Heavy Ball Neural Ordinary Differential Equations

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

We propose heavy ball neural ordinary differential equations (HBNODEs), leveraging the continuous limit of the classical momentum accelerated gradient descent, to improve neural ODEs (NODEs) training and inference. HBNODEs have two properties that imply practical advantages over NODEs: (i) The adjoint state of an HBNODE also satisfies an HBNODE, accelerating both forward and backward ODE solvers, thus significantly reducing the number of function evaluations (NFEs) and improving the utility of the trained models. (ii) The spectrum of HBNODEs is well structured, enabling effective learning of long-term dependencies from complex sequential data. We verify the advantages of HBNODEs over NODEs on benchmark tasks, including image classification, learning complex dynamics, and sequential modeling. Our method requires remarkably fewer forward and backward NFEs, is more accurate, and learns long-term dependencies more effectively than the other ODE-based neural network models.

1 Introduction

Neural ordinary differential equations (NODEs) are a family of continuous-depth machine learning (ML) models whose forward and backward propagations rely on solving an ODE and its adjoint equation [4]. NODEs model the dynamics of hidden features $\boldsymbol{h}(t) \in \mathbb{R}^N$ using an ODE, which is parametrized by a neural network $f(\boldsymbol{h}(t), t, \theta) \in \mathbb{R}^N$ with learnable parameters θ, i.e.,

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta). \quad (1)$$

Starting from the input $\boldsymbol{h}(t_0)$, NODEs obtain the output $\boldsymbol{h}(T)$ by solving (1) for $t_0 \leq t \leq T$ with the initial value $\boldsymbol{h}(t_0)$, using a black-box numerical ODE solver. The number of function evaluations (NFEs) that the black-box ODE solver requires in a single forward pass is an analogue for the continuous-depth models [4] to the depth of networks in ResNets [17]. The loss between NODE prediction $\boldsymbol{h}(T)$ and the ground truth is denoted by $\mathcal{L}(\boldsymbol{h}(T))$; we update parameters θ using the following gradient [41]

$$\frac{d\mathcal{L}(\boldsymbol{h}(T))}{d\theta} = \int_{t_0}^{T} \boldsymbol{a}(t) \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \theta} dt, \quad (2)$$

where $\boldsymbol{a}(t) := \partial \mathcal{L} / \partial \boldsymbol{h}(t)$ is the adjoint state, which satisfies the adjoint equation

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t) \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}}. \quad (3)$$

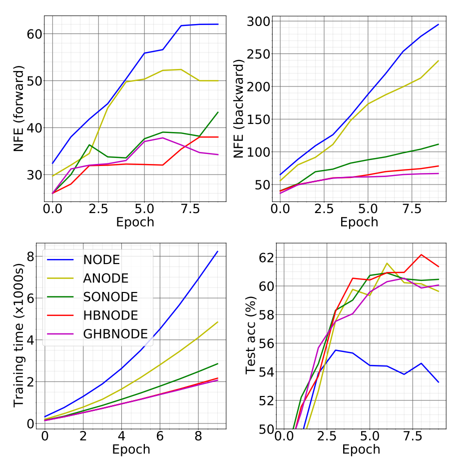

Figure 1: Contrasting NODE, ANODE, SONODE, HBNODE, and GHBNODE for CIFAR10 classification in NFEs, training time, and test accuracy. (Tolerance: 10^{-5}, see Sec. 5.2 for experimental details.)

NODEs are flexible in learning from irregularly-sampled sequential data and particularly suitable for learning complex dynamical systems [4, 44, 58, 37, 9, 24], which can be trained by efficient algorithms [42, 7, 59]. NODE-based continuous generative models have computational advantages over the classical normalizing flows [4, 15, 57, 13]. NODEs have also been generalized to neural stochastic differential equations, stochastic processes, and graph NODEs [22, 29, 39, 52, 21, 36]. The drawback of NODEs is also prominent. In many ML tasks, NODEs require very high NFEs in both training and inference, especially in high accuracy settings where a lower tolerance is needed. The NFEs increase rapidly with training; high NFEs reduce computational speed and accuracy of NODEs and can lead to blow-ups in the worst-case scenario [15, 10, 31, 37]. As an illustration, we train NODEs for CIFAR10 classification using the same model and experimental settings as in [10], except using a tolerance of 10^{-5}; Fig. 1 shows both forward and backward NFEs and the training time of different ODE-based models; we see that NFEs and computational times increase very rapidly for NODE, ANODE [10], and SONODE [37]. More results on the large NFE and degrading utility issues for different benchmark experiments are available in Sec. 5. Another issue is that NODEs often fail to effectively learn long-term dependencies in sequential data [27], discussed in Sec. 4.

1.1 Contribution

We propose heavy ball neural ODEs (HBNODEs), leveraging the continuous limit of the classical momentum accelerated gradient descent, to improve NODE training and inference. At the core of HBNODE is replacing the first-order ODE (1) with a heavy ball ODE (HBODE), i.e., a second-order ODE with an appropriate damping term. HBNODEs have two theoretical properties that imply practical advantages over NODEs:

- The adjoint equation used for training a HBNODE is also a HBNODE (see Prop. 1 and Prop. 2), accelerating both forward and backward propagation, thus significantly reducing both forward and backward NFEs. The reduction in NFE using HBNODE over existing benchmark ODE-based models becomes more aggressive as the error tolerance of the ODE solvers decreases.
- The spectrum of the HBODE is well-structured (see Prop. 4), alleviating the vanishing gradient issue in back-propagation and enabling the model to effectively learn long-term dependencies from sequential data.

To mitigate the potential blow-up problem in training HBNODEs, we further propose generalized HBNODEs (GHBNODEs) by integrating skip connections [18] and gating mechanisms [20] into the HBNODE. See Sec. 3 for details.

1.2 Organization

We organize the paper as follows: In Secs 2 and 3, we present our motivation, algorithm, and analysis of HBNODEs and GHBNODEs, respectively. We analyze the spectrum structure of the adjoint equation of HBNODEs/GHBNODEs in Sec. 4, which indicates that HBNODEs/GHBNODEs can learn long-term dependency effectively. We test the performance of HBNODEs and GHBNODEs on benchmark point cloud separation, image classification, learning dynamics, and sequential modeling in Sec. 5. We discuss more related work in Sec. 6, followed by concluding remarks. Technical proofs and more experimental details are provided in the appendix.

2 Heavy Ball Neural Ordinary Differential Equations

2.1 Heavy ball ordinary differential equation

Classical momentum method, a.k.a., the heavy ball method, has achieved remarkable success in accelerating gradient descent [40] and has significantly improved the training of deep neural networks [49]. As the continuous limit of the classical momentum method, heavy ball ODE (HBODE) has been studied in various settings and has been used to analyze the acceleration phenomenon of the momentum methods [26, 46]. For the ease of reading and completeness, we derive the HBODE from the classical momentum method. Starting from initial points $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$, gradient descent with classical momentum searches a minimum of the function $F(\boldsymbol{x})$ through the following iteration

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - s\nabla F(\boldsymbol{x}^k) + \beta(\boldsymbol{x}^k - \boldsymbol{x}^{k-1}), \tag{4}$$

where $s > 0$ is the step size and $0 \leq \beta < 1$ is the momentum hyperparameter. For any fixed step size s, let $\boldsymbol{m}^k := (\boldsymbol{x}^{k+1} - \boldsymbol{x}^k)/\sqrt{s}$, and let $\beta := 1 - \gamma\sqrt{s}$, where $\gamma \geq 0$ is another hyperparameter. Then we can rewrite (4) as

$$\boldsymbol{m}^{k+1} = (1 - \gamma\sqrt{s})\boldsymbol{m}^k - \sqrt{s}\nabla F(\boldsymbol{x}^k);\ \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \sqrt{s}\boldsymbol{m}^{k+1}. \tag{5}$$

Let $s \to 0$ in (5); we obtain the following system of first-order ODEs,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{m}(t); \ \frac{d\boldsymbol{m}(t)}{dt} = -\gamma \boldsymbol{m}(t) - \nabla F(\boldsymbol{x}(t)). \tag{6}$$

This can be further rewritten as a second-order heavy ball ODE (HBODE), which also models a damped oscillator,

$$\frac{d^2\boldsymbol{x}(t)}{dt^2} + \gamma \frac{d\boldsymbol{x}(t)}{dt} = -\nabla F(\boldsymbol{x}(t)). \tag{7}$$

We compare the dynamics of HBODE (7) and the following ODE limit of the gradient descent (GD)

$$\frac{d\boldsymbol{x}}{dt} = -\nabla F(\boldsymbol{x}). \tag{8}$$

In particular, we solve the ODEs (7) and (8) with $F(\boldsymbol{x})$ defined as a `Rosenbrock` [43] or `Beale` [14] function (see Appendix E.6 for experimental details). Fig. 2 shows that with the same numerical ODE solver, HBODE converges to the stationary point (marked by stars) faster than (8). The fact that HBODE can accelerate the dynamics of the ODE for a gradient system motivates us to propose HBNODE to accelerate forward propagation of NODE.

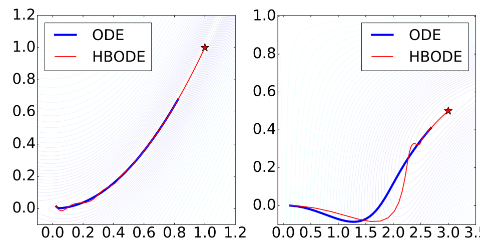

Figure 2: Comparing the trajectory of ODE and HBODE when $F(\boldsymbol{x})$ is the Rosenbrock (left) and Beale (right) functions.

2.2 Heavy ball neural ordinary differential equations

Similar to NODE, we parameterize $-\nabla F$ in (7) using a neural network $f(\boldsymbol{h}(t), t, \theta)$, resulting in the following HBNODE with initial position $\boldsymbol{h}(t_0)$ and momentum $\boldsymbol{m}(t_0) := d\boldsymbol{h}/dt(t_0)$,

$$\frac{d^2\boldsymbol{h}(t)}{dt^2} + \gamma \frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta), \tag{9}$$

where $\gamma \geq 0$ is the damping parameter, which can be set as a tunable or a learnable hyperparmater with positivity constraint. In the trainable case, we use $\gamma = \epsilon \cdot \text{sigmoid}(\omega)$ for a trainable $\omega \in \mathbb{R}$ and a fixed tunable upper bound ϵ (we set $\epsilon = 1$ below). According to (6), HBNODE (9) is equivalent to

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{m}(t); \quad \frac{d\boldsymbol{m}(t)}{dt} = -\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta). \tag{10}$$

Equation (9) (or equivalently, the system (10)) defines the forward ODE for the HBNODE, and we can use either the first-order (Prop. 2) or the second-order (Prop. 1) adjoint sensitivity method to update the parameter θ [37].

Proposition 1 (Adjoint equation for HBNODE)**.** *The adjoint state* $\boldsymbol{a}(t) := \partial\mathcal{L}/\partial\boldsymbol{h}(t)$ *for the HBNODE* (9) *satisfies the following HBODE with the same damping parameter* γ *as that in* (9)*,*

$$\frac{d^2\boldsymbol{a}(t)}{dt^2} - \gamma \frac{d\boldsymbol{a}(t)}{dt} = \boldsymbol{a}(t)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t), t, \theta). \tag{11}$$

Remark 1. *Note that we solve the adjoint equation* (11) *from time* $t = T$ *to* $t = t_0$ *in the backward propagation. By letting* $\tau = T - t$ *and* $\boldsymbol{b}(\tau) = \boldsymbol{a}(T - \tau)$*, we can rewrite* (11) *as follows,*

$$\frac{d^2\boldsymbol{b}(\tau)}{d\tau^2} + \gamma \frac{d\boldsymbol{b}(\tau)}{d\tau} = \boldsymbol{b}(\tau)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(T-\tau), T-\tau, \theta). \tag{12}$$

Therefore, the adjoint of the HBNODE is also a HBNODE and they have the same damping parameter.

We can also employ (10) and its adjoint for the forward and backward propagations, respectively.

Proposition 2 (Adjoint equations for the first-order HBNODE system)**.** *The adjoint states* $\boldsymbol{a_h}(t) := \partial\mathcal{L}/\partial\boldsymbol{h}(t)$ *and* $\boldsymbol{a_m}(t) := \partial\mathcal{L}/\partial\boldsymbol{m}(t)$ *for the first-order HBNODE system* (10) *satisfy*

$$\frac{d\boldsymbol{a_h}(t)}{dt} = -\boldsymbol{a_m}(t)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t), t, \theta); \quad \frac{d\boldsymbol{a_m}(t)}{dt} = -\boldsymbol{a_h}(t) + \gamma \boldsymbol{a_m}(t). \tag{13}$$

Remark 2. *Let $\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t) = d\boldsymbol{a}_{\boldsymbol{m}}(t)/dt$, then $\boldsymbol{a}_{\boldsymbol{m}}(t)$ and $\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t)$ satisfies the following first-order heavy ball ODE system*

$$\frac{d\boldsymbol{a}_{\boldsymbol{m}}(t)}{dt} = \tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t); \quad \frac{d\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t)}{dt} = \boldsymbol{a}_{\boldsymbol{m}}(t)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t), t, \theta) + \gamma\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t). \tag{14}$$

Note that we solve this system backward in time in back-propagation. Moreover, we have $\boldsymbol{a}_{\boldsymbol{h}}(t) = \gamma\boldsymbol{a}_{\boldsymbol{m}}(t) - \tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t)$.

Similar to [37], we use the coupled first-order HBNODE system (10) and its adjoint first-order HBNODE system (13) for practical implementation, since the entangled representation permits faster computation [37] of the gradients of the coupled ODE systems.

3 Generalized Heavy Ball Neural Ordinary Differential Equations

In this section, we propose a generalized version of HBNODE (GHBNODE), see (15), to mitigate the potential blow-up issue in training ODE-based models. In our experiments, we observe that $\boldsymbol{h}(t)$ of ANODEs [10], SONODEs [37], and HBNODEs (10) usually grows much faster than that of NODEs. The fast growth of $\boldsymbol{h}(t)$ can lead to finite-time blow up. As an illustration, we compare the performance of NODE, ANODE, SONODE, HBNODE, and GHBNODE on the Silverbox task as in [37]. The goal of the task is to learn the voltage of an electronic circuit that resembles a Duffing oscillator, where the input voltage $V_1(t)$ is used to predict the output $V_2(t)$. Similar to the setting in [37], we first augment ANODE by 1 dimension with 0-augmentation and augment SONODE, HBNODE, and GHBNODE with a dense network. We use a simple dense layer to parameterize f for all five models, with an extra input term for $V_1(t)$[1]. For both HBNODE and GHBNODE, we set the damping parameter γ to be $\mathrm{sigmoid}(-3)$. For GHBNODE (15) below, we set $\sigma(\cdot)$ to be the `hardtanh` function with bound $[-5, 5]$ and $\xi = \ln(2)$. The detailed architecture can be found in Appendix E. As shown in Fig. 3, compared to the vanilla NODE, the ℓ_2 norm of $\boldsymbol{h}(t)$ grows much faster when a higher order NODE is used, which leads to blow-up during training. Similar issues arise in the time series experiments (see Sec. 5.4), where SONODE blows up during long term integration in time, and HBNODE suffers from the same issue with some initialization.

To alleviate the problem above, we propose the following generalized HBNODE

$$\begin{aligned} \frac{d\boldsymbol{h}(t)}{dt} &= \sigma(\boldsymbol{m}(t)), \\ \frac{d\boldsymbol{m}(t)}{dt} &= -\gamma\boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta) - \xi\boldsymbol{h}(t), \end{aligned} \tag{15}$$

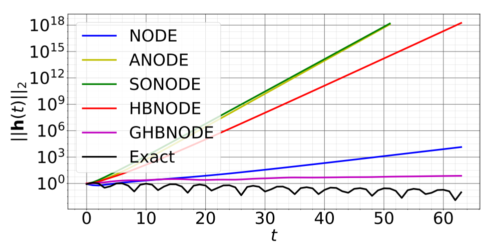

Figure 3: Contrasting $\boldsymbol{h}(t)$ for different models. $\boldsymbol{h}(t)$ in ANODE, SONODE, and HBNODE grows much faster than that in NODE. GHBNODE controls the growth of $\boldsymbol{h}(t)$ effectively when t is large.

where $\sigma(\cdot)$ is a nonlinear activation, which is set as $\tanh$ in our experiments. The positive hyperparameters $\gamma, \xi > 0$ are tunable or learnable. In the trainable case, we let $\gamma = \epsilon \cdot \mathrm{sigmoid}(\omega)$ as in HBNODE, and $\xi = \mathrm{softplus}(\chi)$ to ensure that $\gamma, \xi \geq 0$. Here, we integrate two main ideas into the design of GHBNODE: (i) We incorporate the gating mechanism used in LSTM [20] and GRU [6], which can suppress the aggregation of $\boldsymbol{m}(t)$; (ii) Following the idea of skip connection [18], we add the term $\xi\boldsymbol{h}(t)$ into the governing equation of $\boldsymbol{m}(t)$, which benefits training and generalization of GHBNODEs. Fig. 3 shows that GHBNODE can indeed control the growth of $\boldsymbol{h}(t)$ effectively.

Proposition 3 (Adjoint equations for GHBNODEs)**.** *The adjoint states $\boldsymbol{a}_{\boldsymbol{h}}(t) := \partial\mathcal{L}/\partial\boldsymbol{h}(t)$, $\boldsymbol{a}_{\boldsymbol{m}}(t) := \partial\mathcal{L}/\partial\boldsymbol{m}(t)$ for the GHBNODE* (15) *satisfy the following first-order ODE system*

$$\frac{\partial\boldsymbol{a}_{\boldsymbol{h}}(t)}{\partial t} = -\boldsymbol{a}_{\boldsymbol{m}}(t)\Big(\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t), t, \theta) - \xi\mathbf{I}\Big), \quad \frac{\partial\boldsymbol{a}_{\boldsymbol{m}}(t)}{\partial t} = -\boldsymbol{a}_{\boldsymbol{h}}(t)\sigma'(\boldsymbol{m}(t)) + \gamma\boldsymbol{a}_{\boldsymbol{m}}(t). \tag{16}$$

Though the adjoint state of the GHBNODE (16) does not satisfy the exact heavy ball ODE, based on our empirical study, it also significantly reduces the backward NFEs.

[1]Here, we exclude an $\boldsymbol{h}^3$ term that appeared in the original Duffing oscillator model because including it would result in finite-time explosion.

4 Learning long-term dependencies – Vanishing gradient

It is known that the vanishing and exploding gradients are two bottlenecks for training recurrent neural networks (RNNs) with long-term dependencies [2, 38] (see Appendix C for a brief review on the exploding and vanishing gradient issues in training RNNs). The exploding gradients issue can be effectively resolved via gradient clipping, training loss regularization, etc [38]. Thus in practice the vanishing gradient is the major issue for learning long-term dependencies [38]. As the continuous analogue of RNN, NODEs as well as their hybrid ODE-RNN models, may also suffer from vanishing in the adjoint state $\boldsymbol{a}(t) := \partial\mathcal{L}/\partial\boldsymbol{h}(t)$ [27]. When the vanishing gradient issue happens, $\boldsymbol{a}(t)$ goes to $\mathbf{0}$ quickly as $T-t$ increases, then $d\mathcal{L}/d\theta$ in (2) will be independent of these $\boldsymbol{a}(t)$. We have the following expressions for the adjoint states of the NODE and HBNODE (see Appendix C for detailed derivation):

- For NODE, we have

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_t} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T}\frac{\partial\boldsymbol{h}_T}{\partial\boldsymbol{h}_t} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T}\exp\Big\{-\int_T^t \frac{\partial f}{\partial\boldsymbol{h}}(\boldsymbol{h}(s), s, \theta)ds\Big\}. \tag{17}$$

- For GHBNODE[2], from (13) we can derive

$$\begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_t} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_t}\end{bmatrix} = \begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_T}\end{bmatrix}\begin{bmatrix}\frac{\partial\boldsymbol{h}_T}{\partial\boldsymbol{h}_t} & \frac{\partial\boldsymbol{h}_T}{\partial\boldsymbol{m}_t}\\ \frac{\partial\boldsymbol{m}_T}{\partial\boldsymbol{h}_t} & \frac{\partial\boldsymbol{m}_T}{\partial\boldsymbol{m}_t}\end{bmatrix} = \begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_T}\end{bmatrix}\exp\Big\{\underbrace{-\int_T^t \begin{bmatrix}\mathbf{0} & \frac{\partial\sigma}{\partial\boldsymbol{m}}\\ \frac{\partial f}{\partial\boldsymbol{h}} - \xi\mathbf{I} & -\gamma\boldsymbol{I}\end{bmatrix}ds}_{:=\boldsymbol{M}}\Big\}. \tag{18}$$

Note that the matrix exponential is directly related to its eigenvalues. By Schur decomposition, there exists an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{U}$, where the diagonal entries of $\boldsymbol{U}$ are eigenvalues of $\boldsymbol{Q}$ ordered by their real parts, such that

$$-\boldsymbol{M} = \boldsymbol{Q}\boldsymbol{U}\boldsymbol{Q}^\top \Longrightarrow \exp\{-\boldsymbol{M}\} = \boldsymbol{Q}\exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top. \tag{19}$$

Let $\boldsymbol{v}^\top := \begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_T}\end{bmatrix}\boldsymbol{Q}$, then (18) can be rewritten as

$$\begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_t} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_t}\end{bmatrix} = \begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_T}\end{bmatrix}\exp\{-\boldsymbol{M}\} = \begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_T}\end{bmatrix}\boldsymbol{Q}\exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top = \boldsymbol{v}^\top\exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top. \tag{20}$$

By taking the ℓ_2 norm in (20) and dividing both sides by $\big\|\begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_T}\end{bmatrix}\big\|_2$, we arrive at

$$\frac{\big\|\begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_t} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_t}\end{bmatrix}\big\|_2}{\big\|\begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_T}\end{bmatrix}\big\|_2} = \frac{\big\|\boldsymbol{v}^\top\exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top\big\|_2}{\|\boldsymbol{v}^\top\boldsymbol{Q}^\top\|_2} = \frac{\big\|\boldsymbol{v}^\top\exp\{\boldsymbol{U}\}\big\|_2}{\|\boldsymbol{v}\|_2} = \big\|\boldsymbol{e}^\top\exp\{\boldsymbol{U}\}\big\|_2, \tag{21}$$

i.e., $\big\|\begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_t} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_t}\end{bmatrix}\big\|_2 = \big\|\boldsymbol{e}^\top\exp\{\boldsymbol{U}\}\big\|_2\big\|\begin{bmatrix}\frac{\partial\mathcal{L}}{\partial\boldsymbol{h}_T} & \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}_T}\end{bmatrix}\big\|_2$ where $\boldsymbol{e} = \boldsymbol{v}/\|\boldsymbol{v}\|_2$.

Proposition 4. *The eigenvalues of $-\boldsymbol{M}$ can be paired so that the sum of each pair equals $(t-T)\gamma$.*

For a given constant $a > 0$, we can group the upper triangular matrix $\exp\{\boldsymbol{U}\}$ as follows

$$\exp\{\boldsymbol{U}\} := \begin{bmatrix}\exp\{\boldsymbol{U}_L\} & \boldsymbol{P}\\ \mathbf{0} & \exp\{\boldsymbol{U}_V\}\end{bmatrix}, \tag{22}$$

where the diagonal of $\boldsymbol{U}_L$ ($\boldsymbol{U}_V$) contains eigenvalues of $-\boldsymbol{M}$ that are no less (greater) than $(t-T)a$. Then, we have $\|\boldsymbol{e}^\top\exp\{\boldsymbol{U}\}\|_2 \geq \|\boldsymbol{e}_L^\top\exp\{\boldsymbol{U}_L\}\|_2$ where the vector $\boldsymbol{e}_L$ denotes the first m columns of $\boldsymbol{e}$ with m be the number of columns of $\boldsymbol{U}_L$. By choosing $0 \leq \gamma \leq 2a$, for every pair of eigenvalues of $-\boldsymbol{M}$ there is at least one eigenvalue whose real part is no less than $(t-T)a$. Therefore, $\exp\{\boldsymbol{U}_L\}$ decays at a rate at most $(t-T)a$, and the dimension of $\boldsymbol{U}_L$ is at least $N\times N$. We avoid exploding gradients by clipping the ℓ_2 norm of the adjoint states similar to that used for training RNNs.

In contrast, all eigenvalues of the matrix $\int_T^t \partial f/\partial\boldsymbol{h}ds$ in (17) for NODE can be very positive or negative, resulting in exploding or vanishing gradients. As an illustration, we consider the benchmark Walker2D kinematic simulation task that requires learning long-term dependencies effectively [27, 3]. We train ODE-RNN [44] and (G)HBNODE-RNN on this benchmark dataset, and the detailed experimental settings are provided in Sec. 5.4. Figure 4 plots $\|\partial\mathcal{L}/\partial\boldsymbol{h}_t\|_2$ for ODE-RNN and $\|[\partial\mathcal{L}/\partial\boldsymbol{h}_t\ \partial\mathcal{L}/\partial\boldsymbol{m}_t]\|_2$ for (G)HBNODE-RNN, showing that the adjoint state of ODE-RNN vanishes quickly, while that of (G)HBNODE-RNN does not vanish even when the gap between T and t is very large.

[2]HBNODE can be seen as a special GHBNODE with $\xi = 0$ and σ be the identity map.

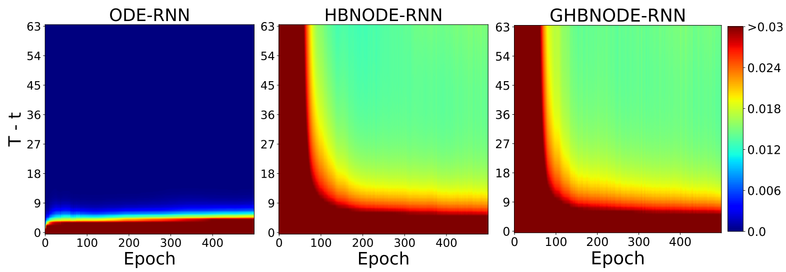

Figure 4: Plot of the the ℓ_2-norm of the adjoint states for ODE-RNN and (G)HBNODE-RNN back-propagated from the last time stamp. The adjoint state of ODE-RNN vanishes quickly when the gap between the final time T and intermediate time t becomes larger, while the adjoint states of (G)HBNODE-RNN decays much more slowly. This implies that (G)HBNODE-RNN is more effective in learning long-term dependency than ODE-RNN.

5 Experimental Results

In this section, we compare the performance of the proposed HBNODE and GHBNODE with existing ODE-based models, including NODE [4], ANODE [10], and SONODE [37] on the benchmark point cloud separation, image classification, learning dynamical systems, and kinematic simulation. For all the experiments, we use Adam [25] as the benchmark optimization solver (the learning rate and batch size for each experiment are listed in Table 1) and Dormand–Prince-45 as the numerical ODE solver. For HBNODE and GHBNODE, we set $\gamma = \mathrm{sigmoid}(\theta)$, where θ is a trainable weight initialized as $\theta = -3$. The network architecture used to parameterize $f(\boldsymbol{h}(t), t, \theta)$ for each experiment below are described in Appendix E. All experiments are conducted on a server with 2 NVIDIA Titan Xp GPUs.

Table 1: The batch size and learning rate for different datasets.

Dataset	Point Cloud	MNIST	CIFAR10	Plane Vibration	Walker2D
Batch Size	50	64	64	64	256
Learning Rate	0.01	0.001	0.001	0.0001	0.003

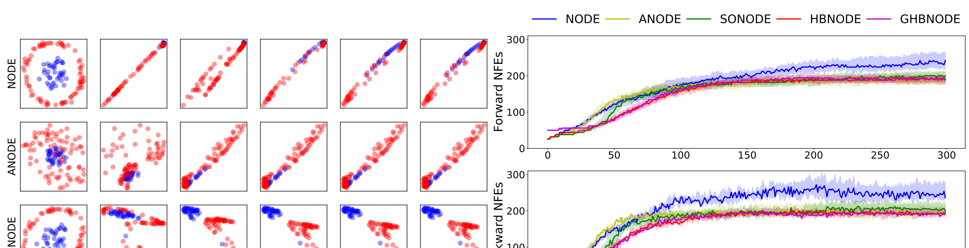

Figure 5: Comparison between NODE, ANODE, SONODE, HBNODE, and GHBNODE for two-dimensional point cloud separation. HBNODE and GHBNODE converge better and require less NFEs in both forward and backward propagation than the other benchmark models.

5.1 Point cloud separation

In this subsection, we consider the two-dimensional point cloud separation benchmark. A total of 120 points are sampled, in which 40 points are drawn uniformly from the circle $\|\boldsymbol{r}\| < 0.5$, and 80 points are drawn uniformly from the annulus $0.85 < \|\boldsymbol{r}\| < 1.0$. This experiment aims to learn effective features to classify these two point clouds. Following [10], we use a three-layer neural network to parameterize the right-hand side of each ODE-based model, integrate the ODE-based model from $t_0 = 0$ to $T = 1$, and pass the integration results to a dense layer to generate the classification results. We set the size of hidden layers so that the models have similar sizes, and the number of

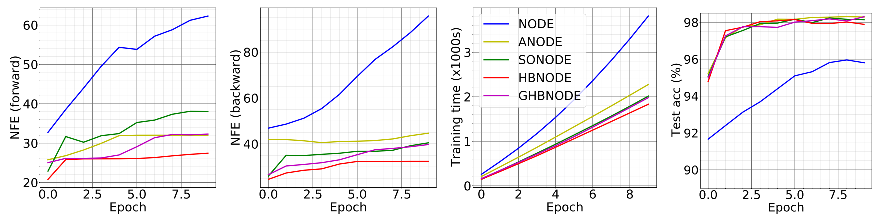

Figure 6: Contrasting NODE [4], ANODE [10], SONODE [37], HBNODE, and GHBNODE for MNIST classification in NFE, training time, and test accuracy. (Tolerance: 10^{-5}).

parameters of NODE, ANODE, SONODE, HBNODE, and GHBNODE are 525, 567, 528, 568, and 568, respectively. To avoid the effects of numerical error of the black-box ODE solver we set tolerance of ODE solver to be 10^{-7}. Figure 5 plots a randomly selected evolution of the point cloud separation for each model; we also compare the forward and backward NFEs and the training loss of these models (100 independent runs). HBNODE and GHBNODE improve training as the training loss consistently goes to zero over different runs, while ANODE and SONODE often get stuck at local minima, and NODE cannot separate the point cloud since it preserves the topology [10].

5.2 Image classification

We compare the performance of HBNODE and GHBNODE with the existing ODE-based models on MNIST and CIFAR10 classification tasks using the same setting as in [10]. We parameterize $f(\boldsymbol{h}(t), t, \theta)$ using a 3-layer convolutional network for each ODE-based model, and the total number of parameters for each model is listed in Table 2. For a given input image of the size $c \times h \times w$, we first augment the number of channel from c to $c + p$ with the augmentation dimension p dependent on each method[3]. Moreover, for SONODE, HBNODE and GHBNODE, we further include velocity or momentum with the same shape as the augmented state.

Table 2: The number of parameters for each models for image classification.

Model	NODE	ANODE	SONODE	HBNODE	GHBNODE
#Params (MNIST)	85,315	85,462	86,179	85,931	85,235
#Params (CIFAR10)	173,611	172,452	171,635	172,916	172,916

NFEs. As shown in Figs. 1 and 6, the NFEs grow rapidly with training of the NODE, resulting in an increasingly complex model with reduced performance and the possibility of blow up. Input augmentation has been verified to effectively reduce the NFEs, as both ANODE and SONODE require fewer forward NFEs than NODE for the MNIST and CIFAR10 classification. However, input augmentation is less effective in controlling their backward NFEs. HBNODE and GHBNODE require much fewer NFEs than the existing benchmarks, especially for backward NFEs. In practice, reducing NFEs implies reducing both training and inference time, as shown in Figs. 1 and 6.

Accuracy. We also compare the accuracy of different ODE-based models for MNIST and CIFAR10 classification. As shown in Figs. 1 and 6, HBNODE and GHBNODE have slightly better classification accuracy than the other three models; this resonates with the fact that less NFEs lead to simpler models which generalize better [10, 37].

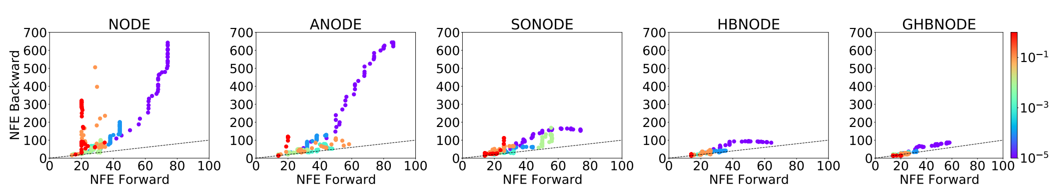

Figure 7: NFE vs. tolerance (shown in the colorbar) for training ODE-based models for CIFAR10 classification. Both forward and backward NFEs of HBNODE and GHBNODE grow much more slowly than that of NODE, ANODE, and SONODE; especially the backward NFEs. As the tolerance decreases, the advantage of HBNODE and GHBNODE in reducing NFEs becomes more significant.

[3]We set $p = 0, 5, 4, 4, 5/0, 10, 9, 9, 9$ on MNIST/CIFAR10 for NODE, ANODE, SONODE, HBNODE, and GHBNODE, respectively.

NFEs vs. tolerance. We further study the NFEs for different ODE-based models under different tolerances of the ODE solver using the same approach as in [4]. Figure 7 depicts the forward and backward NFEs for different models under different tolerances. We see that (i) both forward and backward NFEs grow quickly when tolerance is decreased, and HBNODE and GHBNODE require much fewer NFEs than other models; (ii) under different tolerances, the backward NFEs of NODE, ANODE, and SONODE are much larger than the forward NFEs, and the difference becomes larger when the tolerance decreases. In contrast, the forward and backward NFEs of HBNODE and GHBNODE scale almost linearly with each other. This reflects that the advantage in NFEs of (G)HBNODE over the benchmarks become more significant when a smaller tolerance is used.

5.3 Learning dynamical systems from irregularly-sampled time series

In this subsection, we learn dynamical systems from experimental measurements. In particular, we use the ODE-RNN framework [4, 44], with the recognition model being set to different ODE-based models, to study the vibration of an airplane dataset [35]. The dataset was acquired, from time 0 to 73627, by attaching a shaker underneath the right wing to provide input signals, and 5 attributes are recorded per time stamp; these attributes include voltage of input signal, force applied to aircraft, and acceleration at 3 different spots of the airplane. We randomly take out 10% of the data to make the time series irregularly-sampled. We use the first 50% of data as our train set, the next 25% as validation set, and the rest as test set. We divide each set into non-overlapping segments of consecutive 65 time stamps of the irregularly-sampled time series, with each input instance consisting of 64 time stamps of the irregularly-sampled time series, and we aim to forecast 8 consecutive time stamps starting from the last time stamp of the segment. The input is fed though the the hybrid methods in a recurrent fashion; by changing the time duration of the last step of the ODE integration, we can forecast the output in the different time stamps. The output of the hybrid method is passed to a single dense layer to generate the output time series. In our experiments, we compare different ODE-based models hybrid with RNNs. The ODE of each model is parametrized by a 3-layer network whereas the RNN is parametrized by a simple dense network; the total number of parameters for ODE-RNN, ANODE-RNN, SONODE-RNN, HBNODE-RNN, and GHBNODE-RNN with 16, 22, 14, 15, 15 augmented dimensions are 15,986, 16,730, 16,649, 16,127, and 16,127, respectively. To avoid potential error due to the ODE solver, we use a tolerance of 10^{-7}.

In training those hybrid models, we regularize the models by penalizing the L2 distance between the RNN output and the values of the next time stamp. Due to the second-order natural of the underlying dynamics [37], ODE-RNN and ANODE-RNN learn the dynamics very poorly with much larger training and test losses than the other models even they take smaller NFEs. HBNODE-RNN and GHBNODE-RNN give better prediction than SONODE-RNN using less backward NFEs.

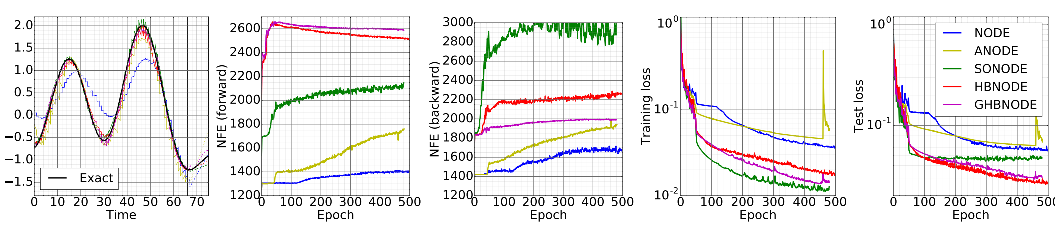

Figure 8: Contrasting ODE-RNN, ANODE-RNN, SONODE-RNN, HBNODE-RNN, and GHBNODE-RNN for learning a vibrational dynamical system. Left most: The learned curves of each model vs. the ground truth (Time: <66 for training, 66-75 for testing).

5.4 Walker2D kinematic simulation

In this subsection, we evaluate the performance of HBNODE-RNN and GHBNODE-RNN on the Walker2D kinematic simulation task, which requires learning long-term dependency effectively [27]. The dataset [3] consists of a dynamical system from kinematic simulation of a person walking from a pre-trained policy, aiming to learn the kinematic simulation of the MuJoCo physics engine [51]. The dataset is irregularly-sampled where 10% of data are removed from the simulation. Each input is consisted of 64 time stamps and fed though the the hybrid methods in a recurrent fashion, and the outputs of hybrid methods is passed to a single dense layer to generate the output time series. The target is to provide auto-regressive forecast so that the output time series is as close as the input sequence shifted 1 time stamp to the right. We compare ODE-RNN (with 7 augmentation), ANODE-RNN (with 7 ANODE style augmentation), HBNODE-RNN (with 7 augmentation), and

GHBNODE-RNN (with 7 augmentation) [4]. The RNN is parametrized by a 3-layer network whereas the ODE is parametrized by a simple dense network. The number of parameters of the above four models are 8,729, 8,815, 8,899, and 8,899, respectively. In Fig. 9, we compare the performance of the above four models on the Walker2D benchmark; HBNODE-RNN and GHBNODE-RNN not only require significantly less NFEs in both training (forward and backward) and in testing than ODE-RNN and ANODE-RNN, but also have much smaller training and test losses.

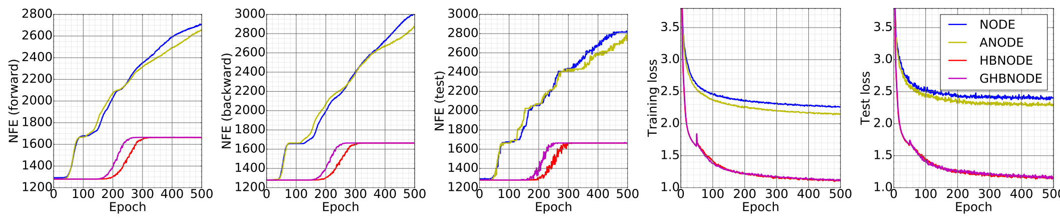

Figure 9: Contrasting ODE-RNN, ANODE-RNN, SONODE-RNN, HBNODE-RNN, and GHBNODE-RNN for the Walker-2D kinematic simulation.

6 Related Work

Reducing NFEs in training NODEs. Several techniques have been developed to reduce the NFEs for the forward solvers in NODEs, including weight decay [15], input augmentation [10], regularizing the learning dynamics [13], high-order ODE [37], data control [31], and depth-variance [31]. HBNODEs can reduce both forward and backward NFEs at the same time.

Second-order ODE accelerated dynamics. It has been noticed in both optimization and sampling communities that second-order ODEs with an appropriate damping term, e.g., the classical momentum and Nesterov's acceleration in discrete regime, can significantly accelerate the first-order gradient dynamics (gradient descent), e.g., [40, 34, 5, 48, 55]. Also, these second-order ODEs have been discretized via some interesting numerical schemes to design fast optimization schemes, e.g., [47].

Learning long-term dependencies. Learning long-term dependency is one of the most important goals for learning from sequential data. Most of the existing works focus on mitigating exploding or vanishing gradient issues in training RNNs, e.g., [1, 56, 23, 54, 32, 19, 50]. Attention-based models are proposed for learning on sequential data concurrently with the effective accommodation of learning long-term dependency [53, 8]. Recently, NODEs have been integrated with long-short term memory model [20] to learn long-term dependency for irregularly-sampled time series [27]. HBNODEs directly enhance learning long-term dependency from sequential data.

Momentum in neural network design. As a line of orthogonal work, the momentum has also been studied in designing neural network architecture, e.g., [33, 50, 45], which can also help accelerate training and learn long-term dependencies. These techniques can be considered as changing the neural network f in (1). We leave the synergistic integration of adding momentum to f with our work on changing the left-hand side of (1) as a future work.

ResNet-style models. Interpreting ResNet as an ODE model has been an interesting research direction [11, 16], which has lead to interesting neural network architectures and analysis from the numerical ODE solvers and differential equation theory viewpoints, e.g., [16, 30, 28].

7 Concluding Remarks

We proposed HBNODEs to reduce the NFEs in solving both forward and backward ODEs, which also improve generalization performance over the existing benchmark models. Moreover, HBNODEs alleviate vanishing gradients in training NODEs, making HBNODEs able to learn long-term dependency effectively from sequential data. In the optimization community, Nesterov acceleration [34] is also a famous algorithm for accelerating gradient descent, that achieves an optimal convergence rate for general convex optimization problems. The ODE counterpart of the Nesterov's acceleration corresponds to (9) with γ being replaced by a time-dependent damping parameter, e.g., $t/3$ [48]. The adjoint equation of the Nesterov's ODE [48] is no longer a Nesterov's ODE. We notice that directly using Nesterov's ODE cannot improve the performance of the vanilla neural ODE. How to integrate Nesterov's ODE with neural ODE is an interesting future direction.

[4]Here, we do not compare with SONODE-RNN since SONODE has some initialization problem on this dataset, and the ODE solver encounters failure due to exponential growth over time. This issue is originally tackled by re-initialization [37]. We re-initialized SONODE 100 times; all failed due to initialisation problems.

References

[1] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. cite arxiv:1606.01540.

[4] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6572–6583, 2018.

[5] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *International conference on machine learning*, pages 1683–1691, 2014.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] Talgat Daulbaev, Alexandr Katrutsa, Larisa Markeeva, Julia Gusak, Andrzej Cichocki, and Ivan Oseledets. Interpolation technique to speed up gradients propagation in neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16689–16700. Curran Associates, Inc., 2020.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Jianzhun Du, Joseph Futoma, and Finale Doshi-Velez. Model-based reinforcement learning for semi-markov decision processes with neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19805–19816. Curran Associates, Inc., 2020.

[10] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

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

[12] Jeffrey L Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[13] Chris Finlay, Joern-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ODE: the world of Jacobian and kinetic regularization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3154–3164. PMLR, 13–18 Jul 2020.

[14] Amilcar dos Santos Gonçalves. A Version of Beale's Method Avoiding the Free-Variables. In *Proceedings of the 1971 26th Annual Conference*, ACM '71, page 433–441, New York, NY, USA, 1971. Association for Computing Machinery.

[15] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019.

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

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[19] Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled Cayley transform. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1969–1978, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[21] Zijie Huang, Yizhou Sun, and Wei Wang. Learning continuous system dynamics from irregularly-sampled partial observations. In *Advances in Neural Information Processing Systems*, 2020.

[22] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[23] Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1733–1741. JMLR. org, 2017.

[24] Patrick Kidger, James Morrill, James Foster, and Terry J. Lyons. Neural controlled differential equations for irregular time series. In *NeurIPS*, 2020.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Nikola B. Kovachki and Andrew M. Stuart. Continuous time analysis of momentum methods. *Journal of Machine Learning Research*, 22(17):1–40, 2021.

[27] Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series. *arXiv preprint arXiv:2006.04418*, 2020.

[28] Qianxiao Li, Ting Lin, and Zuowei Shen. Deep learning via dynamical systems: An approximation perspective. *arXiv preprint arXiv:1912.10382*, 2019.

[29] Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3870–3882. PMLR, 26–28 Aug 2020.

[30] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018.

[31] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asma. Dissecting neural odes. In *34th Conference on Neural Information Processing Systems, NeurIPS 2020*. The Neural Information Processing Systems, 2020.

[32] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2401–2409. JMLR. org, 2017.

[33] Thomas Moreau and Joan Bruna. Understanding the learned iterative soft thresholding algorithm with matrix factorization. *arXiv preprint arXiv:1706.01338*, 2017.

[34] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. Akad. Nauk Sssr*, volume 269, pages 543–547, 1983.

[35] Jean-Philippe Noël and M Schoukens. F-16 aircraft benchmark based on ground vibration test data. In *2017 Workshop on Nonlinear System Identification Benchmarks*, pages 19–23, 2017.

[36] Alexander Norcliffe, Cristian Bodnar, Ben Day, Jacob Moss, and Pietro Liò. Neural {ode} processes. In *International Conference on Learning Representations*, 2021.

[37] Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Liò. On second order behaviour in augmented neural odes. In *Advances in Neural Information Processing Systems*, 2020.

[38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[39] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

[40] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[41] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

[42] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. Snode: Spectral discretization of neural odes for system identification. In *International Conference on Learning Representations*, 2020.

[43] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 01 1960.

[44] Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[45] Michael E. Sander, Pierre Ablin, Mathieu Blondel, and Gabriel Peyré. Momentum residual neural networks. *arXiv preprint arXiv:2102.07870*, 2021.

[46] Bin Shi, Simon S. Du, Michael I. Jordan, and Weijie J. Su. Understanding the acceleration phenomenon via high-resolution differential equations. *arXiv preprint arXiv:1810.08907*, 2018.

[47] Bin Shi, Simon S Du, Weijie Su, and Michael I Jordan. Acceleration via symplectic discretization of high-resolution differential equations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[48] Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pages 2510–2518, 2014.

[49] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

[50] Tan M. Nguyen and Richard G. Baraniuk and Andrea L. Bertozzi and Stanley J. Osher and Bao Wang. MomentumRNN: Integrating momentum into recurrent neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9154–9164, 2020.

[51] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[52] Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[54] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3570–3578. JMLR. org, 2017.

[55] Ashia C. Wilson, Benjamin Recht, and Michael I. Jordan. A Lyapunov Analysis of Momentum Methods in Optimization. *arXiv preprint arXiv:1611.02635*, 2018.

[56] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

[57] Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. ODE2VAE: Deep generative second order ODEs with Bayesian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[58] Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E Gonzalez, Kurt Keutzer, Michael W Mahoney, and George Biros. ANODEV2: A Coupled Neural ODE Framework. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[59] Juntang Zhuang, Nicha C Dvornek, sekhar tatikonda, and James s Duncan. {MALI}: A memory efficient and reverse accurate integrator for neural {ode}s. In *International Conference on Learning Representations*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 4.1.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 4.1.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Supplementary Materials
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]