Improving Neural Ordinary Differential Equations with Nesterov's Accelerated Gradient Method

Nghia H. Nguyen*
FPT Software AI Center
Ha Noi, Vietnam
nghianhh1@fsoft.com.vn

Tan M. Nguyen*
Department of Mathematics
University of California, Los Angeles
tanmnguyen89@ucla.edu

Huyen K. Vo
FPT Software AI Center
Ha Noi, Vietnam
huyenvtk1@fsoft.com.vn

Stanley J. Osher
Department of Mathematics
University of California, Los Angeles
sjo@math.ucla.edu

Thieu N. Vo
Ton Duc Thang University and FPT Software Ho Chi Minh
Ho Chi Minh City, Vietnam
vongocthieu@tdtu.edu.vn

Abstract

We propose the Nesterov neural ordinary differential equations (NesterovNODEs), whose layers solve the second-order ordinary differential equations (ODEs) limit of Nesterov's accelerated gradient (NAG) method, and a generalization called GNesterovNODEs. Taking the advantage of the convergence rate $\mathcal{O}(1/k^2)$ of the NAG scheme, GNesterovNODEs speed up training and inference by reducing the number of function evaluations (NFEs) needed to solve the ODEs. We also prove that the adjoint state of a GNesterovNODEs also satisfies a GNesterovNODEs, thus accelerating both forward and backward ODE solvers and allowing the model to be scaled up for large-scale tasks. We empirically corroborate the advantage of GNesterovNODEs on a wide range of practical applications, including point cloud separation, image classification, and sequence modeling. Compared to NODEs, GNesterovNODEs require a significantly smaller number of NFEs while achieving better accuracy across our experiments.

1 Introduction

Dynamical systems have been recently integrated into deep neural networks for modeling high-dimensional data. The advantage of this approach is that well-developed mathematical modeling techniques from dynamical systems can be employed to improve neural networks. Along this research direction, the correspondence between residual networks, a popular class of neural networks with skip connections, and the numerical solution of ordinary differential equations (ODEs) have been vastly studied in [15, 57, 47, 4]. The resulting Neural ODEs (NODEs) model when taking the the discretization step to zero have shown great promises in a wide range of applications including scientific discovery [24, 62], irregular time series modeling [46, 5], mean-field games [48], and generative modeling [13, 61]. NODEs model the dynamics of hidden state $\boldsymbol{h}(t) \in \mathbb{R}^N$ in a neural

* Co-first authors. Please correspond to: nghianhh1@fsoft.com.vn or tanmnguyen89@ucla.edu or vongocthieu@tdtu.edu.vn

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

network by an ODE

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta), \;\; \boldsymbol{h}(0) = h(t_0), \tag{1}$$

where function f captures the dynamics and is chosen to be a neural network with parameters θ that are learned from the data. Starting from the input $h(t_0)$ at the initial time t_0, NODEs compute the output $h(T)$ at time T by solving the Initial Value Problem in Eq. (1) for some time $T \geq t_0$. NODEs are trained by optimizing the loss $L(\boldsymbol{h}(T))$ between the prediction $\boldsymbol{h}(T)$ and the ground truth where the parameters θ are updated using the following gradient [42]

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

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

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

NODEs solve both the ODE Eq. (1) in its forward pass and the ODEs (2) and (3) in its backward pass using black-box numerical ODE solvers. The literature refers to this approach as either the continuous adjoint method or optimise-then-discretise [21]. The number of function evaluations (NFEs) that these solvers need in a single forward and backward pass is among the main factors that decide the computational efficiency of the model, i.e. how fast the model is. Unfortunately, in many applications, NODEs require high NFEs in both training and inference, especially when the error tolerances of the solvers are set to small values for obtaining high accuracy. Furthermore, the NFEs increase rapidly when training progresses. High NFEs deteriorate the efficiency of NODEs, reduce the accuracy of the trained model, and results in instability during training, making it difficult to scale up the models to large-scale tasks [14, 8, 30, 37, 11].

1.1 Contribution

We propose the Nesterov Neural ODEs (NesterovNODEs) that leverage the continuous limit of the Nesterov's accelerated gradient (NAG) descent [52] and the convergence rate $\mathcal{O}(1/k^2)$ of the NAG scheme to enhance NODE training and inference. Our contributions are four-fold:

1. We formulate the NesterovNODE that solves Nesterov ODEs, i.e. second-order ODEs with a time-dependent damping term, instead of first-order ODEs (1). To improve computational efficiency of the model, we convert these second-order ODEs into equivalent systems of first-order differential-algebraic equations that are solved in both forward and backward propagations of the NesterovNODE.
2. To eliminate the potential blow-up problem in training NesterovNODEs, we further develop the Generalized NesterovNODEs (GNesterovNODEs) by introducing skip connections [16] and gating mechanisms [18] into NesterovNODEs. In general, GNesterovNODEs form a wide class of neural differential equations which are represented by differential-algebraic systems and contain NesterovNODEs as a subclass.
3. We prove that the continuous adjoint equation used to compute the gradients for updating the parameters θ in a GNesterovNODE also follows a generalized Nesterov ODE. Thus, the NFEs in both forward and backward passes of GNesterovNODEs are significantly reduced, especially when the solvers are used with small error tolerances.
4. We prove that the spectrum of the GNesterovNODE is well-structured. This property of GNesterovNODEs helps alleviate the vanishing gradient issue during training and allows the model to capture long-term dependencies in the data.

We empirically demonstrate the advantages of the NesterovNODEs/GNesterovNODEs over the baseline NODE and the state-of-the-art neural ODE models including the Heavy Ball NODEs (HBNODEs), which solve the continuous limit of the heavy ball momentum accelerated gradient descent [60] on a wide range of applications including point cloud separation, image classification, and kinetic simulation. In all experiments, our proposed models achieve better accuracy and smaller NFEs than the baselines.

1.2 Organization

We structure this paper as follows: In Section 2, we review HBNODEs and NesterovODEs. In Section 3, we present the algorithm and analysis of the NesterovNODEs and GNesterovNODEs. We study the spectrum structure of the adjoint equations of NesterovNODEs/GNesterovNODEs to show that NesterovNODEs/GNesterovNODEs can learn long-term dependencies effectively in Section 4. In Section 5 and 6, we empirically validate the advantages of NesterovNODEs/GNesterovNODEs and analyze our models with ablation studies. We discuss related works in Section 7. The paper ends with concluding remarks. Proofs and additional experimental details are provided in the Appendix.

2 An Integration of Nesterov ODEs into NODEs

We first establish a connection between NODEs and gradient descent (GD), then review the Heavy Ball Neural ODEs (HBNODEs), and motivate the integration of Nesterov ODEs into NODEs.

ODE limit of gradient descent and connections to NODEs Gradient descent (GD) has been among the methods of choice in optimization and machine learning for training complex systems. Starting from initial point $\boldsymbol{x}_0 \in \mathbb{R}^d$, GD iterates as $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} - s\nabla F(\boldsymbol{x}_k)$ with $s > 0$ being the step size in order to find a minimum of the function $F(\boldsymbol{x})$. Let $s \to 0$, we obtain the following ODE limit of the GD

$$\frac{d\boldsymbol{x}}{dt} = -\nabla F(\boldsymbol{x}_t). \tag{4}$$

Comparing Eq. (1) and (4), we observe that a NODE solves the ODE limit of the GD where the gradient $-\nabla F(\boldsymbol{x}_t)$ is parameterized by a neural network $f(\boldsymbol{x}(t), t, \theta)$.

Heavy ball neural ordinary differential equations HBNODEs are proposed in [60]. This model takes advantage of the acceleration of heavy ball (HB) momentum [41] to reduce the NFEs needed for solving the ODEs and speed up NODEs. In particular, HBNODEs replace the first-order ODE limit of GD by the following second-order ODE limit of heavy ball momentum method:

$$\frac{d^2\boldsymbol{x}(t)}{dt^2} + \gamma\frac{d\boldsymbol{x}(t)}{dt} = -\nabla F(x(t)). \tag{5}$$

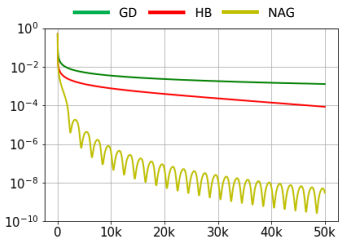

Figure 1: Comparing the convergence of GD, HB and NAG for solving the optimization problem $\min_{\boldsymbol{x}} F(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \mathbf{L}\boldsymbol{x} - \boldsymbol{x}^\top \boldsymbol{b}$ where $\mathbf{L} \in \mathbb{R}^{d\times d}$ is the Laplacian of a cycle graph and $\boldsymbol{b}$ is a d-dimensional vector whose first entry is 1 and all the other entries are 0.

Similar to NODEs, [60] parameterize $-\nabla F(x(t))$ by a neural network $f(\boldsymbol{x}(t), t, \theta)$ and formulate HBNODEs as follows

$$\frac{d^2\boldsymbol{x}(t)}{dt^2} + \gamma\frac{d\boldsymbol{x}(t)}{dt} = f(\boldsymbol{x}(t), t, \theta), \tag{6}$$

where $\gamma > 0$ is the damping parameter, which can be a hyperparameter or a learnable parameter.

Nesterov's accelerated gradient (NAG) momentum Even though HB improves the convergence and accelerates GD, both GD and HB share the same convergence rate of $O(1/k)$ for convex smooth optimization. A breakthrough due to Nesterov [33] replaces the constant momentum γ with $(k-1)/(k+2)$, a.k.a. NAG momentum, improves the convergence rate to $O(1/k^2)$, which is proved to be optimal for convex and smooth objective functions [33, 52]. We demonstrate the faster convergence of NAG in comparison with GD and HB on a quadratic optimization problem in Figure 1. The much faster convergence rate of NAG than that of GD and HB motivates us to incorporate the second-order ODE limit of NAG into a NODE and propose the NesterovNODE. Nesterov acceleration gradient method [33] takes the following form: given initial points $x_0 \in \mathbb{R}^N$ and $y_0 = x_0$, the sequence $\{(x_k, y_k)\}_k$ is defined inductively as:

$$\begin{cases} x_k = y_{k-1} - s\nabla F(y_{k-1}), \\ y_k = x_k + \dfrac{k-1}{k+2}(x_k - x_{k-1}). \end{cases} \tag{7}$$

The continuous limit of the Nesterov scheme is obtained by setting $x_k = \mathbf{h}(k\sqrt{s}) = \mathbf{h}(t)$ with $t = k\sqrt{s}$ and some smooth function $\mathbf{h}$ from $\mathbb{R}$ to $\mathbb{R}^N$. According to [52], the function $\mathbf{h}$ satisfies the Nesterov ODE

$$\mathbf{h}''(t) + \frac{3}{t}\mathbf{h}'(t) + \nabla F(\mathbf{h}(t)) = 0, \tag{8}$$

with the initial conditions $\mathbf{h}(0) = \mathbf{h}_0$, $\mathbf{h}'(0) = 0$.

Remark 1 (Nesterov factor)**.** *The constant* 3 *in the coefficient of* $\mathbf{h}'(t)$ *in Eq.* (8) *is originally from the approximation* $(k-1)/(k+2) = 1 - 3/k + \mathcal{O}(1/k^2)$*. This constant will be replaced by a constant* r *if the factor* $(k-1)/(k+2)$ *in Eq.* (7) *is replaced by* $(k-1)/(k+r-1)$*. It is proved in [52] that the Nesterov ODE still holds the quadratic convergence rate when 3 is replaced by any number* $r > 3$*.*

Remark 2 (Numerical stability)**.** *If the Euler method is used, then to keep the numerical solution close to the exact solution, the step size chosen in the Euler method must be small enough. However, the smaller the step size, the more expensive the computation. The maximum stable step size, which is the maximum value for which the step size can be chosen so that the numerical solution remains close to the exact solution, reflects the numerical stability of the ODEs. It is proved in [52] that the maximum stable step size of the Nesterov ODE is much larger than that of the ODE* (4)*, thus showing the numerical stability advantage of the Nesterov ODE.*

3 Generalize Nesterov ODEs to Differential-Algebraic Systems

One can parameterize $\nabla F(\mathbf{h}(t))$ in Eq. (8) by a neural network $f(\mathbf{h}(t), t, \theta)$ with learnable parameters θ in a similar way as NODE. This results in the following Nesterov Neural ODE (NesterovNODE)

$$\mathbf{h}''(t) + \frac{3}{t}\mathbf{h}'(t) + f(\mathbf{h}(t), t, \theta) = 0, \tag{9}$$

This second-order NesterovNODE can be written in term of the first-order NesterovNODE as

$$\begin{cases} \mathbf{h}'(t) = \mathbf{m}(t), \\ \mathbf{m}'(t) = -\frac{3}{t}\mathbf{m}(t) - f(\mathbf{h}(t), t, \theta). \end{cases} \tag{10}$$

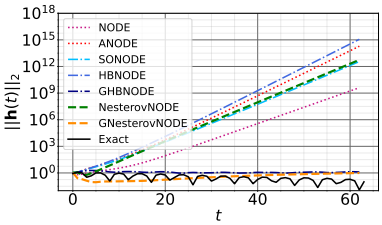

Figure 2: Contrasting the increase in the $l2$-norm of $h(t)$ for NODE, ANODE, SONODE, HBNODE, GHBNODE, NesterovNODE, and GNesterovNODE over long integration time on the Silverbox Initialization task (More details in Appendix D.1).

However, because of the singularity created by the coefficient $\frac{3}{t}$, the training process based directly on Eq. (9) and (10) will be unstable. To avoid the instability issue, we set $\mathbf{h}(t) = k(t)\mathbf{x}(t)$ with $k(t) = t^{-\frac{3}{2}}e^{\frac{t}{2}}$. Then Eq. (8) becomes

$$k(t)\mathbf{x}''(t) + \left(2k'(t) + \frac{3}{t}k(t)\right)\mathbf{x}'(t) + \left(k''(t) + \frac{3}{t}k'(t)\right)\mathbf{x}(t) + \nabla F(\mathbf{h}(t)) = 0. \tag{11}$$

We observe that $2k'(t) + \frac{3}{t}k(t) = k(t)$. By dividing both sides of Eq. (11) by $k(t)$, we obtain:

$$\mathbf{x}''(t) + \mathbf{x}'(t) + f(\mathbf{h}(t), t) = 0, \tag{12}$$

where

$$f(\mathbf{h}(t), t) = \frac{1}{4}\left(t^2 - 3\right) t^{-\frac{1}{2}} e^{-\frac{t}{2}} \mathbf{h}(t) + t^{\frac{3}{2}} e^{\frac{-t}{2}} \nabla F\left(\mathbf{h}(t)\right).$$

Let $\mathbf{m}(t) = \mathbf{x}'(t)$. Then Eq. (8) is equivalent to the following differential-algebraic system

$$\begin{cases} \mathbf{h}(t) = t^{\frac{-3}{2}} e^{\frac{t}{2}} \mathbf{x}(t), \\ \mathbf{x}'(t) = \mathbf{m}(t), \\ \mathbf{m}'(t) = -\mathbf{m}(t) - f(\mathbf{h}(t), t). \end{cases} \tag{13}$$

We parameterize $f(\mathbf{h}(t), t)$ as a neural network, recalled as $f(\mathbf{h}(t), t, \theta)$ with learnable parameter θ, and obtain the differential-algebraic version of NesterovNODE

$$\begin{cases} \mathbf{h}(t) = t^{\frac{-3}{2}} e^{\frac{t}{2}} \mathbf{x}(t), \\ \mathbf{x}'(t) = \mathbf{m}(t), \\ \mathbf{m}'(t) = -\mathbf{m}(t) - f(\mathbf{h}(t), t, \theta). \end{cases} \tag{14}$$

The singularity at $t = 0$ of the NesterovNODE in Eqs. (9) and (10) is now moved to the algebraic equation of the above differential-algebraic system.

It is often the case when training ODE-based models that some functions diverge or explode at a finite time. This phenomenon is called blow-up, which we demonstrate in Fig. 2. In order to alleviate the blow-up problem, we introduce a generalized version of NesterovNODE, termed Generalized NesterovNODE (GNesterovNODE). For the NesterovNODE, the potential blow-up is due to the oscillation inherited from NAG scheme as can be seen in Fig. 1. The blow-up can also occur because of the singularity caused by the factor $t^{\frac{-3}{2}}e^{\frac{t}{2}}$ in the algebraic equation. Following the techniques presented in [60], we address these potential blow-up by applying an activation function σ to the function $f(\mathbf{h}(t), t, \theta)$, the factor $t^{\frac{-3}{2}}e^{\frac{t}{2}}$, and the momentum state $\boldsymbol{m}(t)$ of the NesterovNODE. The activation function can be any activation function commonly used. In our experiments, we use $\tanh$ and $\mathrm{hardtanh}$. In addition, the residual term $\xi\mathbf{h}(t)$, which stands for a skip connection [16], is also added into the governing equation of $\boldsymbol{m}(t)$, which benefits training and generalization of GNesterovNODEs. The GNesterovNODE differential-algebraic system is then given by

$$\begin{cases} \mathbf{h}(t) = \sigma(t^{\frac{-3}{2}}e^{\frac{t}{2}})\mathbf{x}(t), \\ \mathbf{x}'(t) = \sigma(\mathbf{m}(t)), \\ \mathbf{m}'(t) = -\mathbf{m}(t) - \sigma(f(\mathbf{h}(t), t, \theta)) - \xi\mathbf{h}(t), \end{cases} \tag{15}$$

with the initial conditions $\mathbf{h}(0) = \mathbf{h}_0$, $\mathbf{x}(0) = \mathbf{x}_0$, $\mathbf{m}(0) = \mathbf{m}_0$. It is noted that, from $\mathbf{h}(0) = \mathbf{h}_0$ and $\mathbf{h}'(0) = 0$, we must have $\mathbf{x}_0 = \mathbf{h}_0 \lim_{t\to 0} \sigma(t^{\frac{-3}{2}}e^{\frac{t}{2}})^{-1}$ and $\mathbf{m}_0 = \mathbf{h}_0 \lim_{t\to 0} \frac{d}{dt}\sigma(t^{\frac{-3}{2}}e^{\frac{t}{2}})^{-1}$. Fig. 2 shows that GNesterovNODE can indeed control the growth of $\boldsymbol{h}(t)$ effectively.

In general, GNesterovNODEs form a wide class of neural differential equations which are represented by differential-algebraic systems and contain NesterovNODEs as a subclass. Eqs. (9) and (15) define the forward ODE for the (G)NesterovNODE. To efficiently update the parameters during the training process based on (G)NesterovNODEs, we use the continuous adjoint sensitivity given in Propositions 1 and 2 below.

Proposition 1 (Continuous adjoint equation for the second-order NesterovNODE)**.** *The continuous adjoint state* $\mathbf{a}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}$ *of the NesterovNODE given in Eq.* (9) *satisfies the following NesterovNODE*

$$\mathbf{a}''(t) - \frac{3}{t}\mathbf{a}'(t) + \mathbf{a}(t)\frac{\partial f(\mathbf{h}(t), t, \theta)}{\partial \mathbf{h}} + \frac{3}{t^2}\mathbf{a}(t) = 0. \tag{16}$$

Proposition 2 (Continuous adjoint equation for GNesterovNODE)**.** *The continuous adjoint state functions* $\mathbf{a_h}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}$, $\mathbf{a_x}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t)}$ *and* $\mathbf{a_m}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{m}(t)}$ *of the GNesterovNODE system given in Eq.* (15) *satisfy the following differential-algebraic adjoint system*

$$\begin{cases} \mathbf{a_h}(t) = \sigma(t^{\frac{-3}{2}}e^{\frac{t}{2}})^{-1}\mathbf{a_x}(t), \\ \mathbf{a}'_{\mathbf{x}}(t) = t^{\frac{-3}{2}}e^{\frac{t}{2}}\mathbf{a_m}(t)\left[\frac{\partial \sigma(f(\mathbf{h}(t), t))}{\partial \mathbf{h}} + \xi\mathbf{I}\right], \\ \mathbf{a}'_{\mathbf{m}}(t) = \mathbf{a_m}(t) - \mathbf{a_x}(t)\sigma'(\mathbf{m}(t)), \end{cases} \tag{17}$$

with the final value conditions $\mathbf{a_h}(T) = \frac{\partial L}{\partial \mathbf{h}_T}$, $\mathbf{a_x}(T) = \frac{\partial L}{\partial \mathbf{x}_T}$ *and* $\mathbf{a_m}(T) = \frac{\partial L}{\partial \mathbf{m}_T}$.

4 The Effectiveness of GNesterovNODE in Alleviating Vanishing Gradients

In neural networks with many layers, the vanishing gradient is one of the major issues [40]. In the cases of NODEs and their hybrid ODE-RNN variants, the vanishing gradient issue may occur when the adjoint state $\mathbf{a}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}$ goes to 0 quickly as $T - t$ increases. In this section, we will prove that this vanishing gradient issue can be avoided in GNesterovNODEs.

For the GNesterovNODE given in Eq. (15), the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$ can be determined from $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_t}$ via the algebraic relation:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t}\frac{\partial \mathbf{x}_t}{\partial \mathbf{h}_t} = \sigma(t^{\frac{-3}{2}}e^{\frac{t}{2}})^{-1}\frac{\partial \mathcal{L}}{\partial \mathbf{x}_t}.$$

While the gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_t}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{m}_t}$ satisfy the following proposition.

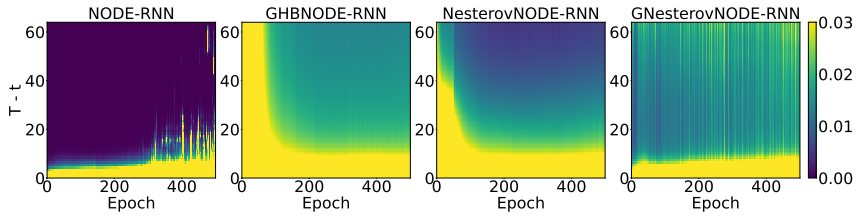

Figure 3: Plot of the $l2$-norm of the adjoint states for ODE-RNN, GHBNODE-RNN, NesterovNODE-RNN and GNesterovNODE-RNN back-propagated from the last time stamp. The term $T - t$ demonstrates the gap between the final time T and intermediate time t. When the gap $T - t$ becomes larger, NesterovNODE-RNN and GNesterovNODE-RNN can address the vanishing gradient problem due to the adjoint states of these methods' decay slowly.

Proposition 3. *For every $t \in (0, T)$, there exist a unit length row vector $v \in \mathbb{C}^N$ and an upper triangular matrix $U \in \mathbb{C}^{N \times N}$ such that*

$$\left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix} \right\|_2 = \| v \exp(U) \|_2 \left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_T} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_T} \end{bmatrix} \right\|_2,$$

and at least $\frac{N}{2}$ complex values in the diagonal of U have the real parts greater than or equal to $\frac{t-T}{2}$.

The term $v \exp(U)$ in the above proposition plays the essential role in the nonvanishing gradient issue. Without loss of generality, we can assume that $U = \begin{bmatrix} U_{\text{large}} & P \\ \mathbf{0} & U_{\text{small}} \end{bmatrix}$ where all complex numbers in the diagonal of U_{large} (resp. U_{small}) have the real parts greater than or equal to (resp. smaller than) $\frac{1}{2}(t - T)$. According to Proposition 3, the size of U_{large} is at least $N/2$. Then we have,

$$\exp(U) = \begin{bmatrix} \exp(U_{\text{large}}) & \tilde{P} \\ \mathbf{0} & \exp(U_{\text{small}}) \end{bmatrix} \quad \text{and} \quad \| v \exp(U) \|_2 \geq \| v_{\text{large}} \exp(U_{\text{large}}) \|_2 .$$

Here, the vector v_{large} is the first m columns of v, and m is the size of U_{large}. Since the real parts of elements in the diagonal of U_{large} is no less than $\frac{1}{2}(t - T)$, $\exp(U_{\text{large}})$ decays at a rate at most $\frac{1}{2}(t - T)$. This results in the nonvanishing gradient of $\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix}$, hence so is $\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix}$.

To illustrate, we take the Walker2D kinematic simulation task [2] in consideration, which requires learning long-term dependency effectively [27]. We train ODE-RNN [46], GHBNODE-RNN [60], NesterovNODE-RNN and GNesterovNODE-RNN on this benchmark dataset (More details in Appendix D.4). Fig. 3 plots $\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \right\|_2$ for ODE-RNN, $\left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix} \right\|_2$ for GHBNODE-RNN and $\left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix} \right\|_2$ for NesterovNODE-RNN and GNesterovNODE-RNN, showing that when the gap between the final time T and intermediate time t becomes larger, the adjoint states of NesterovNODE-RNN, GNesterovNODE-RNN and GHBNODE-RNN decay much more slowly than NODE-RNN. Thus, NesterovNODE-RNN and GNesterovNODE-RNN have the ability to tackle the vanishing gradient issue.

Remark 3. *The gradient exploding problem can be effectively resolved via gradient clipping, training loss regularization, etc. [40, 10]. Therefore, in practice the vanishing gradient problem is the major issue for training deep neural networks [40].*

5 Experimental Results

In this section, we empirically study the advantages of our proposed NesterovNODE/GNesterovNODE over the baseline NODEs and other popular NODE-based architectures, including the augmented NODE (ANODE) [9], the Second Order NODE (SONODE) [37], HBNODE/GHBNODE [60] on a variety of benchmarks including point cloud separation, image classification, and kinetic simulation which involve different data modalities ranging from point cloud to images and time series. ANODEs augments the space on which the ODE is solved while SONODEs and (G)HBNODEs solve a second-order ODE. We aim to show that: (i) NesterovNODEs/GNesterovNODEs require significantly fewer NFEs while attaining similar or even better accuracy as the baselines; (ii) GNesterovNODEs avoid the blow-up of $\boldsymbol{h}(t)$ and thus improve over NesterovNODEs; (iii) NesterovNODEs/GNesterovNODEs

Table 1: The parameters count for the models in point cloud separation, image classifications, and the Walker2D kinematic simulation tasks and the test accuracy on CIFAR10/MNIST. Our methods are able to reach similar or better test accuracy than the baseline methods on CIFAR10 while retaining a similar test accuracy on MNIST.

	Number of parameters				Test Accuracy	
Model	*PC*	*MNIST*	*CIFAR10*	*Walker2D*	*CIFAR10*	*MNIST*
NODE	545	85316	173611	9929	0.5466 ± 0.0051	0.9531 ± 0.0042
ANODE	587	85462	172452	10019	0.6025 ± 0.0032	0.9816 ± 0.0024
SONODE	541	86179	171635	11471	0.6132 ± 0.0073	$\mathbf{0.9824 \pm 0.0013}$
HBNODE	582	85931	172916	10099	0.5989 ± 0.0035	0.9814 ± 0.0011
GHBNODE	582	85931	172916	10099	0.6085 ± 0.0050	0.9817 ± 0.0005
NesterovNODE	581	85930	172915	10098	0.5996 ± 0.0033	$\mathbf{0.9824 \pm 0.0015}$
GNesterovNODE	581	85930	172915	10098	$\mathbf{0.6172 \pm 0.0064}$	0.9807 ± 0.0013

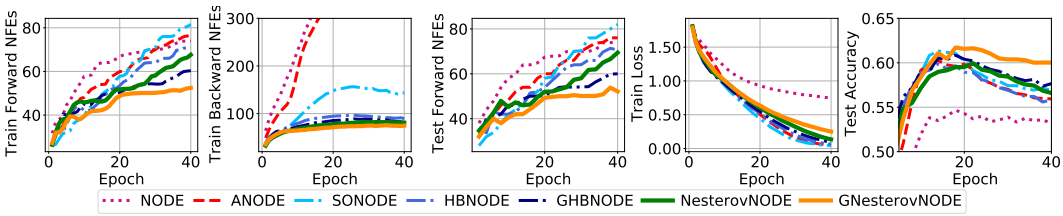

Figure 4: Contrasting the NFEs and accuracy of NODE-based baselines and our methods NesterovNODE/GNesterovNODE on the CIFAR10 dataset (Tolerance: 10^{-5}).

capture better long-term dependencies than the baselines and achieve better results in long-sequence modeling tasks.

For all the experiments, we use Adam [23] as the optimizer and Dormand-Prince 5(4) [7] as the numerical ODE solver. We choose the network architecture used to parameterize $f(\boldsymbol{h}(t), t, \theta)$ so that our proposed models and the baselines have similar numbers of parameters in our experiments as shown in Table 1. Other training/model/dataset details are provided in Appendix D. All results are averaged over 5 runs with different seeds. We conduct the experiments on a server with 6 NVIDIA 2080Ti GPUs with 11GB of GPU memory. Our PyTorch code with documentation can be found at https://github.com/minhtannguyen/NesterovNODE.

5.1 Image classification

We validate the accuracy and efficiency advantage of NesterovNODE/GNesterovNODE for image classification on MNIST [6] and CIFAR10 [25] in comparison with other ODE-based baselines. We follow the same training and model settings as in [60].

NFEs. As shown in Fig. 4, our NesterovNODE and GNesterovNODE reduce the NFEs in both the forward and the backward propagations compared to the baseline models. Although the augmented input dimensions in ANODE help reduce the NFEs compared to NODE, second-order methods reduce the NFEs more significantly. Compared to the other second-order methods i.e. SONODE, HBNODE, and GHBNODE, GNesterovNODE achieve much better NFE reductions on both MNIST (see Fig 12 in the Appendix) and CIFAR10, indicating the improvement in efficiency and stability of our methods over the other second-order baseline models. Such advancement is an essential step to scale GNesterovNODE to larger and more complex practical tasks.

Accuracy. Table 1 shows that our GNesterovNODE achieves the highest test accuracy on CIFAR10. On MNIST, NesterovNODE attains the second-highest test accuracy and very close to the best result from SONODE while being much more efficient than SONODE. This advantage in terms of accuracy can be associated with the small number of NFEs needed by NesterovNODE and GNesterovNODE above, which reduces the model complexity and leads to better generalization.

Note that GNesterovNODE improves over NesterovNODE in efficiency and accuracy (on larger and more challenging benchmark like CIFAR10). This justifies the effectiveness of our solution that introduces an additional bounded activation function σ and the residual term $\xi \boldsymbol{h}(t)$ to prevent the blow-up of $\boldsymbol{h}(t)$ as explained in Section 3.

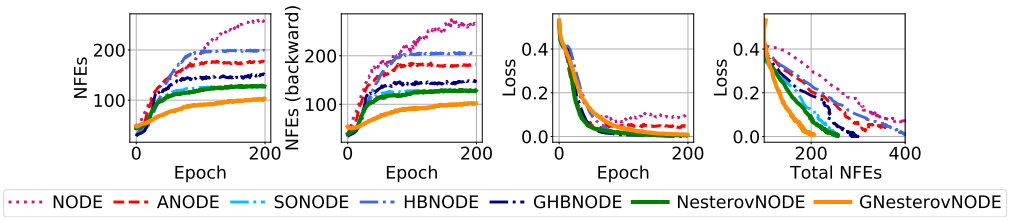

Figure 5: Contrasting the NFEs and training loss of NODE-based baselines and our NesterovNODE/GNesterovNODE on the point cloud benchmark. Results are averaged over 50 runs (Tolerance: 10^{-7}).

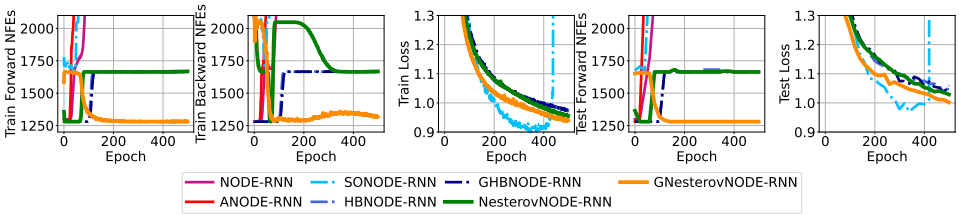

Figure 6: Contrasting the NFEs and losses of NODE-RNN [4], ANODE-RNN [9], HBNODE-RNN/GHBNODE-RNN [60], and our NesterovNODE-RNN/GNesterovNODE-RNN on the Walker2D dataset (Tolerance: 10^{-7}).

5.2 Point cloud separation

We perform experiments on a point cloud separation task in order to verify that NesterovNODEs/GNesterovNODEs can learn effective features to separate two sets of point clouds. The first set consists of 40 points drawn from a circle with the radius $||\boldsymbol{r}|| < 0.5$, while the second set comprises 80 points drawn from an annulus with the inner and outer radius of 0.85 and 1, respectively, i.e. $0.85 < ||\boldsymbol{r}|| < 1.0$. Fig. 5 shows that our NesterovNODE and GNesterovNODE models are able to converge to 0 loss consistently while other methods have difficulty to reach 0 loss. In addition, NesterovNODE and especially the GNesterovNODE require significantly fewer NFEs in forward and backward passes compared to the baselines. Thus, NesterovNODE and GNesterovNODE help improve both the training and the efficiency of the model. We plot the evolution of the point cloud separation through 100 epochs for a random run of each model in Appendix D.2. Like SONODE, HBNODE/GHBNODE, NesterovNODE/GNesterovNODE learn effective features that allow good separation between the two classes of point clouds in these experiments while NODE and ANODE fail for this task.

5.3 Walker2D kinematic simulation

In this section, we investigate NesterovNODE/GNesterovNODE when applied on time-series data. In particular, we use the ODE-RNN framework [46], with the recognition model being set to different ODE-based models, to study Walker2D kinematic simulation task, which requires learning long-term dependency effectively [26]. As shown in Fig. 6, our NesterovNODE-RNN and GNesterovNODE-RNN not only reduce the NFEs both in the forward and the backward stages, but also achieve smaller test loss compared to the baseline models that we compare with, including (G)HBNODE-RNN, ANODE-RNN, and NODE-RNN. Although SONODE-RNN achieves much smaller losses, its NFEs are too high, thus training SONODE-RNN for this task is much more time-consuming compared to our Nesterov-based methods.

5.4 Continuous Normalizing Flows for MNIST

We compare the GNesterovNODE with the baseline NODE and GHBNODE for use in variational inference with the continuous normalizing flow model trained on the MNIST dataset. The continuous normalizing flow model we use is the FFJORD in [14]. We summarize our results in Figure 7. Compared to the FFJORD-NODE and FFJORD-GHBNODE, the FFJORD-GNesterovNODE significantly reduces the NFEs in both forward and backward passes while improving the negative ELBO on the test set. This experiment demonstrates that our method GNesterovNODE accelerates NODE-based models on both discriminative tasks and generative tasks.

6 Observed Properties of NesterovNODE in GNesterovNODE

In this section, we verify empirically that with the addition of the activation function σ and the skip connection, GNesterovNODEs still preserve important properties of NesterovNODEs as stated in Re-

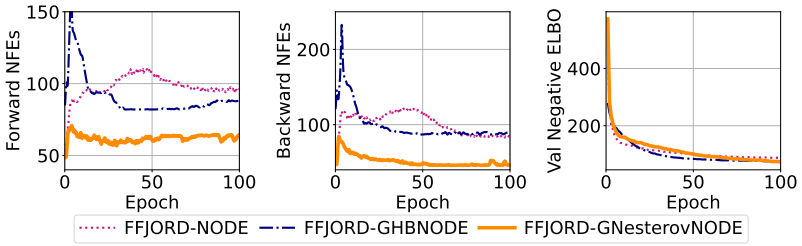

Figure 7: Contrasting the NFEs and the validation negative ELBO of the FFJORD-NODE, the FFJORD-HBNODE, and our FFJORD-GNesterovNODE for the variational inference task with a continuous normalizing flow model, i.e. FFJORD [14], on the binarized MNIST dataset (Tolerance: 10^{-5}).

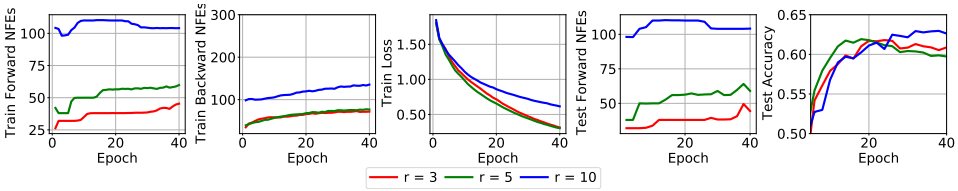

Figure 8: Contrasting different values of the factor r on CIFAR10 with GNesterovNODE (Tolerance: 10^{-5}).

mark 1 and Remark 2. This hints that GNesterovNODEs have the same behaviors as NesterovNODEs while enjoying more stable training.

Effect of the Nesterov factor on NesterovNODE (Remark 1)

As stated in [52], the "magic constant" 3 can be replaced by a constant $r > 3$ while maintaining a convergence rate of $O(1/k^2)$. In this work's experiments, a larger r leads to a better test loss eventually despite a smaller r outperforming in the beginning, but their experiments are based on Nesterov ODE. This section investigates whether this behavior extends to our generalized method GNesterovNODE. As shown in Fig. 8 and Table 2, a larger r also leads to a higher test accuracy, and for $r = 3$ and $r = 5$, the model converges to its best test accuracy sooner than $r = 10$. Although the training loss of $r = 10$ is larger than $r = 3$ and $r = 5$, we observe that all three values of r reach their best testing accuracy when the training loss is approximately in the range of $[0.65, 0.75]$, which hints that decreasing the loss further leads to overfitting. One more interesting observation is that a higher value of r increases the forward and backward NFEs. Intuitively speaking, the term $-\frac{r}{t}\mathbf{h}'(t)$ in the NesterovNODE, which is opposite to the product of the damping parameter r/t and the velocity $\mathbf{h}'(t)$, represents the friction force of the model (see [52, Section 4]). When r is larger, the friction force resists the movement of $\mathbf{h}(t)$ along the trajectory stronger, which slows down the convergence of training loss.

Stability of NesterovNODE (Remark 2)

The NesterovNODE is more numerically stable than NODE in the sense that the step size in the Euler method for solving the NesterovNODE can be chosen larger while the stability of the numerical solution is still guaranteed (see Remark 2). We hypothesize that this fact still holds for the GNesterovNODE in comparison with both NODE and GHBNODE. To illustrate this fact, we perform experiments on CIFAR10 using Euler solvers with large step sizes $(0.1, 0.2, 0.5)$. Due to the instability of large step size, GHBNODE moves fast to the maximum accuracy points and then goes down, as shown in Fig. 9, while NODE achieves high train loss and low test accuracy. Even when the step size is big, GNesterovNODE's training is stable as the curve evolves gradually. More interestingly, as shown in Fig. 10, GNesterovNODE solved with rk4 (Fourth-order Runge-Kutta with 3/8 rule) solver using a large step size outperforms the same model solved with the adaptive step size solvers like dopri5 (Dormand-Prince of order 5) solver. This show the promise of using large step sizes in GNesterovNODE to obtain models with better accuracy and efficiency.

Table 2: Test accuracy for GNesterovNODE on CIFAR10 with varying Nesterov factors r (Tolerance: 10^{-5}).

Value of r	*Test accuracy*
3	0.6180
5	0.6192
10	0.6296

7 Related Work

Increasing efficiency of training NODEs. Several methods have been proposed to reduce the NFEs in NODEs and increase the model efficiency. Among them are works that use weight decay [14] and

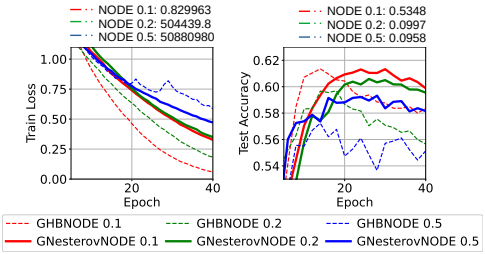

Figure 9: Train loss and test accuracy for large step sizes of Euler solver on CIFAR10 training with NODE, GHBNODE and GNesterovNODE. In the legend, we place the step size next to the model's name.

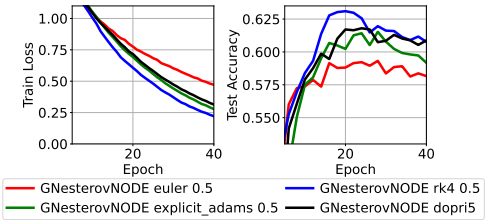

Figure 10: Train loss and test accuracy for various solvers with large step size 0.5 on CIFAR10 training with GNesterovNODE. In the legend, we place the step size next to the model's name.

other regularizers applied on the solver and the learned dynamics [11, 20, 12, 39]. Other works employ input augmentation [8], data-control [30] and depth-variance [30, 36] to reduce the NFEs needed to compute the ODE solution. Another method replaces norms with seminorms in the backward continous adjoint stage to reduce backward NFEs [22]. NesterovNODE solves second-order Nesterov ODE to reduce both forward and backward NFEs.

Second-order dynamical systems. Second-order ODEs have also been employed to speed up NODEs. SONODE [37] replaces the first-order ODE in Eq. (1) by a second ODE which can be solved as a system of first-order ODEs. HBNODE [60] also solves a second-order ODE but with an additional constant damping parameter, which corresponds to the ODE limit of the HB momentum method. NesterovNODE solves the second-order Nesterov ODE with a time-dependent damping parameter, which corresponds to the ODE limit of the NAG momentum method. These momentum-based systems have also been employed in designing deep neural networks as in [32, 28, 49, 35].

Learning long-term dependencies. The ability of a model to learn long-term dependencies is highly needed to scale up the model to large-scale tasks that involve very long sequences. Existing works try to alleviate exploding or vanishing gradient issues happened during the training of recurrent neural networks, including [1, 59, 19, 55, 31, 17, 34]. Recently, learning long-term dependencies with NODEs has been explored. For example, [26] integrate a long-short term memory cell into NODEs. Among the hallmarks of NesterovNODEs is that our proposed models can directly capture long-term dependencies in long sequences.

8 Concluding Remarks

In this paper, we propose the NesterovNODE and its generalized version GNesterovNODE that solve the second-order ODE limit of NAG. These models take advantage of the convergence rate $\mathcal{O}(1/k^2)$ of the NAG scheme to gain acceleration over NODEs and the existing NODE-based models such as HBNODE by reducing the NFEs in solving both forward and backward ODEs. Our Nesterov-based NODEs also achieve better accuracy than NODEs and outperform or at least are on par with other NODE-based models in our experiments while requiring much fewer NFEs. We also prove that NesterovNODEs and GNesterovNODEs can avoid the vanishing gradient issue and can capture long-term dependencies in long sequences effectively. It is worth mentioning that NesterovNODEs/GNesterovNODEs do not introduce any inherently negative societal impact. The high-resolution ODE in [51, 50] is an alternative way to take a continuous-time limit of the NAG scheme and heavy-ball momentum method. This high-resolution ODE introduces a gradient correction that allows the NAG scheme to achieve an inverse cubic rate for minimizing the squared gradient norm, which is better than the inverse square rate in the low-resolution ODE that our method uses. A limitation of our paper is that we have not incorporated restart schemes [38, 45, 56] into the NesterovNODE, and we leave this as future work.

Acknowledgements

This material is based on research sponsored by the AFOSR MURI FA9550-18-1-0502, the ONR grant N00014-20-1-2093, the MURI N00014-20-1-2787, and the NSF under Grant# 2030859 to the Computing Research Association for the CIFellows Project (CIF2020-UCLA-38). NH acknowledges support from the NSF IFML 2019844 and the NSF AI Institute for Foundations of Machine Learning.

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] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[5] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed 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.

[6] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[7] John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[8] 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.

[9] 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.

[10] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney. Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*, 2020.

[11] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M. Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *ICML*, pages 3154–3164, 2020.

[12] Arnab Ghosh, Harkirat Behl, Emilien Dupont, Philip Torr, and Vinay Namboodiri. Steer : Simple temporal regularization for neural ode. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14831–14843. Curran Associates, Inc., 2020.

[13] 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.

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

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

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[17] 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.

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

[19] 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.

[20] Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4370–4380. Curran Associates, Inc., 2020.

[21] Patrick Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

[22] Patrick Kidger, Ricky TQ Chen, and Terry J Lyons. " hey, that's not an ode": Faster ode adjoints via seminorms. In *ICML*, pages 5443–5452, 2021.

[23] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[24] Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: exact likelihood generative learning for symmetric densities. In *International Conference on Machine Learning*, pages 5361–5370. PMLR, 2020.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series, 2020.

[27] Mathias Lechner and Ramin M. Hasani. Learning long-term dependencies in irregularly-sampled time series. *CoRR*, abs/2006.04418, 2020.

[28] Huan Li, Yibo Yang, Dongmin Chen, and Zhouchen Lin. Optimization algorithm inspired deep neural network structure design. In *Asian Conference on Machine Learning*, pages 614–629. PMLR, 2018.

[29] M Luštrek, B Kaluža, Rok Piltaver, Jana Krivec, and Vedrana Vidulin. Localization data for person activity data set, 2010.

[30] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting 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 3952–3963. Curran Associates, Inc., 2020.

[31] 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.

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

[33] 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.

[34] Tan M Nguyen, Richard Baraniuk, Andrea Bertozzi, Stanley Osher, and Bao Wang. Momentumrnn: Integrating momentum into recurrent neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1924–1936. Curran Associates, Inc., 2020.

[35] Tan M Nguyen, Richard Baraniuk, Robert Kirby, Stanley Osher, and Bao Wang. Momentum transformer: Closing the performance gap between self-attention and its linearization. In *Mathematical and Scientific Machine Learning*, pages 189–204. PMLR, 2022.

[36] Tan M Nguyen, Animesh Garg, Richard G Baraniuk, and Anima Anandkumar. Infocnf: An efficient conditional continuous normalizing flow with adaptive solvers. *Asilomar Conference*, 2022.

[37] Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Lió. On second order behaviour in augmented 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 5911–5921. Curran Associates, Inc., 2020.

[38] Brendan O'donoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15(3):715–732, 2015.

[39] Avik Pal, Yingbo Ma, Viral Shah, and Christopher V Rackauckas. Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8325–8335. PMLR, 18–24 Jul 2021.

[40] 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.

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

[42] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[43] Christopher Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl - A julia library for neural differential equations. *CoRR*, abs/1902.02376, 2019.

[44] Christopher Rackauckas and Qing Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1), 2017. Exported from https://app.dimensions.ai on 2019/05/05.

[45] Vincent Roulet and Alexandre d'Aspremont. Sharpness, restart and acceleration. In *Advances in Neural Information Processing Systems*, pages 1119–1129, 2017.

[46] 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.

[47] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[48] Lars Ruthotto, Stanley J. Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences*, 117(17):9183–9193, 2020.

[49] Michael E. Sander, Pierre Ablin, Mathieu Blondel, and Gabriel Peyré. Momentum residual neural networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9276–9287. PMLR, 18–24 Jul 2021.

[50] Bin Shi, Simon S Du, Michael I Jordan, and Weijie J Su. Understanding the acceleration phenomenon via high-resolution differential equations. *Mathematical Programming*, pages 1–70, 2021.

[51] Bin Shi, Simon S Du, Weijie Su, and Michael I Jordan. Acceleration via symplectic discretization of high-resolution differential equations. *Advances in Neural Information Processing Systems*, 32, 2019.

[52] 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.

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

[54] Ch Tsitouras. Runge–kutta pairs of order 5 (4) satisfying only the first column simplifying assumption. *Computers & Mathematics with Applications*, 62(2):770–775, 2011.

[55] 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.

[56] Bao Wang*, **Tan M Nguyen***, Andrea L Bertozzi, Richard G Baraniuk, and Stanley J Osher. Scheduled restart momentum for accelerated stochastic gradient descent. *SIAM Journal on Imaging Sciences*, 2022.

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

[58] Torbjörn Wigren and Johan Schoukens. Three free data sets for development and benchmarking in nonlinear system identification. In *2013 European control conference (ECC)*, pages 2933–2938. IEEE, 2013.

[59] 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.

[60] Hedi Xia, Vai Suliafu, Hangjie Ji, Tan Minh Nguyen, Andrea Bertozzi, Stanley Osher, and Bao Wang. Heavy ball neural ordinary differential equations. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[61] 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.

[62] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2020.

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 8
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 8
   (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]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix
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? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (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]

Supplementary materials for
Improving Neural Ordinary Differential Equations with Nesterov's Accelerated Gradient Method

A Review of the adjoint equation and the gradient for the first-order NODEs

A NODE for a hidden feature $\mathbf{z} : \mathbb{R} \to \mathbb{R}^N$ takes of the form

$$\mathbf{z}'(t) = g(\mathbf{z}(t), t, \theta), \quad \mathbf{z}(0) = \mathbf{z}_0, \tag{18}$$

where $g(\mathbf{z}(t), t, \theta) \in \mathbb{R}^N$ is a neural network with learnable parameters θ. For a scalar loss function $\mathcal{L}$, the adjoint state $\mathbf{a}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{z}(t)}$ satisfies the following differential equation

$$\mathbf{a}'(t) = -\mathbf{a}(t)\frac{\partial g(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}, \quad \mathbf{a}(T) = \frac{\partial \mathcal{L}}{\partial \mathbf{z}(T)}. \tag{19}$$

B The adjoint equations for the NesterovNODEs and GNesterovNODEs

The adjoint equation for the NesterovNODEs (Proposition 1) is an implication of [37, Proposition 3.1] for Nesterov differential equations. In this section, we give the proofs for Proposition 2. The proofs will be intrinsically based on Eq. (19).

Proof of Proposition 2 - the adjoint equation for GNesterovNODE

The GNesterovNODE is formulated as the following differential-algebraic system:

$$\begin{cases} \mathbf{h}(t) = \sigma(k(t))\mathbf{x}(t), \\ \mathbf{x}'(t) = \sigma(\mathbf{m}(t)), \\ \mathbf{m}'(t) = -\mathbf{m}(t) - \sigma(f(\mathbf{h}(t), t)) - \xi \mathbf{h}(t), \end{cases} \tag{20}$$

where $k(t) = t^{\frac{-3}{2}} e^{\frac{t}{2}}$. The adjoints of this system are given by $\mathbf{a_h}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}$, $\mathbf{a_x}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t)}$ and $\mathbf{a_m}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{m}(t)}$. From the first equation in the system, we have

$$\mathbf{a_h}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t)}\frac{\partial \mathbf{x}(t)}{\partial \mathbf{h}(t)} = \frac{1}{\sigma(k(t))}\mathbf{a_x}(t). \tag{21}$$

To determine the dynamics of $\mathbf{a_x}(t)$ and $\mathbf{a_m}(t)$, we rewrite the last two equations in system (20) as the first-order system

$$\begin{cases} \mathbf{x}'(t) = \sigma(\mathbf{m}(t)), \\ \mathbf{m}'(t) = -\mathbf{m}(t) - \sigma(f(k(t)\mathbf{x}(t), t)) - \xi k(t)\mathbf{x}(t). \end{cases}$$

Set $\mathbf{z}(t) = [\mathbf{x}(t) \quad \mathbf{m}(t)]^T$ and

$$g(\mathbf{z}(t), t, \theta) = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} \sigma(\mathbf{m}(t)) \\ -\mathbf{m}(t) - \sigma(f(k(t)\mathbf{x}(t), t)) - \xi k(t)\mathbf{x}(t) \end{bmatrix},$$

then the first-order NesterovNODE can be rewritten as

$$\mathbf{z}'(t) = g(\mathbf{z}(t), t, \theta).$$

In this case, we have

$$\begin{aligned} \frac{\partial g(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} &= \begin{bmatrix} \frac{\partial g_1}{\partial \mathbf{x}} & \frac{\partial g_1}{\partial \mathbf{m}} \\ \frac{\partial g_2}{\partial \mathbf{x}} & \frac{\partial g_2}{\partial \mathbf{m}} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0} & \sigma'(\mathbf{m}(t)) \\ -k(t)\frac{\partial \sigma[f(\mathbf{h}(t), t, \theta)]}{\partial \mathbf{h}} - \xi k(t)\mathbf{I} & -\mathbf{I} \end{bmatrix}. \end{aligned}$$

Thus, by using Eq. (19), we obtain the first-order differential system for the adjoints $\mathbf{a_x}$ and $\mathbf{a_m}$ as

$$\begin{cases} \mathbf{a_x}'(t) = \mathbf{a_m}(t)\left[k(t)\frac{\partial \sigma(f(\mathbf{h}(t), t, \theta))}{\partial \mathbf{h}} + \xi k(t)\mathbf{I}\right], \\ \mathbf{a_m}'(t) = -\mathbf{a_x}(t)\sigma'(\mathbf{m}(t)) + \mathbf{a_m}(t). \end{cases} \tag{22}$$

Together with Eq. (21), we obtain the differential-algebraic system for the adjoints of the GNesterovNODE in Proposition 2.

C Proof of Proposition 3 - the nonvanishing gradient for GNesterovNODEs

Following the lines in [60], for a NODE given by $\mathbf{z}'(t) = g(\mathbf{z}(t), t, \theta)$, we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_T} \exp\left\{ -\int_T^t \frac{\partial g(\mathbf{z}(s), s, \theta)}{\partial \mathbf{z}} ds \right\}. \tag{23}$$

For the GNesterovNODE given in Eq. (15), the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$ can be determined from $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_t}$ via the algebraic relation:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{h}_t} = \sigma(t^{\frac{-3}{2}} e^{\frac{t}{2}})^{-1} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t}.$$

While the gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_t}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{m}_t}$ can be determined by using Eq. (23) as

$$\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_T} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_T} \end{bmatrix} \cdot \exp(M), \tag{24}$$

where

$$M = -\int_T^t \begin{bmatrix} \mathbf{0} & \sigma'(\mathbf{m}(s)) \\ \frac{\partial \sigma(f)}{\partial \mathbf{x}} - \xi t^{-\frac{3}{2}} e^{\frac{t}{2}} \mathbf{I} & -\mathbf{I} \end{bmatrix} ds.$$

Let $M = QUQ^\top$ be a Schur decomposition of M where Q is an orthogonal matrix and U is an upper triangular matrix with eigenvalues of M in the diagonal. Then from Eq. (24), we have

$$\left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_T} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_T} \end{bmatrix} Q \exp(U) Q^\top \right\|_2.$$

Set $v = \frac{1}{\left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_T} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_T} \end{bmatrix} \right\|_2} \cdot \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_T} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_T} \end{bmatrix} Q$, then v is a unit length vector and

$$\left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_t} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix} \right\|_2 = \|v \exp(U)\|_2 \left\| \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_T} & \frac{\partial \mathcal{L}}{\partial \mathbf{m}_T} \end{bmatrix} \right\|_2.$$

To finish the proof of Proposition 3, we need to prove that at least half of complex numbers in the diagonal of U have the real parts greater than or equal to $\frac{t-T}{2}$.

We first claim that the eigenvalues of M can be paired in such a way that each pair has the sum $t - T$. To prove the claim, we write $M = \begin{bmatrix} \mathbf{0} & A \\ B & (t-T)\mathbf{I} \end{bmatrix}$, where

$$A = -\int_T^t \sigma'(\mathbf{m}(s)) ds, \quad \text{and} \quad B = -\int_T^t \left[\frac{\partial \sigma(f)}{\partial \mathbf{x}} - \xi t^{-\frac{3}{2}} e^{\frac{t}{2}} \mathbf{I} \right] ds.$$

Since the matrices $(t-T)\mathbf{I}$ and A commute, the characteristic polynomial of M can be determined as

$$\det(\lambda \mathbf{I} - M) = \det(\lambda(\lambda - t + T)\mathbf{I} - BA)$$

which is the value of the characteristic polynomial of the matrix BA at $\lambda(\lambda - t + T)$. Over the field of complex numbers, the characteristic polynomial of BA is splitted completely and it has the form

$$\det(\lambda \mathbf{I} - BA) = \prod_{i=1}^{N} (\lambda - \lambda_{BA,i}),$$

where $\lambda_{BA,i}$ are the eigenvalues of BA. Then the characteristic polynomial of M has the form

$$\det(\lambda \mathbf{I} - M) = \prod_{i=1}^{N} (\lambda(\lambda - t + T) - \lambda_{BA,i}).$$

Thus the eigenvalues of M can be paired in such a way that each pair is the roots of the quadratic equation

$$\lambda(\lambda - t + T) - \lambda_{BA,i}.$$

Such pairs always have the sum $t - T$. The claim is proved.

Since the diagonal of U are exactly the eigenvalues of M, the claim implies that at least half of complex numbers in the diagonal of U are greater than or equal to $\frac{t-T}{2}$. The proposition is then proved.

D Implementation details

Table 3: The hyperparameters for the models.

Model	NODE	ANODE	SONODE	(G)HBNODE	(G)NesterovNODE
n (Initialization)	1	2	1	1	1
n (Point Cloud)	2	3	2	2	2
h (Point Cloud)	20	20	13	14	14
p (MNIST)	0	5	4	4	4
n (MNIST)	1	6	5	5	5
h (MNIST)	92	64	50	50	50
p (CIFAR10)	0	10	9	9	9
n (CIFAR10)	3	13	12	12	12
h (CIFAR10)	125	64	50	51	51

Table 4: The hyperparameters for ODE-RNN integration models.

Model	NODE	ANODE	SONODE	(G)HBNODE	(G)NesterovNODE
d	1	1	2	2	2
n (Walker2D)	24	24	23	24	24
h_1 (Walker2D)	72	72	46	48	48
h_2 (Walker2D)	48	48	46	48	48

Table 5: The ξ hyperparameters for GHBNODE and GNesterovNODE.

Model	ξ
Initialization	learnable
MNIST	learnable
CIFAR	1.5
Point Cloud	2
Walker 2D	learnable

To solve the NesterovNODE in Eq. (9), we rewrite it in the differential-algebraic form given by Eq. (15). We solve Eq. (15) as follows. First, given $h(t_0)$, we compute $x(t_0)$. Using the computed $x(t_0)$, we solve the ODE given by the last two equations in Eq. (15), $x'(t) = m(t)$ and $m'(t) = -m(t) - f(h(t), t, \theta)$ by using an ODE solver. To get $h(t)$ for the second differential equation, we compute $h(t)$ from $x(t)$. Alternatively, the second differential equation can be expressed as $m'(t) = -m(t) - f(x(t), t, \theta)$, which we can think of applying a composite function $f(g(h(t), t, \theta)$ to $h(t)$. Finally, we compute $h(t_n)$ from $x(t_n)$.

We list some experimental details that are common to all experiments before presenting more details for each experiment. For ODE solvers in the experiments (`dopri5`, `euler`, `rk4`, `explicit_adams`), we use the implementation provided by `torchdiffeq` [1].

- fc_n is a fully-connected layer with the output dimension of size n.

[1] https://github.com/rtqichen/torchdiffeq

- HTanh: HardTanh(-5, 5).
- LReLU: LeakyRELU(0.3).
- tpad: Padding a set of features with the value of the time t. Specifically, this is equivalent to augmenting the feature tensor with shape $c \times x \times y$ to with a time tensor with shape $1 \times x \times y$ filled with the value of t, creating a time-augmented feature tensor with shape $(c+1) \times x \times y$.
- We use a learnable γ with for HBNODE/GHBNODE for all tasks.
- The values of ξ used for GHBNODE and GNesterovNODE used in all tasks are in Table 5.
- Except for the experiments in Section 6 where we investigate the effect of the variation of the Nesterov factor r, we choose $r = 3$ for the remaining experiments.
- $n^* = n$ for all models except for SONODE where $n^* = 2n$. The bounded activation function σ is chosen to be either tanh or hardtanh in PyTorch.
- The tolerance (for adaptive solvers) and step sizes (for fixed step-size solvers) used in the experiments are in Table 6.
- Other hyperparameters are in Table 3 and Table 4. The difference in the architecture between first-order NeuralODE methods (NODE, ANODE) and second-order NeuralODE methods (SONODE, HBNODE/GHBNODE, NesterovNODE/GNesterovNODE) happens because of the structural difference in their dynamics function. In particular, second-order NeuralODE methods use extra states to model the momentum of the original hidden states. To get roughly similar numbers of parameters for first-order NeuralODE methods and second-order NeuralODE methods, there must be differences in their architecture. In choosing the architecture for our method, we have made sure that there is no meaningful architecture difference in our methods compared to the baseline second-order Neural ODE methods (SONODE and HBNODE/GHBNODE).

D.1 Experimental details used in Section 3 Silverbox Initialization Test

The Silverbox dataset [58] is as follows: Given the input voltage $V1(t)$, the models must predict the output voltage $V2(t)$ and the experiment is evaluated over 64 time steps. In this experiment, we use the dopri5 solver (implemented in torchdiffeq) with a tolerance of 10^{-7} for adaptive step sizes in both forward and backward passes. Similar to [60], we parameterize the dynamic f of all methods with a dense layer. The network architecture is:

- ODE layer: $\text{input}_{n^*+1} \to \text{fc}_n$

D.2 Experimental details used in the Point Cloud benchmark in Section 5.2

Following the setting in [60, 9], we use a 3-layer neural network to parameterize the function $f(\boldsymbol{h}(t), t, \theta)$ on the right hand side of the ODE-based models in our study. We integrate the ODE from $t_0 = 1$ to $T = 2$, and pass the output $\boldsymbol{h}(T)$ at time T to a dense classifier. We also set the tolerance of the ODE solvers to be 10^{-7} so that the effect of numerical error is minimized. Visualization of the point cloud evolution for a random run of each model is in Fig. 11.

- Initial Velocity layer: $\text{input}_2 \to \text{fc}_h \to \text{HTanh} \to \text{fc}_h \to \text{HTanh} \to \text{fc}_n$
- ODE layer: $\text{input}_{n^*} \to \text{fc}_h \to \text{ELU} \to \text{fc}_h \to \text{ELU} \to \text{fc}_n$
- Output layer: $\text{input}_n \to \text{fc}_1 \to \text{Tanh}$

D.3 Experimental details for MNIST/CIFAR10 in Section 5.1

In our experiment, we parameterize $f(\boldsymbol{h}(t), t, \theta)$ in the NODE-based layer using a 3-layer neural network. The output of this NODE-based layer is then passed through a dense layer to perform the classification task. Following the augmentation approach in ANODE [8], we add p additional channels to input image, augmenting the number of image channels from c to $c + p$ where p is differently chosen for each method. The values of p chosen for each model are specified in Table 3. For SONODE, HBNODE, GHBNODE, NesterovNODE and GNesterovNODE, we also incorporate velocity or momentum of the same shape as the augment state. We also present extra experiment results for MNIST in Fig. 12.

The architecture for MNIST:

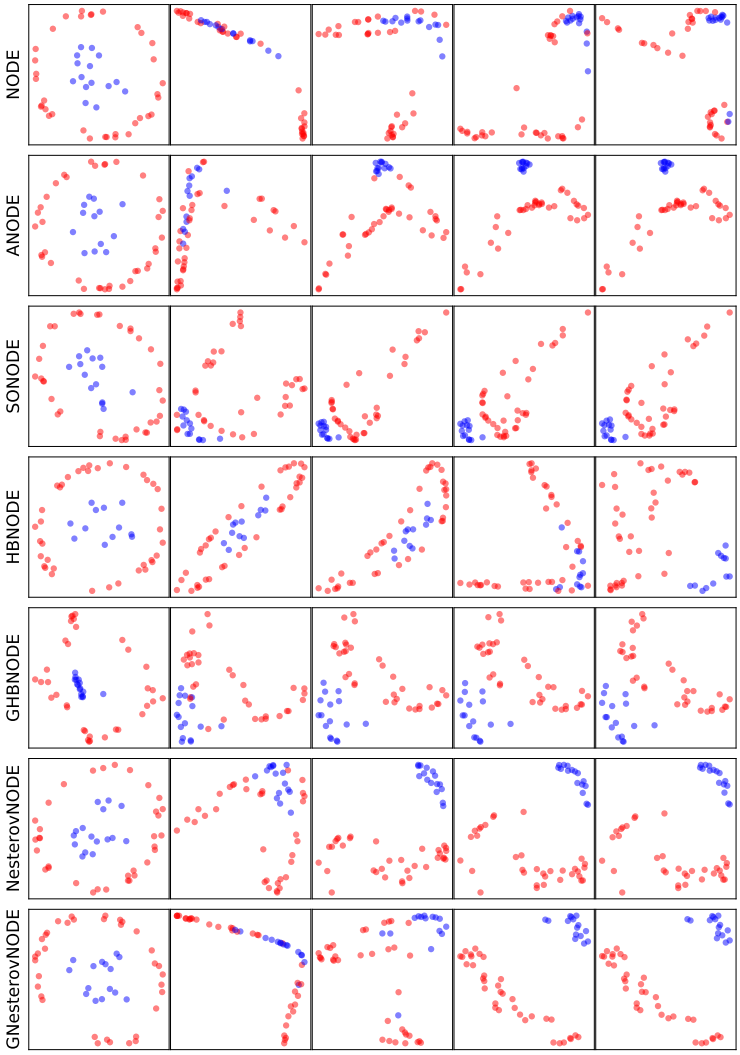

Figure 11: Visualization of the features transform by NODE, ANODE, SONODE, HBNODE, GHBNODE, NesterovNODE and GNesterovNODE after the first 100 epochs of a random run.

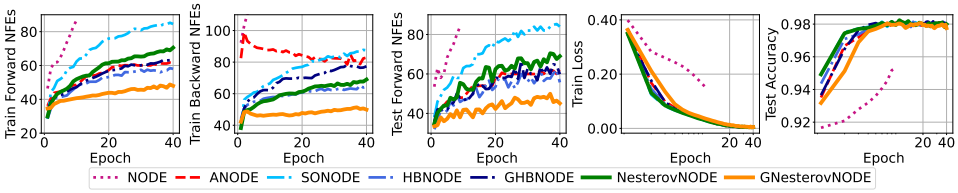

Figure 12: Contrasting the NFEs, training time, training loss, and test accuracy of NODE [4], ANODE [9], HBNODE/GHBNODE [60], and our methods NesterovNODE/GNesterovNODE on the MNIST dataset (Tolerance: 10^{-5}). The run for NODE is stopped due to long running time. The x-axes of the plots for training loss and testing accuracy are scaled logarithmically for visibility.

- Initial Velocity layer: $\text{input}_{1\times28\times28} \to \text{conv}_{h,1} \to \text{LReLU} \to \text{conv}_{h,3} \to \text{LReLU} \to \text{conv}_{2n-1,1}$
- ODE layer: $\text{input}_{n^*\times28\times28} \to \text{tpad} \to \text{conv}_{h,1} \to \text{ReLU} \to \text{tpad} \to \text{conv}_{h,3} \to \text{ReLU} \to \text{tpad} \to \text{conv}_{n,1}$
- Output layer: $\text{input}_{n\times28\times28} \to \text{fc}_{10}$

The architecture for CIFAR10:

- Initial Velocity layer: $\text{input}_{3\times28\times28} \to \text{conv}_{h,1} \to \text{LReLU} \to \text{conv}_{h,3} \to \text{LReLU} \to \text{conv}_{2n-3,1}$
- ODE layer: $\text{input}_{n^*\times32\times32} \to \text{tpad} \to \text{conv}_{h,1} \to \text{ReLU} \to \text{tpad} \to \text{conv}_{h,3} \to \text{ReLU} \to \text{tpad} \to \text{conv}_{n,1}$
- Output layer: $\text{input}_{n\times32\times32} \to \text{fc}_{10}$

D.4 Experimental details for Walker2D in Section 5.3

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 [53]. Following the procedure in HBNODE [60], we randomly take out 10% of the data to make the time series irregularly-sampled. Each input sequence consists of 64 timestamps, which are recurrently fed through a hybrid technique, with the output of the hybrid method being transferred to a single dense layer to form the output time series. The goal is to generate an auto-regressive forecast with an output time series that is as close as the input sequence when shifted one time stamp to the right. The RNN and ODE are parameterized by a 3-layer network. The network architecture is:

- ODE layer: $\text{input}_{n^*} \to \text{fc}^*_n \to \text{Tanh} \to \text{fc}_n \to \text{Tanh} \to \text{fc}_n$
- RNN layer: $\text{input}_{dn+k} \to \text{fc}_{h_1} \to \text{Tanh} \to \text{fc}_{h_2} \to \text{Tanh} \to \text{fc}_{dn}$
- Output layer: $\text{input}_n \to \text{fc}_{17}$

Table 6: Solvers, tolerance, and step sizes used for the experiments.

Experiments	Solvers	Tolerance	Step sizes
Silverbox Initialization (Section 3)	dopri5	10^{-7}	N/A
Point Cloud separation (Section 5.2)	dopri5	10^{-7}	N/A
MNIST/CIFAR10 (Section 5.1)	dopri5	10^{-5}	N/A
Walker2D (Section 5.3)	dopri5	10^{-7}	N/A
Stability of NesterovNODE (Section 6)	Euler, RK4, Explicit Adams, dopri5	10^{-5} for dopri5	0.1, 0.2, 0.5 (for details, refer to Fig. 9 and Fig. 10)

D.5 Experimental details for Continous Normalizing Flow for VAE on the MNIST dataset in Section 5.4

Our training and the baseline FFJORD-NODE model follow the setting in Section 4.3 in [14]. We use the dopri5 solver with a tolerance of 10^{-5}. We use the identity function as σ, and $\xi = 2$ for GHBNODE and GNesterovNODE. Aggregated results about the experiment can be found in Table 7.

Table 7: Test negative ELBO (lower is better) and mean NFEs over all epochs of FFJORD-NODE, FFJORD-HBNODE and our method FFJORD-GNesterovNODE for use in variational inference with a continuous normalizing flow model, i.e. FFJORD [14], trained on the binarized MNIST dataset. We also include the reported results from [14] (in parentheses) in addition to our reproduced results. (Tolerance: 10^{-5}).

Method	Mean Forward NFEs	Mean Backward NFEs	Negative ELBO
FFJORD-NODE	97.92	100.76	88.30 (82.82)
FFJORD-GHBNODE	90.33	98.77	75.87
FFJORD-GNesterovNODE	**62.04**	**51.07**	**72.16**

E Additional experiments

E.1 Integration time

In the the differential-algebraic version of NesterovNODE, we reparametrize from $\mathbf{h}$ to $\mathbf{x}$ in order to eliminate the time-dependent damping coefficient $\frac{3}{t}$ in the Nesterov ODE given by Eq. 8. In

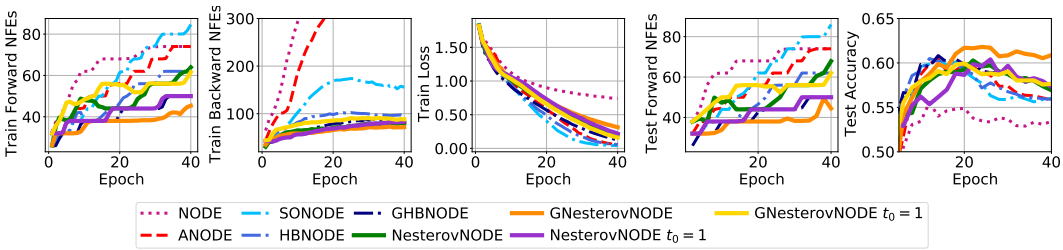

Figure 13: Contrasting the NFEs and accuracy of NODE [4], ANODE [9], HBNODE/GHBNODE [60], our methods NesterovNODE/GNesterovNODE and the methods NesterovNODE/GNesterovNODE with starting integration time $t_0 = 1$ on the CIFAR10 dataset (Tolerance: 10^{-5}).

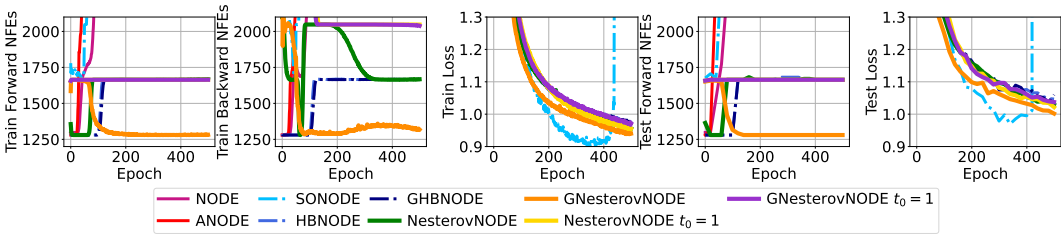

Figure 14: Contrasting the NFEs and losses of NODE [4], ANODE [9], HBNODE/GHBNODE [60], our methods NesterovNODE/GNesterovNODE and the methods NesterovNODE/GNesterovNODE with starting integration time $t_0 = 1$ on the Walker2D dataset (Tolerance: 10^{-7}).

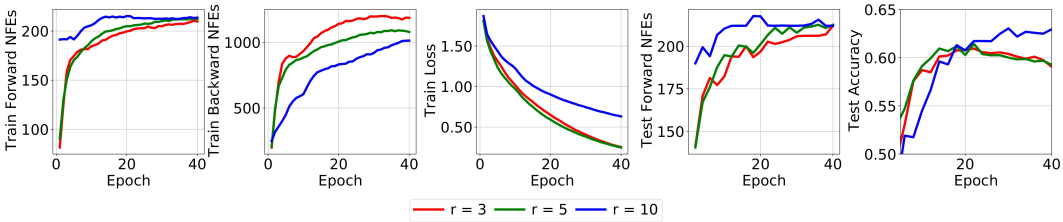

Figure 15: Test accuracy for GNesterovNODE on CIFAR10 with varying Nesterov factors with a lower tolerance value (Tolerance: 10^{-7}).

particular, we set $\mathbf{h}(t) = k(t)\mathbf{x}(t)$ and then find $k(t)$ such that Eq. 8 written in terms of $x(t)$ matches the Heavy-Ball ODE with a constant damping coefficient given by Eq. 5. We find that $k(t) = t^{-\frac{3}{2}} e^{\frac{t}{2}}$ satisfies this requirement. Solving this Heavy-Ball ODE with respect to $\mathbf{x}(t)$ is easier and more stable than solving the Nesterov ODE with respect to $\mathbf{h}(t)$. To strengthen our proposed trick, we conduct two experiments (CIFAR 10 and Walker2d) using the original version of Nesterov in Eq. 10 with the starting integration time $t_0 = 1$. We have to change the starting integration to $t_0 = 1$ due to the singularity in $t_0 = 0$. The empirical results on CIFAR10 (Fig. 13) and Walker2d (Fig. 14) benchmarks show that the approach of changing starting integration time shows much worse accuracy and efficiency than using our reparametrization trick.

E.2 The Effect of Nesterov factor regarding the solver's tolerance

We run another CIFAR10 expreriment using smaller tolerance (10^{-7}) in order to study the effect of Nesterov factor regarding the solver's tolerance. As shown in Fig. 15, when the tolerance are shrunk small enough, the forward NFEs of the smaller factor (3, 5 compared to 10) are still smaller, although the opposite is true for backward NFEs. Moreover, the larger factor converges to a better solution (higher accuracy) despite converging to its best solution later than the smaller factor.

Table 8: GPU memory consumption of the ODE-based method on the CIFAR10 dataset.

Method	Maximum CUDA memory consumption (GiB)
NODE	3.2046
ANODE	2.3292
SONODE	2.2034
HBNODE	2.2346
GHBNODE	2.2456
NesterovNODE	2.2346
GNesterovNODE	2.2580

E.3 GPU memory consumption

We present the maximum CUDA memory consumption of the models used in the CIFAR10 experiments in Table 8. We extract the numbers using the function `max_memory_allocated` [2] in PyTorch.

E.4 Wall-clock time advantage of NesterovNODEs/GNesterovNODEs

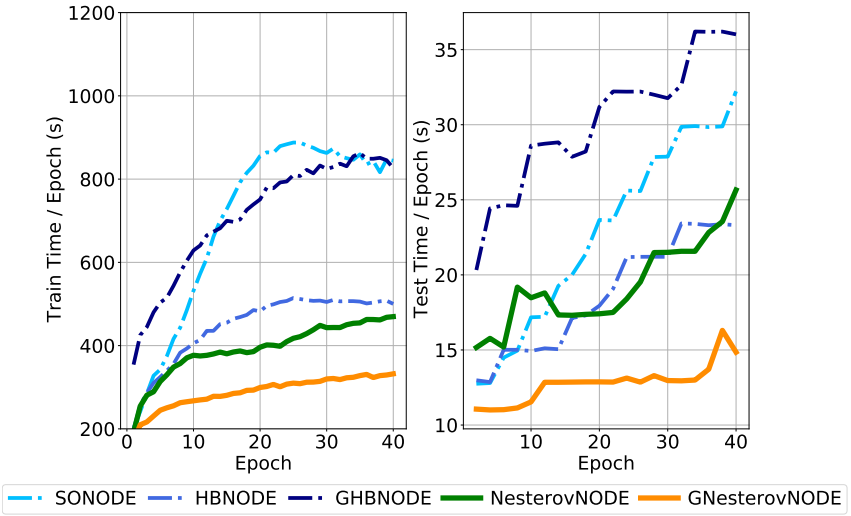

Figure 16: The total wall-clock training and testing time of SONODE, HBNODE, GHBNODE, NesterovNODE, and GNesterovNODE.

In this Figs. 16 and 17, we show the advantage of NesterovNODEs in wall-clock time. NesterovNODEs/GNesterovNODEs uses less time to achieve a comparable accuracy compared to other methods.

E.5 The Effect of using Seminorm for reducing backward NFEs on GNesterovNODE

We tested the effect of the seminorm method [22] on GNesterovNODE for the CIFAR10 classification task and demonstrated the results in Fig. 18. We observe that using the adjoint seminorm method slightly reduces the NFEs compared to using only GNesterovNODE while also considerably reducing the test accuracy. This effect is expected because while the adjoint seminorm method helps reduce the NFEs, its semi-strong constraints might affect the final ODE solutions.

E.6 Human Activity Dataset

We verify the advantage of the GNesterovNODE over the baseline NODE and GHBNODE for time series data on the Human Activity dataset [29]. This dataset consists of time series from five individuals doing various activities: walking, sitting, lying, etc. The data contains 3D positions of tags attached to those individuals' belt, chest and ankles (12 features in total). After preprocessing, the dataset has 6554 sequences of 211 time points. In this task, the model classifies each time point

[2] https://pytorch.org/docs/stable/generated/torch.cuda.max_memory_allocated.html

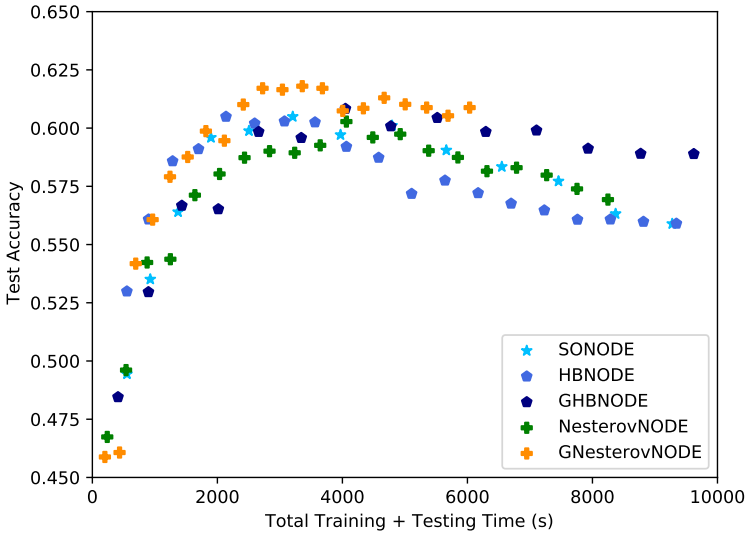

Figure 17: Test accuracy vs the corresponding wall-clock time for SONODE, HBNODE, NesterovNODE, and GNesterovNODE.

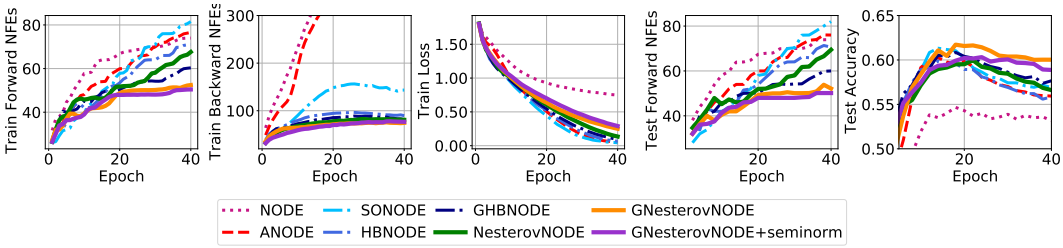

Figure 18: CIFAR10 with the adjoint seminorm method [22].

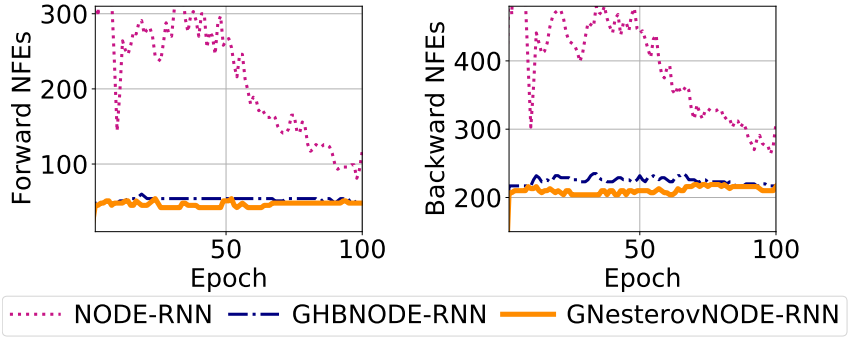

Figure 19: Contrasting the NFEs of NODE-RNN, GHBNODE-RNN and our GNesterovNODE-RNN on the Human Activity benchmark (Per-time-point classification) [29] (Tolerance: 10^{-7}).

into one of seven types of activities (walking, sitting, etc.). We use the ODE-RNN architecture described in Section 4.5 of [46] as the baseline model, with dopri5 adaptive solver and tolerance 10^{-7}. For the GNesterovNODE, we use $\sigma = \tanh$, and ξ is a learnable scalar. Figure 19 and Table 9 show that GNesteroveNODE-RNN achieves the best accuracy and the smallest NFEs in both forward and backward pass.

F Solving Neural Differential-Algebraic Equations

Our implementation uses an ODE to calculate the DAE. An alternative approach is using DAE solvers to solve the DAE, which are implemented by the `DifferentialEquations.jl` library [44, 43].

Table 9: Test accuracy and mean NFEs over all epochs of NODE-RNN, GHBNODE-RNN and our method GNesterovNODE-RNN on the Human Activity benchmark (Per-time-point classification) [29] (Tolerance: 10^{-7}).

Method	Mean Forward NFEs	Mean Backward NFEs	Accuracy
NODE-RNN	220.74	396.42	0.829 ± 0.016
GHBNODE-RNN	51.85	221.26	0.838 ± 0.017
GNesterovNODE-RNN	**46.44**	**210.84**	$\mathbf{0.840} \pm \mathbf{0.016}$

Although Dormand-Prince 5(4) [7] is a common choice for adaptive ODE solver, Tsitouras 5(4) [54] is a more efficient method, which the libraries `DifferentialEquations.jl` [3] [44] and `Diffrax` [4] [21] have implemented.

[3] https://github.com/SciML/DifferentialEquations.jl
[4] https://github.com/patrick-kidger/diffrax