Improving Neural ODE Training with Temporal Adaptive Batch Normalization

Su Zheng[1]*, **Zhengqi Gao**[2]*, **Fan-Keng Sun**[2], **Duane S. Boning**[2], **Bei Yu**[1], **Martin Wong**[1]
[1]Department of CSE, CUHK [2] Department of EECS, MIT

Abstract

Neural ordinary differential equations (Neural ODEs) is a family of continuous-depth neural networks where the evolution of hidden states is governed by learnable temporal derivatives. We identify a significant limitation in applying traditional Batch Normalization (BN) to Neural ODEs, due to a fundamental mismatch — BN was initially designed for discrete neural networks with no temporal dimension, whereas Neural ODEs operate continuously over time. To bridge this gap, we introduce temporal adaptive Batch Normalization (TA-BN), a novel technique that acts as the continuous-time analog to traditional BN. Our empirical findings reveal that TA-BN enables the stacking of more layers within Neural ODEs, enhancing their performance. Moreover, when confined to a model architecture consisting of a single Neural ODE followed by a linear layer, TA-BN achieves 91.1% test accuracy on CIFAR-10 with 2.2 million parameters, making it the first `unmixed` Neural ODE architecture to approach MobileNetV2-level parameter efficiency. Extensive numerical experiments on image classification and physical system modeling substantiate the superiority of TA-BN compared to baseline methods.

1 Introduction

Originally derived as the continuous limit of a residual connection [4], neural ordinary differential equations (Neural ODEs) [4, 7, 37, 45, 8, 42, 30, 18, 32, 19, 31] is a family of continuous-depth neural networks where the evolution of hidden states is governed by learnable temporal derivatives. These models exhibit several intriguing features, such as the capability of temporal reversibility, which enables generative modeling [4, 15].

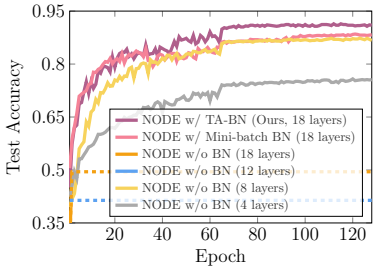


Figure 1: Test accuracies are depicted over the training epochs on CIFAR-10, utilizing Neural ODEs with different numbers of convolution layers as the backbones for learnable derivatives. Dashed horizontal lines denote instances of training failures.

Previous studies on Neural ODEs parameterize the learnable temporal derivatives using a shallow neural network with a limited number of parameters [7, 30]. Without special treatment, merely stacking additional layers in the temporal derivatives does not necessarily enhance Neural ODE performance. Furthermore, deeper networks might increase the stiffness of the ODE system, leading to challenges with the ODE solver, such as excessively small step sizes or even failures due to infinite state values, as shown in Figure 1.

In efforts to deepen the model and address instabilities, one might instinctively consider Batch Normalization (BN) [5, 29, 16, 39, 2] as a remedy to stabilize the intermediate state values of Neural ODEs [13, 35], because BN was originally proposed to accelerate

*The first two authors contribute equally. Correspond to Bei Yu at byu@cse.cuhk.edu.hk.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

the training of (discrete) neural networks and consolidate them against internal covariate shift [29]. However, preliminary works surprisingly found that the introduction of BN into Neural ODEs can even degrade performance, an effect for which the underlying causes remain unclear [13, 35].

In this paper, we demystify the aforementioned phenomenon and demonstrate that directly applying traditional BN to Neural ODEs is fundamentally flawed. The primary complication arises because the forward pass of Neural ODEs employs an adaptive step size solver that discretizes time variably. Consequently, it is not assured that two forward passes will coincide with the same discretized time grids. This variability precludes the possibility of retrieving *population statistics* [16] at test time. Additionally, relying on *mini-batch statistics* for BN renders the outputs of Neural ODEs non-deterministic and vulnerable to outliers and small batch size.

Motivated by these observations, we introduce temporal adaptive BN (TA-BN), a novel technique designed as the continuous-time counterpart to traditional BN, tailored specifically for Neural ODEs. Our empirical findings demonstrate that TA-BN facilitates the up-scaling of Neural ODE model sizes without encountering performance saturation (See Figure 1). Moreover, when confined to a model architecture consisting of a single Neural ODE followed by a linear layer, TA-BN achieves 91.1% test accuracy on CIFAR-10 with 2.2 million parameters, making it the first `unmixed` Neural ODE architecture to approach MobileNetV2-level parameter efficiency. Extensive numerical experiments on image classification and physical system modeling substantiate the superiority of TA-BN compared to baseline methods.

2 Preliminary

Neural ODEs Neural ODEs [4, 7, 37, 45, 8, 42, 30, 18, 32, 19, 31] forge a significant linkage between differential equations and neural networks. They model the continuous dynamics of hidden states with a learnable ODE system:

$$\frac{d\mathbf{h}(t)}{dt} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{h}(t), t) \tag{1}$$

where $\mathbf{f}_{\boldsymbol{\theta}}(\cdot,\cdot) \in \mathbb{R}^D \times \mathbb{R} \to \mathbb{R}^D$ is a neural network parameterized by learnable parameter $\boldsymbol{\theta}$ controlling the temporal evolution, and $\mathbf{h}(t) \in \mathbb{R}^D$ is the ODE state variable at time t. The practical usage of Neural ODEs is by solving an initial value problem: given the initial state $\mathbf{h}(0)$ as input, the state $\mathbf{h}(T)$ at a later time T is computed using an ODE solver and returned as the output. The gradients required for training are computed via the adjoint method [4, 31, 18] reverse in time.

Subsequent works have expanded the capabilities and scope of Neural ODEs by introducing modifications, such as improving expressivity through state augmentation [7], coupling the evolution of both weights and activations [43], accelerating convergence via semi-norm techniques [18], extending to graph neural networks [37] and second-order ODEs [34], adapting to irregular time-series data [19, 32], and drawing connections to diffusion models [15, 28]. Of particular interest, two preliminary studies [13, 35] have indicated that applying BN within Neural ODEs may compromise model performance. Our work aims to unravel this phenomenon and propose a version of BN tailored to operate effectively with Neural ODEs.

Batch Normalization BN [5, 29, 16, 39, 2, 21] performs a re-centering and a re-scaling operation on the given input by subtracting the mean and dividing by the standard deviation:

$$\mathrm{BN}(x_i) = \mathrm{BN}_{\gamma,\alpha}(x_i) = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \alpha \tag{2}$$

where $\epsilon \in \mathbb{R}^+$ ensuring division validity, $\{\gamma \in \mathbb{R}, \alpha \in \mathbb{R}\}$ are learnable parameters, and $x_i \in \mathbb{R}$ represents the i-th data in one batch of size B. During training, the mini-batch statistics [2] $\mu = 1/B \sum_{i=1}^{B} x_i$ and $\sigma^2 = 1/B \sum_{i=1}^{B} (x_i - \mu)^2$ are used for efficiency. In contrast, at test time, BN uses population statistics aggregated across the entire training dataset to ensure deterministic outputs [16]. BN can be extended to multiple dimensions/channels by processing each one separately.

BN has been widely applied in training deep neural networks, as it can stabilize the training process and accelerate convergence. Despite its strong empirical performance, the underlying mechanisms

[2] Throughout our paper, statistics refer to mean and standard deviation (std) or equivalently variance.

of BN have been subject to various interpretations, such as reducing internal covariate shift [16], smoothing the optimization landscape [39], and decoupling the learning of length and direction [21]. Originally, BN [16] was developed for neural networks with hidden units that do not explicitly depend on time. However, recent advancements have extended BN to accommodate time-dependent models, including recurrent neural networks (RNNs)[5, 23] and spiking neural networks (SNNs) [20, 44, 6, 17]. Unlike these models, Neural ODEs exploit adaptive time discretization, necessitating a specialized modification of BN for its application.

3 Traditional Batch Normalization in Neural ODEs

Let us consider a simplified one-dimensional Neural ODE when incorporating BN as example:

$$\frac{dh}{dt} = \mathrm{BN}_{\gamma,\alpha}(wh + b) \tag{3}$$

where $\boldsymbol{\theta} = \{\gamma \in \mathbb{R}, \alpha \in \mathbb{R}, w \in \mathbb{R}, b \in \mathbb{R}\}$ are the learnable parameters. Given a batch of input $\{h_i(0)\}_{i=1}^B$ (i.e., initial values of the ODE), the forward process employs an ODE solver to discretize Eq. (3) over time, visiting N sequential points $\mathcal{T} = (t_0 = 0, t_1, t_2, \cdots, t_N = T)$ and computing the outputs $\{h_i(T)\}_{i=1}^B$. An adaptive step size solver (e.g., Runge-Kutta method) is usually preferred because of its higher efficiency compared to a fixed step size solver (e.g., Forward Euler method). Thus, the differences between two consecutive time steps might not be identical (e.g., $t_2 - t_1 \neq t_1 - t_0$), and the value of N also depends on the batch of data. The temporal discretization indicates that we need to invoke BN at every $t \in \mathcal{T}$ for every sample:

$$\mathrm{BN}(x_{i,j}) = \mathrm{BN}_{\gamma,\alpha}(x_{i,j}) = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}\gamma + \alpha, \quad \text{where } x_{i,j} = w \cdot h_i(t_j) + b \tag{4}$$

for every $j = 1, 2, \cdots, N$ and $i = 1, 2, \cdots, B$, where μ_j and σ_j^2 are statistics associated with t_j.

During training To perform the normalization shown in Eq. (4), BN applies mini-batch mean $\mu_j = 1/B \sum_{i=1}^B x_{i,j}$ and variance $\sigma_j^2 = 1/B \sum_{i=1}^B (x_{i,j} - \mu_j)^2$. Although the training can proceed under normal conditions, the statistics cannot be successfully accumulated across batches. For instance, the time grid $\mathcal{T}$ utilized for the first batch might differ significantly from the grid $\mathcal{T}'$ used for a subsequent batch. Consequently, specific time points $t'_j \in \mathcal{T}'$ might not coincide with any time points in $\mathcal{T}$, and vice versa. This might result in an impractical collection of infinite statistics μ_j and σ_j^2. Moreover, except those associated with $t = 0$ and $t = T$, most stored statistics will be calculated and updated based on a limited number of batches.

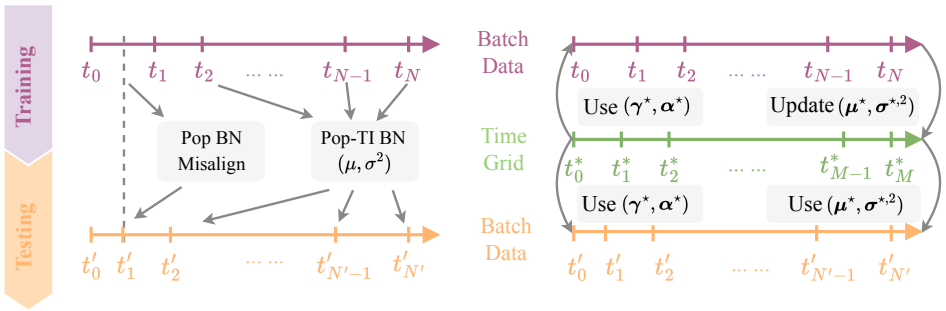


Figure 2: Left: The failure of Pop BN in Neural ODEs stems from the misalignment of discretized time grids. Pop-TI BN aggregates all running mini-batch statistics into a single pair of (μ, σ^2), implicitly assuming time-independent population statistics. Right: Our proposed TA-BN automatically conducts temporal interpolation to accumulate statistics and update parameters during training and testing.

During inference Conventionally, BN should perform the normalization shown in Eq. (4) using population statistics during inference. However, as shown in the left part of Figure 2, the population statistics associated with the time point $t'_j \in \mathcal{T}'$, required by the temporal discretization during inference, might not be available if the time value t'_j is never encountered during training. Moreover, even if the population statistics exist, they are likely to be inaccurate as discussed above.

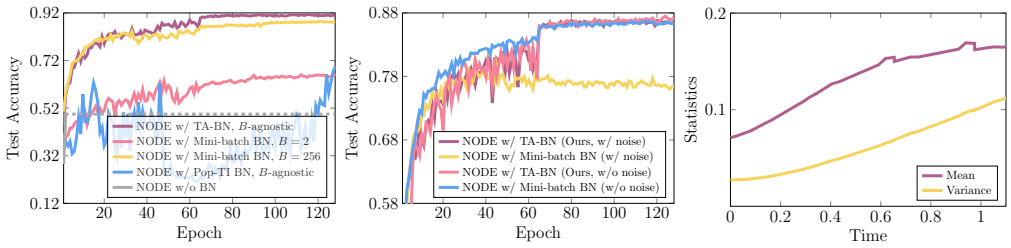


Figure 3: Left: We train a Neural ODE with a U-Net backbone as the learnable derivatives on CIFAR-10. Mini-batch BN shows degraded accuracies with a batch size of 2, while TA-BN can maintain high accuracies under varying batch sizes, because it uses the estimated population statistics during testing. Pop-TI BN aggregates running statistics encountered at any time points into a single pair of μ and σ^2. This approach assumes time-independent statistics, leading to erroneous predictions and erratic test loss curves. Middle: When noisy data exist on average in one out of every test batch, Mini-batch BN's performance deteriorates, because the noise affects the mini-batch statistics. The backbone for learnable derivatives in this experiment consists of 6 convolution layers. Right: We plot the output statistics from the first layer of U-Net over time; they are time-dependent.

Summary A seemingly straightforward remedy could be to consistently use mini-batch statistics during both training and inference, thus avoiding the need for accumulating population statistics. However, this approach has a critical limitation: the output $h_i(T)$ becomes highly dependent on their respective batch's specific characteristics. This will make the outputs inaccurate when batch size is very small (e.g., $B = 2$) or the occurrence of outliers in batch, and even fail in single-run scenarios ($B = 1$) because running standard deviation cannot be calculated. As shown in the left of Figure 3, when the batch size is reduced to 2, mini-batch BN displays degraded accuracies. Furthermore, when noisy data exist, mini-batch BN performance deteriorates, as shown in the middle of Figure 3.

For clarity, we will henceforth refer to the use of population statistics at test time as Pop BN, and the use of mini-batch statistics as Mini-batch BN. Notably, both methods utilize mini-batch statistics during training. It is important to emphasize that in practice, when implementing BN in Neural ODEs with popular deep learning frameworks (e.g., PyTorch with TorchDiffeq [4]), actually a variant of Pop BN is used, which we will refer to as Pop-TI BN. Pop-TI BN accumulates statistics at all encountered time points during training into a single pair of μ and σ^2, which implicitly assumes that all time points share the same statistics. This aggregation approach can result in outputs during test time without errors concerning missing population statistics which we would anticipate as in Pop BN. Unfortunately, these predictions are meaningless and often lead to poor performance, as illustrated in the left of Figure 3. This issue highlights the challenges identified in recent studies [13, 35]. [3] In a nutshell, our observations can be succinctly summarized as follows:

- Pop BN fails when Neural ODEs employ an adaptive step size solver.
- Pop-TI BN implicitly assumes time-independent statistics, leading to erroneous outputs.
- Mini-batch BN fails when Neural ODEs operate with small batch sizes or outliers exist.

4 Temporal Adaptive Batch Normalization

The fundamental reason why Pop-TI BN and Mini-batch BN fail in Neural ODEs is attributed to the added temporal dimension, where the distributions at different time points vary, effectively constituting a stochastic process as depicted in the right of Figure 3. This observation suggests that BN should be defined to be time-dependent in Neural ODEs. Ideally, we would want BN to store population statistics at every $t \in [0, T]$, with corresponding learnable parameters $\gamma(t)$ and $\beta(t)$ defined at every t. However, implementing such a model is impractical, because not every t will be encountered in training, and the update of $\gamma(t)$, $\beta(t)$, and the population statistics will be naturally sparse on t, resulting in inefficiency.

[3] We examined several open-source codes provided by previous works. They align with this implementation and exhibit similar erratic behavior to the Pop-TI BN's test loss curve in the left of Figure 3.

To address the aforementioned problem, we propose an intuitive method termed temporal adaptive BN (TA-BN) for usage in Neural ODEs. Let us illustrate it with the example shown in Eq. (3). To begin with, we evenly divide the time span $[0, T]$ defining $(M+1)$ time grids $\mathcal{T}^\star = (t_0^\star = 0, t_1^\star = \frac{T}{M}, t_2^\star = \frac{2T}{M}, \cdots, t_M^\star = T)$. We associate the time grid $t_m^\star$ with population mean $\mu_m^\star$ and population variance $\sigma_m^{\star,2}$, as well as learnable parameters $\gamma_m^\star$ and $\alpha_m^\star$ for every $m = 0, 1, 2, \cdots, M$. For later simplicity, we denote $\boldsymbol{\gamma}^\star = [\gamma_0^\star, \gamma_1^\star, \cdots, \gamma_M^\star]^T$ and $\boldsymbol{\alpha}^\star = [\alpha_0^\star, \alpha_1^\star, \cdots, \alpha_M^\star]^T$. Similar notations apply to $\boldsymbol{\mu}^\star$ and $\boldsymbol{\sigma}^\star$.

During training As shown in the right of Figure 2, given a batch of data $\{h_i(0)\}_{i=1}^B$, the forward pass of Neural ODE might discretize the time as $\mathcal{T} = (t_0, t_1, \cdots, t_N)$ which differs from the grids $\mathcal{T}^\star$ of TA-BN. However, TA-BN can calculate the temporal derivative for the i-th data at t_j:

$$\text{TABN}_{\boldsymbol{\gamma}^\star, \boldsymbol{\alpha}^\star}(x_{i,j}) = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \gamma_j + \alpha_j \quad \text{where } x_{i,j} = w \cdot h_i(t_j) + b \tag{5}$$

where (μ_j, σ_j^2) are still the mini-batch mean and variance based on $\{x_{i,j}\}_{i=1}^B$. Here γ_j and α_j are interpolated based on $\boldsymbol{\gamma}^\star$ and $\boldsymbol{\alpha}^\star$, respectively:

$$\gamma_j = G(t_j, \boldsymbol{\gamma}^\star, \mathcal{T}^\star), \quad \alpha_j = G(t_j, \boldsymbol{\alpha}^\star, \mathcal{T}^\star) \tag{6}$$

where $G(t, \mathbf{a}, \mathcal{T})$ is a function to interpolate the value $a(t)$ given an array of values $\mathbf{a} \in \mathbb{R}^{M+1}$ and their corresponding $(M+1)$ time points $\mathcal{T}$. There are many choices for $G(\cdot, \cdot, \cdot)$, such as linear, cubic spline, and kernel smoothing. We empirically observe that linear interpolation based on two nearest neighbors suffices for our experiments, and we advocate it due to its implementation simplicity:

$$G(t, \mathbf{a}, \mathcal{T}) = \frac{t_{l+1} - t}{t_{l+1} - t_l} a_l + \frac{t - t_l}{t_{l+1} - t_l} a_{l+1} \tag{7}$$

where l is the index which makes t_l to be the largest value smaller than t in $\mathcal{T}$. Using a linear $G(\cdot, \cdot, \cdot)$ implies that we approximate the underlying $a(t)$ in a piece-wise linear manner, and as long as the number of time grids is sufficiently large, $G(t, \mathbf{a}, \mathcal{T})$ can approximate any continuous $a(t)$ arbitrarily well, i.e., $|G(t, \mathbf{a}, \mathcal{T}) - a(t)| \to 0$ when $M \to \infty$. In practice, we set M close to the number of ODE-discretized time grids. Please refer to Section 5 for ablations on the choice of $G(\cdot, \cdot, \cdot)$.

During training, we also need to accumulate the encountered running mini-batch statistics (μ_j, σ_j^2) to the population statistics $(\mu_m^\star, \sigma_m^{\star,2})$, so that later they can be used for inference. As shown in the right part of Figure 2, this requires the interpolation to be performed in the opposite direction:

$$\begin{aligned} \mu_m^\star &\leftarrow (1-\eta) \cdot \mu_m^\star + \eta \cdot G(t_m^\star, \boldsymbol{\mu}, \mathcal{T}), & \boldsymbol{\mu} &= [\mu_1, \mu_2, \cdots, \mu_N]^T \\ \sigma_m^{\star,2} &\leftarrow (1-\eta) \cdot \sigma_m^{\star,2} + \eta \cdot G(t_m^\star, \boldsymbol{\sigma^2}, \mathcal{T}), & \boldsymbol{\sigma^2} &= [\sigma_1^2, \sigma_2^2, \cdots, \sigma_N^2]^T \end{aligned} \tag{8}$$

where $\eta \in [0, 1]$ is a momentum constant to perform moving averaging to update the population statistics based on current estimate of $\mu_m^\star$ and $\sigma_m^{\star,2}$ using the new observed values $G(t_m^\star, \boldsymbol{\mu}, \mathcal{T})$ and $G(t_m^\star, \boldsymbol{\sigma^2}, \mathcal{T})$, respectively. Note that the training does not involve the gradients with respect to t.

During inference After the training phase, TA-BN will have well-trained parameters $\boldsymbol{\gamma}^\star$ and $\boldsymbol{\alpha}^\star$, and accurately record population statistics $\boldsymbol{\mu}^\star$ and $\boldsymbol{\sigma}^\star$. During inference, when provided with a batch of data and denoting the discretized time grids as $\mathcal{T}'$, as depicted in the right part of Figure 2, we again proceed with the interpolation step:

$$\begin{aligned} &\text{Param.} & \gamma_j' &= G(t_j', \boldsymbol{\gamma}^\star, \mathcal{T}^\star), & \alpha_j' &= G(t_j', \boldsymbol{\alpha}^\star, \mathcal{T}^\star) \\ &\text{Pop. Stat.} & \mu_j' &= G(t_j', \boldsymbol{\mu}^\star, \mathcal{T}^\star), & \sigma_j'^{,2} &= G(t_j', \boldsymbol{\sigma}^{\star,2}, \mathcal{T}^\star) \end{aligned} \tag{9}$$

This interpolation differs from the training phase in that now the interpolation directions are the same for parameters and the population statistics, because now we utilize them for the new time grids $\mathcal{T}'$.

Implementations and summary In practice, we observe that the interpolations during training outlined in Eq. (6)-(8) have opposite directions, potentially leading to increased computational overhead. To elaborate, when encountering a time point t_j discretized by the ODE solver during training, the interpolation for obtaining γ_j and α_j in Eq. (6) using Eq. (7) can be immediately executed by identifying the index l such that $t_l^\star$ is the largest value smaller than t_j. However, the interpolation

in Eq. (8) cannot be carried out at this time. It becomes feasible only after the ODE solver completes its execution and we gather all t_j's into $\mathcal{T}$. This sequential nature degrades computational efficiency. To mitigate this issue, we have slightly adjusted the interpolation for population statistics during training so that it can be performed concurrently with the parameter interpolation at time t_j. The key steps of our proposed TA-BN are summarized in Algorithm 1. It will be called as a subroutine for every discretized time point $t_j \in \mathcal{T}$ required by the ODE solver.

Algorithm 1 outlines the case involving a single one-dimensional neuron and one TA-BN layer within the Neural ODE. In more general cases, we may have Q TA-BN layers within a Neural ODE, and each neuron may possess D dimensions. In such cases, we can perform the normalization in a dimensional-wise manner. Thus, we must maintain $2(M+1)Q$ population statistics and $2(M+1)Q$ learnable parameters, each with D dimensions. In contrast, a traditional feedforward neural network with Q BNs has $2Q$ population statistics and $2Q$ learnable parameters, each with D dimensions. Note that for image inputs, we perform channel-wise normalization, so D will be the number of channels.

Finally, since our linear interpolation in Algorithm 1 relies on two nearest neighbors, it is prudent to ensure that the number of TA-BN time grids $(M+1)$ are roughly at the same level as the number of ODE-discretized time grids. Denser grids would be unnecessary as those not adjacent to any t_j would remain unused. Alternatively, as our numerical results will demonstrate, employing other interpolation methods that require all time grids may lead to increased computational overhead without significant improvements. Additionally, we employ L2 regularization $\|\boldsymbol{\gamma}^\star - 1\|^2$ and $\|\boldsymbol{\alpha}^\star\|^2$ to avoid significant discrepancies among the elements in $\boldsymbol{\gamma}^\star$ and $\boldsymbol{\alpha}$, making the training more stable.

Algorithm 1 The forward pass of a TA-BN layer at time t_j

Input: Batched input $\mathbf{x} = \{x_{i,j}\}_{i=1}^{B}$ at time t_j

Get $t_l^\star$ and $t_{l+1}^\star$ such that $t_l^\star$ is the largest value smaller than t_j in $\mathcal{T}^*$;

$\omega_1 = \frac{t_{l+1}^\star - t_j}{t_{l+1}^\star - t_l^\star}, \omega_2 = \frac{t_j^\star - t_l}{t_{l+1}^\star - t_l^\star}$ ▷ Compute the weights for linear interpolation;

if the model is in the `training` mode **then**

  $\mu_j = \texttt{mean}(\mathbf{x}), \quad \sigma_j^2 = \texttt{var}(\mathbf{x})$; ▷ Compute the mini-batch statistics;

  $\gamma_j = \omega_1\gamma_l^\star + \omega_2\gamma_{l+1}^\star, \quad \alpha_j = \omega_1\alpha_l^\star + \omega_2\alpha_{l+1}^\star$;

  $\mu_l^\star \leftarrow (1-\eta\cdot\omega_1)\mu_l^\star + \eta\cdot\omega_1\mu_j, \quad \mu_{l+1}^\star \leftarrow (1-\eta\cdot\omega_2)\mu_{l+1}^\star + \eta\cdot\omega_2\mu_j$;

  $\sigma_l^{\star,2} \leftarrow (1-\eta\cdot\omega_1)\sigma_l^{\star,2} + \eta\cdot\omega_1\sigma_j^2, \quad \sigma_{l+1}^{\star,2} \leftarrow (1-\eta\cdot\omega_2)\sigma_{l+1}^{\star,2} + \eta\cdot\omega_2\sigma_j^2$;

else if the model is in the `evaluation` mode **then**

  $\mu_j = \omega_1\mu_l^\star + \omega_2\mu_{l+1}^\star, \quad \sigma_j^2 = \omega_1\sigma_l^{\star,2} + \omega_2\sigma_{l+1}^{\star,2}$;

  $\gamma_j = \omega_1\gamma_l^\star + \omega_2\gamma_{l+1}^\star, \quad \alpha_j = \omega_1\alpha_l^\star + \omega_2\alpha_{l+1}^\star$;

end if

return $\frac{\mathbf{x}-\mu_j}{\sqrt{\sigma_j^2+\epsilon}}\gamma_j + \alpha_j$;

5 Numerical Results

Various network architecture designs have been explored in existing Neural ODE studies. For instance, the original study [4] integrates a feature extractor (e.g., CNN-based), a Neural ODE module, and an MLP in sequence for classification. Another study structures neural networks with alternating Neural ODE modules and convolutional/linear layers [11]. While these solutions represent viable approaches for optimizing performance on given ML tasks through the synergy of conventional (discrete) neural networks and Neural ODEs, they become inappropriate for investigating the effect of specific architectural modifications (i.e., TA-BN) on Neural ODEs. This is because the Neural ODE module can be entirely bypassed if its trained weights approach zero in extreme cases, thereby overshadowing the intended modification. Hence, our model comprises **a Neural ODE module followed by a single linear layer** as advocated by [7], which we refer to as the `unmixed` Neural ODE architecture. Our code is developed based on PyTorch [36], TorchDiffeq [3], and a customized TA-BN layer. All experiments are run on a Linux server with RTX 3090 GPUs.

5.1 Image Classification

Being consistent in experimental scales of previous Neural ODE studies, we conduct image classification across datasets including MNIST [26], SVHN [33], CIFAR-10, CIFAR-100 [22], and

Tiny-ImageNet [24]. We employ the dopri5 solver with a tolerance of 10^{-3} for ODE solving and adopt the AdamW optimizer [27] with a learning rate of 10^{-3} to train the neural networks for 128 epochs. The training batch size is 256. We set $M = 100$ for TA-BN.

Table 1 compares the test top-1 accuracy and number of parameters of various Neural ODEs. TA-BN outperforms Mini-batch BN and Neural ODE w/o BN on SVHN, CIFAR-10, CIFAR-100, and Tiny-ImageNet. Moreover, TA-BN achieves superior accuracies over Aug-NODE [7] and STEER [12]. On MNIST and CIFAR-10, TA-BN has fewer parameters than Aug-NODE and STEER, which indicates that TA-BN can also help training in the small Neural ODE regime. We also visualize Neural ODEs' performance changes as the number of parameters varies in Figure 4, by including four additional baselines IL-NODE, 2nd-Ord [30], HBNODE, and GHBNODE [41]. For reference, the popular convolutional-based MobileNetV2 [38], known for its parameter efficiency, achieves approximately 94% accuracy with about 2M parameters. Our TA-BN assisted Neural ODE is the first to approach this level of performance using the `unmixed` architecture, while most previous Neural ODE literature either falls short of this performance or/and does not follow this `unmixed` architecture.

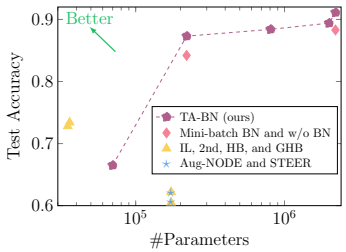


Figure 4: Comparison between different Neural ODEs on CIFAR-10. The baselines marked by yellow triangles do not adhere to the `unmixed` structure and are not strictly comparable to ours. It is unknown whether increasing the number of parameters inside their ODEs can lead to better accuracy.

Table 1: Comparison of test accuracies and number of parameters between different Neural ODEs[†].

Model	MNIST		CIFAR10		SVHN		CIFAR100		Tiny-Imagenet	
	Accuracy	#Params	Accuracy	#Params	Accuracy	#Params	Accuracy	#Params	Accuracy	#Params
Aug-NODE [7]	0.982	84k	0.606	172k	0.835	172k	N/A	N/A	N/A	366k
STEER [12]	0.986	84k	0.621	172k	0.841	172k	N/A	N/A	N/A	N/A
w/o BN	**0.989±0.001**	37k	0.517±0.049	2.2M	0.096±0.025	2.2M	0.246±0.084	2.2M	-	2.2M
w/ Pop-TI BN	0.973±0.011	37k	0.548±0.087	2.2M	0.241±0.123	2.2M	0.251±0.112	2.2M	0.044±0.007	2.2M
w/ Mini-batch BN	0.962±0.013	37k	0.822±0.095	2.2M	0.906±0.031	2.2M	0.492±0.176	2.2M	0.200±0.006	2.2M
w/ TA-BN (ours)	0.988±0.001	37k	0.748±0.059	70k	0.953±0.002	220k	0.576±0.016	220k	0.436±0.013	220k
	0.988±0.001	220k	**0.910±0.010**	2.2M	**0.958±0.004**	2.2M	**0.664±0.025**	2.2M	**0.512±0.008**	2.2M

[†] 'N/A' indicates values are not available in the original literature. '-' indicates training failure at the first epoch. Results are reported with a test batch size of 256. The error bars are shown using the format mean±std.

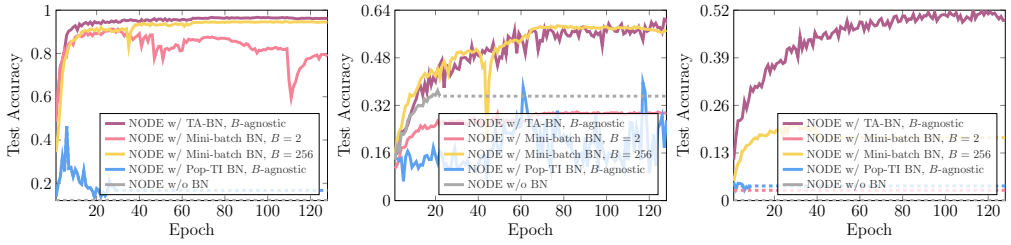


Figure 5: Comparison between Neural ODEs w/ TA-BN, w/ Mini-batch BN, w/ Pop-TI BN, and w/o BN, using an 18-layer U-Net backbone for learnable derivatives. The total number of parameters including the outside linear layer is 2.2M. The datasets are SVHN (left), CIFAR-100 (middle), and Tiny-ImageNet (right). The results on CIFAR-10 have been shown in the left part of Figure 3.

The left part of Figure 3 in Section 3 monitors the test accuracy over training epochs on CIFAR-10. It indicates that training a large Neural ODE without special processes might fail due to numerical instability. Pop-TI BN performs poorly because of the time-independent statistics assumption. While Mini-batch BN achieves satisfactory accuracy, our TA-BN outperforms it without encountering the issues associated with Mini-batch BN, such as batch-dependent outcomes and vulnerability to outliers. Similarly, the left and middle parts of Figure 5 display test accuracy versus training epochs on the SVHN and CIFAR-100 datasets. TA-BN consistently shows better and more stable accuracy than Mini-batch BN on these datasets. On the Tiny-ImageNet dataset (right part of Figure 5), TA-BN demonstrates superior accuracy and faster convergence.

Neural ODE Up-scaling Enabled by TA-BN As shown in the left part of Figure 6, Neural ODEs with a limited number of layers perform normally with acceptable accuracy; however, when the layer count exceeds 10, training fails due to numerical instability. In contrast, the incorporation of TA-BN enables deeper layers within Neural ODE as the learnable derivatives, scaling up the model size and enhancing accuracy, as exemplified by the middle and right sections of Figure 6. Please see Appendix A.1 for architecture details.

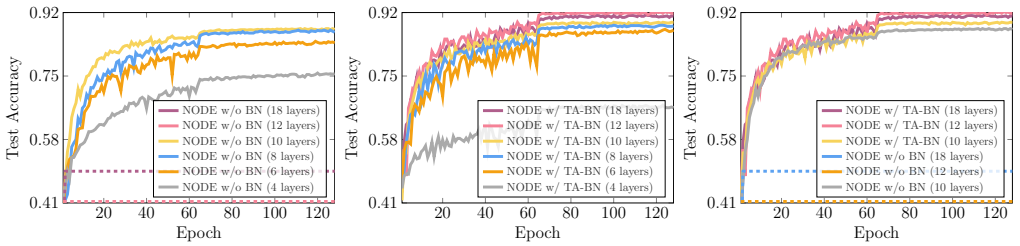


Figure 6: CIFAR-10 accuracies with increasing sizes of the backbones for learnable derivatives. These figures illustrate the scaling up of Neural ODEs without BN (left) and Neural ODEs with TA-BN (middle). We also compare the accuracies of these two settings in one figure (right).

Ablations on TA-BN We integrate TA-BN into Aug-NODE [7], HBNODE [41], and SONODE [34]. Table 2 reports model accuracies w/ and w/o TA-BN. It indicates that our method can also work in compatible with existing Neural ODE variants and boost their performances.

Table 2: Comparison between Neural ODE variations with and without TA-BN on CIFAR-10†.

Accuracy	Aug-NODE	HBNODE	SONODE
w/o TA-BN	0.848	0.851	0.302
w/ TA-BN	**0.862**	**0.867**	**0.853**

† We use 6 convolution layers as the backbone for learnable derivatives.

To show the impact of different interpolation functions G, we test the following interpolation methods: (1) Linear interpolation demonstrated by Eq. (7). (2) Cubic interpolation: We use cubic spline interpolation [1]. (3) Kernel smoothing: Given time t, we calculate $G(t, \mathbf{a}, \mathcal{T})$ by the weighted sum of elements in $\mathbf{a}$. The weight coefficient for t_i is $K(t, t_i) = \exp(-0.5b^{-2}(t - t_i)^2)$ with $b = 0.1T$. (4) Gaussian process: We fit a Gaussian process [10] to perform interpolation in each forward pass. As illustrated by Figure 7, in addition to linear interpolation, the cubic interpolation and kernel smoothing methods also achieve good performance. However, these methods can be much slower than linear interpolation due to more complicated interpolation mechanisms. With a larger network architecture, we observe that these methods suffer from unaffordable running time and instable performance. Thus, we adopt linear interpolation as our default setting. Please see Appendix B for further discussion. Future work can focus on improving the interpolation strategy for TA-BN.

5.2 Physical System Modeling

We first evaluate TA-BN on physical dynamical system modeling tasks using the Walker2d-v2 [25] and HalfCheetah-v2 [14] datasets. These datasets consist of trajectories of 3D robot systems generated by the Mujoco physics engine [40]. Following [42], we perform temporal autoregressive prediction and use the treatments from [42] to avoid collisions. We employ TA-BN in conjunction with a 12-layer MLP serving as the backbone for learnable derivatives. TA-BN with $M = 100$ is applied to the first 5 layers, as is done with Mini-batch BN and Pop-TI BN. We utilize the dopri5 solver with its default settings for solving ODEs and employ the RMSprop optimizer with a learning rate of 10^{-3} for training the neural networks over 100 epochs. The batch size is 64 for Walker2d-v2 and 32 for HalfCheetah-v2. Figure 8 presents the mean absolute error (MAE) along the training epochs on these datasets. Compared with Neural ODEs w/o BN, w/ Mini-batch BN, and w/ Pop-TI BN, using TA-BN can achieve lower errors and faster convergence. Moreover, Table 3 summarizes the MAE results and compares them with the results in [42].

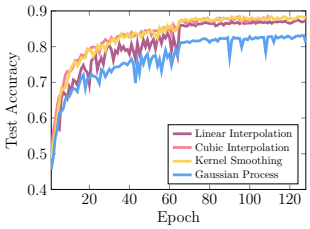


Figure 7: Ablations on interpolation method using CIFAR-10 with 6 convolution layers as the backbone for learnable derivatives. Linear interpolation has a gap of only 0.01 compared to the best result. However, other methods have degraded accuracies and unaffordable runtime when we up-scale the model.

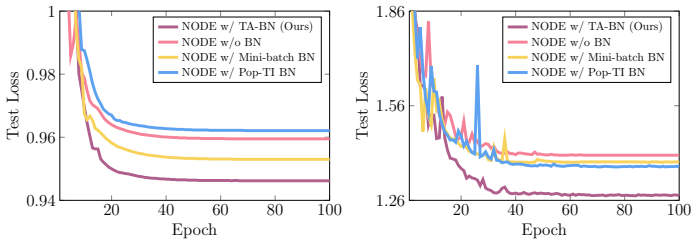


Figure 8: Comparison of Neural ODEs with TA-BN and baselines on Walker2d-v2 (left) and HalfCheetah-v2 (right). The Neural ODE backbone for learnable derivatives is an MLP with 12 layers. Please see Appendix A.2 for architecture details.

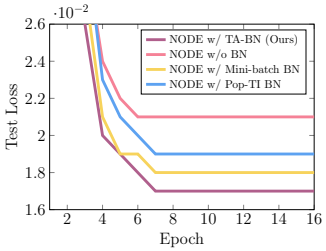


Figure 9: Comparison of different Neural ODEs on Charge Pump circuit modeling. The backbone for learnable derivatives is an 8-layer MLP.

Table 3: MAE comparison between different Neural ODEs on Walker2d-v2 and HalfCheetah-v2.

Dataset	Walker2d-v2	HalfCheetah-v2
Baseline [42]	1.02	1.46
w/o BN	0.959	1.40
w/ Mini-batch BN	0.953	1.38
w/ Pop-TI BN	0.962	1.37
w/ TA-BN (ours)	**0.946**	**1.28**

Additionally, we investigate the effectiveness of TA-BN by modeling the temporal electrical current of a Charge Pump (CP) circuit [9]. The CP is an analog integrated circuit composed of MOS transistors and is commonly utilized in commercial electrical products. To simulate the semiconductor manufacturing impact on the CP, we generate 200 circuit instances based on the process design kit (PDK) file. Concretely, we simulate the CP's behavior from 0 seconds to 200 nanoseconds using a time grid of 2 picoseconds and record the electrical currents at the CP output every 200 picoseconds.

In this experiment, we utilize 16 circuit features (representing the widths of the MOS transistors) and the electrical currents at the present 20 time points (with each point comprising 2 values) as the Neural ODE input, resulting in 56 input features in total. Our objective is to train the Neural ODE to predict the electrical currents at the subsequent 20 time points, yielding 40 outputs in total. To construct the dataset, we randomly sample 20k data from the simulation results. The training set includes 90% of them, and the remaining data are used as the testing set. We use 8 linear layers to parameterize the derivative of the Neural ODE, with each layer containing 56 neurons. Mini-batch BN, Pop-TI BN, and TA-BN are applied to the first 3 layers. We use $M = 100$ for TA-BN. The final linear layer outside the Neural ODE maps the 56 features to 40 prediction values. For ODE solving, we use the dopri5 method with a tolerance level of 10^{-3}. We use the SGD optimizer with a learning rate of 10^{-2} to train the model for 16 epochs. The batch size is 200.

Figure 9 presents the mean square error (MSE) results of Neural ODEs w/o BN, w/ Mini-batch BN, w/ Pop-TI BN, and w/ TA-BN. BN can improve the prediction, and TA-BN surpasses Mini-batch BN. It indicates that TA-BN is suitable for the circuit modeling task.

6 Conclusions and Limitations

In this paper, we demystify the previously unknown reason why batch normalization (BN) may lead to performance degradation when used with Neural ODEs. The fundamental mismatch arises from BN being initially designed for discrete neural networks without a temporal dimension, whereas Neural ODEs operate continuously over time. To address this challenge, we propose temporal adaptive batch normalization (TA-BN), specifically tailored for Neural ODEs. Our key is associating population statistics and learnable parameters with predefined regular time grids, with temporal interpolation automatically performed during training and testing. As a continuous-time counterpart to traditional BN, TA-BN ensures training stability, thereby facilitating the scaling of Neural ODE model sizes. Our extensive experiments demonstrate TA-BN's ability to enhance the performance of Neural ODEs compared to baseline methods in tasks such as image classification and physical system modeling.

The interpolation process used in TA-BN inevitably introduces runtime overhead, which slows down the execution of Neural ODEs. The parameter-free linear interpolation technique is empirically found to be stable, time-efficient, and capable of providing performance improvement. However, future work can focus on enhancing the temporal interpolation used in TA-BN to further optimize its efficiency and performance.

References

[1] Interpolating natural cubic splines using PyTorch. `https://github.com/patrick-kidger/torchcubicspline`.

[2] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. *Advances in Neural Information Processing Systems*, 31, 2018.

[3] Ricky T. Q. Chen. torchdiffeq. `https://github.com/rtqichen/torchdiffeq`, 2018.

[4] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.

[5] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. In *International Conference on Learning Representations*, 2017.

[6] Chaoteng Duan, Jianhao Ding, Shiyan Chen, Zhaofei Yu, and Tiejun Huang. Temporal effective batch normalization in spiking neural networks. *Advances in Neural Information Processing Systems*, 35:34377–34390, 2022.

[7] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *Advances in neural information processing systems*, 32, 2019.

[8] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *International conference on machine learning*, pages 3154–3164. PMLR, 2020.

[9] Zhengqi Gao, Jun Tao, Fan Yang, Yangfeng Su, Dian Zhou, and Xuan Zeng. Efficient performance trade-off modeling for analog circuit based on bayesian neural network. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[10] R Jacob Gardner, Geoff Pleiss, Bindel David, Kilian Q Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix gaussian process inference with GPU acceleration. *Advances in Neural Information Processing Systems*, 2018.

[11] Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

[12] Arnab Ghosh, Harkirat Behl, Emilien Dupont, Philip Torr, and Vinay Namboodiri. STEER: Simple temporal regularization for neural ode. *Advances in Neural Information Processing Systems*, 33:14831–14843, 2020.

[13] Julia Gusak, Larisa Markeeva, Talgat Daulbaev, Alexandr Katrutsa, Andrzej Cichocki, and Ivan Oseledets. Towards understanding normalization in neural odes. *arXiv preprint arXiv:2004.09222*, 2020.

[14] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7657–7666, 2021.

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.

[17] Haiyan Jiang, Vincent Zoonekynd, Giulia De Masi, Bin Gu, and Huan Xiong. Tab: Temporal accumulated batch normalization in spiking neural networks. In *International Conference on Learning Representations*, 2024.

[18] 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.

[19] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

[20] Youngeun Kim and Priyadarshini Panda. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Frontiers in neuroscience*, 15:773954, 2021.

[21] Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 806–815. PMLR, 2019.

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

[23] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2657–2661, 2016.

[24] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. 2015.

[25] Mathias Lechner and Ramin Hasani. Mixed-memory RNNs for learning long-term dependencies in irregularly sampled time series. 2022.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[28] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

[29] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization. *arXiv preprint arXiv:1809.00846*, 2018.

[30] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020.

[31] Takashi Matsubara, Yuto Miyatake, and Takaharu Yaguchi. Symplectic adjoint method for exact gradient of neural ode with minimal memory. *Advances in Neural Information Processing Systems*, 34:20772–20784, 2021.

[32] James Morrill, Cristopher Salvi, Patrick Kidger, and James Foster. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pages 7829–7838. PMLR, 2021.

[33] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 7. Granada, Spain, 2011.

[34] Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Liò. On second order behaviour in augmented neural odes. *Advances in neural information processing systems*, 33:5911–5921, 2020.

[35] Viktor Oganesyan, Alexandra Volokhova, and Dmitry Vetrov. Stochasticity in neural odes: An empirical study. *arXiv preprint arXiv:2002.09779*, 2020.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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

[38] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[39] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in Neural Information Processing Systems*, 31, 2018.

[40] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[41] Hedi Xia, Vai Suliafu, Hangjie Ji, Tan Nguyen, Andrea Bertozzi, Stanley Osher, and Bao Wang. Heavy ball neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 34:18646–18659, 2021.

[42] Wei Xiao, Tsun-Hsuan Wang, Ramin Hasani, Mathias Lechner, Yutong Ban, Chuang Gan, and Daniela Rus. On the forward invariance of neural odes. In *International conference on machine learning*, pages 38100–38124. PMLR, 2023.

[43] Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E Gonzalez, Kurt Keutzer, Michael W Mahoney, and George Biros. Anodev2: A coupled neural ode framework. *Advances in Neural Information Processing Systems*, 32, 2019.

[44] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11062–11070, 2021.

[45] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *International Conference on Machine Learning*, pages 11639–11649. PMLR, 2020.

A Implementation Details

A.1 Image Classification

Our model comprises a Neural ODE module followed by a single linear layer, which we refer to as the `unmixed` Neural ODE architecture. The single linear layer maps the activations provided by Neural ODE to the final class probability (or logit). We have 3 settings for the Neural ODE module.

Multilayer Perceptron (MLP): We use 2 linear layers inside the Neural ODE module. On MNIST, each linear layer has 28×28 hidden features, matching the number of pixels in an MNIST image.

Convolutional Neural Network (CNN): We use $4 \sim 12$ convolution layers inside the Neural ODE module. The kernel size of all layers is 3×3. The numbers of output channels are:

4 layers: $3(\text{input}) \to 32_{\downarrow} \to 64_{\downarrow} \to 32_{\uparrow} \to 3_{\uparrow}$.
6 layers: $3(\text{input}) \to 32_{\downarrow} \to 64_{\downarrow} \to 128_{\downarrow} \to 64_{\uparrow} \to 32_{\uparrow} \to 3_{\uparrow}$.
$7 + L_c$ layers: $3(\text{input}) \to 32_{\downarrow} \to 64_{\downarrow} \to 128_{\downarrow} \ (\to 256) \times L_c \to 128 \to 64_{\uparrow} \to 32_{\uparrow} \to 3_{\uparrow}$.

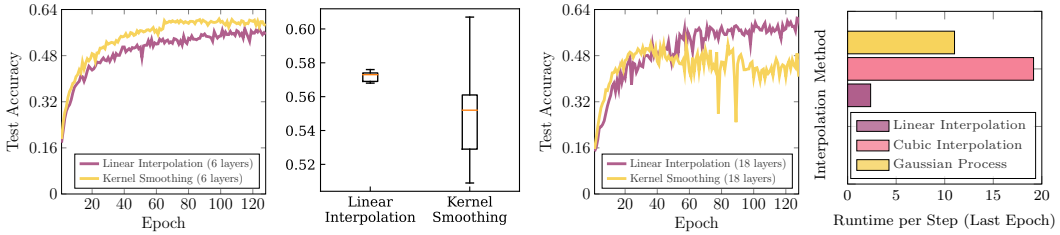
Note that a layer with $\downarrow$ downscales the feature maps using a stride of 2. A layer with $\uparrow$ is a transposed convolution layer that upscales the feature maps using a stride of 2.

U-Net: We use a U-Net with 18 convolution layers. The kernel size of all layers is 3×3. The numbers of output channels are designed as $3(\text{input}) \to 32_{\downarrow} \to 32 \to 64_{\downarrow} \to 64 \to 128_{\downarrow} \to 128 \to 256 \to 256 \to 128_{\uparrow} \to 128 \to 128 \to 64_{\uparrow} \to 64 \to 64 \to 32_{\uparrow} \to 32 \to 32 \to 3$.

A.2 Physical Dynamical Systems Modeling

The HalfCheetah-v2 and Walker2d-v2 datasets consist of trajectories of 3D robot systems generated by the Mujoco physics engine [40]. Each trajectory represents a sequence of a 17-dimensional vector describing the system's state, such as the robot's joint angles and poses. The Neural ODEs are required to predict the trajectories in an autoregressive manner, as in [25, 42].

In these tasks, the input size is 17, and the Neural ODE module is designed as $17(\text{input}) \ (\to 64) \times 11 \to 17$. BN is applied to the first 5 layers. Based on the code from [42], we do the prediction via the invariance set propagation and data-controlled neural ODE proposed in [42]. Since the input size is the same as the output size, we don't need a linear layer outside the Neural ODE module.



(a) (b) (c) (d)

Figure 10: (a) Kernel smoothing can achieve slightly better accuracy on CIFAR-100 with 6 convolution layers for learnable derivatives. (b) Linear interpolation achieves more stable performance across 5 runs. (c) Linear interpolation performs better with a larger model size. (d) We compare different methods on the runtime per step at the last epoch to show the efficiency of linear interpolation.

B Discussion on Interpolation Methods

In Figure 7, we compare the performance of four interpolation methods, including linear interpolation, cubic interpolation, kernel smoothing, and Gaussian process. The cubic interpolation and kernel smoothing methods achieve good performance with 6 convolution layers for learnable derivatives. The results on CIFAR-100 also support this point, as shown in Figure 10(a). However, Figure 10(b) indicates that linear interpolation achieves more stable performance, and Figure 10(c) clearly shows the superiority of linear interpolation with a larger model size. Furthermore, linear interpolation is more efficient than cubic interpolation and Gaussian process, as shown in Figure 10(d). Therefore, we use linear interpolation by default, which can provide superior performance and stability.

C Additional Ablation Studies

C.1 Ablation Study on ODE Solvers

We have conducted extra experiments on CIFAR-10 using fixed-step solvers like Euler method, with the 8-layer backbone. The results are shown in Table 5. Regardless of the solver, TA-BN achieves the best performance among the techniques. We also explored midpoint and rk4 solvers; however, they are much slower and haven't finished in the limited time constraint.

Table 4: Ablation study on ODE solvers.

Method	ODE Solver	Accuracy
w/o BN	Euler	0.839±0.002
w/ Pop-TI BN	Euler	0.631±0.203
w/ Mini-batch BN	Euler	0.864±0.002
w/ TA-BN (ours)	Euler	0.872±0.003
w/o BN	Dopri5	0.843±0.004
w/ Pop-TI BN	Dopri5	0.332±0.090
w/ Mini-batch BN	Dopri5	0.865±0.004
w/ TA-BN (ours)	Dopri5	0.874±0.001

C.2 Ablation Study on Time Grids

Regarding the grid size hyperparameter M, we ran experiments without BN and found that the number of function evaluations (NFE) is around hundreds. Thus, we set $M = 100$ in our paper. We have performed extra ablation studies on it, as reported in the following table. Using $M > 100$ brings no improvement but too much runtime overhead. We don't have the confidence interval of $M = 500$ due to the time limit.

Table 5: Ablation study on ODE solvers.

Method	Time Grids M	Accuracy
w/ TA-BN (ours)	10	0.851±0.015
w/ TA-BN (ours)	50	0.851±0.019
w/ TA-BN (ours)	100	0.874±0.001
w/ TA-BN (ours)	500	0.870

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We clearly describe our main claims such as the findings, contributions, and results in the abstract and introduction. They match our idea, methodology, and experimental results that we present in the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We describe the limitations of our method in Section 5 "Conclusion and Limitations".

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Our method is based on batch normalization. We do not introduce additional assumptions and theories.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We have fully disclosed the details of neural ODE architectures, ODE solvers, and optimizers in the paper, along with the used Python packages and hardware platform. Combined with the detailed experimental settings in our paper, the provided information can be used to reproduce the main experimental results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We put part of the code for reproduciblity in supplementary. It will be released upon acceptance.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We have specified all the training and test details in the experiment section and the appendix. They follow existing papers and open-source code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report error bars in the major result table. We also show box plots for reporting error bars in some experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have described the hardware platform in the experiment section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We strictly conform the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: For the existing assets we use, we have cited the papers and repos. They are all open-source.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.