How Deep Do We Need: Accelerating Training and Inference of Neural ODEs via Control Perspective

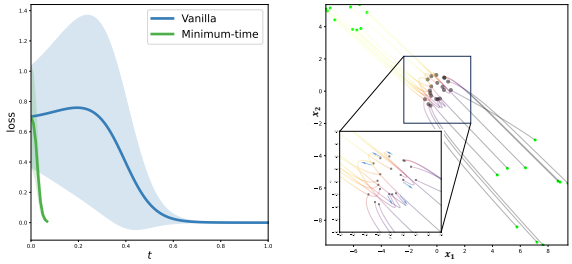
Keyan Miao [1] **Konstantinos Gatsis** [2]

Abstract

Neural Ordinary Differential Equations (ODEs) have shown promise in learning continuous dynamics. However, their slow training and inference speed hinder wider applications. In this paper, we propose to optimize Neural ODEs from a spatial and temporal perspective, drawing inspiration from control theory. We aim to find a reasonable depth of the network, accelerating both training and inference while maintaining network performance. Two approaches are proposed. One reformulates training as a minimum-time optimal control problem directly in a single stage to search for the terminal time and network weights. The second approach uses pre-training coupled with a Lyapunov method in an initial stage, and then at a secondary stage introduces a safe terminal time updating mechanism in the forward direction. Experimental results demonstrate the effectiveness of speeding up Neural ODEs.

1. Introduction

Deep Neural Networks (DNNs) have transformed AI and achieved remarkable success in complex tasks (LeCun et al., 2015; Krizhevsky et al., 2012). Increasing depth has been a common strategy for enhancing capabilities, yet this is not always reliable and may lead to diminishing returns or adverse effects. Challenges such as gradient vanishing and exploding arise in deeper architectures, causing instability (Bengio et al., 1994). These networks also incur a higher computational cost, slowing training and inference. Moreover, excessively complex models can suffer from overfitting and poor robustness (Sun et al., 2016; Huang et al., 2021; Zhu et al., 2022; Hassani & Javanmard, 2024).



(a) Loss via time (b) State trajectories

Figure 1. Performance of Neural ODEs on Concentric Annuli problem: (a) Prediction loss vs. time on 20 correctly classified examples by Vanilla and Minimum-time Neural ODEs; (b) State trajectories of 20 examples by Vanilla Neural ODEs

To address this, Residual Networks (ResNets) (He et al., 2016) emerged as a breakthrough. ResNets introduced skip connections that bypass certain layers, alleviating gradient issues and enhancing training of deeper models. Building upon the success of ResNets, the exploration of Neural Ordinary Differential Equations (Neural ODEs) (Chen et al., 2018) has gained prominence. These models treat networks as dynamic systems governed by differential equations, offering adaptability and flexibility (Weinan, 2017; Ee et al., 2018; Li et al., 2017).

Neural ODEs have recently seen diverse applications in classical machine learning, enhancing tasks like image classification, generative models and time-series problems (Grathwohl et al., 2018; Rubanova et al., 2019; Kidger, 2022). In control systems, Neural ODEs offer a natural approach for modeling and optimizing dynamic behaviors. Their ability to handle continuous-time dynamics has implications in trajectory planning, state observer design, and system identification (Liang et al., 2021; Miao & Gatsis, 2023; Djeumou et al., 2022; Zhao et al., 2024).

1.1. Challenges / Motivations

Despite the rapid progress and promising results, challenges remain in understanding the full potential of Neural ODEs. Researchers are actively investigating ways to improve their

[1]Department of Engineering, University of Oxford, Oxford, United Kingdom [2]School of Electronics and Computer Science, University of Southampton, Southampton, United Kingdom. Correspondence to: Keyan Miao <keyan.miao@eng.ox.ac.uk>, Konstantinos Gatsis <k.gatsis@soton.ac.uk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

stability, expressivity, robustness and convergence during training (Kang et al., 2021; Dupont et al., 2019; Massaroli et al., 2020; Yan et al., 2019; Huang et al., 2022; Liu et al., 2020; 2021). However, the training and inference of Neural ODEs can be prohibitively computationally expensive, which limits their broader applicability. This is attributed to the slow and computationally expensive ODE solver within the Neural ODEs architecture. The uncertainty about the necessary depth of the underlying neural network, linked to the number of solver evaluations in Neural ODEs, results in dynamic changes comparable to numerous layers.

In some cases, the computational costs are necessary, while overly deep networks are often unnecessary and can even lead to overfitting issues. Taking the 2-D Concentric Annuli problem as an example, it can be observed from Figure 1(a) that the system learned by Vanilla Neural ODEs trained within $[0, 1]$, achieves the desired state within 0.6 time units, rendering subsequent computations redundant. Furthermore, Figure 1(a) and Figure 1(b) reveal an initial error increase and trajectory divergence from the optimal direction within the first 0.2 units. An overly deep network contains more complex dynamics, which makes the state trajectory become more convoluted. In Figure 1(b), some points corresponding to the final target in the lower right direction will flow in the exact opposite direction (upper left) at the beginning.This suggests that Vanilla Neural ODEs may follow convoluted paths, incurring needless computational and temporal expenses. This behavior aligns with the training process focusing solely on terminal state requirements.

1.2. Related Works

Numerous research efforts have been devoted to reducing the computational cost of Neural ODEs. Most of them focus on encouraging Neural ODEs to learn relatively simple dynamics and accelerate computation from the spatial perspective of dynamics. For instance, some studies proposed additional regularization terms (Kelly et al., 2020; Finlay et al., 2020; Pal et al., 2021) to facilitate learning of Neural ODEs that are easier to integrate. Research efforts adopt a temporal standpoint: certain studies delve into the optimization of numerical solvers, for instance, Poli et al. (2020) advocates augmenting training with an extra neural network to capture the higher-order components of the numerical solver, leading to reduced computation time; while the STEER technique introduced by Ghosh et al. (2020) imparts regularization to Neural ODEs by employing stochastic sampling of the ODE's terminal time during the training process. Nevertheless, these techniques predominantly expedite the inference phase, with their influence on expediting training being minor or, in some cases, resulting in supplementary training expenditures. Notably, the STEER approach demonstrates limited enhancement in both domains. Some work proposed to optimize the network structure within the framework of second-order optimization by optimizing the integration terminal time (Liu et al., 2021), but exclusively based on the adjoint sensitivity method, which is a speed-up for training under limited conditions, as the adjoint method is inherently slower because it has to solve additional ODEs. Furthermore, there is research focusing on determining explicit depth bounds for deep neural networks by exploring the turnpike property (Faulwasser et al., 2021), however, it relies on *a posteriori* computation based on trained deep neural networks instead of *a priori* estimation.

1.3. Our Contributions

In contrast to conventional techniques like pruning and early exit (Han et al., 2015; Teerapittayanon et al., 2016) in standard deep neural networks, which aim to enhance network efficiency, we address the issue of network depth in deep learning by leveraging Neural ODEs. Our approach is to learn an efficient model by design, as opposed to making a network more efficient after learning which is often followed in the above works. Our approaches involve adjusting the integral time span to modify the network's depth, thereby refining its architecture as Figure 2 shows and facilitating acceleration from temporal and spatial perspectives. From the spatial angle, we strive for Neural ODEs to acquire more direct trajectories, leading to swift and straightforward state elevation towards the target. From the temporal perspective, our objective is to determine an appropriate terminal time of Neural ODEs that minimizes superfluous computational resources and time consumption and also upholds network performance. In summary, our contributions are as follows:

- Extending the framework of Neural ODEs, we propose a novel concept to address the depth complexity of DNN by by formulating it as the problem of adjusting the interval of ODE integration, primarily focusing on optimal terminal time to enhance the efficiency of Neural ODEs;
- Our first approach addresses this challenge by framing depth complexity as a Minimum-time Optimal Control problem. This allows us to directly optimize the terminal time in a single stage for acceleration, while still maintaining network performance;
- Our second method employs the Lyapunov approach from control theory during pre-training. This ensures dynamic convergence with guaranteed speed, followed by an iterative process to refine the terminal time for optimized performance;
- Experiments on supervised learning problems and generative models demonstrate that our approaches achieve an order of magnitude faster training and inference compared to the baseline of Vanilla Neural ODEs, optimizing both dynamics spatial and temporal performance.

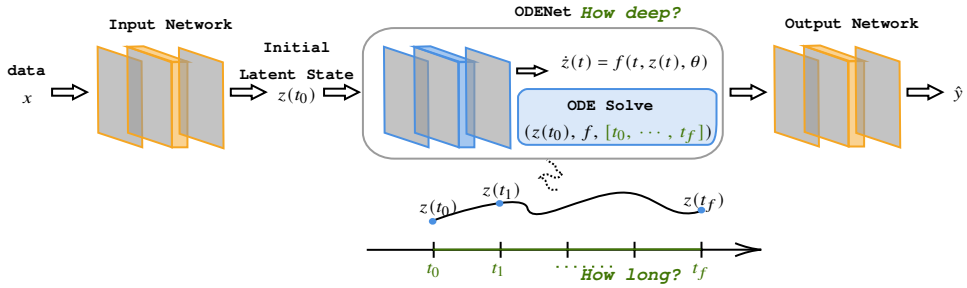


Figure 2. Depth Problem of Neural ODEs

2. Preliminaries

2.1. Optimal Control

Optimal control is a significant branch of modern control theory, encompassing various approaches such as calculus of variations, the Pontryagin maximum principle (PMP) (Pontryagin, 1987), and dynamic programming (DP) (Bellman, 1957). When dealing with unconstrained control vectors, the application of calculus of variations becomes relevant. However, in the presence of constraints, Pontryagin introduced and proved the maximum principle, a fundamental result in optimal control theory and calculus of variations. The maximum principle yields an open-loop control solution. At its core, the PMP introduces the Hamiltonian function, which combines the system dynamics and the performance criterion. The Hamiltonian function captures the total instantaneous cost associated with the system's state and control inputs. By optimizing the Hamiltonian with respect to the control inputs, the PMP helps identify the control strategy that yields the best performance.

Mathematically, the PMP establishes a set of necessary conditions for an optimal control trajectory. For a system governed by the state equations $\dot{z} = f(z, \theta)$ where θ represents control variable, and the Hamiltonian function $H(z, \theta, p)$, where p represents the adjoint state (Lagrange multiplier) associated with the state variable z, the PMP states that an optimal control trajectory (z^*, θ^*) must satisfy the following conditions: state dynamics $\dot{z}^* = \frac{\partial H(z^*, \theta^*, p^*)}{\partial p}$; adjoint dynamics $\dot{p}^* = -\frac{\partial H(z^*, \theta^*, p^*)}{\partial z}$; transversality condition $p^*(t_f) = \frac{\partial \Phi(z(t_f))}{\partial z(t_f)}$ where Φ represents the terminal cost; optimization of the Hamiltonian condition $\min H(z^*, \theta, p^*) = H(z^*, \theta^*, p^*)$. The PMP can be effectively applied to address a variety of optimal control problems, encompassing scenarios such as minimum fuel consumption and minimum time problems. Of particular focus in this study is the latter category, which pertains to the minimization of time required to accomplish certain tasks.

Complementing optimal control, turnpike theory explores the long-term behavior of dynamic systems over extended periods. It investigates the phenomenon where the system, despite having various feasible control inputs, tends to converge and remain close to a specific optimal trajectory known as the turnpike. The turnpike phenomenon offers valuable insights into system behavior, stability, and performance over prolonged time intervals.

2.2. Neural ODEs

DNNs have demonstrated their capability to learn nonlinear mappings and generalize effectively to unseen data under specific conditions across a wide range of problems. However, the depth problem once posed significant challenges for neural networks. The breakthrough came with the introduction of ResNet. By incorporating skip connections, ResNet achieved a remarkable performance boost. Recently, its universal approximation power has been explained from a non-linear control perspective (Tabuada & Gharesifard, 2020). Coinciding with ResNet's emergence, the idea of treating the hidden layers of neural networks as states of a dynamical system gained popularity. Building upon this concept, Chen (Chen et al., 2018) proposed an ODE specified by the neural network to parameterize continuous dynamics:

Definition 2.1 (Neural ODEs)**.** With $h_x : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$, $h_y : \mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$ representing the input network and output network respectively, a Neural ODE is a system of the form

$$\begin{cases} \dot{z}(t) = f(t, z(t), \theta) \\ z(t_0) = h_x(x) \\ \hat{y}(t) = h_y(z(t)) \end{cases} \qquad t \in \mathcal{S} \tag{1}$$

where $\mathcal{S} := [t_0, t_f]$ $(t_0, t_f \in \mathbb{R}^+)$ is the depth domain and f is a neural network called ODENet which is chosen as part of the machine learning model with parameter θ.

The solution of this ODE at some time t_f from an initial value $z(t_0)$ is $z(t_f)$, obtained with a differential equation solver by means of a specific solution scheme according to desired accuracy. The number of times the ODE solver evaluates the function in one forward pass can be interpreted as the number of hidden layers of the neural network, i.e., the depth. Now ResNet can be seen a special case of Neural ODEs using Euler discretization.

For common machine learning scenarios, such as image

classification, an input data point x is mapped by Neural ODEs to $\hat{y}$, e.g., a label, and y represents the true label. In other words, the inference of Neural ODEs is carried out by solving the initial value problem (IVP):

$$\hat{y} = h_y \left(h_x (x) + \int_{t_0}^{t_f} f (t, z (t) , \theta) \, dt \right) \tag{2}$$

The training of the Neural ODEs with parameters θ proposed in (Chen et al., 2018) considers only a loss function $\Phi \left(z \left(t_f \right) \right)$ that depends on the terminal state $z(t_f)$, which is also the common scenario for supervised learning problems where the terminal state is compared to a true label. In this case, the training can be cast into a *Mayer* optimal control problem where only terminal cost is considered. However, for the purpose of our paper, it is valuable to introduce an additional integral term over the latent state trajectories measured by a function $L(\cdot)$ as

$$\ell := \Phi \left(z \left(t_f \right) \right) + \int_{t_0}^{t_f} L (z (t) , \theta, t) \, dt \tag{3}$$

since in the framework of Neural ODEs, the latent states evolve through layers, and then generate outputs. Also, the comprehensive evaluation of the entire trajectory's performance plays a pivotal role in facilitating the acquisition of more efficient dynamics.

With such a loss function, the training can be cast into an optimization problem of the following form where x represents training data:

$$\begin{aligned} & \min_{\theta \in U} \quad \ell \\ s.t. \quad & \dot{z} (t) = f (t, z (t) , \theta) , t \in \mathcal{S} \\ & z (t_0) = h_x (x) \end{aligned} \tag{4}$$

which is a *Bolza* optimal control problem, and θ is the control variable here. The problem can be solved recursively by gradient descent (GD) and then optimal control theory comes in handy to provide formulas for computing these gradients.

Supervised Learning problem In the context of supervised learning problems, particularly exemplified by image recognition tasks in machine learning, it is noted, as previously mentioned, that for common scenarios such as image classification, an input data point x is transformed by Neural ODEs into an output $\hat{y}$, typically representing a classification label. Generally, the terminal loss Φ shown in loss function (3) is usually designed as the cross-entropy between the predicted label $\hat{y}$ and the true label y.

Generative Models Neural ODEs have proven their versatility not only in supervised learning but also in the field of generative models. Chen et al. (2018) pioneered the concept of Continuous Normalizing Flows (CNF), utilizing the formula for the instantaneous change of variables to tackle specific challenges in this domain. Essentially, CNF seeks to create complex distributions from simpler ones by dynamically evolving the data points according to the ODE, thus facilitating the generation of target distribution. The central process of CNF involves generating data points x that conform to a target distribution $p(x)$. This is achieved by transforming data from a basic distribution, typically a Gaussian distribution, through a parametrically defined ODE, expressed as $f = (t, z(t), \theta)$. The IVP is:

$$\begin{aligned} \partial_t \begin{bmatrix} z(t) \\ \log p(z(t)) \end{bmatrix} &= \begin{bmatrix} f (t, z(t), \theta) \\ - \operatorname{Tr} \left(\frac{\partial f}{\partial z(t)} \right) \end{bmatrix} \\ \begin{bmatrix} z(t_0) \\ \log p(z(t_0)) - \log p(x) \end{bmatrix} &= \begin{bmatrix} x \\ 0 \end{bmatrix} \end{aligned} \tag{5}$$

where $t \in \left[t_0, t_f \right]$. Then $z(t_f)$ and $\log p(z(t_f)) - \log p(x)$ can be obtained by solving the IVP. Finally, the target distribution can be computed as $\log p(x) = \log p(z(t_f)) - \int_{t_0}^{t_f} \left(- \operatorname{Tr} \frac{\partial f}{\partial z(t)} \right) dt$ where $z(t_f)$ should follow a foundational distribution. The terminal loss can be designed as negative log-likelihood:

$$\Phi = -\mathbb{E}_{p_x} \left\{ \log p(z(t_f)) - \int_{t_0}^{t_f} \left(- \operatorname{Tr} \frac{\partial f}{\partial z(t)} \right) dt \right\} \tag{6}$$

2.3. Lyapunov Conditions for Convergence Rate

When considering the stability and convergence properties of dynamic systems, the application of Lyapunov methods becomes indispensable. Lyapunov methods are mathematical techniques employed to analyze the stability and evolution characteristics of dynamic systems. These methods involve the construction of an energy function (Lyapunov function) to monitor the system's evolution, thereby guaranteeing that the system tends towards a stable state. In control theory, the concept of a stable dynamical system indicates that all solutions within a certain region around an equilibrium tend to converge towards that point. Lyapunov theory (Ames et al., 2014) extends this idea by considering the convergence of solutions to states that minimize a potential function V.

Theorem 1. For the ODE described in (1), a continuously differentiable function V that is also positive except for the equilibrium and is radially unbounded, then it is an exponentially stabilizing Lyapunov function if there exits a constant $\kappa > 0$ such that:

$$\min_{\theta} \left[\left. \frac{\partial V}{\partial z} \right|_z f (t, z, \theta) + \kappa V (z) \right] \leq 0 \tag{7}$$

holds for all z and $t \in \left[t_0, t_f \right]$. The existence of this Lyapunov function implies that there is a θ irrespective of z that can achieve

$$\left. \frac{\partial V}{\partial z} \right|_z f (t, z, \theta) + \kappa V (z) \leq 0 \tag{8}$$

and the ODE using θ is exponentially stable with respect to V and constant κ:

$$V\left(z\left(t\right)\right) \leq V\left(z\left(t_0\right)\right) e^{-\kappa t} \tag{9}$$

The importance of exponential stability in a system lies in its ability to guarantee that the system can rapidly converge to the desired state which is defined by V within a finite time. Recently, Lyapunov theory has gained considerable attention in the field of machine learning. For instance, Kang (Kang et al., 2021) utilized Lyapunov stability to enhance the robustness of Neural ODEs. Additionally, this approach provides a safeguard for ensuring the convergence rate of the system, offering a novel perspective for enhancing the training of neural networks (Rodriguez et al., 2022; Zhao et al., 2023).

3. Methods

The theoretical insights from the field of control theory mentioned above provide perspectives and tools for addressing the structural challenges of Neural ODEs. In this section, we present our approaches to determine an appropriate network depth, focusing on optimizing from the perspective of time span. We propose two methods: the first involves directly transforming the problem into a minimum-time optimal control problem, which we refer to as Minimum-time Neural ODEs, where the terminal time is treated as a learnable parameter for training; the second approach, as mentioned in the preceding section, focuses on evaluating the performance of the entire trajectory of the system dynamics, which we refer to as a pre-training approach based on convergence guarantee, grounded in control theory perspectives.

First, as mentioned in the previous section, for the system (1), and the objective function $\ell := \Phi\left(z\left(t_f\right)\right) + \int_{t_0}^{t_f} L\left(z\left(t\right), \theta, t\right) dt$, the optimal control problem we recast is as (4) shown where the design of running cost L will be discussed later. The ODE that describes the dynamics of latent state $z\left(t\right)$ is represented by a neural network f whose parameters are θ, taking $z\left(t\right)$ as input.

Gradients of ℓ with respect to parameter θ can be computed via automatic differentiation (Paszke et al., 2017), or via adjoint sensitivity analysis as in the following proposition which has only $O(N_f)$ memory cost during training (Zhuang et al., 2020) where N_f denotes the number of layers of f.

Proposition 3.1. *Consider Problem (1)-(4). The gradient of loss ℓ with respect to parameter θ is*

$$\nabla_\theta \ell = \mu\left(t_0\right) \tag{10}$$

where $z\left(t\right)$, $p\left(t\right)$ and $\mu\left(t\right)$ satisfy the boundary value problem:

$$\begin{aligned} &\dot{z}\left(t\right) = f,\ z\left(t_0\right) = z_0 \\ &\dot{p}\left(t\right) = -p\left(t\right)\frac{\partial f}{\partial z} - \frac{\partial L}{\partial z},\ p\left(t_f\right) = \frac{\partial \Phi}{\partial z\left(t_f\right)} \\ &\dot{\mu}\left(t\right) = -p\left(t\right)\frac{\partial f}{\partial \theta} - \frac{\partial L}{\partial \theta},\ \mu\left(t_f\right) = \mathbb{0}_{n_\theta} \end{aligned} \tag{11}$$

The proof is provided in the supplementary material.

Algorithm 1 Minimum-Time Neural ODEs

Input: Initial time $t_0 > 0$, number of iterations $n > 0$
Result: t_f^*, θ^*

```
Initialize t_f, θ
for i < n do
    z ← ODESolver(f(t, z, θ), z(t_0), t_0, t_f)
    θ ← Optimizer(∇_θ ℓ, θ)      ▷ Update neural network parameters
    t_f ← Optimizer(∇_{t_f} ℓ, t_f)      ▷ Update terminal time
end for
```

3.1. Approach 1: Minimum-time Neural ODEs

We propose the general framework of Minimum-time Neural ODEs. In our setting, it is noteworthy that the terminal time, denoted as t_f, is a learnable parameter rather than a fixed value. Specifically, after each back-propagation, t_f is updated which is utilized in the subsequent forward pass. As to the gradient of ℓ with respect to t_f, the computation is related to taking the derivative with respect to the upper limit of a variable integral:

Proposition 3.2. *Consider Problem (1)-(4). The gradient of loss ℓ with respect to terminal time t_f is*

$$\begin{aligned} \frac{d\ell}{dt_f} &= \frac{\partial \Phi}{\partial z\left(t_f\right)}\frac{dz(t_f)}{dt_f} + \frac{d}{dt_f}\int_{t_0}^{t_f} L\left(z\left(t\right), \theta, t\right) \\ &= \frac{\partial \Phi}{\partial z\left(t_f\right)} f\left(z\left(t_f\right)\right) + L\left(t_f\right) \end{aligned} \tag{12}$$

As to CNF problems whose terminal function is designed as (6), the first term on the right side of (12) is $-\mathbb{E}_{p_x}\left\{\frac{\partial \log p(z(t_f))}{\partial z(t_f)} f(z(t_f))\right\} + \mathbb{E}_{p_x}\left\{-\operatorname{Tr}\frac{\partial f}{\partial z(t_f)}\right\}$.

In our experiments, we jointly update the network parameters and terminal time. However, for finer adjustments, it is advisable to consider employing different optimizers and learning rates for updates and coordinate descent methods could be used for optimization as in Algorithm 1.

Temporal Regularization In order to find the minimum-time optimal control strategy, the loss function is designed as

$$\ell := \Phi\left(z\left(t_f\right)\right) + \lambda \int_{t_0}^{t_f} 1 dt \tag{13}$$

which can be seen as $L = \lambda$. In this case, $\nabla_{t_f}\ell = \frac{\partial\Phi}{\partial z(t_f)} f\left(z\left(t_f\right)\right) + \lambda$. This is equivalent to the method of adding a regularization term $\lambda\left|t_f - t_0\right|$ directly to the loss function, where λ denotes the power of regularization on the training process and can be seen as the trade-off between final performance and integral time span.

In addition, to ensure the safety of terminal time updates and prevent overly abrupt changes, we propose to apply clipping to the updates:

$$t_f = \begin{cases} T, & t_f > T \\ t_0 + \epsilon, & t_f < t_0 + \epsilon \\ t_f, & \text{else} \end{cases} \tag{14}$$

where ϵ denotes the step size used when using a fixed-step ODE solver, and in other cases, it can be chosen as a small value. When the initial upper bound of integration is chosen to be sufficiently large, T can be set as the initial upper bound T_f, otherwise alternative values may be considered.

Since we simultaneously train both neural network parameters and terminal time, the training process, especially in its initial stages, may exhibit instability, and the choice of multipliers could have a significant impact. In our approach, we suggest starting with a relatively large initial terminal time and optimize it in a backward manner. In this setting, even when we consider only terminal loss in addition to time regularization, after each step of backward updating t_f, the terminal state is also pushed backward. As a result, it gradually demands the learned dynamics to reach the target state earlier and thus requires the learning of more direct and faster dynamics in the spatial perspective.

Besides, it is possible that dynamics learned in this manner become very fast. If there is a limitation on the speed of dynamics for the sake of stability, specifically the magnitude of $\dot{z}$, it can be enforced through clipping within the network.

3.2. Approach 2: Convergence-rate-based Neural ODEs

The loss function employed in the aforementioned approach in (13), apart from incorporating an explicit penalty on time, solely takes into account considerations related to the terminal state. Although it is demonstrating promising performance, including learning more direct and faster dynamics, convergence guarantees are lacking. To overcome this limitation, we propose to first pre-train a dynamics model with guaranteed convergence speed by including trajectory tracking in the loss function via Lyapunov method from the spatial perspective on a small time interval, and then search for the minimum terminal time satisfying the terminal state requirements in the forward direction. This is because, as observed in Figure 1(a), in the context of learning problems, the temporal extent (or number of layers) does not need to be excessively large to achieve a small error under control. In other words, the guaranteed rate of convergence implies that any layer beyond a certain time can be dropped from training theoretically.

Algorithm 2 Convergence-rate-based Pre-training algorithm

Input: Initial time $t_0 > 0$, number of iterations $n > 0$, threshold $\varepsilon > 0$, update step size for terminal time $\gamma > 0$
Result: t_f^*, θ^*

```
Initialize t_f, θ
for i < n do
    z ← ODESolver(f(t, z, θ), z(t_0), t_0, t_f)
    θ ← Optimizer(∇_θ ℓ, θ)   ▷ Update neural network parameters
end for
while Φ(t_f) > ε do
    t_f ← t_f + γ             ▷ Update terminal time
    z ← ODESolver(f(t, z, θ), z(t_0), t_0, t_f)
    θ ← Optimizer(∇_θ ℓ, θ)
end while
```

Lyapunov method Inspired by LyaNet (Rodriguez et al., 2022), for a given training data pair and supervised loss Φ, the potential function V can be designed as

$$V_{\hat{y}}(\cdot) \coloneqq \Phi\left(z\left(\cdot\right)\right) \tag{15}$$

For example, when standard cross-entropy loss is considered, we consider using the truncated cross entropy loss defined as $\Phi(\cdot) \coloneqq \max\{0, \Phi_{ce}\}$. Then a point-wise Lyapunov loss can be designed as

$$\mathcal{V} \coloneqq \max\left\{0, \frac{\partial V_{\hat{y}}}{\partial z} f\left(t, z, \theta\right) + \kappa V_{\hat{y}}\left(z\right)\right\} \tag{16}$$

Equation (38) signifies the local violation of invariance condition specified in (8). When $\mathcal{V} = 0$ holds for all data in the time interval, as per Theorem 1, the inference dynamics exhibit exponential convergence towards a prediction that minimizes the loss. The Lyapunov loss for the dynamic system (1) is

$$\ell \coloneqq \mathbb{E}\left[\int_{t_0}^{t_f} \mathcal{V} dt\right] \tag{17}$$

Theorem 2 ((Rodriguez et al., 2022)). Consider the Lyapunov loss above. If there exists a parameter θ^* of the dynamic system that satisfies $\ell(\theta^*) = 0$, then:

- The potential function $V_{\hat{y}}$ is an exponentially stabilizing Lyapunov function with θ^*;
- For $t \in \left[t_0, t_f\right]$, the dynamic satisfies the convergence speed with respect to the loss Φ:

$$\Phi\left(z(t)\right) \leq \Phi\left(z(t_0)\right) e^{-\kappa t} \tag{18}$$

The Lyapunov method provides a guarantee of convergence rate for a broader range of problems and affords us the

opportunity to manually select the rate of convergence, as there might be instances where too fast dynamics are not preferred.

Learning procedure It is noteworthy that, for instance, the utilization of Lyapunov loss ensures a convergence rate but does not precisely guarantee the convergence of the terminal state to a specific value since Lyapunov loss lacks constraints specifically targeting the terminal state itself. This dependence is influenced by the magnitude of the terminal time, t_f. If t_f is very small, the learned dynamics might not have sufficient time to elevate the state to the desired value. Conversely, if t_f is excessively large, the network could expend excessive depth on processing already converged states, resulting in unnecessary time consumption. Therefore, we introduce a pre-training approach. Initially, we commence with a smaller initial value of t_f, implying the utilization of a shallower network to train a swift dynamic along the integration path. Subsequently, we iteratively augment the time span, e.g. network depth, until the model loss reaches the predefined threshold. The algorithm is shown in Algorithm 2. This algorithm can be viewed as a greedy approach, as it aims to identify the shortest required time span, corresponding to the shallowest neural network, thereby minimizing unnecessary layer complexity.

Approach 1 is convenient and intuitive, accomplished in a single stage, yet the selection of parameter λ is a challenge, potentially resulting in performance degradation. Besides, it currently lacks a theoretical foundation to guarantee the convergence rate of the learned underlying dynamics. On the other hand, Approach 2 provides dynamics with convergence guarantees, and it can lead to an overall faster training since a much shorter time span is selected during pre-training. However, this approach requires a two-stage process and becomes more challenging when Neural ODEs are not positioned at the end of the network architecture. This is because the states along the trajectory need to be passed into the subsequent neural network, which is a limitation of LyaNet itself Note that the second approach can be combined with the previous Minimum-time Neural ODEs approach by incorporating time regularization and a terminal loss, treating t_f as a learnable parameter for training in a single stage. However, in preliminary experiments this combined approach does not exhibit significant advantages compared to the individual approaches as shown in Appendix F.2.

4. Experiments

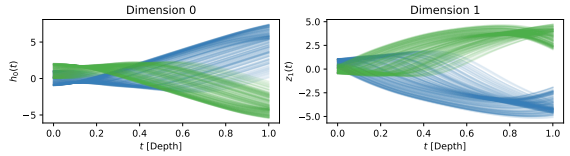
In this section, we demonstrate the benefits of the proposed methods on a variety of machine learning tasks. We compare the results among Vanilla Neural ODEs, our methods, LyaNet, STEER, TayNODE (Kelly et al., 2020) and SNOpt. Our code is available at https://github.com/KYMiao/Accelerating-Neural-ODEs.

4.1. Supervised Learning

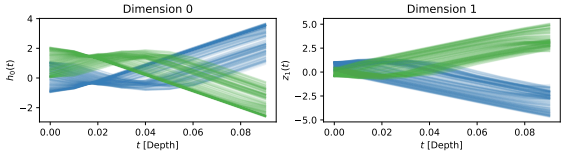
Binary Classification (two moons) We test our methods on the two moons dataset for binary classification. The results are displayed in Table 1 and Figure 3. The time of pre-training and second phase training are also indicated in Table 1 for the Convergence-rate-based method. Our methods have learned considerably faster dynamics that distinguish between the two data groups quickly, leading to an order of magnitude reduction in training and inference time.

Table 1. Performance on Supervised Learning Problems

	Method	t_f	Training	Inference
Binary	Vanilla	1	46.64s	71.88ms
	Minimum-time	0.0904	7.33s	8.4ms
	Convergence-rate-based	0.10	Pre: 5.42s Sec: 0.79s	8.44ms
CA	Vanilla	1	123s	86.6ms
	Minimum-time	0.0819	27s	10.52ms
	Convergence-rate-based	0.09	Pre: 15.98s Sec: 1.02s	10.24ms



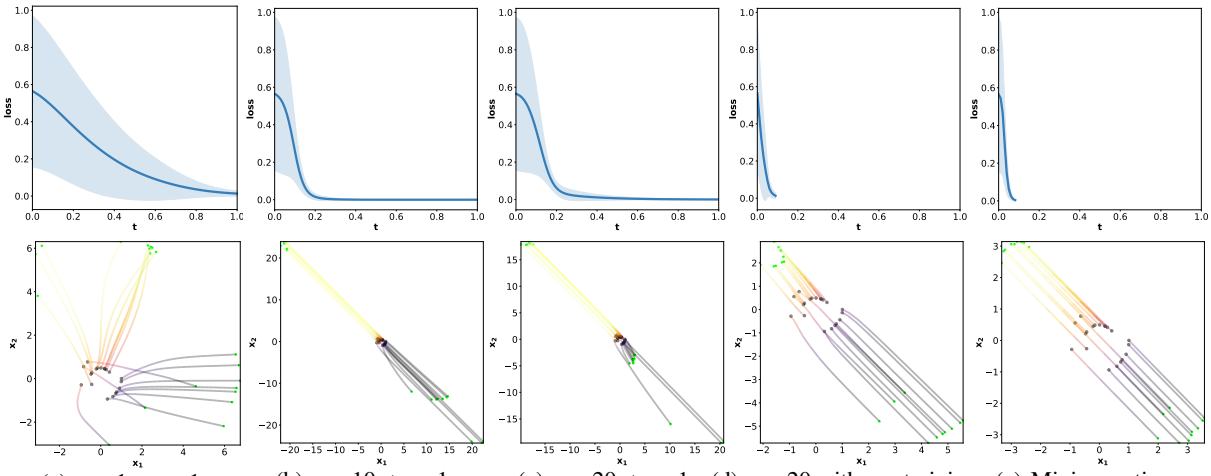
(a) State trajectory learned by Vanilla Neural ODEs



(b) State trajectory learned by Minimum-time Neural ODEs

Figure 3. Performance on two moons dataset

Binary Classification (Concentric Annuli) We present our results on Concentric Annuli (CA) problem (introduced in Appendix F.2) in Table 1 and Figure 4, which includes the loss curve over the time span and the state trajectories. An interesting observation emerges when we apply the same value of κ to training on the interval $[0, 1]$, akin to the approach taken by LyaNet. We discover that, for $\kappa = 20$, faster dynamics are not learned as shown in Figure 4(a)-4(c) compared to the case $\kappa = 10$. However, when considering shorter time spans, a faster dynamic is attainable. This phenomenon can be elucidated: over a longer time span, although theoretically feasible, it becomes numerically difficult to sustain a substantial rate of descent once the loss



(a) $\kappa = 1, t_f = 1$ (b) $\kappa = 10, t_f = 1$ (c) $\kappa = 20, t_f = 1$ (d) $\kappa = 20$ with pre-training (e) Minimum-time

Figure 4. Performance on Concentric Annuli Problem using different methods: (a)-(c) shows the results obtained by Lyapunov method trained with $\kappa = 1, 10, 20$ respectively on $[0, 1]$; (d) demonstrates the results obtained by our Convergence-rate-based method with pre-training on $[0, 0.05]$, $\kappa = 20$, (e) shows the results obtained by Minimum-time Neural ODEs

Table 2. Image Classification using Neural ODEs

	Method		Terminal Time	Test Accuracy	Training Time	Inference Time	NFE
MNIST	Vanilla NODEs		1	99.57%	4h47min23s	0.546s	402.6
	Minimum-time NODEs		0.0343	99.53%	20min23s	0.03s	16.1
Fashion MNIST	Vanilla NODEs		1	93.07%	4h41min19s	0.494s	402.6
	Minimum-time NODEs		0.0564	92.68%	22min24s	0.044s	24.2
CIFAR-10	Vanilla NODEs		1	82.60%	>10h	0.531s	404.9
	Minimum-time NODEs	$\lambda = 1$	0.1103	76.38%	42min37s	0.098s	47.3
		$\lambda = 0.5$	0.1975	78.41%	50min57s	0.124s	81
		$\lambda = 0.1$	0.3606	81.74%	2h22min15s	0.278s	146.1

has converged to a relatively small value. This difficulty may be exacerbated with larger values of κ, as violations of the Lyapunov function are likely to occur a lot over the longer time span. However, over a shorter time span, especially when the loss has not yet reached a minimum, such training is more feasible. This observation can be seen as an additional advantage of the pre-training approach. The results of the combined method and SNOpt (Liu et al., 2021) are shown in Appendix F.2.

Image classification We evaluated the performance of our approach on several benchmark datasets for image classification tasks, including MNIST (LeCun, 1998), Fashion-MNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky et al., 2009). The results are shown in Table 2 where NFE measures the number of evaluations required by the ODE solver to determine the solution trajectory. And the comparison of our method and other popular approaches such as TayN-ODEs (Kelly et al., 2020) on computation time are shown in Figure 5. Also, (Liu et al., 2021) reports that its SNOpt method can reduce the training time by 30% on the MNIST dataset and 19% on the CIFAR-10 dataset, based on the adjoint sensitivity method and adaptive solvers as shown in Appendix F.3. However, adjoint sensitivity method is inherently much slower itself, our approach demonstrates a greater advantage in speed improvements. It is evident that our approach achieves significant speed enhancements in both training and inference compared to Vanilla Neural ODEs across these three datasets, with only minimal sacrifice in test accuracy. Particularly noteworthy is the experimentation on the CIFAR-10 dataset, where we achieved a trade-off between accuracy and training/inference time by tuning the parameter λ.

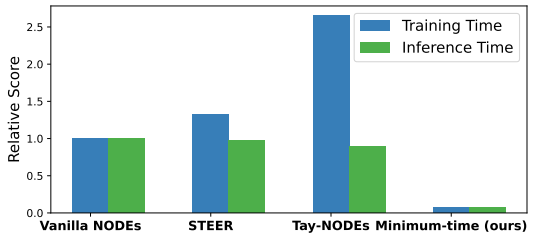


Figure 5. Comparison of different methods' time consumption for classification task on MNIST dataset

MNIST classification is less complex than CIFAR10 classification. Our results show that when we use the same network architecture, desired performance can be achieved with a smaller depth (integration time) in MNIST, while a longer depth is needed for CIFAR10. As a result, this gives us inspiration for choosing hyper parameters. In more complex problems, consider choosing slightly smaller λ since our ultimate task is to find the appropriate depth that can guarantee the performance of the model. The results provide us also with inspiration on how this model can be scaled up to more complex problems; to solve more complex problems, we might consider 1) designing a more sophisticated network structure, and 2) finding a longer integration interval.

4.2. Generative Models

Figure 6 demonstrates our experiments on generative models using the two moons dataset. Vanilla Neural ODEs took more than 18 minutes to train and 100 steps to generate the target pattern, while our minimum-time method took around 6 minutes and 20 steps respectively.

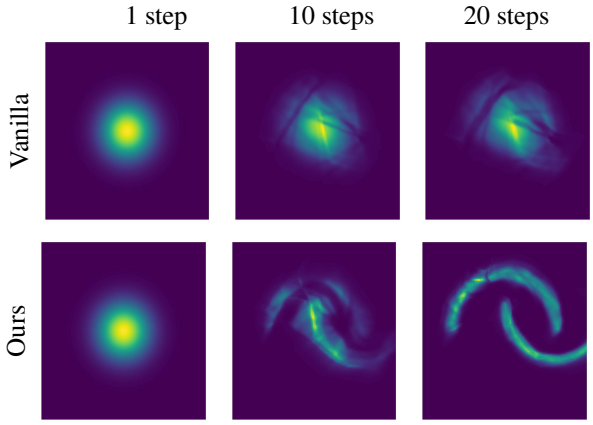


Figure 6. CNF Results on two-moons dataset

5. Discussions and Future work

In this work, we propose a control teory perspective, borrowing techniques such as minimum-time optimal control and Lyapunov methods from temporal and spatial angles, to learn more efficient dynamics, determine a more suitable network depth, optimize network structure, and consequently accelerate the training and inference of Neural ODEs while mitigating unnecessary computations. Future research directions include integrating our approach with acceleration targeting ODE solvers to further enhance computational efficiency especially for the case that adaptive ODE solvers are used. Additionally, the combination of our method with efforts to enhance network robustness presents another promising avenue. Furthermore, we intend to explore time-variant Neural ODEs whose training aligns more closely with the optimal control problems, thus enabling a more profound and elucidating investigation of the interplay between turnpike theory and the depth of Neural ODEs.

Acknowledgements

The authors would like to express sincere gratitude to Prof. Antonis Papachristodoulou for his invaluable suggestions and refinement of this paper. His insightful feedback and support improved the quality of this work. Appreciation is also extended to the Engineering and Physical Sciences Research Council (EPSRC) for the financial support provided through the grant EP/T517811/1.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ames, A. D., Galloway, K., Sreenath, K., and Grizzle, J. W. Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics. *IEEE Transactions on Automatic Control*, 59(4):876–891, 2014.

Bellman, R. *Dynamic Programming*. Princeton University Press, New Jersey, 1957.

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Djeumou, F., Neary, C., Goubault, E., Putot, S., and Topcu, U. Neural networks with physics-informed architectures and constraints for dynamical systems modeling. In *Learning for Dynamics and Control Conference*, pp. 263–277. PMLR, 2022.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes. *Advances in neural information processing systems*, 32, 2019.

Ee, W., Han, J., and Li, Q. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6, 12 2018. doi: 10.1007/s40687-018-0172-y.

Faulwasser, T., Hempel, A.-J., and Streif, S. On the turnpike to design of deep neural nets: Explicit depth bounds. *arXiv preprint arXiv:2101.03000*, 2021.

Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. How to train your neural ode: the world of jacobian and kinetic regularization. In *International conference on machine learning*, pp. 3154–3164. PMLR, 2020.

Ghosh, A., Behl, H., Dupont, E., Torr, P., and Namboodiri, V. Steer: Simple temporal regularization for neural ode. *Advances in Neural Information Processing Systems*, 33: 14831–14843, 2020.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Hassani, H. and Javanmard, A. The curse of overparametrization in adversarial training: Precise analysis of robust generalization for random features regression. *The Annals of Statistics*, 52(2):441–465, 2024.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Huang, H., Wang, Y., Erfani, S., Gu, Q., Bailey, J., and Ma, X. Exploring architectural ingredients of adversarially robust deep neural networks. *Advances in Neural Information Processing Systems*, 34:5545–5559, 2021.

Huang, Y., Yu, Y., Zhang, H., Ma, Y., and Yao, Y. Adversarial robustness of stabilized neural ode might be from obfuscated gradients. In *Mathematical and Scientific Machine Learning*, pp. 497–515. PMLR, 2022.

Kang, Q., Song, Y., Ding, Q., and Tay, W. P. Stable neural ode with lyapunov-stable equilibrium points for defending against adversarial attacks. *Advances in Neural Information Processing Systems*, 34:14925–14937, 2021.

Kelly, J., Bettencourt, J., Johnson, M. J., and Duvenaud, D. K. Learning differential equations that are easy to solve. *Advances in Neural Information Processing Systems*, 33: 4370–4380, 2020.

Kidger, P. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.

Li, Q., Chen, L., Tai, C., et al. Maximum principle based algorithms for deep learning. *arXiv preprint arXiv:1710.09513*, 2017.

Liang, Y., Ouyang, K., Yan, H., Wang, Y., Tong, Z., and Zimmermann, R. Modeling trajectories with neural ordinary differential equations. In *IJCAI*, pp. 1498–1504, 2021.

Liu, G.-H., Chen, T., and Theodorou, E. A. Ddpnopt: Differential dynamic programming neural optimizer. *arXiv preprint arXiv:2002.08809*, 2020.

Liu, G.-H., Chen, T., and Theodorou, E. Second-order neural ode optimizer. *Advances in Neural Information Processing Systems*, 34:25267–25279, 2021.

Massaroli, S., Poli, M., Park, J., Yamashita, A., and Asama, H. Dissecting neural odes. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020.

Miao, K. and Gatsis, K. Learning robust state observers using neural odes. In *Learning for Dynamics and Control Conference*, pp. 208–219. PMLR, 2023.

Pal, A., Ma, Y., Shah, V., and Rackauckas, C. V. Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In *International Conference on Machine Learning*, pp. 8325–8335. PMLR, 2021.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Poli, M., Massaroli, S., Yamashita, A., Asama, H., and Park, J. Hypersolvers: Toward fast continuous-depth models. Jul 19, 2020.

Pontryagin, L. *Mathematical theory of optimal processes*. CRC press, 1987.

Rodriguez, I. D. J., Ames, A., and Yue, Y. Lyanet: A lyapunov framework for training neural odes. In *International Conference on Machine Learning*, pp. 18687–18703. PMLR, 2022.

Rubanova, Y., Chen, R. T., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Sun, S., Chen, W., Wang, L., Liu, X., and Liu, T.-Y. On the depth of deep neural networks: A theoretical view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Tabuada, P. and Gharesifard, B. Universal approximation power of deep residual neural networks via nonlinear control theory. *arXiv preprint arXiv:2007.06007*, 2020.

Teerapittayanon, S., McDanel, B., and Kung, H.-T. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.

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

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yan, H., Du, J., Tan, V. Y., and Feng, J. On robustness of neural ordinary differential equations. *arXiv preprint arXiv:1910.05513*, 2019.

Zhao, L., Gatsis, K., and Papachristodoulou, A. A barrier-lyapunov actor-critic reinforcement learning approach for safe and stable control. *arXiv preprint arXiv:2304.04066*, 2023.

Zhao, L., Miao, K., Gatsis, K., and Papachristodoulou, A. Nlbac: A neural ordinary differential equations-based framework for stable and safe reinforcement learning. *arXiv preprint arXiv:2401.13148*, 2024.

Zhu, Z., Liu, F., Chrysos, G., and Cevher, V. Robustness in deep learning: The good (width), the bad (depth), and the ugly (initialization). *Advances in neural information processing systems*, 35:36094–36107, 2022.

Zhuang, J., Dvornek, N., Li, X., Tatikonda, S., Papademetris, X., and Duncan, J. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *International Conference on Machine Learning*, pp. 11639–11649. PMLR, 2020.

A. Optimal Control and Maximum Principle

The approaches to solving optimal problems includes variation method, the Pontryagin maximum principle, and dynamic programming. Applying the variation method to solve the optimal control problem requires that the control vector is not subject to any constraints, and the Hamiltonian function is required to be continuously differentiable to the control vector. However, in actual engineering problems, control variables are often subject to certain restrictions. In order to solve the constrained variation problem, Pontryagin proposed and proved the principle of maximum value. Its conclusion has many similarities with the conclusion of the variation method. It can be applied to the situation where the control variable is limited by the boundaries and does not require the Hamiltonian function is continuously differentiable to the control vector. So it has been widely used.

For a minimization problem, given a system of ordinary differential equations

$$\dot{z}(t) = f(z(t), \theta(t), t) \tag{19}$$

where $z \in R^n$ is a phase vector, $\theta \in R^m$ is a control parameter, and f is a continuous vector function in the variable z and θ, which is continuously differentiable with respect to z. U is a certain set of admissible values of control parameter θ. The initial time t_0 is fixed. Boundaries can be fixed, free or constrained by trajectory. Among all admissible controls transferring the phase point from the position $z(t_0)$ to the position $z(t_f)$, it is required to find an optimal control, a function $\theta^*(t)$ for which the objective function

$$\ell = \int_{t_0}^{t_f} L(z(t), \theta(t), t)\, dt + \Phi(z(t_f), t_f) \tag{20}$$

takes smallest possible value.
Define the Hamiltonian function

$$H(z(t), \theta(t), p(t), t) = p^T(t) f(z(t), \theta(t), t) + L(z(t), \theta(t), t) \tag{21}$$

Then the optimal control $\theta^*(t)$, the optimal state trajectory $z^*(t)$ and the optimal adjoint trajectory $p^*(t)$ satisfy the following conditions (Pontryagin, 1987):

The Canonical Equation:

$$\dot{z}^*(t) = \frac{\partial H(z^*(t), \theta^*(t), p^*(t), t)}{\partial p} \tag{22}$$

The Adjoint Equation:

$$-\dot{p}^*(t) = \frac{\partial H(z^*(t), \theta^*(t), p^*(t), t)}{\partial z} \tag{23}$$

The Minimization of the Hamiltonian Condition:

$$\min_{\theta \in U} H(z^*(t), \theta(t), p^*(t), t) = H(z^*(t), \theta^*(t), p^*(t), t) \tag{24}$$

or

$$H(z^*(t), \theta(t), p^*(t), t) \geq H(z^*(t), \theta^*(t), p^*(t), t) \quad \forall t \in [t_0, t_f], \forall \theta(t) \in U \tag{25}$$

The Transversality Condition:

$$p^*(t_f) = \frac{\partial \Phi(z(t_f), t_f)}{\partial z(t_f)} \tag{26}$$

Constancy of the Hamiltonian for Autonomous Problems:

$$\begin{aligned} &H(z^*(t), \theta^*(t), p^*(t), t) = constant && \text{if } t_f \text{ is fixed} \\ &H(z^*(t), \theta^*(t), p^*(t), t) = 0 && \text{if } t_f \text{ is free and positive} \end{aligned} \tag{27}$$

Note that if $z(t_0)$ (respectively $z(t_f)$ is fixed, then $p(t_0)$ (respectively $p(t_f)$) is free, if $z(t_0)$ (respectively $z(t_f)$) is free, then $p(t_0)$ (respectively $p(t_f)$) is fixed. The advantage of the Pontryagin Maximum Principle is that it can be used to solve the optimal control problems with constraints of control parameter and give the conditions which optimal control should follows. It is noteworthy that what Pontryagin Maximum Principle provides are only necessary conditions.

B. Proof of Proposition 3.1

Proof. Define the augmented objective function with a Lagrange multiplier p as

$$\mathcal{L} = \ell - \int_{t_0}^{t_f} p(t)\left[\dot{z}(t) - f\left(t, z(t), \theta\right)\right] dt \tag{28}$$

and $\dot{z} - f = 0$ always holds by construction, so $p(t)$ can be freely assigned while $\frac{d\mathcal{L}}{d\theta} = \frac{d\ell}{d\theta}$. As to the integration part on right hand side of (28), we have

$$\begin{aligned}\int_{t_0}^{t_f} p(t)\left(\dot{z} - f\right) dt = p(t)z(t)\Big|_{t_0}^{t_f} \\ - \int_{t_0}^{t_f} \dot{p}(t)z(t)dt - \int_{t_0}^{t_f} p(t)f dt \\ = p\left(t_f\right) z\left(t_f\right) \\ - p\left(t_0\right) z\left(t_0\right) - \int_{t_0}^{t_f} \left(\dot{p}(t)z(t) + p(t)f\right) dt\end{aligned}$$

Hence,

$$\begin{aligned}\mathcal{L} = \Phi\left(z\left(t_f\right)\right) - p\left(t_f\right) z\left(t_f\right) + p\left(t_0\right) z\left(t_0\right) \\ + \int_{t_0}^{t_f} \left(\dot{p}(t)z(t) + p(t)f + L\right) dt\end{aligned}$$

Then the gradient of ℓ with respect to θ can be computed as

$$\begin{aligned}\frac{d\ell}{d\theta} = \frac{d\mathcal{L}}{d\theta} = \left(\frac{\partial\Phi}{\partial z\left(t_f\right)} - p(t_f)\right)\frac{dz(t_f)}{d\theta} \\ + \int_{t_0}^{t_f} \left(\dot{p}(t)\frac{dz(t)}{d\theta} + p(t)\left(\frac{\partial f}{\partial\theta} + \frac{\partial f}{\partial z}\frac{dz}{d\theta}\right) + \frac{\partial L}{\partial\theta} + \frac{\partial L}{\partial z}\frac{dz}{d\theta}\right) dt\end{aligned} \tag{29}$$

Now if we set this Lagrange multiplier as the adjoint state for the Hamiltonian function

$$H\left(z, p, \theta\right) = pf + L \tag{30}$$

and according to PMP, the optimality conditions requires

$$\dot{z} = \frac{\partial H}{\partial p} = f,\ \dot{p} = -\frac{\partial H}{\partial z} = -p\frac{\partial f}{\partial z} - \frac{\partial L}{\partial z} \tag{31}$$

with initial conditions $z(t_0) = z_0$ and $p(t_f) = \frac{\partial\Phi}{\partial z(t_f)} = \frac{\partial\Phi}{\partial z\left(t_f\right)}$. Substituting (31) into (29), it can be obtained that

$$\begin{aligned}\frac{d\ell}{d\theta} = \int_{t_0}^{t_f} \left(p(t)\frac{\partial f}{\partial\theta} + \frac{\partial L}{\partial\theta}\right) dt \\ = \int_{t_f}^{t_0} \left(-p(t)\frac{\partial f}{\partial\theta} - \frac{\partial L}{\partial\theta}\right) dt = \int_{t_f}^{t_0} -\frac{\partial H}{\partial\theta} dt\end{aligned} \tag{32}$$

proving the result. □

C. Proof of gradient w.r.t. time

Proof. As to the gradient of ℓ with respect to t_f, the computation is related to taking the derivative with respect to the upper limit of a variable integral:

$$\begin{aligned}\frac{d\ell}{dt_f} = \frac{\partial\Phi}{\partial z\left(t_f\right)}\frac{dz(t_f)}{dt_f} + \frac{d}{dt_f}\int_{t_0}^{t_f} L\left(z\left(t\right), \theta, t\right) \\ = \frac{\partial\Phi}{\partial z\left(t_f\right)} f\left(z\left(t_f\right)\right) + L\left(t_f\right)\end{aligned} \tag{33}$$

In fact, we can optimize the initial time t_0 as well. The derivative of the loss with respect to t_0 is $\nabla_{t_0}\ell = \frac{d\ell}{dt_f} - \int_{t_f}^{t_0}\left(-p(t)\frac{\partial f}{\partial t} - \frac{\partial L}{\partial t}\right)dt$.

Let $p_{aug} = \begin{bmatrix} p \\ p_t \end{bmatrix}$ where $p_t = \frac{d\ell}{dt(t)}$. The Jacobian of f has the form

$$\frac{\partial f_{aug}}{\partial [z,t]} = \begin{bmatrix} \frac{\partial f}{\partial z} & \frac{\partial f}{\partial t} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}(t) \tag{34}$$

then

$$\frac{dp_{aug}}{dt} = \begin{bmatrix} p(t) & p_t(t) \end{bmatrix}\frac{\partial f_{aug}}{\partial [z,t]} - \begin{bmatrix} 1 & 1 \end{bmatrix}\frac{\partial L}{\partial [z,t]} = -\begin{bmatrix} p(t)\frac{\partial f}{\partial z} & p_t(t)\frac{\partial f}{\partial t} \end{bmatrix} - \begin{bmatrix} \frac{\partial L}{\partial z} & \frac{\partial L}{\partial t} \end{bmatrix} \tag{35}$$

Hence, the gradients with respect to t_0 is

$$\nabla_{t_0}\ell = p_t(t_0) = \frac{d\ell}{dt_f} - \int_{t_f}^{t_0}\left(-p(t)\frac{\partial f}{\partial t} - \frac{\partial L}{\partial t}\right)dt \tag{36}$$

proving the result. □

However, in our setting, we are primarily concerned with the length of the time interval, i.e. the depth. Therefore, we only need to learn the terminal time.

D. Proof of Theorem 3

Starting by rearranging the terms of the integral:

Proof.

$$\ell \coloneqq \mathbb{E}_{(z_0,\hat{y})\sim D}\left[\int_{t_0}^{t_f}\mathcal{V}dt\right] = \int_D\int_{t_0}^{t_f}\mathcal{V}dtdD((z_0,\hat{y})) = \int_{D\times[t_0,t_f]}\mathcal{V}dD(t,(z_0,\hat{y})) \tag{37}$$

Recall that

$$\mathcal{V} \coloneqq \max\left\{0, \frac{\partial V_{\hat{y}}}{\partial z}f\left(t,z,\theta\right) + \kappa V_{\hat{y}}\left(z\right)\right\} \tag{38}$$

It can be found that $\mathcal{V}$ is beging integrated over a bounded domain and it satisties the following properties:

- $\mathcal{V} \geq 0$ for all values of $z_0, \hat{y}, t$.
- $\mathcal{V}$ is continuous since it is themaximum of two continuous functions.

Then it can be concluded that $\ell(\theta^*) = 0$ only can be achieved when $\mathcal{V} = 0$ for all data. This follows from the standard calculus argument that if the function weren't zero at a point there would be a region surrounding that point that would integrate to a strictly positive value. Then it can only be satisfied when

$$\frac{\partial V_{\hat{y}}}{\partial z}f\left(t,z,\theta^*\right) + \kappa V_{\hat{y}}\left(z\right) \leq 0 \tag{39}$$

for all data. According to Theorem 1, we have that the potential function $V_{\hat{y}}$ is an exponentially stabilizing Lyapunov function with θ^* and since $V_{\hat{y}}(\cdot) \coloneqq \Phi\left(z\left(\cdot\right)\right)$, it can be shown that

$$\Phi\left(z(t)\right) \leq \Phi\left(z(t_0)\right)e^{-\kappa t} \tag{40}$$

□

E. Update of t_f in Minimum-time Neural ODEs

To ensure the safety of terminal time updates and prevent overly abrupt changes, we propose to apply clipping to the updates:

$$t_f = \begin{cases} T, & t_f > T \\ t_0 + \epsilon, & t_f < t_0 + \epsilon \\ t_f, & \text{else} \end{cases} \tag{41}$$

where ϵ denotes the step size used when using a fixed-step ODE solver, and in other cases, it can be chosen as a small value. When the initial upper bound of integration is chosen to be sufficiently large, T can be set as the initial upper bound T_f, while in other situations, alternative values may be considered. For instance, $T = 2T_f - t_0 - \epsilon$, and in this case the centre of the feasible interval for t_f is T_f where T_f denotes the initial upper bound.

F. Experiments Details

F.1. Binary Classification

The moon dataset is shown as Figure 7. The experiments are run on Apple M1 Pro chip. We fit a three–layer network of

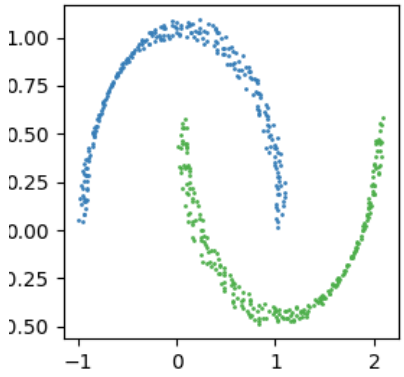


Figure 7. moon dataset

hidden dimensions 16. ODE-solver is chosen as RK4 with step size 0.01. Optimizer is Adam with learning rate as 0.1.

The results obtainded by Lyapunov method are shown in Figure 8 respectively.

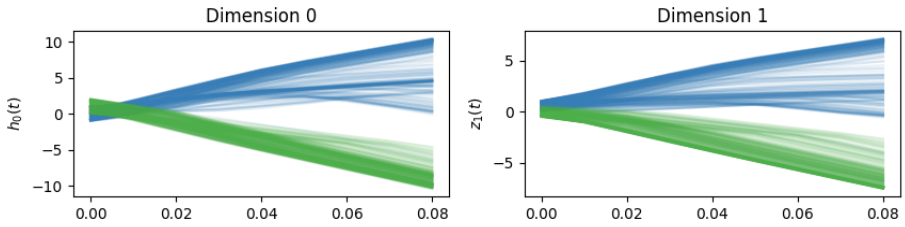


Figure 8. State trajectory learned by Lyapunov method with pre-training

F.2. Concentric Annuli

The Concentric Annuli is shown as Figure 9. The problem is described as:

$$f(x) = \begin{cases} 0 & \text{if} \quad \|x\| = r_1 \\ 1 & \text{if} \quad \|x\| = r_2 \end{cases} \tag{42}$$

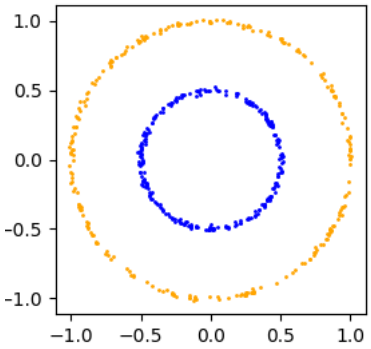


Figure 9. Concentric Annuli

Table 3. Performance of Combined Method on Concentric Annuli

	Method	Terminal Time	Training Time	Inference Time
Concentric Annuli	Vanilla	1	123s	86.6ms
	Minimum-time	0.0819	27s	10.52ms
	Convergence-rate-based	0.09	Pre: 15.98s Sec: 1.02s	10.24ms
	Combined	0.0687	16.03s	10.01ms

where x represents 2-dimensional data.
The experiments are run on Apple M1 Pro chip. The basic sturcture is chosen as Augmented Neural ODEs to address the problem of intersection of integral trajectory. We fit a three–layer network of hidden dimensions 32. ODE-solver is chosen as RK4 with step size 0.01. Optimizer is Adam with learning rate as 0.01, scheduler as ExponentialLR.

Especially, for Lyapunov method, which is pre-trained on $[0, 0.05]$, the pre-trained results are shown in Figure 10. Then, the

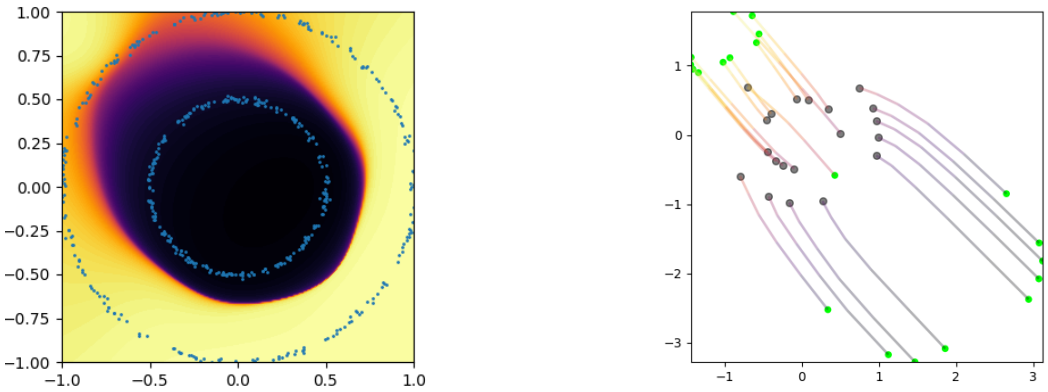


Figure 10. Pre-trained results. left: Boundary Decision; right: State Trajectories

threshold of loss is set as 0.01, the results after finishing the iteration to find the safe terminal time are shown in Figure 11.

As the paper points out, these two approaches can be combined to complement each other's strengths by using $\ell = \ell_{lya} + \lambda_1\phi + \lambda_2 \int_{t_0}^{t_f} 1dt$. The results of the combined method are shown in Table 3.
The results of using the network structure optimization in SNOpt (Liu et al., 2021) in the Concentric Annuli problem are shown in Table 4. The speedup of this method is also significant, but its total elapsed time is still long because it is based on

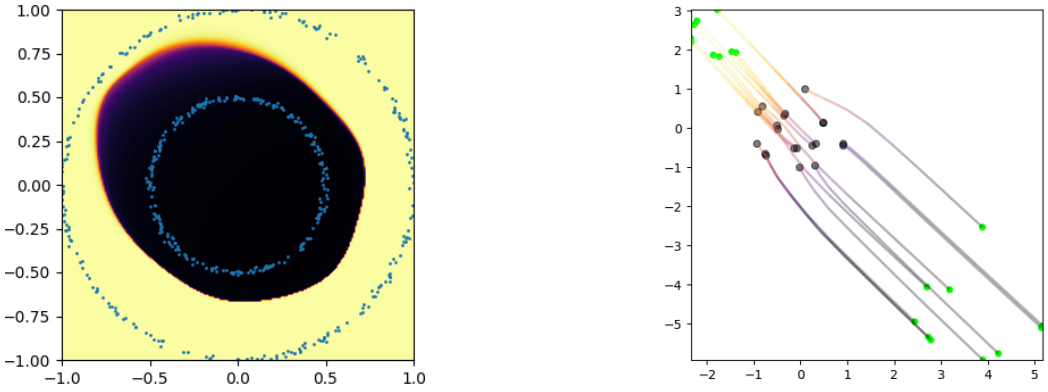


Figure 11. Final results. left: Boundary Decision; right: State Trajectories

Table 4. Results Comparison with SNOpt (Liu et al., 2021)

	Method	Training Time	Inference Time
Concentric Annuli	Vanilla (Auto Diff)	123s	86.6ms
	Minimum-time	27s	10.52ms
	Convergence-rate-based	Pre: 15.98s Sec: 1.02s	10.24ms
	Vanilla (ASM + adaptive)	1307s	0.91s
	SNOpt (adaptive)	927s	0.54s
	Vanilla (ASM + fixed)	465s	0.33s
	SNOpt (fixed)	89s	34.2ms

the adjoint sensitivity method (ASM). When using a fixed stepsize ODE solver, the total depth of the network is directly linked to the terminal time. In this paper, we use the fixed step solver for consideration, because in most cases of our target tasks, the performance of the fixed step solver is sufficient. Our method can be equally applied to the case that uses adaptive stepsize solver, where again it has the effect of accelerating training and inference. But the acceleration will not be as significant as in the fixed stepsize method, because in addition to the terminal time, the adaptive step size also affects the overall network depth. The results on Concentric Annuli are as follows in Table 5.

F.3. Image Classification

The experiments are run on NVIDIA Tesla V100 GPU. The network sturcture is based on the code from Neural ODEs paper. The ODE-solver is chosen as RK4 with step size 0.01, and integral time span for Vanilla Neural ODEs is set as $[0, 1]$. Optimizer is SGD with learning rate starting as 0.1 with decay rate [0.1, 0.01, 0.001] at epoch [60, 100, 140]. Training epoch is chosen as 160, batch size is 128. Data augmentation technique are used on all the dataset, MNIST, Fashion-MNIST and CIFAR.

The reduction in training time using minimum-time Neural ODEs and network structure optimization scheme in SNOpt (Liu et al., 2021) for MNIST and CIFAR-10 is shown in the table, where the results for SNOpt are taken from its paper. We use the same network architecture as (Liu et al., 2021; Chen et al., 2018) for image classification on MNIST and CIFAR-10. It should be pointed out that our percentage training time reduction is based on Vanilla Neural ODEs using automatic

Table 5. Performance with Adaptive Solvers on CA

	Method	t_f	Training Time	Inference Time
Concentric Annuli	Vanilla (adaptive)	1	168s	117.288ms
	Minimum-time (adaptive)	0.0724	129s	86.493ms

Table 6. Training Time Reduction with SNOpt and Our Method on Image Datasets

Dataset	Method	t_f	Test Accuracy	Percentage Training Time Reduction
MNIST	Minimum-time NODEs	0.0349	99.53%	94%
	SNOpt (Liu et al., 2021)	0.38	98.99%	30%
CIFAR-10	Minimum-time NODEs $\lambda = 0.1$	0.3606	81.74%	76%
	SNOpt (Liu et al., 2021)	0.49	77.82%	19%

differentiation, while SNOpt's dropped time is based on that using adjoint method, and the latter one is significantly slower than the one based on automatic differentiation (may reach 10 times). Although the network structure optimization scheme part of (Liu et al., 2021) also contains an update of the terminal time, it still remains within the purpose of the whole paper, which is to improve the convergence efficiency of the training with a small memory cost which is different with ours.

F.4. CNF

The experiments are run on NVIDIA Tesla V100 GPU. The network sturcture is based on the code from Neural ODEs paper. The ODE-solver is chosen as RK4 with step size 0.01, and integral time span for Vanilla Neural ODEs is set as $[0, 1]$. Optimizer is Adam with learning rate starting as 0.05. Training iterations are chosen as 1000, batch size is 512. λ used is here is 0.35.
Figure 12 demonstrates our experiments on generative models on two-circles dataset.

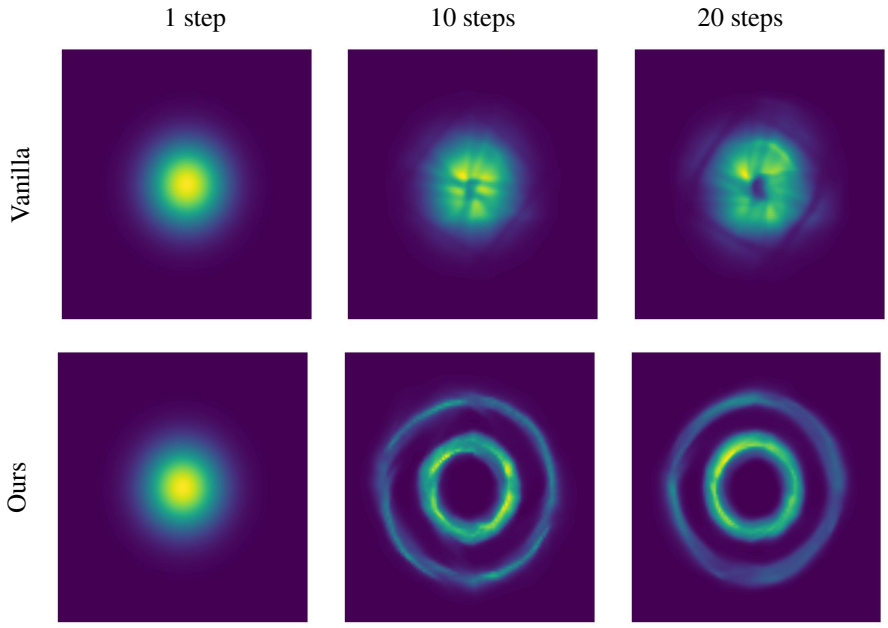


Figure 12. CNF Results on two-circles dataset