Nesterov Neural Ordinary Differential Equations

Anonymous Authors[1]

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. Taking the advantage of the convergence rate $\mathcal{O}(k^2)$ of the NAG scheme, NesterovNODEs 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 NesterovNODE also satisfies a NesterovNODE, 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 NesterovNODEs on a wide range of practical applications including point cloud separation, image classification, and sequence modeling. Compared to NODEs, NesterovNODEs 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 (Haber & Ruthotto, 2017; Weinan, 2017; Ruthotto & Haber, 2019; Chen et al., 2018). 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 (Köhler et al., 2020; Zhong et al., 2020), irregular time series modeling (Rubanova et al., 2019b; De Brouwer et al., 2019), mean-field games (Ruthotto et al., 2020), and generative modeling (Grathwohl et al., 2019b; Yildiz et al., 2019). NODEs model the dynamics of hidden state $\boldsymbol{h}(t) \in \mathbb{R}^N$ in neural 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 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 (Pontryagin, 2018)

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

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

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

NODEs solve both the ODE 1 in its forward pass and the ODEs 2 and 3 in its backward pass using black-box numerical ODE solvers. 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 (Grathwohl et al., 2019a; Dupont et al., 2019b; Massaroli et al., 2020; Norcliffe et al., 2020; Finlay et al., 2020).

1.1 Contribution

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

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

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. We prove that the adjoint equation used to compute the gradients for updating the parameters θ in a NesterovNODE also follows a Nesterov ODE. Thus, the NFEs in both forward and backward passes of NesterovNODEs are significantly reduced, especially when the solvers are used with small error tolerances.
3. We prove that the spectrum of the NesterovNODE is well-structured. This property of NesterovNODEs helps alleviate the vanishing gradient issue during training and allows the model to capture long-term dependencies in the data.
4. To eliminate the potential blow-up problem in training NesterovNODEs, we further develop the Generalized NesterovNODEs (GNesterovNODEs) by introducing skip connections (He et al., 2016) and gating mechanisms (Hochreiter & Schmidhuber, 1997) into NesterovNODEs.

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 (Xia et al., 2021) 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 then present the algorithm and analysis of the NesterovNODE. In Section 3, we propose the 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 up with concluding remarks. Proofs and additional experimental details are provided in the Appendix.

2 Nesterov Neural Ordinary Differential Equations

2.1 An Optimization Perspective of Neural ODEs

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.

NODEs implicitly perform gradient descent 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 $\boldsymbol{x}_t = \boldsymbol{x}_{t-1} - s\nabla F(\boldsymbol{x}_t)$ 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 solve the ODE limit of the GD where the gradient $-\nabla F(\boldsymbol{x}_t)$ is parameterized by a neural network $f(\boldsymbol{h}(t), t, \theta)$.

Heavy ball neural ordinary differential equations HBNODEs are proposed in (Xia et al., 2021). This model takes advantage of the acceleration of heavy ball (HB) momentum (Polyak, 1964) 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:

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

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

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

where $\gamma > 0$ is the damping parameter, which can be set to a tunable constant 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 (Nesterov, 1983) 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 (Nesterov, 1983; Su et al., 2014). 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.

2.2 Nesterov Ordinary Differential Equation

For the completeness of the paper, we recall the Nesterov acceleration method and its exact limit as a second-order ODE. More detail can be found in (Su et al., 2014).

Nesterov acceleration gradient method (Nesterov, 1983) takes the following form: given initial points x_0 and $y_0 = x_0$, the sequence $\{(x_k, y_k)\}_k$ is defined inductively as:

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

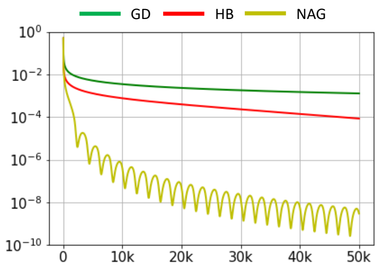

Figure 1. Comparing the convergence of GD, HB and NAG for solving the quadratic optimization problem $\min_{\boldsymbol{x}} F(\boldsymbol{x}) = 1/2\, \boldsymbol{x}^\top \mathbf{L}\boldsymbol{x} - \boldsymbol{x}^\top \boldsymbol{b}$ where $\mathbf{L} \in \mathbb{R}^{d\times d}$ is the Laplacian of a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

By combining two equations in system 7, we obtain

$$\frac{x_{k+1}-x_k}{\sqrt{s}} = \frac{k-1}{k+2}\frac{x_k - x_{k-1}}{\sqrt{s}} - \sqrt{s}\nabla F(y_k). \quad (8)$$

To derive the continuous limit of the Nesterov scheme, we set $x_k = \mathbf{h}(k\sqrt{s}) = \mathbf{h}(t)$ with $t = k\sqrt{s}$ and some smooth function $\mathbf{h}$. Then by using Taylor's expansion, we can write

$$x_{k+1} = \mathbf{h}(t+\sqrt{s}) = \mathbf{h}(t) + \mathbf{h}'(t)\sqrt{s} + \frac{1}{2}\mathbf{h}''(t)s + o(s),$$

$$x_{k-1} = \mathbf{h}(t-\sqrt{s}) = \mathbf{h}(t) - \mathbf{h}'(t)\sqrt{s} + \frac{1}{2}\mathbf{h}''(t)s + o(s).$$

Therefore,

$$\frac{x_{k+1}-x_k}{\sqrt{s}} = \mathbf{h}'(t) + \frac{1}{2}\mathbf{h}''(t)\sqrt{s} + o(\sqrt{s}),$$

$$\frac{k-1}{k+2}\frac{x_k - x_{k-1}}{\sqrt{s}} = \mathbf{h}'(t) - \left(\frac{1}{2}\mathbf{h}''(t) + \frac{3}{t}\mathbf{h}'(t)\right)\sqrt{s} + o(\sqrt{s}),$$

$$\nabla F(y_k) = \nabla F(\mathbf{h}(t)) + o(\sqrt{s}).$$

By substituting the above equations to Eq. 8 and then comparing the terms with $\sqrt{s}$, we obtain the Nesterov ODE

$$\mathbf{h}''(t) + \frac{3}{t}\mathbf{h}'(t) + \nabla F(\mathbf{h}(t)) = 0. \quad (9)$$

Remark 1 (Nesterov factor)**.** *The constant 3 in the coefficient of* $\mathbf{h}'(t)$ *in Eq. 9 is originally from the approximation* $(k-1)/(k-3) = 1 - 3/k + \mathcal{O}(1/k^2)$*. This constant will be replaced by a constant* r *if the factor* $(k-1)/(k-3)$ *in Eq. 7 is replaced by* $(k-1)/(k+r-1)$*. It is proved in (Su et al., 2014) that the Nesterov ODE still holds the quadratic convergence rate when 3 is replaced by any number* $r > 3$*.*

Remark 2 (Numerical stability)**.** *In computation, both the ODE 4 and the Nesterov ODE are solved using the Euler method. 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 (Su et al., 2014) 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.*

2.3 Nesterov Neural Ordinary Differential Equation

One can parameterize $\nabla F(\mathbf{h}(t))$ in Eq. 9 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)

$$\mathbf{h}''(t) + \frac{3}{t}\mathbf{h}'(t) + f(\mathbf{h}(t), t, \theta) = 0, \quad (10)$$

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

$$\begin{cases} \mathbf{h}'(t) = \mathbf{m}(t), \\ \mathbf{m}'(t) = -\frac{3}{t}\mathbf{m}(t) - f(\mathbf{h}(t), t, \theta). \end{cases} \quad (11)$$

However, because of the singularity created by the coefficient $\frac{3}{t}$, the training process based directly on Eq. 10 and 11 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. 9 becomes

$$k(t)\mathbf{x}''(t) + \left(2k'(t) + \frac{3}{t}k(t)\right)\mathbf{x}'(t) + \left(k''(t) + \frac{3}{t}k'(t)\right)\mathbf{x}(t) + \nabla F(\mathbf{h}(t)) = 0. \quad (12)$$

We observe that $2k'(t) + \frac{3}{t}k(t) = k(t)$. By dividing both sides of Eq. 12 by $k(t)$, we obtain:

$$\mathbf{x}''(t) + \mathbf{x}'(t) + f(\mathbf{h}(t), t) = 0, \quad (13)$$

where

$$f(\mathbf{h}(t), t) = \frac{k''(t) + \frac{3}{t}k'(t)}{k(t)}\mathbf{x}(t) + \frac{1}{k(t)}\nabla F(\mathbf{h}(t)) = \frac{1}{4}\left(t^2 - 3\right)t^{-\frac{1}{2}}e^{-\frac{t}{2}}\mathbf{h}(t) + t^{\frac{3}{2}}e^{\frac{-t}{2}}\nabla F\left(\mathbf{h}(t)\right).$$

Let $\mathbf{m}(t) = \mathbf{x}'(t)$. Then Eq. 9 is equivalent to the following differential-algebraic system

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

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

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

The singularity at $t = 0$ of the NesterovNODE in Eqs. 10 and 11 is now moved to the algebraic equation of the above differential-algebraic system. This helps us to avoid the instability issue caused by the singularity inside the ODE solvers when solving the NesterovNODE. Eqs. 10 and 15 define the forward ODE for the NesterovNODE. To efficiently update the parameters during the training process based on NesterovNODEs, we use the adjoint sensitivity given in Propositions 1 and 2 below.

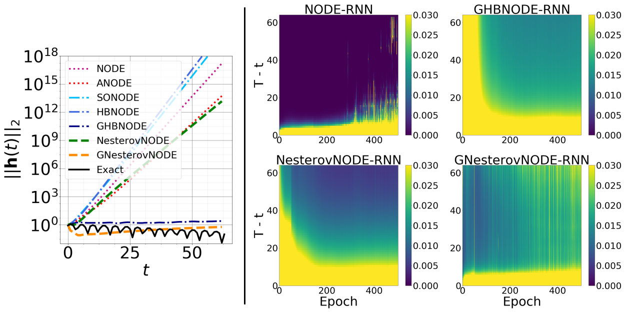

Figure 2. (Left) Contrasting the increase in the $l2$-norm norm of $h(t)$ for NODE, ANODE, SONODE, HBNODE, GHBNODE, NesterovNODE, and GNesterovNODE over long integration time on the Silverbox Initialization task. (Right) Plot of the $l2$-norm of the adjoint states for ODE-RNN, GHBNODE-RNN, NesterovNODE-RNN and GNesterovNODE-RNN backpropagated from the last time stamp.

Proposition 1 (Adjoint equation for the second-order NesterovNODE)**.** *The adjoint state* $\mathbf{a}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}$ *of the NesterovNODE given in Eq. 10 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 (Adjoint equation for the differential-algebraic NesterovNODE)**.** *The 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 differential-algebraic NesterovNODE system given in Eq. 15 satisfy the following differential-algebraic adjoint system*

$$\begin{cases} \mathbf{a_h}(t) = t^{\frac{3}{2}} e^{-\frac{t}{2}} \mathbf{a_x}(t), \\ \mathbf{a}'_\mathbf{x}(t) = t^{-\frac{3}{2}} e^{\frac{t}{2}} \mathbf{a_m}(t) \cdot \dfrac{\partial f(\mathbf{h}(t), t, \theta)}{\partial \mathbf{h}}, \\ \mathbf{a}'_\mathbf{m}(t) = \mathbf{a_m}(t) - \mathbf{a_x}(t). \end{cases} \tag{17}$$

3 Generalized Nesterov Neural Ordinary Differential Equations

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. 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. We address these potential blow-up by applying the bounded 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 $m(t)$ of the NesterovNODE. In addition, the residual term $\xi \mathbf{h}(t)$, which stands for a skip connection (He et al., 2016), is also added into the governing equation of $m(t)$, which benefits training and generalization of GNesterovNODEs. The GNesterovNODE differential-algebraic system is then given by

$$\begin{cases} \mathbf{h}(t) = \sigma(t^{\frac{-3}{2}} e^{\frac{t}{2}})\mathbf{x}(t), \\ \mathbf{x}'(t) = \sigma(\mathbf{m}(t)), \\ \mathbf{m}'(t) = -\mathbf{m}(t) - \sigma(f(\mathbf{h}(t), t, \theta)) - \xi \mathbf{h}(t), \end{cases} \tag{18}$$

Fig. 2 shows that GHBNODE can indeed control the growth of $\boldsymbol{h}(t)$ effectively. The following proposition formulates the adjoint sensitivity of the GNesterovNODE.

Proposition 3 (Adjoint equation for GNesterovNODE)**.** *The 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. 18 satisfy the following differential-algebraic adjoint system*

$$\begin{cases} \mathbf{a_h}(t) = \sigma(t^{\frac{-3}{2}} e^{\frac{t}{2}})^{-1} \mathbf{a_x}(t), \\ \mathbf{a}'_\mathbf{x}(t) = t^{\frac{-3}{2}} e^{\frac{t}{2}} \mathbf{a_m}(t) \left[\dfrac{\partial \sigma(f(\mathbf{h}(t), t))}{\partial \mathbf{h}} + \xi \mathbf{I} \right], \\ \mathbf{a}'_\mathbf{m}(t) = \mathbf{a_m}(t) - \mathbf{a_x}(t)\sigma'(\mathbf{m}(t)). \end{cases} \tag{19}$$

4 Learning long-term dependencies – Vanishing gradient

In learning long-term dependencies in the input data, the vanishing gradient is one of the major issues (Pascanu et al., 2013). 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.

Following the lines in (Xia et al., 2021), for a NODE given by $\mathbf{z}'(t) = g(\mathbf{z}(t), t, \theta)$, we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_T} \exp\left\{ -\int_T^t \frac{\partial g(\mathbf{z}(s), s, \theta)}{\partial \mathbf{z}} ds \right\}. \tag{20}$$

For the GNesterovNODE given in Eq. 18, 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. 20 as

$$\begin{bmatrix} \dfrac{\partial \mathcal{L}}{\partial \mathbf{x}_t} & \dfrac{\partial \mathcal{L}}{\partial \mathbf{m}_t} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \mathcal{L}}{\partial \mathbf{x}_T} & \dfrac{\partial \mathcal{L}}{\partial \mathbf{m}_T} \end{bmatrix} \cdot \exp(M), \tag{21}$$

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. 21, 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 $\|v\|_2 = 1$ and

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

The term $v\exp(U)$ plays the essential role in the nonvanishing gradient issue. We prove that

Proposition 4. *The matrix U can always be written in the form $U = \begin{bmatrix}U_{large} & P\\ \mathbf{0} & U_{small}\end{bmatrix}$ 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) $\frac{1}{2}(t-T)$. Moreover, the size of U_{large} is at least half the size of M.*

As a consequence, $\exp(U) = \begin{bmatrix}\exp(U_{\text{large}}) & \tilde{P}\\ \mathbf{0} & \exp(U_{\text{small}})\end{bmatrix}$ and $\|v\exp(U)\|_2 \geq \|v_{\text{large}}\exp(U_{\text{large}})\|_2$. 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 (Brockman et al., 2016) in consideration, which requires learning long-term dependency effectively (Lechner & Hasani, 2020b). We train ODE-RNN (Rubanova et al., 2019a), GHBNODE-RNN (Xia et al., 2021), NesterovNODE-RNN and GNesterovNODE-RNN on this benchmark dataset. Fig. 2 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.

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) (Dupont et al., 2019a), the Second Order NODE (SONODE) (Norcliffe et al., 2020), HBNODE/GHBNODE (Xia et al., 2021) 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. ANODE augments the space on which the ODE is solved, and SONODEs solve a second-order ODE. We aim to show that: (i) NesterovNODEs/GNesterovNODEs require much 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 capture better long-term dependencies than the baselines and achieve better results in long-sequence modeling tasks.

Table 1. The parameters count for the models in point cloud separation, image classifications, and the Walker2D kinematic simulation tasks. The methods have similar numbers of parameters to ensure fair comparison.

Model	*Point Cloud*	*MNIST*	*CIFAR10*	*Walker2D*
NODE	545	85316	173611	9929
ANODE	587	85462	172452	10019
SONODE	541	86179	171635	11471
HBNODE	582	85931	172916	10099
GHBNODE	582	85931	172916	10099
NesterovNODE	581	85930	172915	10098
GNesterovNODE	581	85930	172915	10098

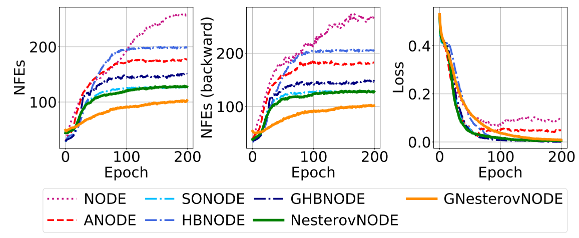

Figure 3. Contrasting the NFEs and training loss of NODE, ANODE, HBNODE/GHBNODE, and our methods NesterovNODE/GNesterovNODE on the point cloud benchmark. The results here are averaged over 50 independent runs (Tolerance: 10^{-7}).

For all the experiments, we use Adam (Kingma & Ba, 2015) as the optimizer and Dormand-Prince-45 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 for fair comparisons. Table 1 lists the number of parameters of the models. Other training/model/dataset details are provided in the Appendix. All of our results are averaged over 5 runs with different seeds. Our experiments are conducted on a server with 6 NVIDIA 2080Ti GPUs with 11GB of memory per GPU.

5.1 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 ra-

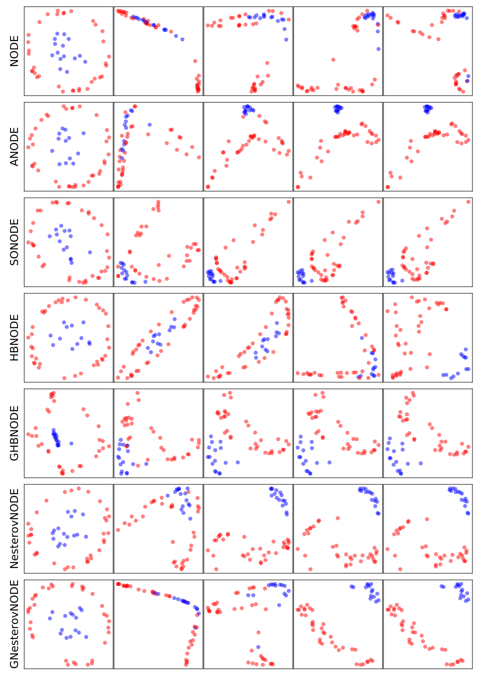

Figure 4. Visualization of the features transform by NODE, ANODE, SONODE, HBNODE, GHBNODE, NesterovNODE and GNesterovNODE after the first 100 epochs of a random run.

dius of 0.85 and 1, respectively, i.e. $0.85 < ||\boldsymbol{r}|| < 1.0$. Following the setting in (Xia et al., 2021; Dupont et al., 2019a), 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. Fig. 3 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. Also, in Fig. 4, we plot the evolution of the point cloud separation through 100 epochs for a random run of each model in our study. Like SONODE, HBNODE and GHBNODE, NesterovNODE and 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.

5.2 Image classification

We validate the accuracy and efficiency advantage of NesterovNODE and GNesterovNODE for image classification tasks on the MNIST (Deng, 2012) and CIFAR10 (Krizhevsky et al., 2009) in comparison with other ODE-based baselines. We follow the same training and model settings as in (Xia et al., 2021; Dupont et al., 2019b).

Table 2. The test accuracy on CIFAR10/MNIST. Previous methods can be divided into two groups, first-order ODE-based methods and second-order ODE-based methods. NODE and ANODE fall into the first group, while SONODE and HBNODE/GHBNODE belong to the second group. Our methods NesterovNODE/GNesterovNODE are more similar to the second group, but we also have an additional algebraic equation in our system. 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.

Model	*CIFAR10*	*MNIST*
NODE	0.5555	0.9657
ANODE	0.6046	0.9809
SONODE	0.6049	**0.9831**
HBNODE	0.6049	0.9823
GHBNODE	0.6083	0.9822
NesterovNODE	0.6028	0.9827
GNesterovNODE	**0.6159**	0.9828

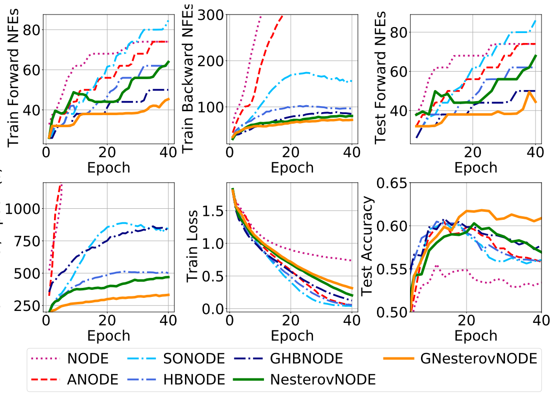

Figure 5. Contrasting the NFEs and accuracy of NODE (Chen et al., 2018), ANODE (Dupont et al., 2019a), HBNODE/GHBNODE (Xia et al., 2021), and our methods NesterovNODE/GNesterovNODE on the CIFAR10 dataset (Tolerance: 10^{-5}).

In particular, following the augmentation approach in ANODE (Dupont et al., 2019b), 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.[1] For SONODE, HBNODE, GHBNODE, NesterovNODE and GNesterovNODE, we also incorporate velocity or momentum of the same shape as the augment state. 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.

NFEs. As shown in Figs. 5 and 6, 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

[1]We choose $p = 0, 5, 4, 4, 4, 4, 4/0, 10, 9, 9, 9, 9, 9$ on MNIST/CIFAR10 for NODE, ANODDE, SONODE, HBNODE, GHBNODE, NesterovNODE, and GNesterovNODE.

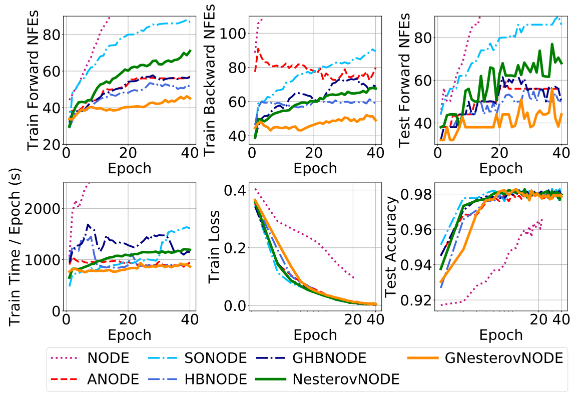

Figure 6. Contrasting the NFEs, training time, training loss, and test accuracy of NODE (Chen et al., 2018), ANODE (Dupont et al., 2019a), HBNODE/GHBNODE (Xia et al., 2021), and our methods NesterovNODE/GNesterovNODE on the MNIST dataset (Tolerance: 10^{-5}). The run for NODE is stopped due to long running time. The x-axes of the plots for training loss and testing accuracy are scaled logarithmically for visibility.

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 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 2 shows that our GNesterovNODE achieves the highest test accuracy on CIFAR10. On MNIST, GNesterovNODE 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 GNesterovNODE above, which reduces the model complexity and leads to better generalization.

Note that GNesterovNODE improves over NesterovNODE in both accuracy and efficiency. 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.

5.3 Walker2D kinematic simulation

In this section, we investigate NesterovNODE/GNesterovNODE when applied on time-series data. In particular, we compare our methods with the baseline methods on the Walker2D kinematic simulation task (Brockman et al., 2016). The data is generated from a kinematic simulation of a person walking using a pre-trained policy, with the goal of learning the MuJoCo physics engine's kinematic simulation. 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

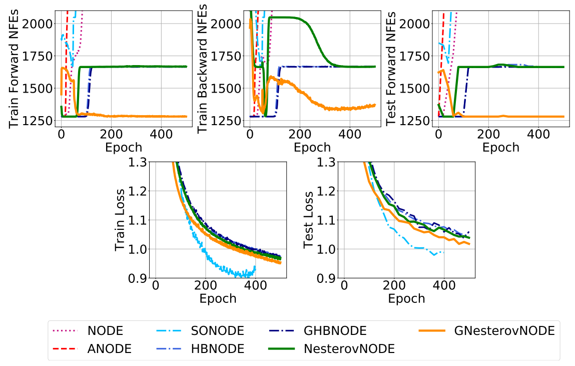

Figure 7. Contrasting the NFEs and losses of NODE (Chen et al., 2018), ANODE (Dupont et al., 2019a), HBNODE/GHBNODE (Xia et al., 2021), and our methods NesterovNODE/GNesterovNODE on the Walker2D dataset (The tolerance is 10^{-7}). The run for SONODE is stopped due to long running time.

an auto-regressive forecast with an output time series that is as similar to the input sequence as possible when moved one timestamp to the right. The RNN and ODE are parameterized by a 3-layer network and the number of parameters are shown in Table 1. As shown in Fig. 7, our NesterovNODE and GNesterovNODE 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. Although SONODE achieves much smaller losses, its NFEs are too high, thus training SONODE for this task is much more time-consuming compared to our Nesterov-based methods.

6 Empirical Analysis

6.1 Effect of the Nesterov Factor on NesterovNODE

As stated in (Su et al., 2014), the "magic constant" 3 can be replaced by a constant $r > 3$ while maintaining a convergence rate of $O(k^2)$. 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 3, 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

Table 3. Test accuracy for GNesterovNODE on CIFAR10 with varying Nesterov factors r.

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

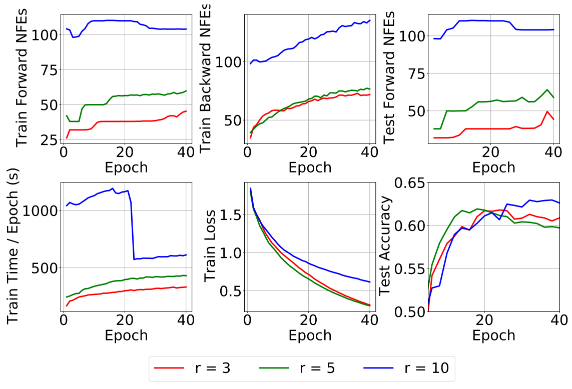

Figure 8. Contrasting different values of the Nesterov factor r on CIFAR10 training with GNesterovNODE (Tolerance: 10^{-5}).

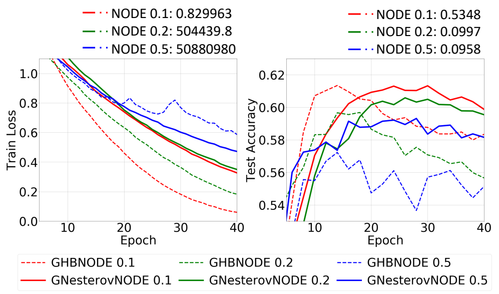

Figure 9. Train loss and test accuracy for various large step sizes $(0.1, 0.2, 0.5)$ of Euler solvers on CIFAR10 training with GHBNODE and GNesterovNODE (Tolerance: 10^{-5}).

$\mathbf{h}'(t)$, represents the friction force of the model (see (Su et al., 2014, Section 4)). When r is larger, the friction force resist the movement of $\mathbf{h}(t)$ along the trajectory stronger, which slows down the convergence of the training loss.

6.2 Stability of NesterovNODE

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.

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 (Grathwohl et al., 2019a) and other regularizers applied on the solver and the learned dynamics (Finlay et al., 2020; Kelly et al., 2020; Ghosh et al., 2020; Pal et al., 2021). Other works employ input augmentation (Dupont et al., 2019b), data-control (Massaroli et al., 2020) and depth-variance (Massaroli et al., 2020) to reduce the NFEs needed to compute the ODE solution. 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 (Norcliffe et al., 2020) replaces the first-order ODE in 1 by a second ODE which can be solved as a system of first-order ODEs. HBNODE (Xia et al., 2021) 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 (Moreau & Bruna, 2017; Li et al., 2018; Sander et al., 2021)

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 (Arjovsky et al., 2016; Wisdom et al., 2016; Jing et al., 2017; Vorontsov et al., 2017; Mhammedi et al., 2017; Helfrich et al., 2018). Recently, learning long-term dependencies with NODEs has been explored. For example, (Lechner & Hasani, 2020a) 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 $\mathcal{O}(k^2)$ 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. The differential equations corresponding to other advanced optimization algorithms such as the alternating direction method of multipliers (ADMM) have been recently developed (Franca et al., 2018). It is interesting to explore how to incorporate those differential equations into NODEs. Another interesting study is to analyze the stiffness of NesterovNODEs, HBNODEs and NODEs, which we leave as future work.

References

Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

De Brouwer, E., Simm, J., Arany, A., and Moreau, Y. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/455cb2657aaa59e32fad80cb0b65b9dc-Paper.pdf.

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

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019a. URL https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019b. URL https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf.

Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. M. How to train your neural ode: the world of jacobian and kinetic regularization. In *ICML*, pp. 3154–3164, 2020. URL http://proceedings.mlr.press/v119/finlay20a.html.

Franca, G., Robinson, D., and Vidal, R. Admm and accelerated admm as continuous dynamical systems. In *International Conference on Machine Learning*, pp. 1559–1567. PMLR, 2018.

Ghosh, A., Behl, H., Dupont, E., Torr, P., and Namboodiri, V. Steer : Simple temporal regularization for neural ode. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 14831–14843. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/a9e18cb5dd9d3ab420946fa19ebbbf52-Paper.pdf.

Grathwohl, W., Chen, R. T., Bettencourt, J., and Duvenaud, D. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019a.

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., and Duvenaud, D. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=rJxgknCcK7.

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Helfrich, K., Willmott, D., and Ye, Q. Orthogonal recurrent neural networks with scaled Cayley transform. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1969–1978, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/helfrich18a.html.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., Tegmark, M., and Soljačić, M. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1733–1741. JMLR. org, 2017.

Kelly, J., Bettencourt, J., Johnson, M. J., and Duvenaud, D. K. Learning differential equations that are easy to solve. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 4370–4380. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/2e255d2d6bf9bb33030246d31f1a79ca-Paper.pdf.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Köhler, J., Klein, L., and Noé, F. Equivariant flows: exact likelihood generative learning for symmetric densities. In *International Conference on Machine Learning*, pp. 5361–5370. PMLR, 2020.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lechner, M. and Hasani, R. Learning long-term dependencies in irregularly-sampled time series, 2020a.

Lechner, M. and Hasani, R. M. Learning long-term dependencies in irregularly-sampled time series. *CoRR*, abs/2006.04418, 2020b. URL https://arxiv.org/abs/2006.04418.

Li, H., Yang, Y., Chen, D., and Lin, Z. Optimization algorithm inspired deep neural network structure design. In *Asian Conference on Machine Learning*, pp. 614–629. PMLR, 2018.

Massaroli, S., Poli, M., Park, J., Yamashita, A., and Asama, H. Dissecting neural odes. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3952–3963. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/293835c2cc75b585649498ee74b395f5-Paper.pdf.

Mhammedi, Z., Hellicar, A., Rahman, A., and Bailey, J. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2401–2409. JMLR. org, 2017.

Moreau, T. and Bruna, J. Understanding the learned iterative soft thresholding algorithm with matrix factorization. *arXiv preprint arXiv:1706.01338*, 2017.

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

Norcliffe, A., Bodnar, C., Day, B., Simidjievski, N., and Lió, P. On second order behaviour in augmented neural odes. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5911–5921. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/418db2ea5d227a9ea8db8e5357ca2084-Paper.pdf.

Pal, A., Ma, Y., Shah, V., and Rackauckas, C. V. Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8325–8335. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/pal21a.html.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

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

Pontryagin, L. S. *Mathematical theory of optimal processes*. Routledge, 2018.

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019a. URL http://arxiv.org/abs/1907.03907.

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019b. URL https://proceedings.neurips.cc/paper/2019/file/42a6845a557bef704ad8ac9cb4461d43-Paper.pdf.

Ruthotto, L. and Haber, E. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pp. 1–13, 2019.

Ruthotto, L., Osher, S. J., Li, W., Nurbekyan, L., and Fung, S. W. 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. ISSN 0027-8424. doi: 10.1073/pnas.1922204117. URL https://www.pnas.org/content/117/17/9183.

Sander, M. E., Ablin, P., Blondel, M., and Peyré, G. Momentum residual neural networks. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9276–9287. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/sander21a.html.

Su, W., Boyd, S., and Candes, E. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pp. 2510–2518, 2014.

Vorontsov, E., Trabelsi, C., Kadoury, S., and Pal, C. On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3570–3578. JMLR. org, 2017.

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

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 4880–4888, 2016.

Xia, H., Suliafu, V., Ji, H., Nguyen, T. M., Bertozzi, A., Osher, S., and Wang, B. Heavy ball neural ordinary differential equations. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=fYLfs9yrtMQ.

Yildiz, C., Heinonen, M., and Lahdesmaki, H. Ode2vae: Deep generative second order odes with bayesian neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/99a401435dcb65c4008d3ad22c8cdad0-Paper.pdf.

Zhong, Y. D., Dey, B., and Chakraborty, A. Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryxmb1rKDS.

Supplementary materials for

Nesterov Neural Ordinary Differential Equations

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

$$\mathbf{z}'(t) = g(\mathbf{z}(t), t, \theta), \quad \mathbf{z}(0) = \mathbf{z}_0, \tag{22}$$

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

$$\mathbf{a}'(t) = -\mathbf{a}(t)\frac{\partial g(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}, \quad \mathbf{a}(T) = -\mathbf{I}. \tag{23}$$

B The adjoint equations for the NesterovNODEs and GNesterovNODEs

The adjoint equation for the NesterovNODEs (Proposition 1) is an implication of (Norcliffe et al., 2020, Proposition 3.1) for Nesterov differential equations. The NesterovNODEs are a special case of the GNesterovNODE when the activation σ is the identity and $\xi = 0$. Therefore, Proposition 2 is a direct consequence of Proposition 3. In this section, we give the proofs for Proposition 3. The proofs will be intrinsically based on Eq. 23.

Proof of Proposition 3 - the adjoint equation for GNesterovNODE

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

$$\begin{cases} \mathbf{h}(t) = \sigma(k(t))\mathbf{x}(t), \\ \mathbf{x}'(t) = \sigma(\mathbf{m}(t)), \\ \mathbf{m}'(t) = -\mathbf{m}(t) - \sigma(f(\mathbf{h}(t), t)) - \xi\mathbf{h}(t), \end{cases} \tag{24}$$

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

$$\mathbf{a}_\mathbf{h}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t)}\frac{\partial \mathbf{x}(t)}{\partial \mathbf{h}(t)} = \frac{1}{\sigma(k(t))}\mathbf{a}_\mathbf{x}(t). \tag{25}$$

To determine the dynamics of $\mathbf{a}_\mathbf{x}(t)$ and $\mathbf{a}_\mathbf{m}(t)$, we rewrite the last two equations in system 24 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. 23, we obtain the first-order differential system for the adjoints $\mathbf{a}_\mathbf{x}$ and $\mathbf{a}_\mathbf{m}$ as

$$\begin{cases} \mathbf{a}_\mathbf{x}'(t) = \mathbf{a}_\mathbf{m}(t)\left[k(t)\frac{\partial \sigma(f(\mathbf{h}(t), t, \theta))}{\partial \mathbf{h}} + \xi k(t)\mathbf{I}\right], \\ \mathbf{a}_\mathbf{m}'(t) = -\mathbf{a}_\mathbf{x}(t)\sigma'(\mathbf{m}(t)) + \mathbf{a}_\mathbf{m}(t). \end{cases} \tag{26}$$

Together with Eq. 25, we obtain the differential-algebraic system for the adjoints of the GNesterovNODE in Proposition 3.

C Proof of Proposition 4 - the nonvanishing gradient for GNesterovNODEs

Recall that the matrix M obtained during the calculation of the gradients has the form

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

and U is the upper triangular matrix in the Schur decomposition of M. Therefore, the complex numbers in the diagonal of U are the eigenvalues of M.

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

$$A = -\int_T^t \sigma'(\mathbf{m}(s))ds$$

and

$$B = -\int_T^t \left[\frac{\partial\sigma(f)}{\partial\mathbf{x}} - \xi t^{-\frac{3}{2}} e^{\frac{t}{2}}\mathbf{I}\right] ds.$$

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.

Without loss of generality, we can assume that the complex numbers in the diagonal of U are arranged in the decreasing values of the real parts. As a consequence of the above claim, The matrix U has the form $U = \begin{bmatrix} U_{\text{large}} & P \\ \mathbf{0} & U_{\text{small}} \end{bmatrix}$ 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) $\frac{1}{2}(t-T)$ and the size of U_{large} is at least N.

D Experimental details

Table 4. The hyperparameters for the models.

Model	NODE	ANODE	SONODE	HBNODE	GHBNODE	NesterovNODE	GNesterovNODE
n (Initialization)	1	2	1	1	1	1	1
n (Point Cloud)	2	3	2	2	2	2	2
h (Point Cloud)	20	20	13	14	14	14	14
n (MNIST)	1	6	5	5	5	5	5
h (MNIST)	92	64	50	50	50	50	50
n (CIFAR10)	3	13	12	12	12	12	12
h (CIFAR10)	125	64	50	51	51	51	51

We list some experimental details that are common to all experiments before presenting more details for each experiment.

Table 5. The hyperparameters for ODE-RNN integration models.

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

Table 6. The ξ hyperparameters for GHBNODE and GNesterovNODE.

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

- fc$_n$ is a fully-connected layer with the output dimension of size n.
- 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 6.
- Except for the experiments in Section 6.1 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 $n^* = 2n$.
- Other hyperparameters are presented in Table 4 and Table 5.

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

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

D.2 Experimental details used in the Point Cloud benchmark in Section 5.1

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

D.3 Experimental details for MNIST in Section 5.2

- Initial Velocity: input$_{1\times 28\times 28} \to$ conv$_{h,1} \to$ LReLU $\to$ conv$_{h,3} \to$ LReLU $\to$ conv$_{2n-1,1}$
- ODE: input$_{n^*\times 28\times 28} \to$ tpad $\to$ conv$_{h,1} \to$ ReLU $\to$ tpad $\to$ conv$_{h,3} \to$ ReLU $\to$ tpad $\to$ conv$_{n,1}$
- Output: input$_{n\times 28\times 28} \to$ fc$_{10}$

D.4 Experimental details for CIFAR10 in Section 5.2

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

D.5 Experimental details for Walker2D in Section 5.3

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