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 $\mathbf{a}_\mathbf{h}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}$, $\mathbf{a}_\mathbf{x}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t)}$ and $\mathbf{a}_\mathbf{m}(t) := \frac{\partial \mathcal{L}}{\partial \mathbf{m}(t)}$. From the first equation in the system, we have

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

To determine the dynamics of $\mathbf{a}_\mathbf{x}(t)$ and $\mathbf{a}_\mathbf{m}(t)$, 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 $\mathbf{a}_\mathbf{x}$ and $\mathbf{a}_\mathbf{m}$ as

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

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:

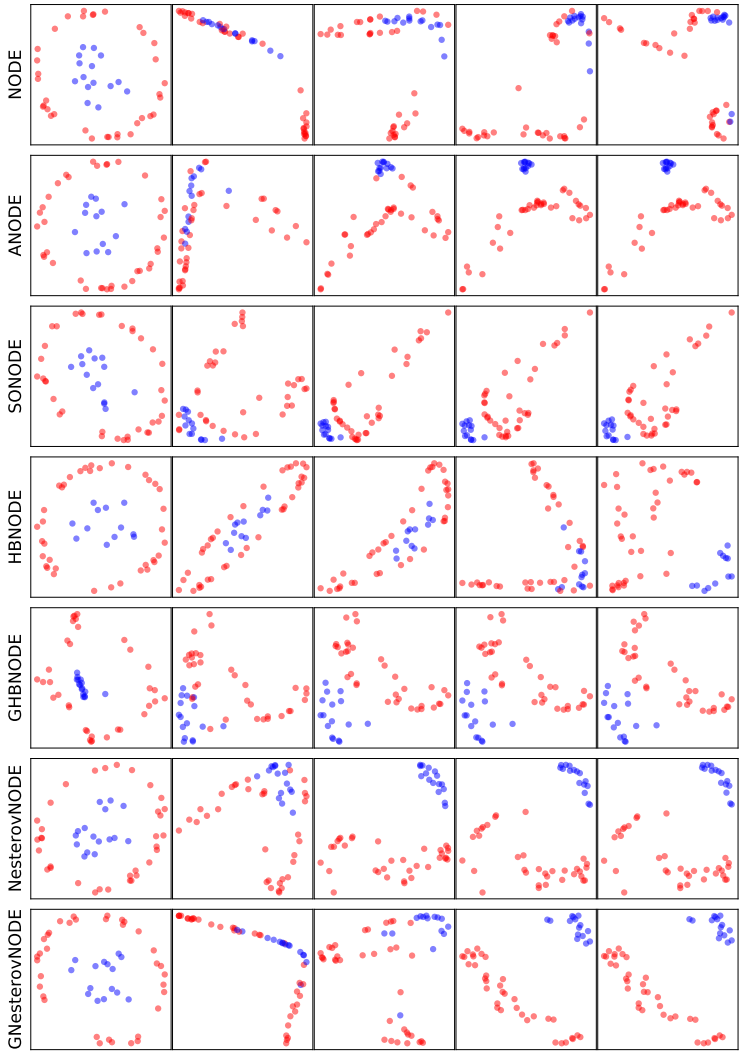

Figure 11: Visualization of the features transform by NODE, ANODE, SONODE, HBNODE, GHBNODE, NesterovNODE and GNesterovNODE after the first 100 epochs of a random run.

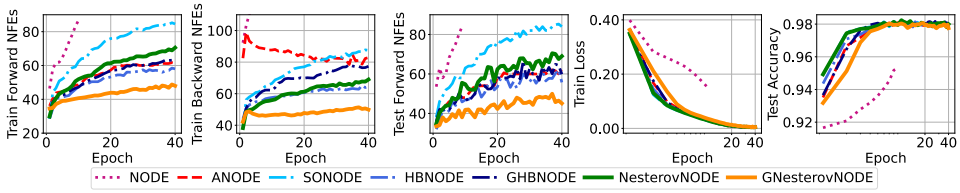

Figure 12: Contrasting the NFEs, training time, training loss, and test accuracy of NODE [4], ANODE [9], HBNODE/GHBNODE [60], and our methods NesterovNODE/GNesterovNODE on the MNIST dataset (Tolerance: 10^{-5}). The run for NODE is stopped due to long running time. The x-axes of the plots for training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

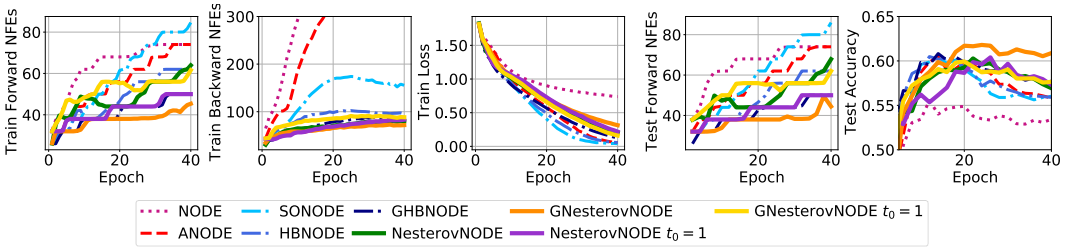

Figure 13: Contrasting the NFEs and accuracy of NODE [4], ANODE [9], HBNODE/GHBNODE [60], our methods NesterovNODE/GNesterovNODE and the methods NesterovNODE/GNesterovNODE with starting integration time $t_0 = 1$ on the CIFAR10 dataset (Tolerance: 10^{-5}).

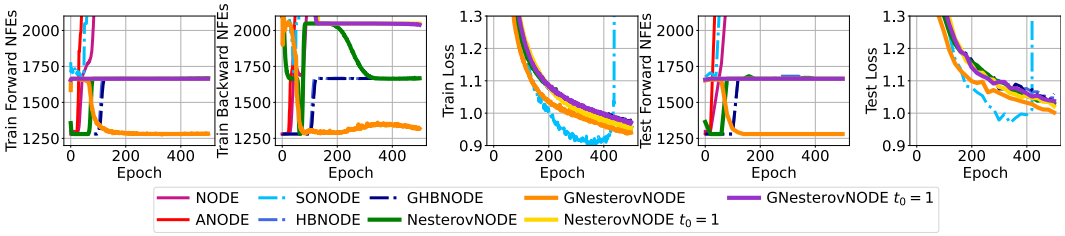

Figure 14: Contrasting the NFEs and losses of NODE [4], ANODE [9], HBNODE/GHBNODE [60], our methods NesterovNODE/GNesterovNODE and the methods NesterovNODE/GNesterovNODE with starting integration time $t_0 = 1$ on the Walker2D dataset (Tolerance: 10^{-7}).

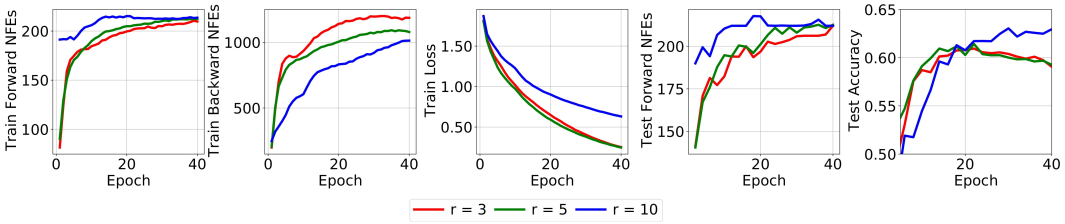

Figure 15: Test accuracy for GNesterovNODE on CIFAR10 with varying Nesterov factors with a lower tolerance value (Tolerance: 10^{-7}).

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

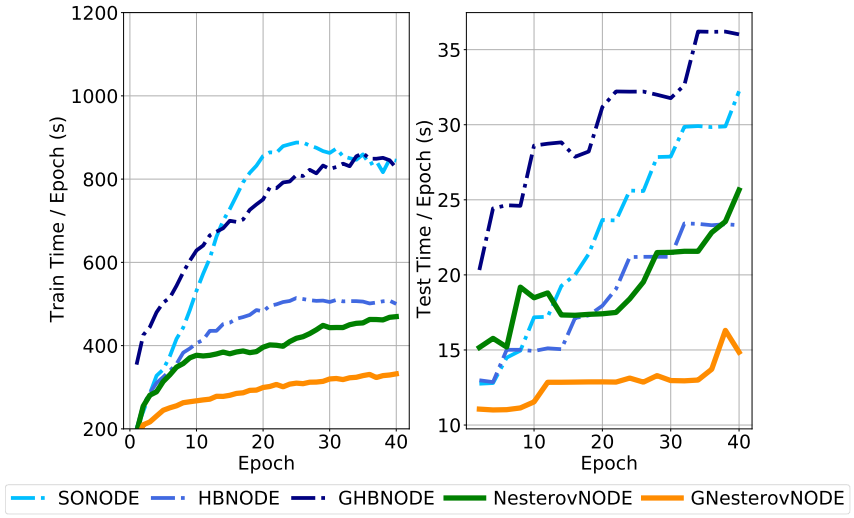

Figure 16: The total wall-clock training and testing time of SONODE, HBNODE, GHBNODE, NesterovNODE, and GNesterovNODE.

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

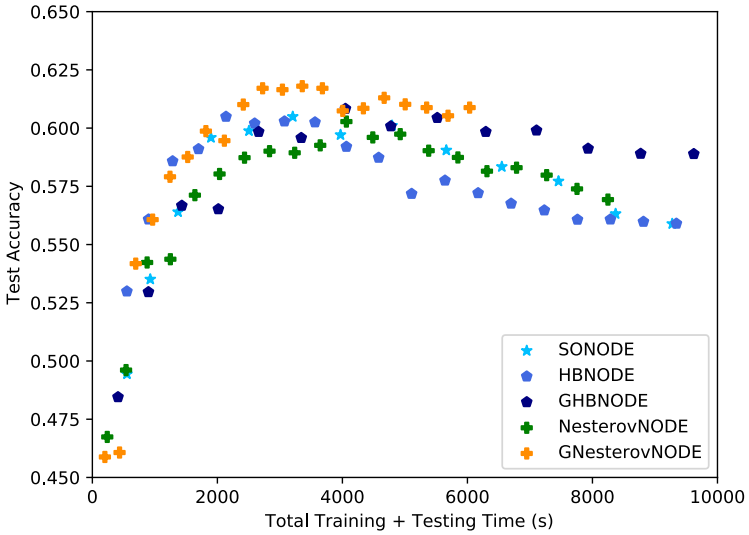

Figure 17: Test accuracy vs the corresponding wall-clock time for SONODE, HBNODE, NesterovNODE, and GNesterovNODE.

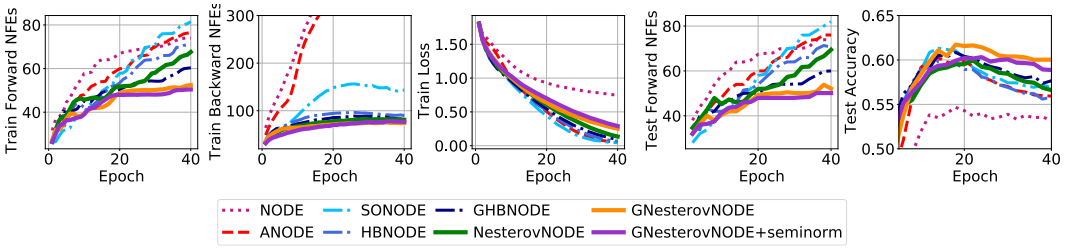

Figure 18: CIFAR10 with the adjoint seminorm method [22].

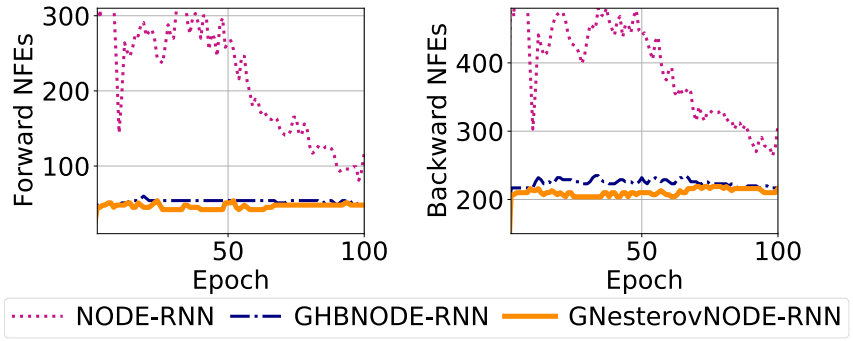

Figure 19: Contrasting the NFEs of NODE-RNN, GHBNODE-RNN and our GNesterovNODE-RNN on the Human Activity benchmark (Per-time-point classification) [29] (Tolerance: 10^{-7}).

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**	$\mathbf{0.840 \pm 0.016}$

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