optimization of a neural ODE (NODE) via [14] and our shooting formulation. Since in practice, we have multiple inputs $\tilde{\mathbf{x}}_i, i = 1, \dots, n$, there is an initial momentum vector $\mathbf{p}_i$ corresponding to each of them. If the shooting formulation is to scale up to a large sample size $n$, we must take care that the parametrization does not grow linearly with $n$. To this end, we propose what we call the *Hamiltonian particle-ensemble parametrization.* It is a finite set of particles, where each particle is a (position, momentum) pair. The initial conditions of these particle pairs $\{(\mathbf{q}_j, \mathbf{p}_j)\}_{j=1}^K$ (where $K \ll n$) completely determine $\theta(t)$. This is illustrated in the rightmost panel of Fig. 1. Once the optimized set of particles has been computed, the computational efficiency of the forward model, similarly to NODE [14], is retained for vector fields $f$ that are linear in their parameters $\theta(t)$.

Our **contributions** are as follows: 1) We introduce a shooting formulation for DNNs, amounting to an initial-value formulation for neural network parametrization. This allows for optimization over the original network parameter space via optimizing over the initial conditions of critical networks only; 2) We propose an efficient implementation of the shooting approach based on a novel particle-ensemble parametrization in which a set of initial particles (the (position, momentum) pairs) describe the space of putative optimal network parameters; 3) We propose the `UpDown` model which gives rise to explicit shooting equations; 4) We prove universality for the flows of the `UpDown` vector field and demonstrate in experiments its good performance on several prediction tasks.

## 2 Related work

We draw inspiration from two separate branches of research: 1) continuous formulations of neural networks [14] and 2) shooting approaches for deformable image registration [2, 28, 30].

**Continuous-depth neural networks.** Continuous equivalents of `ResNets` [21, 22] have been developed in [33, 20], but naïve implementations are memory-demanding since backpropagation requires differentiating through the numerical integrator. Two approaches can address this unfavorable memory footprint. Neural ODE [14] does not store intermediate values in the forward pass, but recomputes them by integrating the forward model backward. This is easily possible only if the forward model is numerically invertible and the formulation is time-continuous [18][1]. Instead, checkpointing [18] is a general approach to reduce memory requirements by selectively recomputing parts of the forward solution [19]. Our work can easily be combined with these numerical approaches.

**Solving implicit equations.** A recent line of works, including deep equilibrium models [7] and implicit residual networks [32], has shown that it may not always be necessary to freely parametrize all the layers in the network. Specifically, in [7] and [32], the parameters of each layer are defined via an implicit equation motivated by *weight tying* thus improving expressiveness and reducing the number of parameters while decreasing the memory footprint via implicit differentiation. Instead, our work increases expressiveness and reduces the number of parameters via particle-based shooting.

**Invertibility and expressiveness.** Based on similarity with continuous time integration, constraining the norm of a layer in a `ResNet` will result in an invertible network such as in [9, 23]. Invertibility is also explored in [42], where it is enforced (as in our setting) via a penalty of the norm. These works show that standard learning tasks can be performed on top of a one-to-one transformation. Recent theoretical developments [43] show that indeed capping a NODE or i-ResNet [9] with a single linear layer gives universal approximation for non-invertible continuous functions. Further, expressiveness can be increased by moving to more complex models, e.g., by introducing additional dimensions as explored in augmented NODE [15]. In [44] (`AnodeV2`), Zhang et al. treat time-dependent $\theta(t)$. Weights are evolved jointly with the state of the continuous DNN. While this weight evolution could, in principle, also be captured by a learned weight network, the authors argue that this would result in a large increase in parameters and therefore opt for explicitly parametrizing these evolutions (e.g., via a reaction diffusion equation). In contrast, our method does not rely on learning a separate weight-network or on explicitly specifying a weight evolution. Instead, our evolving weights are a direct consequence of the shooting equations which, in turn, are a direct consequence of penalizing network parameters (the control cost) over time; a large increase in parameters does not occur.

**Hamiltonian approaches.** Toth et al. [36] proposed Hamiltonian generative networks to learn the Hamiltonian governing the evolution of a physical system. Specifically, they learn Hamiltonian vector fields in the latent space of an image encoder-decoder architecture. Sæmundsson et al. [34] also learn the underlying dynamics of a system from time-dependent data, starting from a discrete Lagrangian combined with a variational integrator. This motivates particular network structures; e.g., Newtonian networks where the potential energy

[1]In a discrete setting, resolving the forward model in the backward direction generally requires costly solving of implicit equations. This can be done (it is, e.g., done for invertible `ResNets` [9]). In general, an explicit numerical solution for forward time-integration becomes implicit in the backward direction and vice versa.

is learned via a neural network. Although sharing common tools, our work completely differs from this line of research in the sense that we exploit Hamiltonian mechanics to parametrize *general* continuous neural networks. In principle, our work applies to most network architectures and is not specific to physical data.

Finally, we mention that shooting approaches have been applied successfully in other areas such as diffeomorphic image matching [28, 2, 30]. However, the decisive difference here is in the dimensionality of the underlying space: in image registration it is typically a 3D image; for DNNs, it is usually a high-dimensional space ($\mathbb{R}^d$) with data sampled from a distribution in that space.

## 3 Shooting formulation of ODE-inspired neural networks

We consider, for simplicity, a supervised learning task where the input and target spaces are $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y}$, resp., and sampled data are denoted by $\{(\tilde{\mathbf{x}}_i, \tilde{\mathbf{y}}_i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$. The goal is to learn the weight $\theta(t)$ in the following flow equation

$$\dot{\mathbf{x}}_i(t) = f(\mathbf{x}_i(t), \theta(t)), \quad \mathbf{x}_i(0) = \tilde{\mathbf{x}}_i, \quad 0 \leq t \leq T \tag{3.1}$$

such that it minimizes the loss $\sum_{i=1}^n \ell(\mathbf{x}_i(T), \tilde{\mathbf{y}}_i)$ for some loss function $\ell$. In existing works, the weight is chosen independent of time, i.e., $\theta(t) = \theta$ [14], or specific evolution equations are postulated for it [26, 44]. Such strategies show the difficulty of addressing infinite dimensional parametrizations of time-dependent $\theta$ and the need for regularization for well-posedness [17, 26, 20]. Instead of parametrizing $\theta(t)$ directly, we aim at penalizing $\theta(t)$ according to the regularity of $f(\cdot, \theta(t))$ to arrive at a well-posed problem. Specifically, we consider a regularization term $R(\theta(t))$ (discussed in §3.1) and propose to minimize, for $\lambda > 0$,

$$\mathcal{E}_n(\theta) = \int_0^T R(\theta(t))\ \mathrm{d}t + \lambda \sum_{i=1}^n \ell(\mathbf{x}_i(T), \tilde{\mathbf{y}}_i) \quad \text{subject to Eq. (3.1)}\ . \tag{3.2}$$

Note that upon discretizing the time $t$ (i.e., having a number of parameters proportional to the number of timesteps) this is similar to a `ResNet` with weight decay. For a `ResNet` or a NODE, optimization is based on computing the parameter gradient via a forward pass followed by backpropagation (see left panel of Fig. 1).

**Optimality equations.** The optimality conditions for Eq. (3.2) in continuous time are:

$$\begin{cases} \dot{\mathbf{x}}_i(t) - f(\mathbf{x}_i(t), \theta(t)) = 0,\ \mathbf{x}_i(0) = \tilde{\mathbf{x}}_i, & \text{Data evolution} \\ \dot{\mathbf{p}}_i(t) + \partial_{\mathbf{x}} f(\mathbf{x}(t), \theta(t))^\top(\mathbf{p}_i) = 0,\ \mathbf{p}_i(T) = -\nabla\ell(\mathbf{x}_i(T), \tilde{\mathbf{y}}_i), & \text{Adjoint evolution} \\ \partial_\theta R(\theta(t)) - \sum_{i=1}^n \partial_\theta f(\mathbf{x}_i(t), \theta(t))^\top(\mathbf{p}_i(t)) = 0\,. & \text{Compatibility} \end{cases} \tag{3.3}$$

The first equation is the forward model and the second equation is the adjoint equation solved backward in time in order to compute the gradient with respect to the parameters. At convergence, the third equation is also satisfied. This last equation encodes the optimality of the layer at timestep $t$, as it is the case for an *argmin layer* or *weight tying* [3]. Its left hand side corresponds to the gradient with respect to the parameter $\theta$, but as we shall see it will allow us to compute $\theta$ directly via our (position, momentum) pairs in our particle shooting formulation. The shooting approach simply replaces the optimization set by the set of critical points of Eq. (3.2) expressed in these optimality conditions. That is, we only optimize over solutions fulfilling Eq. (3.3).

**Shooting principle.** The shooting method is standard in optimal control [11] and can be formulated as follows: Since, at optimality, the system in Eq. (3.3) is satisfied, one can *turn this system into a forward model* defined only by its initial conditions $\{(\mathbf{x}_i(0), \mathbf{p}_i(0))\}_{i=1}^n$ which specify the *entire trajectory* of optimal parameters. We refer to the forward model defined by Eq. (3.3) as the *shooting equations*. Unfortunately, this initial-condition parametrization still requires *all* initial conditions $\mathbf{x}_i(0)$ and their corresponding momenta $\mathbf{p}_i(0)$ for $i = 1, \ldots, n$. Since this does not scale to very large datasets, we propose an approximation using a collection of particles, as described next.

**Hamiltonian particle ensemble.** In the limit and ideal case where the data distribution is known, the optimality equations can be approximated using a collection of particles which follow the Hamiltonian system (see Appendix A). We thus consider a collection of particles $(\mathbf{q}_j, \mathbf{p}_j)_{j=1,\ldots,K} \in \mathbb{R}^d \times \mathbb{R}^d$ that drive the evolution of the entire population $(\mathbf{x}_i)_{i=1,\ldots,n} \in \mathbb{R}^d$ through the following forward model

$$\begin{cases} \dot{\mathbf{x}}_i(t) - f(\mathbf{x}_i(t), \theta(t)) = 0,\ \mathbf{x}_i(0) = \tilde{\mathbf{x}}_i & \text{Data evolution} \\ \left.\begin{aligned} &\dot{\mathbf{q}}_j(t) - f(\mathbf{q}_j(t), \theta(t)) = 0, \\ &\dot{\mathbf{p}}_j(t) + \partial_{\mathbf{x}} f(\mathbf{q}_j(t), \theta(t))^\top (\mathbf{p}_j(t)) = 0, \\ &\partial_\theta R(\theta(t)) - \sum_{j=1}^{K} \partial_\theta f(\mathbf{q}_j(t), \theta(t))^\top (\mathbf{p}_j(t)) = 0\,, \end{aligned}\right\} & \text{Hamiltonian equations} \end{cases} \tag{3.4}$$

with initial conditions $\{(\mathbf{q}_j(0), \mathbf{p}_j(0))\}_{j=1,\ldots,K}$, where the gradient with respect to to this new parametrization is computed via backpropagation. This set of (position, momentum) pairs is termed the *Hamiltonian particle ensemble*. As the number of particles is reduced, so are the number of free parameters, see Appendix B. Indeed, varying the Hamiltonian particle ensemble allows for controlling the tradeoff between reconstruction and network complexity.

## 3.1 Choices of regularization, parametrization and conserved quantities

The main computational bottleneck in the forward model of Eq. (3.4) is the implicit parametrization of $\theta$ by the last equation. Making it explicit is key to render shooting computationally tractable.

**Linear in parameter - quadratic penalty.** In the simplest case, the space of functions $f$ is a linear space parametrized by $\theta(t)$. In this case quadratic penalty amounts to a kinetic penalty. Specifically, as a motivating example, consider the forward model (with component-wise activation function $\sigma$)

$$f(\mathbf{x}(t), \theta(t)) = \mathbf{A}(t)\sigma(\mathbf{x}(t)) + \mathbf{b}(t) \tag{3.5}$$

with $\mathbf{A} \in L^2([0,1], \mathbb{R}^{d^2})$, $\mathbf{b} \in L^2([0,1], \mathbb{R}^d)$ and $\theta(t) = [\mathbf{A}(t), \mathbf{b}(t)]$. With the quadratic regularizer $R(\theta(t)) = \frac{1}{2}\operatorname{Tr}\left(\mathbf{A}(t)^\top \mathbf{M_A} \mathbf{A}(t)\right) + \frac{1}{2}\mathbf{b}(t)^\top \mathbf{M_b} \mathbf{b}(t)$, where $\mathbf{M_A}$, $\mathbf{M_b}$ are positive definite matrices, the particle shooting equations are

$$\begin{cases} \dot{\mathbf{q}}_j(t) &= \mathbf{A}(t)\sigma(\mathbf{q}_j(t)) + \mathbf{b}(t), \\ \dot{\mathbf{p}}_j(t) &= -\,\mathrm{d}\sigma(\mathbf{q}_j(t))^\top \mathbf{A}(t)^\top \mathbf{p}_j(t), \end{cases} \qquad \begin{cases} \mathbf{A}(t) &= \mathbf{M_A}^{-1}(-\sum_{j=1}^{K} \mathbf{p}_j(t)\sigma(\mathbf{q}_j(t))^\top) \\ \mathbf{b}(t) &= \mathbf{M_b}^{-1}(-\sum_{j=1}^{K} \mathbf{p}_j(t))\,, \end{cases} \tag{3.6}$$

with given initial conditions $(\mathbf{p}_j(0), \mathbf{q}_j(0))$. We emphasize that $\theta(t)$ is *explicitly defined* by $\{(\mathbf{p}_j(t), \mathbf{q}_j(t))\}_{j=1}^{K}$ and the computational cost is reduced to matrix multiplications.

As is well-known [4], the Hamiltonian flow preserves the Hamiltonian function. In the "linear in parameter - quadratic regularization" case, this preserved quantity, denoted $H(\mathbf{p}(t), \mathbf{q}(t))$, equals to $R(\theta(t))$, which is the kinetic energy of the system of particles. As a first consequence, the objective functional can be rewritten as, $H(\mathbf{p}(0), \mathbf{q}(0)) + \lambda \sum_{i=1}^{n} \ell(\mathbf{x}_i(T), \tilde{\mathbf{y}}_i)$ . This clearly allows for direct optimization on $(\mathbf{p}(0), \mathbf{q}(0))$, i.e., shooting. As a second consequence, since the vector field has constant norm (its squared norm is the kinetic energy), it gives a quantitative bound on the regularity of the flow map at time $t = T$ explicit in terms of $H(\mathbf{p}(0), \mathbf{q}(0))$. In addition (Appendix A), the Rademacher complexity of the generated flows with bounded $H(\mathbf{p}(0), \mathbf{q}(0)))$ can also be controlled.

**Nonlinear in parameter and non-quadratic penalty.** A standard `ResNet` structure uses vector fields of the type (in convolutional form or not)

$$f(\mathbf{x}(t), \theta(t)) = \theta_1(t)\sigma(\theta_2(t)\mathbf{x}(t) + b_2(t)) + b_1(t) \ , \tag{3.7}$$

where $\theta_1(t) \in L(\mathbb{R}^n, \mathbb{R}^d)$ and $\theta_2(t) \in L(\mathbb{R}^d, \mathbb{R}^n)$. This forward model can also be handled in our shooting approach since the shooting equations in Eq. (3.3) are completely specified by the Hamiltonian $H(\mathbf{p}, \mathbf{x}, \theta) = R(\theta) - \mathbf{p}^\top f(\mathbf{x}, \theta)$. Automatic differentiation can be used (see Appendix C) to implement the forward model

$$\dot{\mathbf{x}}(t) = \frac{\partial H}{\partial \mathbf{p}}(\mathbf{p}(t), \mathbf{x}(t), \theta(t)), \ \dot{\mathbf{p}}(t) = -\frac{\partial H}{\partial \mathbf{x}}(\mathbf{p}(t), \mathbf{x}(t), \theta(t)), \ \theta(t) \in \arg\min H(\mathbf{p}(t), \mathbf{x}(t), \theta(t)). \tag{3.8}$$

Important bottlenecks appear since the third equation is nonlinear and potentially associated with a non-convex optimization problem. This could be addressed by unrolling the optimization corresponding to the last equation, resulting in increased computational cost. In addition, in this nonlinear case, the Hamiltonian function is no longer (in general) equal to $R(\theta(t))$ even in the quadratic regularization setting. Therefore, results on the smoothness or Rademacher complexity would no longer be guaranteed as for the linear - quadratic penalty case. Last, quadratic regularization has no known theoretical results for the Rademacher complexity of functions generated by Eq. (3.7) with bounded norm. Norms for which the Rademacher complexity of this class of functions is known [16] to be bounded are called Barron norms, which are non-smooth and non-convex, and which would add to the difficulty. To circumvent these issues while retaining expressiveness and theoretical guarantees in the linear parametrization setting, we next introduce the `UpDown` model.

### 3.2 The `UpDown` model

The key idea is to transform the vector field of Eq. (3.7) into a model which is linear in parameters on which the quadratic regularization can be applied. To this end, we introduce the additional state $\mathbf{v}(t) = \theta_2(t)\mathbf{x}(t) + b_2(t)$ which we differentiate with respect to time. We obtain $\dot{\mathbf{v}}(t) = \dot{\theta}_2(t)\mathbf{x}(t) + \dot{b}_2(t) + \theta_2(t)\dot{\mathbf{x}}(t)$ and replacing $\dot{\mathbf{x}}(t)$ by its formula, we get $\dot{\mathbf{v}}(t) = \dot{\theta}_2(t)\mathbf{x}(t) + \dot{b}_2(t) + \theta_2(t)(\theta_1(t)\sigma(\mathbf{v}(t)) + b_1(t))$. Thus, using the additional state variable $\mathbf{v}(t)$, we propose the following ODE system, denoted the `UpDown` model, introducing the (integer-valued) inflation factor $\alpha \geq 1$,

$$\dot{\mathbf{x}}(t) = \theta_1(t)\sigma(\mathbf{v}(t)) + b_1(t), \quad \dot{\mathbf{v}} = \theta_2(t)\mathbf{x}(t) + b_2(t) + \theta_3(t)\sigma(\mathbf{v}(t)) \, , \tag{3.9}$$

with $\mathbf{x}(t) \in \mathbb{R}^d$, $\mathbf{v}(t) \in \mathbb{R}^{\alpha d}$. For the data evolution, the initial conditions for $\mathbf{x}(t)$ are given by the data samples $\{\tilde{\mathbf{x}}_i\}$. We parametrize the initial conditions for $\mathbf{v}(t)$ using an affine map via $\mathbf{v}(0) = \Theta_{12}(\mathbf{x}(0)) + \mathbf{b}_{12}$, where $\Theta_{12} \in L(\mathbb{R}^d, \mathbb{R}^{\alpha d})$, $\mathbf{b}_{12} \in L(\mathbb{R}^{\alpha d})$. In Appendix D, we prove the following theorem:

**Theorem 1.** *Given the flow of a time-dependent vector field defined on a compact domain $C$ of $\mathbb{R}^d$, which is time continuous and Lipschitz, we denote by $\varphi(T, \mathbf{x}(0))$ its flow at time $T$ from starting value $\mathbf{x}(0)$. Then, there exists a parametrization of the `UpDown` model for which its solution is $\varepsilon$-close to the flow, $\sup_{\mathbf{x}(0) \in C} \|\varphi(T, \mathbf{x}(0)) - \mathbf{x}(T)\| \leq \varepsilon$.*

Notably, in the proof, the dimension of the hidden state $\mathbf{v}$ is used twice: *first*, for having a sufficient number of neurons in Eq. (3.7) to approximate a stationary vector field (standard universality property of multilayer perceptron) and, *second*, for approximating time-dependent vector fields. Therefore, at the cost of introducing a possibly large number of dimensions, the `UpDown` model is universal in the class of time-dependent NODEs. As shown in Appendix D, this universality result transfers to our shooting formulation. Due to its additional dimensions, it is also likely to be universal in the space of functions (i.e., not necessarily injective). We focus on the `UpDown` model in our experiments. Note also that while we derived our theory for vector-valued evolutions for simplicity, similar linear in parameter evolution equations can for example be derived for convolutional neural networks.

## 4 Experiments

Our goal is to demonstrate that it is possible to learn DNNs by optimizing only over the initial conditions of *critical* networks. This is made possible via shooting and efficient via our particle parametrization. A key difference to prior work is that our approach allows to capture time-dependent (i.e., layer-dependent in the discrete setting) parameters *without* discretizing these parameters at every time-point. Comparisons to other neural-ODE like methods are not straightforward as hyper-parameters and different implementation may make results difficult to compare. For consistency, we provide four different formulations (based on the `UpDown` model of §3.2).

- The **static direct** model foregoes the Hamiltonian particle ensemble, and instead directly optimizes over *time-constant* parameters: $\theta(t) = \theta$. Everything else, including the `UpDown` model, stays unchanged. This model is most closely related to NODE [14] and augmented NODE [15].

- The **static with particles** model is similar to the *static direct* model. However, instead of directly optimizing over a *time-constant* $\theta$, it uses a set of (position, momentum) pairs (i.e., particles, as in our *dynamic with particles* model below) to parametrize $\theta$ indirectly.

- We call our proposed shooting model **dynamic with particles**. It is parametrized via a set of initial conditions of (position, momentum) pairs, which evolve over time and fully specify $\theta(t)$.

- Finally, we consider the **dynamic direct** model which uses a piece-wise time-constant $\theta(t)$. It essentially chains together multiple *static direct* models and is closely related to a discrete `ResNet` in the sense that multiple blocks (we use five) are used in succession. However, each block involves time-integrating the `UpDown` model. While the *dynamic with particles* model captures $\theta(t)$ indirectly via particles and shooting, the *dynamic direct* model requires many more parameters as it represents $\theta(t)$ directly. We show results for the *dynamic direct* model for a subset of the experiments.

All experiments use the `UpDown` model with quadratic penalty function $R$. Detailed experimental settings, including weights for the quadratic penalty function, can be found in Appendix E.

**Simple 1D function regression.** We approximate a simple quadratic-like function $y = x^2 + 3/(1 + x^2)$ which is non-invertible. We use 15 particles for our experiments. Fig. 2 shows the test loss and the network complexity, as measured by the log Frobenius norm integrated over time [29], for the different models as a function of the inflation factor $\alpha$ (cf. §3.2). On average, the *dynamic with particles* model shows the best fits with the lowest complexity measures, indicating the simplest network parametrization. Note that the *static with particles* approach results in the lowest complexity measures only because it cannot properly fit the function as indicated by the high test loss. Additional results for a cubic function $y = x^3$ are in Appendix F.

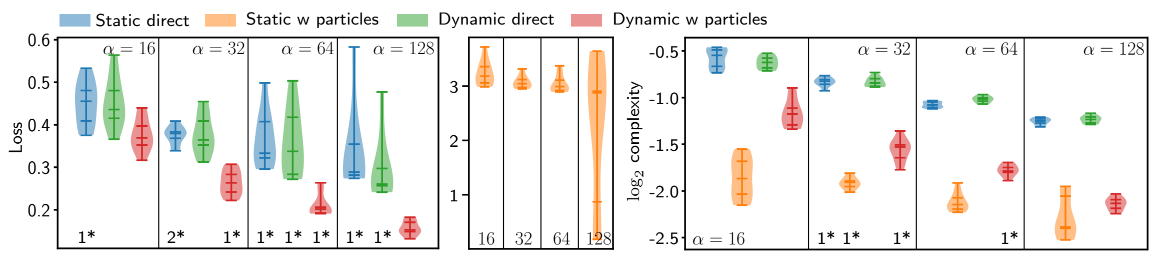
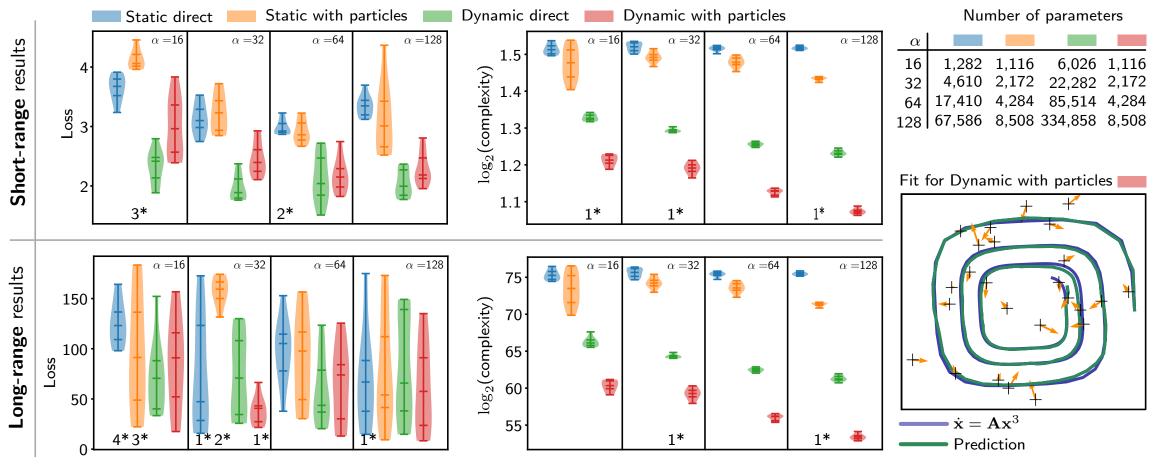


**Figure 2:** Fit for quadratic-like $y = x^2 + 3/(1 + x^2)$ for 10 random initializations. *Left*: Test loss; *Right*: time-integral of $\log_2$ of the Frobenius norm complexity. Lower is better for both measures. * indicates number of removed outliers (outside the interquartile range (IQR) by $\geq 1.5\times$ IQR); $\alpha$ denotes the inflation factor.



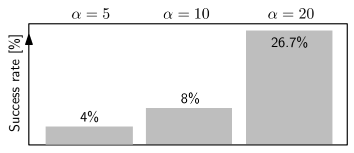
Number of parameters

| $\alpha$ | 🟦 | 🟧 | 🟩 | 🟥 |
|---|---|---|---|---|
| 16 | 1,282 | 1,116 | 6,026 | 1,116 |
| 32 | 4,610 | 2,172 | 22,282 | 2,172 |
| 64 | 17,410 | 4,284 | 85,514 | 4,284 |
| 128 | 67,586 | 8,508 | 334,858 | 8,508 |

**Figure 3:** Fit for spiral (short- and long-range). Losses for the different models as well as the time-integral of $\log_2$ of the Frobenius norm complexity measure. Lower is better for both measures. The * symbol indicates how many outliers were removed and $\alpha$ denotes the inflation factor.

**Spiral.** Next, we revisit the spiral ODE example of [14] following the nonlinear dynamics $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}^3$, $\mathbf{x} \in \mathbb{R}^2$ (where the power is component-wise). We fix $\mathbf{x}(0) = [2, 0]^T$, use $A = [-0.1, 2.0; -2, -0.1]$ and evolve the dynamics for time $T = 10$. The training data consists of snippets from this trajectory, all of the same length. We use an $L^2$ norm loss (calculated on all intermediate time-points) and 15 particles. Our goal is to show that we can obtain the best fit to the training data due to our dynamic model. Fig. 3 (*top*) shows that we can indeed obtain similar or better fits (lower losses) for a similar number of parameters while achieving the lowest network complexity measures. Fig. 3 (*bottom*) shows the corresponding results for the validation data consisting of the original long trajectory starting from initial value $\mathbf{x}(0)$. Interestingly, by pasting together short-range solutions we are successful in predicting the long-range trajectory despite training on short-range trajectory snippets.

**Concentric circles.** To study the impact of the inflation factor $\alpha$ in a classification regime, we replicate the concentric circles setting of [15]. The task is learning to separate points, sampled from two disjoint annuli in $\mathbb{R}^2$. While we are less interested in the learned flow (as in [15]), we study how often the proposed `UpDown` (dynamic with particles) model perfectly fits the training data as a function of $\alpha$. To the right, we show the success rate over 50 training runs for three choices of $\alpha$ and 20 particles. *Notably, the effect of $\alpha$ is only visible if the classification loss is down-weighted so that the regularization, $R$, dominates.* Otherwise, for the tested $\alpha$, the model always fits the data. The experiment is consistent with [15], where it is shown that increasing the space on which



| | MSE $\pm\,\sigma$ |
|---|---|
| †`GPPVAE-DIS` | 0.0309 $\pm$ 0.00002 |
| †`GPPVAE-JOINT` | 0.0288 $\pm$ 0.00005 |
| †`ODE`$^2$`VAE` | 0.0194 $\pm$ 0.00006 |
| †`ODE`$^2$`VAE-KL` | 0.0184 $\pm$ 0.0003 |
| **Ours** (stat. direct) | 0.0126 $\pm$ 0.0064 |
| **Ours** (stat. w particles) | 0.0125 $\pm$ 0.0063 |
| **Ours** (dyn. w particles) | 0.0122 $\pm$ 0.0064 |

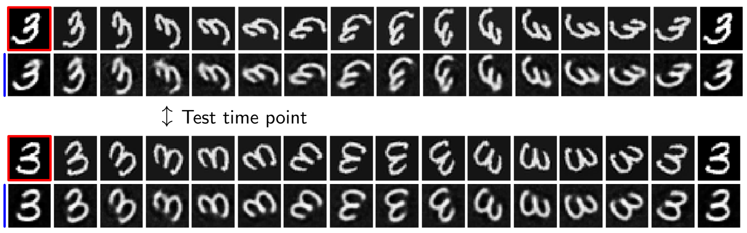
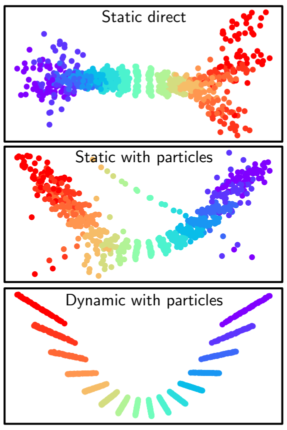


**Figure 4:** *Left*: Image (per-pixel) MSE (measured at the marked time point) averaged over all testing sequences of the rotated MNIST dataset. *Right*: Two testing sequences and predictions (marked blue) for all 16 time points when the image at $t = 0$ is given as input (marked red). Results marked with † are taken from [40].

an ODE is solved allows for easy separation of the data and leads to less complex flows. The latter is also observed for our model; visualizations can be found in Appendix F.

**Rotating MNIST.** Here, we are given sequences of a rotating MNIST digit (along 16 angles, linearly spaced in $[0, 2\pi]$). The task is learning to synthesize the digit at any rotation angle, given only the *first* image of a sequence. We replicate the setup of [40] and consider rotated versions of the digit “3”. We identify each rotation angle as a time point $t_i$ and randomly drop four time points of each sequence during training. One fixed time point is consistently left-out and later evaluated during testing. We use the same convolutional autoencoder of [40] with the `UpDown` model operating in the internal representation space after the encoder. During training, the encoder receives the first image of a sequence (always at angle 0°), the `UpDown` model integrates forward to the desired time points, and the decoder decodes these representations. As loss, we measure the mean-squared-error (MSE) of the decoder outputs. Fig. 4 lists the MSE (at the left-out angle), averaged over all testing sequences and shows two example sequences with predictions for all time points (100 particles, $\alpha = 10$).



While all `UpDown` variants substantially lower the MSE previously reported in the literature, they exhibit comparable performance. To better understand the differences, we visualize the internal representation space of the autoencoder by projecting all 16 internal representations (i.e., the output of the `UpDown` models after receiving the output of the encoder) of each testing image onto the two largest principal components, shown to the right (different colors indicate the different rotation angles). This qualitative result shows that allowing for a time-dependent parametrization leads to a more structured latent space of the autoencoder.

**Bouncing balls.** Finally, we replicate the “bouncing balls” experiment of [40]. This is similar to the rotating MNIST experiment, but the underlying dynamics are more complex. In particular, we are given 10,000 (training) image sequences of bouncing balls at 20 different time points [35]. The task is learning to predict, after seeing the first three images of a sequence, future time points. We use the same convolutional autoencoder of [40] and minimize image (per-pixel) MSE (using all 20 time points for training). Our `UpDown` model operates in the internal representation space of the encoder (50-dimensional in our experiments[2]). In test mode, the network receives the first three image of a sequence and predicts 10 time points ahead. We measure the image (per-pixel) MSE and average the results (per time point) over all 500 testing sequences. For model selection, we rely on the provided validation set. Our `UpDown` (dynamic with particles) model uses 100 particles. Fig. 5 (*left*) lists the averaged

[2]We did not further experiment with this hyperparameter, so potentially better results can be obtained.

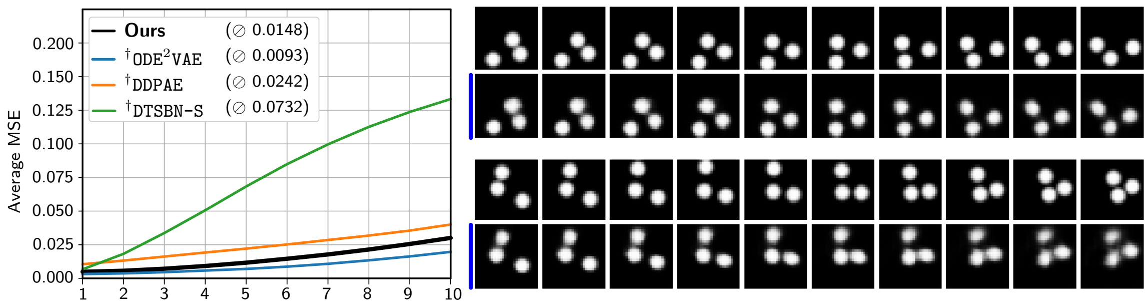


**Figure 5:** *Left*: Image (per-pixel) MSE for predicting 10 time points ahead (after receiving the first three inputs of a sequence), averaged over all testing sequences (numbers in parentheses indicate the MSE when additionally averaged over *all* prediction time points). Results marked with † are taken from [40]. *Right*: Two testing sequences with predictions (marked blue).

MSE per time point, plotted against the approaches listed in [40]. Fig. 5 (*right*) shows two testing sequences with predictions (the three input time points are not shown). Results for the `UpDown` static and static with particles model are ⊘ 0.0154 and ⊘ 0.0150, respectively.

## 5 Discussion and Conclusions

We demonstrated that it is possible to parametrize DNNs via initial conditions of (position, momentum) pairs. While our experiments are admittedly still simple, results are encouraging as they show that 1) the particle-based approach can achieve competitive performance over direct parametrizations and that 2) time-dependent parametrizations are useful to obtaining simpler networks and can be realized with significantly fewer parameters using particle-based shooting. Our work opens up many different follow-up questions and formulations. E.g., we presented our approach for a model with continuous dynamics, but the particle and the shooting formalism can also be applied to discrete-time models. Further, we focused, for simplicity, on continuous variants of multi-layer perceptrons, but similar linear-in-parameter models can be formulated for convolutional neural networks. Models that are nonlinear in their parameters hold the promise for connections with optimal mass transport theory and to theoretical complexity results, which we touched upon for our `UpDown` model. Indeed, this change of paradigm in the parametrization may result in new quantitative results on the generalization properties of the constructed networks. Lastly, how well the approach generalizes to more complex problems, how many particles are needed to switch from a standard deep network to its shooting formulation, and how optimizing over critical points of the original optimization problem via shooting relates to network generalization will be fascinating to explore.

## Broader Impact

Our work advances knowledge in how to parametrize deep neural networks. Specifically, it shifts the parametrization of deep neural networks from a layer-by-layer perspective to an initial-value parametrization. At this point it is theoretical and conceptual in nature and so is its likely current broader impact.

## Acknowledgments and Disclosure of Funding

This research project was initiated during a one-month invitation of M. Niethammer by the Labex Bézout, supported by the French National Research Agency ANR-10-LABX-58.

## References

[1] Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.*, 2(4):303–314, 1989.

[2] Diffeomorphic 3D image registration via geodesic shooting using an efficient adjoint calculation. *IJCV*, 97(2):229–241, 2012.

[3] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017.

[4] V. I. Arnold. *Mathematical Methods of Classical Mechanics*. Springer, 1978.

[5] M. Athans and P. L. Falb. *Optimal control: an introduction to the theory and its applications*. Courier Corporation, 2013.

[6] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.

[7] S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. In *NeurIPS*, 2019.

[8] P. L. Bartlett and S. Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 3(null):463–482, Mar. 2003.

[9] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *ICML*, 2019.

[10] J.-D. Benamou and Y. Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000.

[11] J. F. Bonnans. The shooting approach to optimal control problems. In *IFAC International Workshop on Adaptation and Learning in Control and Signal Processing*, 2013.

[12] M. Bruveris and F.-X. Vialard. On completeness of groups of diffeomorphisms. *J. Eur. Math. Soc. (JEMS)*, 19(5):1507–1544, 2017.

[13] A. Bryson and Y. Ho. Applied optimal control: optimization, estimation, and control. 1975. *Hemisphere, New York*, pages 177–203.

[14] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

[15] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural ODEs. In *NeurIPS*, 2019.

[16] W. E, C. Ma, and L. Wu. Barron spaces and the compositional function spaces for neural network models. *arXiv*, 2019. https://arxiv.org/abs/1906.08039.

[17] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman. How to train your neural ODE: the world of Jacobian and kinetic regularization. In *ICML*, 2020.

[18] A. Gholami, K. Keutzer, and G. Biros. ANODE: Unconditionally accurate memory-efficient gradients for neural odes. In *IJCAI*, 2019.

[19] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. SIAM, 2008.

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

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[23] J.-H. Jacobsen, A. Smeulders, and E. Oyallon. i-RevNet: Deep invertible networks. In *ICLR*, 2018.

[24] Q. Li, L. Chen, C. Tai, and E. Weinan. Maximum principle based algorithms for deep learning. *JMLR*, 18(1):5998–6026, 2017.

[25] G.-H. Liu and E. A. Theodorou. Deep learning theory review: An optimal control and dynamical systems perspective. *arXiv*, 2019. https://arxiv.org/abs/1908.10920.

[26] S. Massaroli, M. Poli, J. Park, A. Yamashita, and H. Asama. Dissecting neural ODEs. *arXiv*, 2020. https://arxiv.org/abs/2002.08071.

[27] A. Maurer. A vector-contraction inequality for rademacher complexities. In R. Ortner, H. U. Simon, and S. Zilles, editors, *Algorithmic Learning Theory*, pages 3–17, Cham, 2016. Springer International Publishing.

[28] M. Miller, A. Trouvé, and L. Younes. Geodesic shooting for computational anatomy. *J. Math. Imaging Vis.*, 24:209–228, 2006.

[29] B. Neyshabur, S. Bhojanapalli, D. Mcallester, and N. Srebro. Exploring generalization in deep learning. In *NeurIPS*. 2017.

[30] M. Niethammer, Y. Huang, and F.-X. Vialard. Geodesic regression for image time-series. In *MICCAI*, 2011.

[31] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge University Press, 2007.

[32] V. Reshniak and C. Webster. Robust learning with implicit residual networks. *arXiv*, 2019. https://arxiv.org/abs/1905.10479.

[33] L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *J. Math. Imaging Vis.*, pages 1–13, 2019.

[34] S. Sæmundsson, A. Terenin, K. Hofmann, and M. P. Deisenroth. Variational integrator networks for physically meaningful embeddings. *arXiv*, 2019. https://arxiv.org/abs/1910.09349.

[35] I. Sutskever, G. Hinton, and G. Taylor. The recurrent temporal restricted Boltzmann machine. In *NeurIPS*, 2009.

[36] P. Toth, D. J. Rezende, A. Jaegle, S. Racanière, A. Botev, and I. Higgins. Hamiltonian generative networks. In *ICLR*, 2020.

[37] L. N. Trefethen. Finite difference and spectral methods for ordinary and partial differential equations. 1996.

[38] J. L. Troutman. *Variational calculus and optimal control: optimization with elementary convexity*. Springer Science & Business Media, 2012.

[39] M. J. Wainwright. *High-Dimensional Statistics: A Non-asymptotic Viewpoint*, volume 48. Cambridge University Press, 2019.

[40] C. Yildiz, M. Heinonen, and H. Lahdesmaki. ODE$^2$VAE: Deep generative second order ODEs with Bayesian neural networks. In *NeurIPS*, 2019.

[41] L. Younes. *Shapes and Diffeomorphisms*. Springer, 2010.

[42] L. Younes. Diffeomorphic learning. *arXiv*, 2018. https://arxiv.org/abs/1806.01240.

[43] H. Zhang, X. Gao, J. Unterman, and T. Arodz. Approximation Capabilities of Neural ODEs and Invertible Residual Networks. In *ICML*, 2020.

[44] T. Zhang, Z. Yao, A. Gholami, J. E. Gonzalez, K. Keutzer, M. W. Mahoney, and G. Biros. ANODEV2: A coupled neural ODE framework. In *NeurIPS*, 2019.

## Supplementary material

The following sections discuss additional aspects of our approach. Specifically, Sec. A shows that the optimality condition underlying our shooting formulation can be approximated via particles. Sec. B discusses the number of free parameters of our shooting approach in relation to the number of free parameters for direct optimization. Sec. C explains how the shooting equations can be automatically derived via automatic differentiation. Sec. D shows the universality of our `UpDown` model. Sec. E provides details on our experimental setup. Lastly, Sec. F shows some additional experimental results.

## A Expectation approximation of optimality equations

In this section, we show that the particle shooting approach can be viewed as approximating the optimality equation associated with a collection of particles.

### A.1 Variational setup

Suppose the data consists of input $X \in \mathbb{R}^d$. Let $f(\cdot, \theta(t))$ be a vector field on $\mathbb{R}^d$, e.g. a shallow hidden layer or a linear (in parameter) layer. Consider the flow $\varphi := \varphi(T, \cdot)$ generated by $f$ according to

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

We consider the general task of minimizing,

$$\mathrm{Reg}(\varphi) + \lambda \mathbb{E}[\ell(\varphi(X))]\,, \tag{A.2}$$

where $\lambda$ is a positive regularization parameter.

Without loss of generality, set the terminal time to $T = 1$. Letting $\rho_0$ denote the probability density of $X$, minimizing Eq. (A.2) is equivalent to minimizing

$$\mathrm{Reg}(\varphi) + \lambda \inf_{\varphi} \int_{\mathbb{R}^d} \ell(\varphi(\mathbf{x}))\rho_0(\mathbf{x})\,\mathrm{d}\mathbf{x}\,.$$

This can be rewritten as

$$\mathrm{Reg}(\varphi) + \lambda \inf_{\varphi} \int_{\mathbb{R}^d} \ell(\mathbf{x}')\rho_1(\mathbf{x}')\,\mathrm{d}\mathbf{x}'\,,$$

where $\rho_1(\mathbf{x}) := \rho(1, \mathbf{x})$ is the flow of the continuity equation

$$\partial_t \rho(t, \mathbf{x}) + \mathrm{div}(\rho(t, \mathbf{x}) f(\mathbf{x}, \theta)) = 0\,, \rho(0, \mathbf{x}) = \rho_0(\mathbf{x})\,,$$

where div is the divergence operator on vector fields. Note $\rho_1$ can be regarded as the density representing the data at time 1.

### A.2 Choice of regularization

We discuss two different possibilities for regularization of the map $\varphi$:

$$\mathrm{Reg}(\varphi) = \int_0^1 R(\theta(t))\,\mathrm{d}t\,. \tag{A.3}$$

The first possibility is a quadratic penalty, where $R(\theta(t)) = \frac{1}{2}\|\theta(t)\|_2^2$ is the Frobenius norm of the parameter $\theta$. Since the space of vector fields is a *finite* dimensional linear space, it can be endowed with a scalar product, which turns this space into a Reproducing Kernel Hilbert Space (RKHS). Therefore, the linear in parameter - quadratic regularization setting is a particular case of vector fields encoded by $f(\cdot,\theta(t)) \in H$, with $H$ a RKHS embedded in $W^{1,\infty}$ vector fields[3]. This setting leverages strong analytical and geometrical foundations [41, 12]: 1) When the activation function (very often, the RKHS is composed of smooth vector fields) is smooth, the flow map $\varphi$ is guaranteed to be a one-to-one smooth map (i.e. a diffeomorphism). For instance, with the `UpDown` model, it is a homeomorphism in $(\mathbf{q},\mathbf{v})$. Moreover, the quadratic penalty induces a right-invariant distance on the set of flows generated by Eq. (A.1) and the distance to identity of the resulting flow can be bounded by $\mathrm{Reg}(\varphi)$ (see [41, 12] for more details in a Sobolev setting). 2) When the activation function is of `ReLU` type, the resulting map is still a $W^{1,\infty}$ one-to-one map (i.e. a homeomorphism) and has Lipschitz regularity.

The other type of regularization we will discuss is the Barron norm regularization for the shallow hidden layer of Eq. (3.7). First, recall Barron's norm [16],

$$\|\theta\|_{\mathcal{B}}^2 := \frac{1}{n}\sum_{j=1}^{n} \|\theta_1^j\|_2^2(\|[\theta_2]_j\|_1 + \|b_2^j\|_1)^2 \,.$$

As discussed in the main text, the reason we consider a Barron norm penalty for the shallow hidden layer rather than the quadratic penalty is because of its theoretical results. Indeed, the Rademacher complexity is bounded for the combination of a shallow hidden layer with a Barron norm penalty, but not when combined with a quadratic penalty.

## A.3 Rademacher complexity of bounded energy flows.

In this section we consider the flows generated by a neural ODE approach with a kinetic energy penalty for which we address the question of bounding the Rademacher complexity of the generated flows. The flow of a vector field $f(\cdot,\theta(t))$ is a vector valued map denoted by $\varphi$. Let us first treat the case of the Rademacher complexity of a component of a the flow map $\varphi^k$.

**Theorem 2.** *Let $\mathcal{F}$ be a space of vector fields defined on a compact space $C \subset \mathbb{R}^d$. Assume that the Rademacher complexity on $n$ points in $C$ of each component of the vector fields $f^k(t,\cdot)$ for $k = 1,\dots,d$ is controlled by $M(n,t)$ which depends on $n$, then the Rademacher complexity of each component of the flows at time 1 is bounded by $\int_0^1 M(n,t)\,\mathrm{d}t$.*

*Proof.* Recall that Rademacher complexity (see [39]) of a class of functions $\mathcal{F}$ is defined as, for $\mathbf{Z} = (\mathbf{z}_1,\dots,\mathbf{z}_n) \in C$,

$$\mathrm{Rad}_{\mathcal{F},\mathbf{Z}} \overset{\text{def.}}{=} \mathbb{E}[\sup_{g\in\mathcal{F}} \sum_{i=1}^{n} \varepsilon_i g(\mathbf{z}_i)] \,,$$

where the $(\varepsilon_i)_{i=1,\dots,n}$ are i.i.d. Rademacher random variables. Our hypothesis ensures $\mathrm{Rad}_{\mathcal{F},\mathbf{Z}} \leq M(n)$. Apply the definition of the flow to get

$$\varphi(1,\mathbf{x}) = \mathbf{x} + \int_0^1 f(\varphi(t,\mathbf{x}),\theta(t))\,\mathrm{d}t \,.$$

Therefore,

$$\begin{aligned}\mathbb{E}[\sup_{\varphi\in\mathcal{F}} \sum_{i=1}^{n} \varepsilon_i \varphi^k(\mathbf{z}_i)] &\leq \mathrm{Rad}_{\{\mathrm{Id}\},\mathbf{Z}} + \int_0^1 \mathbb{E}[\sup_{f(\cdot,\theta(t))} \sum_{i=1}^{n} \varepsilon_i f^k(\varphi(t,\mathbf{z}_i),\theta(t))]\,\mathrm{d}t \,,\\ &\leq 0 + \int_0^1 M(n,t)\,\mathrm{d}t \,.\end{aligned}$$

[3]I.e smoothness asks for Lipschitz regularity of the first derivative, which ensures existence and uniqueness of the flow.

In the previous formula, we used the fact that the Rademacher complexity of just a single map is zero. □

**Corollary 3.** *Let $H$ be a RKHS of vector fields whose kernel $\mathsf{k}$ satisfies $\|\mathsf{k}(\mathbf{x},\mathbf{x})^k\|_\infty \leq 1$ , then, the set of flows denoted by $\mathcal{F}$ at time 1 of time-dependent vector fields in a ball of radius $R$ satisfies $\mathrm{Rad}_{\mathcal{F},n} \leq \frac{2R}{\sqrt{n}}$, where $\mathrm{Rad}_{\mathcal{F},n}$ is the Rademacher complexity for $n$ points.*

*Proof.* The Rademacher complexity of the ball of radius $R$ in the RKHS $H$ [8, Lemma22] is upper bounded by $\mathrm{Rad}_{H,n} \leq \frac{2R}{\sqrt{n}}$. We then directly apply Theorem 2. □

A similar result also holds for vector fields generated by the shallow-hidden-layer of Eq. (3.7), see [16]. Last, we note that the result and its proof also hold if one uses the following Rademacher complexity for vector valued functions [27],

$$\mathrm{Rad}_{\mathcal{F},\mathbf{Z}} \overset{\text{def.}}{=} \mathbb{E}[\sup_{g\in\mathcal{F}} \sum_{i=1}^{n} \sum_{j=1}^{d} \varepsilon_j \varepsilon_i g_j(\mathbf{z}_i)]\,,$$

for $g = (g_j)_{j=1,\ldots,d} \in \mathcal{F}$.

## A.4 Optimality equations and Hamiltonian ensemble approximation

We detail the optimality equations in the case data are represented via a probability measure. As mentioned above, the regularity of the map is enforced via a penalty on the weights at each timepoint and is the integral $\int_0^1 R(\theta(t))\,\mathrm{d}t$ or even more generally $\int_0^1 R(\theta(t),\rho(t))\,\mathrm{d}t$. Using Lagrange multipliers, this constraint can be enforced and minimizers of the energy should be saddlepoints of the energy

$$\begin{aligned}\mathcal{L}(\rho,\theta,p) := \lambda \int_{\mathbb{R}^d} \ell(\mathbf{x})\rho_1(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_0^1 R(\theta(t))\,\mathrm{d}t \\ + \int_0^1 \int_{\mathbb{R}^d} p(t,\mathbf{x})(\partial_t\rho(t,\mathbf{x}) + \mathrm{div}(\rho(t,\mathbf{x})f(\mathbf{x},\theta(t))))\,\mathrm{d}\mathbf{x}\,\mathrm{d}t\,,\end{aligned}$$

where $p(t,\mathbf{x})$ is a time and space dependent function. The optimality equations are then

$$\begin{cases} \partial_t\rho(t,\mathbf{x}) + \mathrm{div}(\rho(t,\mathbf{x})f(t,\mathbf{x},\theta(t))) = 0\,, \\ \partial_t p(t,\mathbf{x}) + \nabla p(t,\mathbf{x}) \cdot f(t,\mathbf{x},\theta(t)) = \frac{\delta R}{\delta\rho}(\theta(t),\rho(t))\,, \\ \frac{\delta R}{\delta\theta}(\theta(t),\rho(t)) - \int_{\mathbb{R}^d} \frac{\delta f}{\delta\theta}(\mathbf{x})^\top(\nabla p(t,\mathbf{x})\rho(t,\mathbf{x})) = 0\,, \end{cases} \tag{A.4}$$

where $\nabla p$ is the gradient w.r.t. $x$ of $p(t,x)$ and $\delta$ denotes differentiation w.r.t. the indicated parameter.

In practice, one does not have access to the full distribution and the variational setup needs to be approximated. As proposed in the main text, we approximate it using a collection of particles that follow the optimality equations which are Hamiltonian evolution equations for this collection of particles. The collection of particles $(\mathbf{q}_i,\mathbf{p}_i)$ are defined by their state and costate; the quantities $p(t,\mathbf{x})$ and $\rho(t,\mathbf{x})$ are simply approximated with $\mathbf{p}_i(t),\mathbf{q}_i(t)$ satisfying the Hamiltonian equations associated with the empirical measure $\rho(t,\cdot) = \frac{1}{N}\sum_{i=1}^N \delta_{\mathbf{q}_i(t)}(\cdot)$, the corresponding function $p(t,\cdot)$ being only defined at points $\mathbf{q}_i$. When the number of particles tends to infinity, the optimal trajectory in the parameter space can be well approximated, and one can expect to recover the optimal trajectory.

However, we do not explore the following question here: *How well can a set of Hamiltonian particles approximate Eq. (A.4)?* We simply remark that this question is directly connected to expressiveness and generalization properties of the constructed neural network and is also probably data dependent.

## A.5 Linear in parameters - quadratic energy

Now let us examine in detail models that are *linear* in parameters and have *quadratic* energy on parameters: this case is the simplest to be studied, and computationally not as demanding as the nonlinear case. As mentioned above, the set of possible vector fields $f(\cdot, \theta(t))$ is a finite dimensional linear space, which is a reproducing kernel Hilbert space when endowed with an $L^2$ norm. Since all Hilbert norms in finite dimensions are equivalent, this choice of regularization is universal in this class of quadratic penalties.

1. The vector field is $f(\cdot, \theta(t)) = \theta \cdot \sigma$, where $\sigma$ is a vector of maps. In this case, the optimality equation reads
$$\frac{\delta f}{\delta \theta}(\mathbf{x})^*(\nabla p(t,\mathbf{x})\rho(t,\mathbf{x})) = \int_{\mathbb{R}^d} \sigma(x)^\top (\nabla p(t,\mathbf{x}))\rho(t,\mathbf{x}))\,\mathrm{d}x\,.$$
2. If the penalty $R$ only depends on $\theta$ and is quadratic: $R(\theta(t)) = \frac{1}{2}\int_0^1 \|\theta\|^2\,\mathrm{d}t$, then one has $\frac{\delta R}{\delta \theta}(\theta(t), \rho(t)) = \theta(t)$.

Thus, under these two conditions, the parameters are *explicit* in terms of $p$, $\rho$ and $\sigma$:
$$\theta(t) = \int_{\mathbb{R}^d} \sigma(\mathbf{x})^\top (\nabla p(t,\mathbf{x})\rho(t,\mathbf{x}))\,\mathrm{d}\mathbf{x}\,. \tag{A.5}$$

**Remark 4.** *If, instead of quadratic regularization on the parameters, we were to choose a RKHS norm (in the infinite dimensional case) as penalty, it would result in the introduction of the kernel applied to the r.h.s. of Eq. (A.5).*

Having a formula for the parameter $\theta(t)$, one could be tempted to derive an evolution equation for $\theta$. This equation is known as the EPDiff equation [41] and is unfortunately not a closed equation on the set of parameters $\theta(t)$ themselves. Therefore, our approach is a possible way to approximate it.

## A.6 Nonlinear in parameters - energy which depends on the distribution

For exposition purposes, we present two cases of interest, which are not contained in the previous section.

**Example of Barron's norm.** Obviously, a shallow-hidden-layer $\mathrm{shl}_\theta$ is not linear in parameters. We have already discussed that it is proper in this case to endow the space with norms such as Barron's norm [16]. For instance, for $\mathrm{shl}_\theta(x)$ one has
$$\mathrm{shl}_\theta(\mathbf{x}) = \theta_1 \sigma(\theta_2(\mathbf{x}) + b_2)\,, \tag{A.6}$$
where $\theta_1 \in L(\mathbb{R}^n, \mathbb{R}^d)$ and $\theta_2 \in L(\mathbb{R}^d, \mathbb{R}^n)$. The Barron norm of the shallow-hidden-layer is then
$$\|\,\mathrm{shl}_\theta\,\|_{\mathcal{B}}^2 := \frac{1}{n}\sum_{j=1}^n \|\theta_1^j\|_2^2 (\|[\theta_2]_j\|_1 + \|[b_2]_j\|_1)^2\,,$$
where $\theta_1^j$ denotes the jth column of $\theta_1$ and $[\theta_2]_j$ denotes the jth row of $\theta_2$.

Let us consider the case of $R(\theta(t)) = \frac{1}{2}\|\,\mathrm{shl}_{\theta(t)}\,\|_{\mathcal{B}}^2$. In this case, one has the following optimality equations to solve
$$\lambda \theta_1^j (\|[\theta_2]_j\|_1 + \||[b_2]_j\|_1)^2 = \int_{\mathbb{R}^d} \sigma([\theta_2]_j \mathbf{x} + [b_2]_j)^\top (\nabla p(t,\mathbf{x})\rho(t,\mathbf{x}))\,\mathrm{d}\mathbf{x}\,,$$
$$\lambda \|\theta_1^j\|_2^2 (\|[\theta_2]_j\|_1 + \|[b_2]_j\|_1)\partial\|[\theta_2]_j^k\|_1 = \int_{\mathbb{R}^d} [\,\mathrm{d}\sigma([\theta_2]_j \mathbf{x} + [b_2]_j)(\mathbf{x}_k)]^\top (\nabla p(t,x)\rho(t,\mathbf{x}))\,\mathrm{d}\mathbf{x}\,,$$
$$\lambda \|\theta_1^j\|_2^2 (\|[\theta_2]_j\|_1 + \|[b_2]_j\|_1)\partial\|[b_2]_k^j\|_1 = \int_{\mathbb{R}^d} [\,\mathrm{d}\sigma([\theta_2]_j \mathbf{x} + [b_2]_j)(\mathbf{x}_k)]^\top (\nabla p(t,\mathbf{x})\rho(t,\mathbf{x}))\,\mathrm{d}\mathbf{x}\,.$$

These equations involve the subdifferential of the $L^1$ norm, and optimization of this type of functions, which involves sparsity, is a well-explored field [6]. We leave experiments with this norm for future work.

**$L^2$ regularization, optimal transport.** Last, we briefly mention a model that is part of our framework which has the advantage of not specifying a norm on the space of vector fields. In case there is no obvious norm to be used on the space of vector fields, it is possible to use an $L^2$ type of penalty, directly on the vector fields themselves. Indeed, one way to be rather independent of the choice of the parametrization of the map consists in introducing a cost that represents the $L^2$ norm of the map. However, $L^2$ depends on the choice of a measure and this measure can be naturally chosen as the density of the data, $\rho(t, \mathbf{x})$. More precisely, one can have

$$R(f, \rho(t)) = \frac{1}{2} \int_{\mathbb{R}^d} \|f(\mathbf{x}, \theta)\|^2 \rho(t, \mathbf{x}) \, \mathrm{d}\mathbf{x} \,. \tag{A.7}$$

In such a case, this formulation resembles finding an optimal transport (OT) map between $\rho_0$ and $\rho_1$. Specifically, optimal transport is an optimization problem which can be solved via a fluid dynamic formulation [10] introducing the kinetic penalty above. However, the two models (OT and the one defined by the regularization Eq. (A.7)) differ in the sense that the optimization set for optimal transport is much larger and the above formulation is a parametrized approximation of OT. This parametrized approximation needs to retain generalization properties of the optimized map. Note however, that in the limit where the number of neurons goes to infinity, optimal transport will be well-approximated since the optimization is performed on a dense subset of all vector fields. Obviously, fixing the choice of a shallow-hidden-layer design implies a choice for $n$ in Eq. (3.7), which thus gives a regularization of the computed approximation of the optimal transport map.

**Computational burden.** In both cases, the implicit equation corresponding to the third equation in Eq. (A.4) has to be solved at each layer of the discretization. We experimented with a simple strategy of unrolling the related minimization scheme. An efficient approach to solve such implicit equations will be necessary for practical implementations.

## B Analysis of the number of free parameters

It is instructive to understand the number of parameters for a shooting approach in comparison to the typical approach of optimizing a neural network (where the parameter-dependency at optimality is only considered implicitly at convergence of the numerical solution rather than explicitly during the shooting). We focus on the cases of affine and convolutional layers for illustration.

Consider a DNN with a depth of $D$ layers, each of them parametrized by a shallow hidden layer of $P$ parameters. The number of free parameters is then $DP$, compared to $2KS$ where $K$ is the number of active particles, each of them of size $2S$[4]. Hence, solutions with less than $DP/(2S)$ particles provide benefits in the number of free parameters. Therefore, as

[4]For example, $S$ for our `UpDown` model simply corresponds to the dimension of its state space: $S = (\alpha+1)d$, where $\alpha$ is the inflation factor and $d$ the data dimension. Note that in our experiments with the `UpDown` model we also learned an affine map from the initial conditions $x(0)$ to the initial conditions $v(0)$. Such a map has $\alpha d(d+1)$ parameters. These parameters are included in the table of Fig. 3 and in Tables 1/2 summarizing the number of mode parameters. However, we will not consider parameters in our discussion here, as they would equally apply to both a shooting and a direct optimization approach and could also be avoided by simply initializing $v(0)$ to zero. A similar initialization to zero approach is, for example, commonly taken in `ResNets` when increasing the number of feature channels [21].

the number of particles is reduced, we may parameterize the DNN with a smaller number of parameters. *Most remarkably, the number of free parameters is always $2KS$ regardless of the number of parameters of a particular layer as the layer parameters are obtained via the shooting equations based on the particle states.* This is a consequence of regularizing the parameters in our loss which couples them across time at optimality. We make this clearer in what follows.

**Affine layers** Recall that in our simple example of Sec. 3.1 the parameters $\theta(t) = [A(t), b(t)]$ of our affine model given as

$$\mathbf{A}(t) = \mathbf{M_A}^{-1}(-\sum_{j=1}^{K} \mathbf{p}_j(t)\sigma(\mathbf{q}_j(t))^\top), \quad \mathbf{b}(t) = \mathbf{M_b}^{-1}(-\sum_{j=1}^{K} \mathbf{p}_j(t))\,.$$

Here, $A(t)$ has $d^2$ and $b(t)$ has $d$ parameters; these parameters are indirectly given by the set of particles $\{(\mathbf{q}_i(t), \mathbf{p}_i(t))\}$ at any given time. Hence, for this model $S = d$ and $P = d(d+1)$. If we assume we have $K$ particles and compare to a discrete layer implementation of this model then the particle-based approach will have less free parameters if

$$2Kd < Dd(d+1).$$

Importantly, the state-space dimension, $d$, only enters the number of free parameters linearly for the particle approach ($2Kd$), while there is a quadratic dependence for direct optimization ($Dd(d+1)$). This is a direct consequence of the optimality condition which couples the parameters $\theta(t)$ across time. One can see this phenomenon in action in Eq. (B), where the matrix $A$ is expressed as the sum of matrices $\mathbf{p}_j(t)\sigma(\mathbf{q}_j(t))^\top$ with rank $\leq 1$. Concretely, a particle-based shooting approach uses less parameters if the number of particles $K < D(d+1)/2$. Another interesting observation based on this example is that even if we would have only considered a linear model (i.e., without the bias term, $b(t)$) the number of parameters for the particles would have still remained at $2KS$. This is again a consequence of optimality and of our parameter regularization. Note that this also means that even though our `UpDown` model

$$\dot{\mathbf{x}}(t) = \theta_1(t)\sigma(\mathbf{v}(t)) + b_1(t), \quad \dot{\mathbf{v}} = \theta_2(t)\mathbf{x}(t) + b_2(t) + \theta_3(t)\sigma(\mathbf{v}(t))\,,$$

has significantly more parameters $\theta(t) = [\theta_1(t), b_1(t), \theta_2(t), b_2(t), \theta_3(t)]$ when directly optimized, this has no direct impact on the number of free parameters of its particle-based parameterization. Only the state-space dimension matters. Concretely, if we were to instead consider a model of the form

$$\dot{\mathbf{x}}(t) = \theta_1(t)\sigma(\mathbf{v}(t)), \quad \dot{\mathbf{v}} = \theta_2(t)\mathbf{x}(t)\,,$$

the particle-based parameterization would stay *unchanged!* Only the way how one infers $\theta(t)$ from the particles changes.

**Convolutional layers** Shooting approaches for convolutional models can also be derived. We did not experiment with such models in this work. However, we show here that the number of free parameters may also be decreased with a particle-based approach. This will be interesting to explore in future work. Specifically, for convolutional layers a particle-based parameterization could be particularly effective as one typically has quadratic complexity in the number of filters between convolutional layers (i.e., if a layer with $N$ feature channels is followed by a layer with $M$ feature channels, this will induce the estimation of $N \times M$ convolutional filters and hence will drastically influence the number of parameters for large

$N$ or $M$). In contrast, a particle-based shooting approach does not increase the number of parameters as it ties them together via the optimality conditions expressed by the shooting equations. As a rough estimate for a standard convolutional `ResNet` for $D = 50$, $P = 100^2 \times 16$, $DP \approx 8.10^6$. Thus, if particles have size 40, we end up with at most $10^5$ active particles.

**General Remarks** Nevertheless, all model parameters (e.g., $[A(t), b(t)]$ or all convolutional filters for a convolutional layer) are still instantiated during computation. It is important to note that regardless of the chosen number of particles, a shooting neural network solution is a possible optimal solution (for a given data set) at any given time, not only at convergence. One optimizes over the family of possible neural network models with the goal of finding the element within this family that best matches the observations.

## C Automatic shooting

The general shooting equations were presented in Eq. (3.3). We then proceeded to explicitly derive the shooting equations for a continuous DNN with linear-in-parameter layers and `UpDown` layers in §3.1 and §3.2, respectively. While this was instructive, it is somewhat cumbersome, in particular, for more complex models or when moving to convolutional networks. Fortunately, in practice these shooting equations do not need to be derived by hand. Indeed, they are completely specified by the Hamiltonian $H(\mathbf{p}, \mathbf{x}, \theta) = \mathbf{p}^\top(\dot{\mathbf{x}} - f(t, \mathbf{x}, \theta)) + R(\theta)$, in the sense that the shooting equations in Eq. (3.3) are computed via differentiation of $H$. Specifically, the shooting equations in Eq. (3.3) are equivalently given by

$$\begin{cases} \dot{\mathbf{x}} = \frac{\partial H(\mathbf{p},\mathbf{x},\theta)}{\partial p}, \\ \dot{\mathbf{p}} = -\frac{\partial H(\mathbf{p},\mathbf{x},\theta)}{\partial x}, \\ \theta \in \arg\min_\theta H(\mathbf{p}, \mathbf{x}, \theta). \end{cases}$$

As discussed above, the last equation can be replaced by solving $\partial_\theta R(\theta) - \sum_{i=1}^N \partial_\theta f(t, \mathbf{x}_i, \theta)^T(\mathbf{p}_i) = 0$. Automatic differentiation can be used to automatically obtain the shooting equations. As fitting a shooting model requires differentiating the shooting equations we in effect end up with differentiating twice. This can be done seamlessly using modern deep learning libraries, such as `PyTorch`.

## D Universality of the `UpDown` model

In this section, we set out to demonstrate that the `UpDown` model is universal in the sense that its associated flow can come $\varepsilon$-close to the flow of any well behaving time-dependent vector field.

Define the shallow-hidden-layer vector field with time-varying parameters $\theta(t) = (\theta_1(t), \theta_2(t), b_2(t))$ as:

$$\mathrm{shl}_\theta(\mathbf{x}) = \theta_1 \sigma(\theta_2 \mathbf{x} + b_2).$$

While shooting with the shallow hidden layer is theoretically appealing as $\mathrm{shl}_\theta$ is universal [1], it would result in implicit shooting equations. We first show that the `UpDown` model introduced in 3.2 can give the same flow as the shallow hidden layer (Lemma 6) and then leverage this relationship to show that the `UpDown` model inherits the universality of the shallow hidden layer (7). We recall the reformulation presented in Section 3.2.

**Lemma 5.** *Let* $\mathrm{shl}_{\theta(t)} : \mathbb{R}^d \mapsto \mathbb{R}^d$ *be a time-dependent vector field with* $\theta_2(t)$ *and* $b_2(t)$ *being piecewise* $C^1$ *and* $\theta_1(t), b_1(t)$ *continuous. Then, there exists a parametrization of the* `UpDown` *model that gives the same flow at a fixed time,* $T = 1$.

*Proof.* We rewrite the differential equation

$$\dot{\mathbf{q}} = \theta_1(t)\sigma(\theta_2(t)\mathbf{q} + b_2(t)) + b_1(t)\,,$$

by introducing the additional state variable $\mathbf{v} = \theta_2(t)\mathbf{q} + b_2(t)$ which we differentiate w.r.t. time. We obtain $\dot{\mathbf{v}} = \dot{\theta}_2(t)\mathbf{q} + \dot{b}_2(t) + \theta_2(t)\dot{\mathbf{q}}$. Replacing $\dot{\mathbf{q}}$ by its formula, we get

$$\dot{\mathbf{v}} = \dot{\theta}_2(t)\mathbf{q} + \dot{b}_2(t) + \theta_2(t)\theta_1(t)(\sigma\mathbf{v}(t)) + b_1(t))\,.$$

The system can be rewritten as

$$\begin{cases} \dot{\mathbf{q}} = \theta_1(t)\sigma(\mathbf{v}(t)) + b_1(t)\,, \\ \dot{\mathbf{v}} = \theta_3(t)\mathbf{q}(t) + \theta_4(t)\sigma(\mathbf{v}(t)) + b_3(t)\,. \end{cases} \tag{D.1}$$

Therefore, with the initial condition $\mathbf{v}(0) = \theta_2(0)\mathbf{q}(0)$ and $\mathbf{q}(0) = \mathbf{q}_0$, the two systems of ordinary differential equations are equivalent. □

Note that the key point in this lemma is the loss of regularity in the evolution of $\theta_2$ since we differentiated once in time. For that reason, we now show that adding more dimensions using the inflation factor $\alpha$ alleviate this issue. It is likely possible that one could prove a universality result using only $\alpha = 1$[5]; we leave this question for future work. However, experimentally, the inflation factor has a crucial effect on the performance of the optimization, as discussed in Sec. 4.

**Lemma 6.** *Let* $\mathrm{shl}_{\theta(t)} : \mathbb{R}^d \mapsto \mathbb{R}^d$ *be a time-dependent vector field with* $\theta(t)$ *which is piecewise continuous. Then, there exists a parametrization of the* `UpDown` *model that gives the same flow.*

*Proof.* Without loss of generality, we only treat the case of one discontinuity in time of the parametrization; We thus assume that $\theta(t)$ is continuous on $[0, t_1[$ and $[t_1, 1]$. We consider $\mathbf{q}, \mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^d$ such that $\mathbf{q}, \mathbf{v}_1$ are defined as in Lemma 5. We now define, up to time $t_1$, $\mathbf{v}_2(t) = \theta(t_1)\mathbf{v}_1(t) + \theta_2(t_1)$ which implies (differentiating w.r.t. time) that $\mathbf{v}_2$ follows an evolution equation similar to $\mathbf{v}_1$ and thus can be encoded in the general form of Eq. (D.1). Now, $\mathbf{q}(t), \mathbf{v}_2(t)$ are defined on $[t_1, 1]$ by the evolution Eq. (D.1) in order to coincide with the flow of $\mathrm{shl}_{\theta(t)}$ on $[t_1, 1]$, $\dot{\mathbf{q}} = \theta_1(t)\sigma(\mathbf{v}_2(t)) + b_1(t)$ and $\dot{\mathbf{v}}_2 = \theta_3(t)\mathbf{q}(t) + \theta_4(t)\sigma(\mathbf{v}_2(t)) + b_3(t)$ for well chosen parameters as in Lemma 5. Since the value of $\mathbf{v}_1(t)$ is not used in the evolution equation of $\mathbf{q}$, we can simply extend it by $\mathbf{v}_1(t) = \mathbf{v}_1(t_1)$ which is a valid evolution equation for Eq. (D.1).

In the general case, we decompose the time interval $[0, 1]$ into $k$ intervals $[t_i, t_{i+1}[$ on which $\theta(t)$ is continuous and the proposed method can be directly extended using an inflation factor $\alpha = k$, introducing $\mathbf{v}_k \in \mathbb{R}^d$. □

Note that the result of this lemma gives an equality between the two flows defined on the *whole* space $\mathbb{R}^d$. The next result is an approximation result which holds on a compact domain $K \subset \mathbb{R}^d$. For a function $f : \mathbb{R}^d \to \mathbb{R}$, we denote $\|f\|_{K,\infty} = \sup_{x\in K} |f(x)|$.

**Proposition 7.** *The* `UpDown` *model is universal in the class of time-dependent vector fields. Let* $K \subset \mathbb{R}^d$ *be a compact domain. For every time-dependent vector field (such that it is time continuous and is Lipschitz in space)* $w : [0, 1] \times \mathbb{R}^d \mapsto \mathbb{R}^d$ *and its associated flow* $\varphi(t, \mathbf{x})$ *there exist time dependent parameters of the* `UpDown` *model such that*

$$\begin{cases} \dot{\mathbf{q}} = \theta_1\sigma(\mathbf{v}) + b_1\,, \\ \dot{\mathbf{v}} = \theta_2(\mathbf{q}) + b_2 + \theta_3\sigma(\mathbf{v})\,, \end{cases}$$

*is* $\varepsilon$*-close to the solution* $\varphi(1, \mathbf{x})$*, e.g.* $\|\varphi(1, \mathbf{x}) - \mathbf{q}(1, \mathbf{x})\|_{K,\infty} \leq \varepsilon$.

[5]Note that the case $\alpha = 1$ is similar in its formulation to a second-order model on $\mathbf{q}$.

*Proof.* The proof is standard and we include it here for self-containedness. It is the consequence of [1] and Lemma 6. Let $B(0,r)$ a ball of radius $r$ in $\mathbb{R}^d$ which contains $\varphi(t,K)$ for all time $t \in [0,1]$. The flow associated with a given time-dependent vector field $v(t,\cdot)$ can be approximated by a vector field which is piecewise constant in time; i.e. let $\varepsilon > 0$ be a positive real, (by continuity in time of $v(t,\cdot)$) there exists a decomposition of $[0,1]$ into $k$ intervals $[t_i, t_{i+1}]$ and Lipschitz vector fields $v_i(x) = \mathrm{shl}_{\theta_i}(x)$ such that $\|v_i(\mathbf{x}) - v(t,\mathbf{x})\|_{B(0,r),\infty} \leq \varepsilon$ for $t \in [t_i, t_{i+1}]$. Denote by $w(t,\cdot)$ the time-dependent vector field defined by $w(t,\cdot) = v_i(\cdot)$ for all $t \in [t_i, t_{i+1}]$. Thus, denoting the flow of $v(t,\cdot)$ by $\varphi^v$ and the flow of $w(t,\cdot)$ by $\varphi^w$, we get

$$\|\varphi^v(1,\mathbf{x}) - \varphi^w(1,\mathbf{x})\| \leq \int_0^1 \|v(t,\varphi^v(t,\mathbf{x})) - v(t,\varphi^w(t,\mathbf{x}))\| + \|v(t,\varphi^w(t,\mathbf{x})) - w(t,\varphi^w(t,\mathbf{x}))\|\,\mathrm{d}t$$

$$\begin{aligned}\|\varphi^v(1,\mathbf{x}) - \varphi^w(1,\mathbf{x})\|_{K,\infty} &\leq \int_0^1 \mathrm{Lip}(v)\|\varphi^v(t,\mathbf{x}) - \varphi^w(t,\mathbf{x})\|_{K,\infty} + \|v(t,\cdot) - w(\cdot)\|_{B(0,r),\infty}\,\mathrm{d}t \\ &\leq \int_0^1 \mathrm{Lip}(v)\|\varphi^v(t,\mathbf{x}) - \varphi^w(t,\mathbf{x})\|_{K,\infty}\,\mathrm{d}t + \varepsilon\,,\end{aligned}$$

where $\mathrm{Lip}(v)$ denotes a bound on the Lipschitz constant of $v(t,\mathbf{x})$ w.r.t. $\mathbf{x} \in B(0,r)$ for all $t \in [0,1]$. Then, the Grönwall lemma [41] gives

$$\|\varphi^v(1,\mathbf{x}) - \varphi^w(1,\mathbf{x})\|_{K,\infty} \leq \varepsilon e^{\mathrm{Lip}(v)}\,. \tag{D.2}$$

By lemma 6, $\varphi^w(1,\mathbf{x})$ can be approximated by the flow of the `UpDown` and the result is obtained by triangular inequality. □

In this section, we focused on a universality result in the space of time-dependent vector fields. Interestingly, due to the additional dimensions, it is likely that the model is universal in the space of functions as well. This conjecture is supported by the the quadratic 1D function regression example which shows that the `UpDown` model is able to capture some maps which are not homeomorphic. We leave this question for future work.

## E Experimental settings

This section describes our experimental settings. We use our `UpDown` model for all experiments and simply use a weighted Frobenius norm penalty for all parameters. Specifically, we weigh this penalty for all parameters with 1 except for, $\theta_3$ which we penalize by 10. In our experiments we have observed better convergence properties for higher penalties on $\theta_3$. This might be due to the special role that $\theta_3$ plays in the model as it subsumes a quadratic term in the original derivation of the `UpDown` model (see Sec. 3.2). In all experiments we also optimize over the affine map from $x(0)$ to $v(0)$ for the data evolution.

**Simple function regression** We use 500 epochs for all experiments. For all particle-based experiments we freeze the positions of the particles for the first 50 epochs. We use a `ReLU` activation function and the MSE loss. We weight the MSE loss by 100 and the parameter norm loss by 1. We use 500 training samples, 1,000 testing samples and 1,000 validation samples and a batch size of 50. Note that for these simple examples there is in practice no real difference between the training, testing, and validation data, as the number of samples is large and the domain is $[-1.5, 1.5]$. We initialize the particle positions uniformly at random in $[-1.5, 1.5]$ and draw the momenta from a Gaussian distribution with zero mean and standard deviation 0.1. All time-integrations are done via a fourth order Runge-Kutta integrator with time-step 0.1. For optimization we use `Adam` with a learning rate of 0.01 and the `ReduceLROnPlateau` learning rate scheduler of `PyTorch` with a learning rate reduction factor of 0.5.

**Spiral** The spiral data is generated between time $t = 0$ and $t = 10$ with 200 uniformly spaced timepoints. Training is only on small time snippets with an approximate length of 0.25 time-units. Evaluation is on these short time snippets as well as on the entire trajectory by pasting together solutions for these short time snippets, i.e., an individual short solution starts where the previous one ends. Settings for the spiral are the same as for the simple function regression with the following exceptions. We use 1,500 epochs. The step-size for the fourth order Runge-Kutta integrator is 0.05. The MSE loss is still weighted by 100, but the parameter norm loss only by 0.01. We randomly draw 100 new training samples during each epoch and use 100 evaluation samples and 1,000 short range samples and 1 long-range testing sample. All samples are randomly drawn from the trajectory. However, as the trajectory is traversed at highly nonuniform speed the samples are drawn from a uniform distribution across the trace of the spiral. As for the simple function regression experiment, there is little practical difference between the training, validation, and testing data as the problem is so simple. However, this is not of concern in these experiments as the prime objective is to study the fitting behavior of the different models. We use a batch size of 100.

**Rotating MNIST** We use the data provided by the authors of [40] and follow the same autoencoder architecture, except that our encoder maps into a 20-dimensional representation space. The number of particles is set to 100 and the inflation factor $\alpha$ is set to 10. For optimization, we use `Adam` with a learning rate of 0.001 and the `CosineAnnealingLR` learning rate scheduler of `PyTorch`. We train for 500 epochs with a batch size of 25 and the parameter norm loss set to 0.1.

**Bouncing balls** As in the rotating MNIST experiment, we rely on the data provided by the authors of [40], follow their autoencoder architecture and set the dimensionality of the representation space of the encoder to 50. The first three images of each sequence are provided to the encoder by concatenating the images along the channel dimension. The inflation factor $\alpha$ is set to 20 and we use 100 particles. We optimize over 100 epoch using `Adam` with the `CosineAnnealingLR` learning rate scheduler of `PyTorch`, the learning rate set to 0.001 and the parameter norm loss set to 0.0001.

## F Additional results

**Simple function regression** In Section 4, we considered approximating a quadratic-like function. Here we show parallel results for approximating a cubic function $y = x^3$. We will also include some additional figures for the quadratic-like regression function. Note that whereas the cubic function is invertible (but not diffeomorphic), the quadratic-like one considered in Section 4 is a simple example of a non-invertible function. Tab. 1 shows the number of parameters for the four different formulations for both regression functions. Fig. 6 shows for the cubic regression the test loss and the network complexity, as measured by the Frobenius norm [29], for the four formulations. On average the particle-based approaches show the best fits with the lowest complexity measures, indicating the simplest network parameterization. Note however that while the dynamic particle approach greatly outperformed the static particle approach for the quadratic-like function (see Fig. 2) this is not the case here. In fact, the static particle approach shows slightly better fits than the dynamic one. This might be because the cubic function is significantly simpler to fit and hence may not benefit as much from the dynamic approach. To illustrate that fitting the quadratic-like function is indeed harder, Figs. 7 and 8 show function fits for different numbers of particles for the cubic function and the quadratic-like function, respectively. All these fits are for the particle-based dynamic `UpDown` model. Clearly, very few particles can achieve reasonable fits for a simple function. As little as two particles already show a good fit for the cubic function, whereas the quadratic-like function requires with more particles. This supports our hypothesis that fitting more complex functions may require more particles.

Since our approach is based on the time-integration of the `UpDown` model it is interesting to see 1) how the mapping is expressed across time and 2) how the parameters, $\theta(t)$, of the `UpDown` model change over time. Fig. 9 shows example mappings for the cubic and the quadratic-like function, respectively. The estimated mappings are highly regular. Lastly, Figs. 10 and 11 show the time-evolutions of the model parameters for the cubic and the quadratic-like function for two different inflation factors. While different parameters show different dynamics, clear changes over time can be observed. In particular, $\theta_2(t)$ and $b_2(t)$ show strong changes. These parameters mostly control the behavior of the hidden high-dimensional state, $v$, as $\theta_3(t)$ is penalized significantly more in our model (see Sec.E) and consequently shows more moderate changes.

**Table 1:** Number of parameters for the simple function regression cubic and quadratic experiments.

| | | **Inflation factor** | | | | | |
|---|---|---|---|---|---|---|---|
| | **#Particles** | 4 | 8 | 16 | 32 | 64 | 128 |
| static/dynamic w/ particles | 2 | 28 | 52 | 100 | 196 | 388 | 772 |
| | 5 | 58 | 106 | 202 | 394 | 778 | 1,546 |
| | 15 | 158 | 286 | 542 | 1,054 | 2,078 | 4,126 |
| | 25 | 258 | 466 | 882 | 1,714 | 3,378 | 6,706 |
| dynamic direct | n/a | 153 | 461 | 1,557 | 5,669 | 21,573 | 84,101 |
| static direct | n/a | 37 | 105 | 337 | 1,185 | 4,417 | 17,025 |

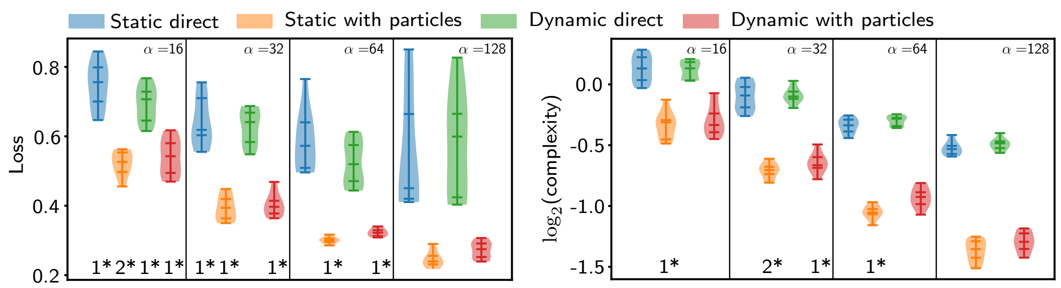


**Figure 6:** Function fit (15 particles) for cubic $y = x^3$ for 10 random initializations. *Left*: Test loss; *Right*: time-integral of $\log_2$ of the Frobenius norm complexity. Lower is better for both measures. * indicates number of removed outliers (outside the interquartile range (IQR) by $\geq 1.5\times$ IQR); $\alpha$ denotes the inflation factor.

**Spiral** Tab. 2 shows the number of parameters in each of the four formulations for the spiral experiment. This table complements the Table in Fig. 3 which only showed the number of parameters when using 15 particles.

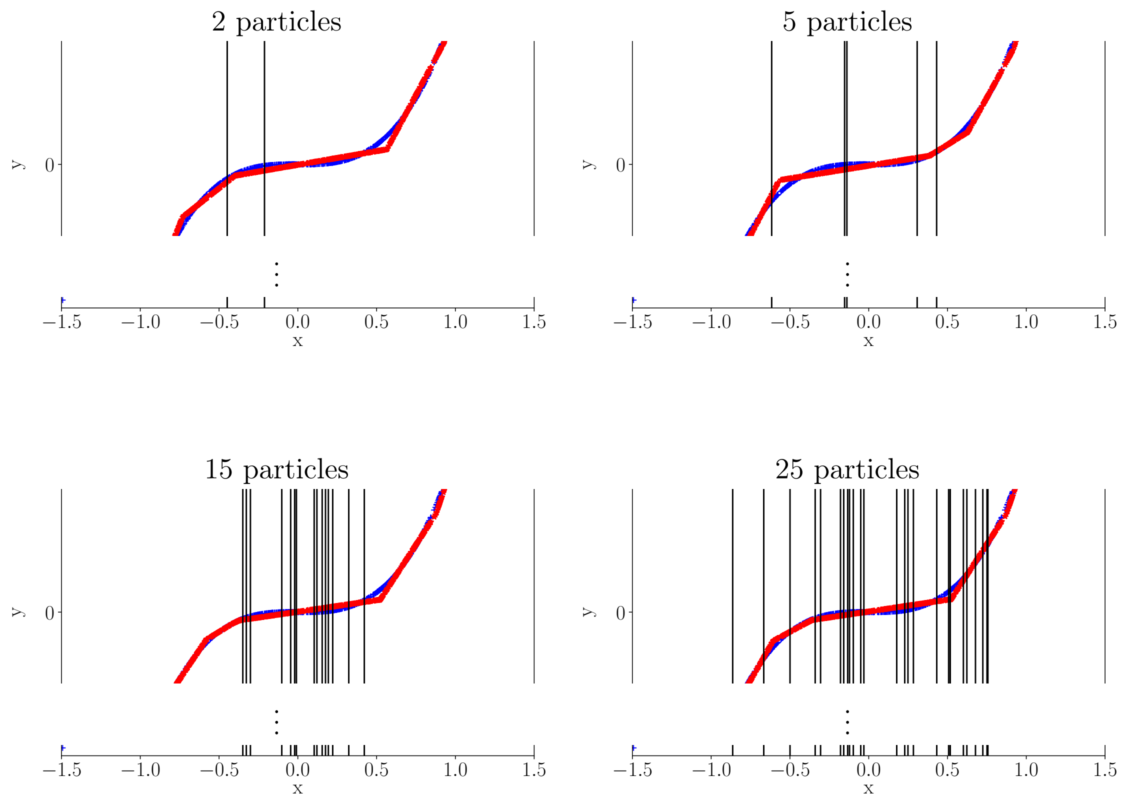


**Figure 7:** Fits for the cubic function with inflation factor 16 and for different numbers of particles. Vertical lines indicate particle positions after optimization. While subtle, the figures suggest that using more particles allows for better approximation of the function. This is confirmed by the test loss values in Figure 6 (bottom left).

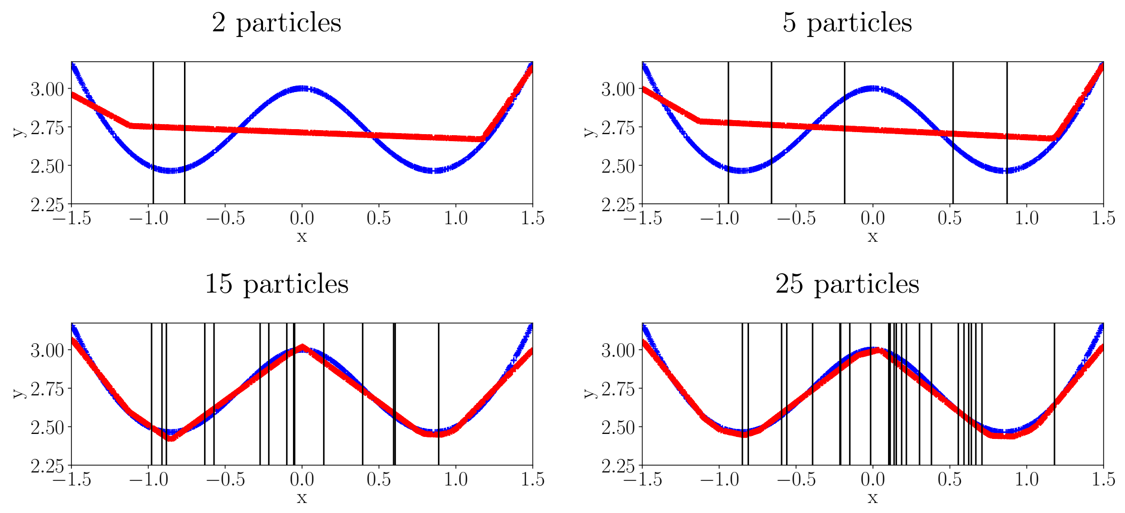


**Figure 8:** Fits for the quadratic-like function for inflation factor 16 with different numbers of particles. Vertical lines indicate particle positions after optimization. As this function is more complex than the cubic function 2 and 5 particles is not sufficient for a fit. But 15 and 25 particles result in a well-fitting approximation.

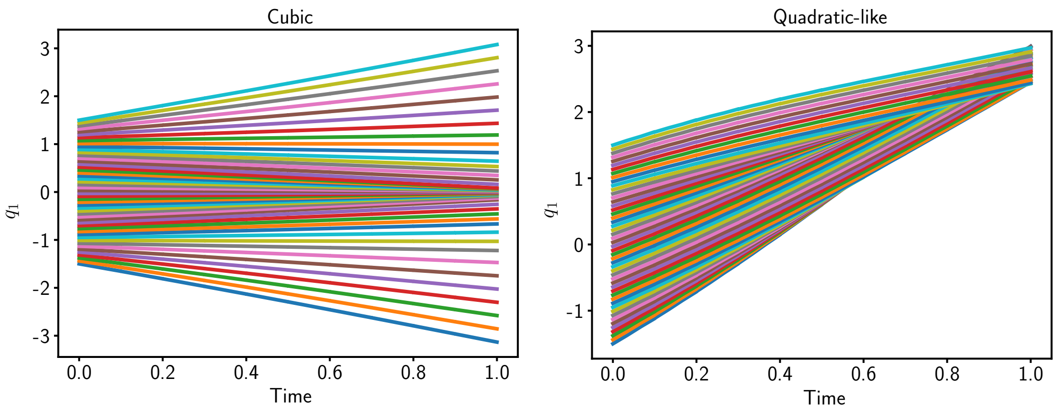


**Figure 9:** Mapping of the cubic function (left) and the quadratic-like function (right). As can be seen, the mappings are *highly regular*.

**Table 2:** Number of parameters for the spiral experiment.

| | | **Inflation factor** | | | |
|---|---|---|---|---|---|
| | **#Particles** | 16 | 32 | 64 | 128 |
| static/dynamic w/ particles | 15 | 1,116 | 2,172 | 4,284 | 8,508 |
| | 25 | 1,796 | 3,492 | 6,884 | 13,668 |
| | 50 | 3,496 | 6,792 | 13,384 | 26,568 |
| static direct | n/a | 1,282 | 4,610 | 17,410 | 67,586 |
| dynamic direct | n/a | 6,026 | 22,282 | 85,514 | 334,858 |

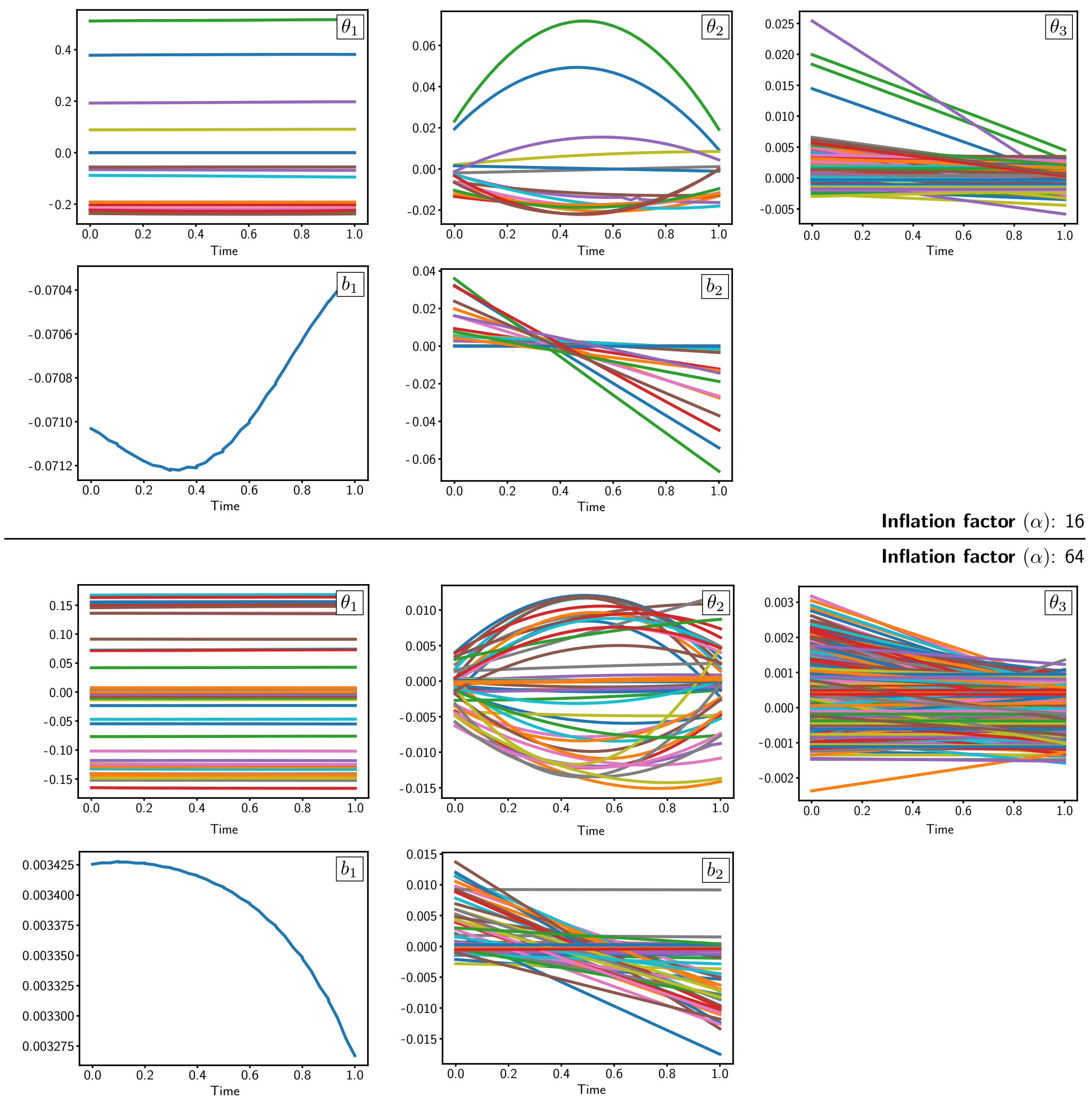


**Figure 10:** Weight evolution across time (i.e., continuous depth) for 15 particles when fitting the cubic function using the `UpDown` model: $\dot{\mathbf{x}}(t) = \theta_1(t)\sigma(\mathbf{v}(t)) + b_1(t)$, $\dot{\mathbf{v}} = \theta_2(t)\mathbf{x}(t) + b_2(t) + \theta_3(t)\sigma(\mathbf{v}(t))$. Results are for the dynamic with particles approach. Top: Inflation factor 16. Bottom: Inflation factor 64. Changes in parameter values can clearly be observed.

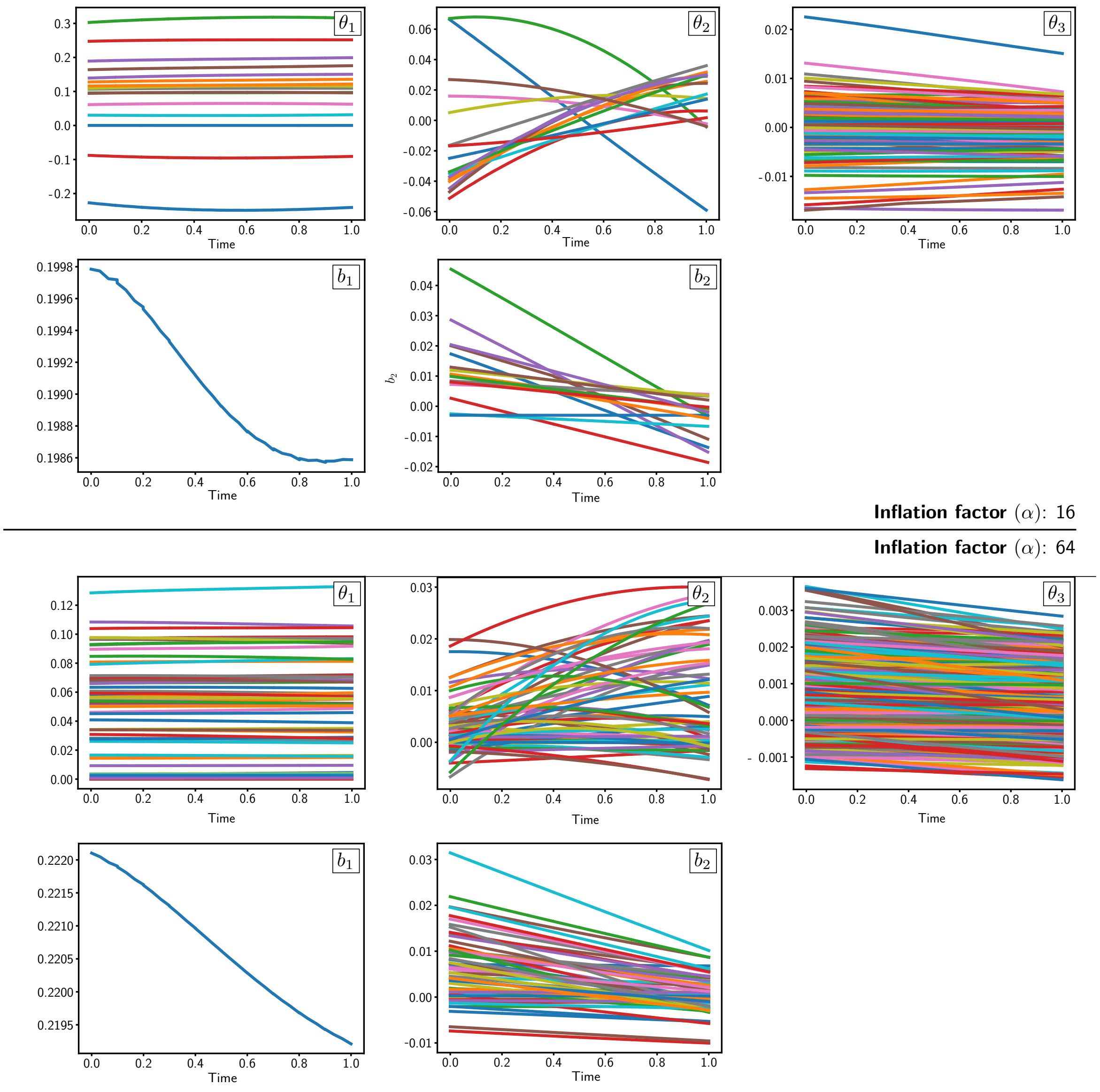


**Figure 11:** Weight evolution across time (i.e., continuous depth) for 15 particles when fitting the quadratic-like function using the `UpDown` model: $\dot{\mathbf{x}}(t) = \theta_1(t)\sigma(\mathbf{v}(t)) + b_1(t),\ \dot{\mathbf{v}} = \theta_2(t)\mathbf{x}(t) + b_2(t) + \theta_3(t)\sigma(\mathbf{v}(t))$. Results are for the dynamic with particles approach. Top: Inflation factor 16. Bottom: Inflation factor 64. Changes in parameter values can clearly be observed.