Towards Optimal Network Depths: Control-Inspired Acceleration of Training and Inference in Neural ODEs

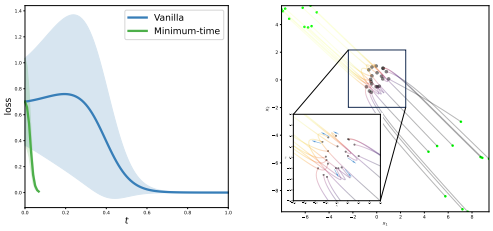
Keyan Miao **Konstantinos Gatsis**
Department of Engineering
University of Oxford
{keyan.miao, konstantinos.gatsis}@eng.ox.ac.uk

Abstract

Neural Ordinary Differential Equations (ODEs) offer potential for learning continuous dynamics, but their slow training and inference limit broader use. This paper proposes spatial and temporal optimization inspired by control theory. It seeks an optimal network depth to accelerate both training and inference while maintaining performance. Two approaches are presented: one treats training as a single-stage minimum-time optimal control problem, adjusting terminal time, and the other combines pre-training with Lyapunov method, followed by safe terminal time updates in a secondary stage. Experiments confirm the effectiveness of addressing Neural ODEs' speed limitations.

1 Introduction

Deep Neural Networks (DNNs) have transformed AI by excelling in complex tasks [1, 2]. However, increasing network depth can lead to issues like gradients problems [3], higher computation, and overfitting. Residual Networks (ResNets) [4] introduced skip connections to address these concerns, allowing for deeper models. Neural Ordinary Differential Equations (Neural ODEs) [5] build on this success by treating neural networks as dynamical systems governed by differential equations for adaptability [6, 7, 8]. Neural ODEs find applications in image classification, generative modeling, and time-series analysis [9, 10, 11]. In control systems, they can be used for trajectory planning and system identification [12, 13, 14]. While Neural ODEs hold promise, challenges persist in enhancing their stability, expressiveness, and robustness through ongoing research [15, 16, 17, 18, 19]. Nevertheless, their computational demands are a limiting factor, stemming from the slow ODE solver and uncertainties about network depth: excessive computational costs arise, and overly deep networks can lead to overfitting. For example, in the 2-D Concentric Annuli problem, Vanilla Neural ODEs achieve the desired performance by 0.6 time units, making subsequent computations redundant. Errors and trajectory deviations within 0.2 units indicate that it may take intricate routes as shown in Figure 1, incurring unnecessary computational expenses.



(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 Neural ODEs and Minimum-time Neural ODEs; (b) State trajectories of 20 examples by Vanilla Neural ODEs

Many efforts aim to reduce Neural ODEs' computational burden. Some simplify Neural ODE dynamics, for example, regularization terms encourage easier integration [20, 21, 22]. Temporally,

DLDE-III Workshop in the 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

studies optimize numerical solvers, like adding an extra neural network to capture the high-order components of the numerical solver [23], speeding up computations. STEER [24] introduces regularization via terminal time sampling. Overall, these methods mainly speed up inference, with limited gains in training. Research also explores DNNs depth bounds using turnpike properties from optimal control theory [25], relying on post-training analysis rather than prior estimation or taking into account the depth during training. We address the deep learning network depth challenge of Neural ODEs by adjusting the integral time span, focusing on proactive design rather than post-learning optimization, aiming for faster, more efficient training and inference. Neural ODEs' differentiable nature with respect to terminal time and convenient depth adjustment feature allow us to optimize network depth, which is an adaptivity not present in standard discrete networks. Spatially, we guide Neural ODEs towards more direct trajectories. Temporally, we aim to find the optimal depth to balance efficiency and performance. Experiments confirm an order of magnitude speed improvement over Vanilla Neural ODEs. In summary, our contributions include:

- We expand Neural ODEs by redefining the network depth challenge as an interval adjustment for ODE integration, with a focus on terminal time determination. We introduce methods from temporal and spatial perspectives for acceleration;
- Temporally, we convert the depth challenge into a minimum-time optimal control problem, aiming to find the shortest terminal time to optimize spatial and temporal performance;
- Spatially, we employ Lyapunov method during pre-training to learn dynamics with guaranteed convergence speeds then iteratively determine the minimum terminal time for optimizing temporal performance.

2 Methods

Incorporating insights from control theory, we address the structural challenges of Neural ODEs by optimizing network depth. We introduce two approaches: Minimum-time Neural ODEs, which transforms the challenge into a minimum-time optimal control problem and treats the terminal time as a learnable parameter during training; Convergence-rate-based Neural ODEs which is grounded in control theory, evaluating the entire trajectory of system dynamics based on convergence guarantees.

We reformulate the Neural ODEs system $\dot{z}(t) = f(t, z(t), \theta)$ and objective function $\ell := \Phi\left(z\left(t_f\right)\right) + \int_{t_0}^{t_f} L(z(t), \theta, t)\, dt$ as an optimal control problem, with considerations for the running cost. The latent state dynamics are described by a neural network f with parameters θ, taking $z(t)$ as input. Efficient gradient computation methods, such as automatic differentiation or adjoint sensitivity analysis inspired by maximum principle in control theory [26], are employed for parameter updates during training.

2.1 Approach 1: Minimum-time Neural ODEs

We propose the Minimum-time Neural ODEs framework with a learnable terminal time parameter t_f, that updates after each back-propagation and influences the subsequent forward pass. The gradient of ℓ with respect to t_f is computed by taking the derivative of an upper limit in a variable integral:

$$\frac{d\ell}{dt_f} = \frac{\partial \Phi}{\partial z\left(t_f\right)} \frac{dz(t_f)}{dt_f} + \frac{d}{dt_f} \int_{t_0}^{t_f} L(z(t), \theta, t) = \frac{\partial \Phi}{\partial z\left(t_f\right)} f\left(z\left(t_f\right)\right) + L\left(t_f\right) \tag{1}$$

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 Algorithm 1 shown in Appendix. In order to find the minimum-time optimal control strategy, the loss function is designed as

$$\ell := \Phi\left(z\left(t_f\right)\right) + \lambda \int_{t_0}^{t_f} 1 dt \tag{2}$$

which can be seen as $L = \lambda$. In this case, $\nabla_{t_f} \ell = \frac{\partial \Phi}{\partial z\left(t_f\right)} f\left(z\left(t_f\right)\right) + \lambda$, 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 updates and prevent overly abrupt changes, we propose to apply clipping to the updates as introduced in Appendix. Besides, it is possible that

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

Training neural network parameters and terminal time together can introduce initial training instability, influenced by multiplier choices. To address this, we begin with a large initial terminal time and optimize it backwards. This approach shifts the terminal state backwards with each t_f update, gradually encouraging faster and more direct spatial dynamics.

2.2 Approach 2: Convergence-rate-based Neural ODEs

Our previous approach emphasizes the terminal state and incorporats a time penalty but lacks guaranteed convergence, ensuring the attainment of the correct solution at a specific rate, such as exponential speed. In essence, guaranteed convergence implies that we can theoretically omit layers beyond a certain time. Our alternative method first pre-trains a dynamics model on a small time interval, emphasizing whole trajectory for guaranteed convergence from a spatial perspective. Subsequently, we determine the minimum terminal time to meet the terminal state requirements. When considering convergence properties of dynamic systems, the application of Lyapunov methods becomes indispensable which involve the construction of an energy function (Lyapunov function) to monitor the system's evolution, thereby assessing whether the system tends towards a stable state [27].

Lyapunov method Inspired by LyaNet[28], for a given training data pair and supervised loss Φ, the potential function V can be designed as $V_{\hat{y}}(\cdot) := \Phi(z(\cdot))$. For example, when standard cross-entropy loss is considered, then we consider using the truncated cross entropy loss defined as $\Phi(\cdot) := \max\{0, \Phi_{ce}\}$. Then a point-wise Lyapunov loss can be designed as

$$\mathcal{V} := \max\left\{0, \frac{\partial V_{\hat{y}}}{\partial z} f(t, z, \theta) + \kappa V_{\hat{y}}(z)\right\} \tag{3}$$

Equation (3) signifies the local violation of invariance condition. When $\mathcal{V} = 0$ holds for all data in the time interval, the inference dynamics exhibit exponential convergence towards a prediction that minimizes the loss. The Lyapunov loss is $\ell := \mathbb{E}\left[\int_{t_0}^{t_f} \mathcal{V} dt\right]$. If there exists a parameter θ^* of the dynamic system that satisfies $\ell(\theta^*) = 0$, the dynamic satisfies the exponential convergence speed with respect to the loss Φ. The Lyapunov method also allows us to manually adjust the convergence rate when necessary, accommodating cases where overly rapid dynamics are undesirable.

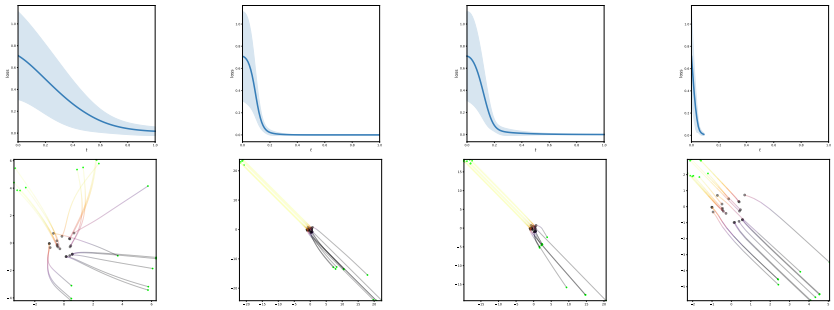
Learning procedure Notably, using Lyapunov loss ensures convergence rate but doesn't guarantee precise terminal state value, depending on t_f. Small t_f may not give the dynamics enough time to reach the desired state, while excessively large t_f results in unnecessary computations. To address this, we introduce a pre-training approach: we start with a small t_f indicating a shallow network for swift dynamic training, then gradually increase the network depth until the loss reaches a threshold (Algorithm 2 in Appendix). This approach is somewhat greedy, striving to find the shallowest network.

Comparison between Approach 1 and 2 Approach 1 is simple but risky in a single-stage process because the choice of λ may lead to performance issues as large λ may result in a shorter terminal time but an undesired final state. In contrast, Approach 2 offers convergence assurances and allows initial pre-training over a shorter time span, potentially expediting training. Both approaches have merits, combining them in a single stage hasn't shown significant advantages in preliminary experiments.

3 Experiments

In this section, we demonstrate the benefits of the proposed methods on a variety of machine learning tasks and compare the results with Vanilla Neural ODEs, LyaNet, STEER and TayNODE [20].

Concentric Annuli In Figure 2, we show results for the Concentric Annuli problem using our Lyapunov-based pre-training approach, including the loss curve and state trajectory. Pre-training uses a terminal time of 0.05 with a later update step of 0.01. The final trained t_f is 0.09, with a 15s-training. In contrast, Vanilla Neural ODEs require $123s$ to train, highlighting the significant improvement due to the shallow network depth during pre-training. Notably, when training over the interval $[0, 1]$ as LyaNet, we find that for $\kappa = 20$, faster dynamics are not learned compared to $\kappa = 10$.



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

Figure 2: 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) demonstrated the results obtained by Lyapunov method with pre-training on $[0, 0.05]$, $\kappa = 20$

However, shorter time spans enable faster dynamics. This is because sustaining rapid improvement after loss convergence over longer intervals, especially with larger κ values leading to more frequent Lyapunov function violations, becomes challenging. In contrast, shorter intervals, especially before loss minimization, allow for more effective training, highlighting the pre-training approach's benefits.

Table 1: 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

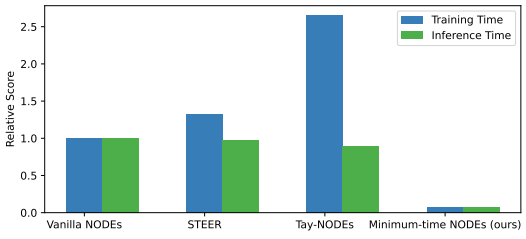


Figure 3: Comparison of different methods' time consumption for classification on MNIST dataset

Image classification We evaluated the performance of our approach on several benchmark datasets for image classification tasks, including MNIST [29], Fashion-MNIST [30] and CIFAR-10 [31]. The results are shown in Table 1 where NFE measures the number of evaluations by the ODE solver. And the comparison of our method and other popular approaches such as TayN-ODEs [20] on time consumption are shown in Figure 3. It is evident that our approach achieves an order of magnitude 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 experiments on the CIFAR-10, where a trade-off between accuracy and training/inference time by tuning the parameter λ is shown.

4 Discussions and Future work

In this work, we propose leveraging a control perspective, including techniques such as minimum-time control and Lyapunov methods, to learn more straightforward dynamics, determine a more suitable network depth, and consequently accelerate the training and inference of Neural ODEs while mitigating unnecessary computations. Future research directions hold intriguing possibilities, including integrating our approach with acceleration targeting ODE solvers to further enhance computational efficiency especially for the case that adaptive ODE solvers are used. Furthermore, we intend to delve into the exploration of time-variant Neural ODEs whose training aligns more closely with optimal control problem, thus enabling a more profound and elucidating investigation of the interconnectedness between turnpike theory and the depth of Neural ODEs.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

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

[7] Weinan Ee, Jiequn Han, and Qianxiao Li. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6, 12 2018.

[8] Qianxiao Li, Long Chen, Cheng Tai, et al. Maximum principle based algorithms for deep learning. *arXiv preprint arXiv:1710.09513*, 2017.

[9] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[10] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

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

[12] Yuxuan Liang, Kun Ouyang, Hanshu Yan, Yiwei Wang, Zekun Tong, and Roger Zimmermann. Modeling trajectories with neural ordinary differential equations. In *IJCAI*, pages 1498–1504, 2021.

[13] Keyan Miao and Konstantinos Gatsis. Learning robust state observers using neural odes. In *Learning for Dynamics and Control Conference*, pages 208–219. PMLR, 2023.

[14] Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. Neural networks with physics-informed architectures and constraints for dynamical systems modeling. In *Learning for Dynamics and Control Conference*, pages 263–277. PMLR, 2022.

[15] Qiyu Kang, Yang Song, Qinxu Ding, and Wee Peng Tay. Stable neural ode with lyapunov-stable equilibrium points for defending against adversarial attacks. *Advances in Neural Information Processing Systems*, 34:14925–14937, 2021.

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

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

[18] Hanshu Yan, Jiawei Du, Vincent YF Tan, and Jiashi Feng. On robustness of neural ordinary differential equations. *arXiv preprint arXiv:1910.05513*, 2019.

[19] Yifei Huang, Yaodong Yu, Hongyang Zhang, Yi Ma, and Yuan Yao. Adversarial robustness of stabilized neural ode might be from obfuscated gradients. In *Mathematical and Scientific Machine Learning*, pages 497–515. PMLR, 2022.

[20] Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. *Advances in Neural Information Processing Systems*, 33:4370–4380, 2020.

[21] 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.

[22] Avik Pal, Yingbo Ma, Viral Shah, and Christopher V Rackauckas. Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In *International Conference on Machine Learning*, pages 8325–8335. PMLR, 2021.

[23] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models. Jul 19, 2020.

[24] 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.

[25] Timm Faulwasser, Arne-Jens Hempel, and Stefan Streif. On the turnpike to design of deep neural nets: Explicit depth bounds. *arXiv preprint arXiv:2101.03000*, 2021.

[26] LS Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[27] Aaron D Ames, Kevin Galloway, Koushil Sreenath, and Jessy W Grizzle. Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics. *IEEE Transactions on Automatic Control*, 59(4):876–891, 2014.

[28] Ivan Dario Jimenez Rodriguez, Aaron Ames, and Yisong Yue. Lyanet: A lyapunov framework for training neural odes. In *International Conference on Machine Learning*, pages 18687–18703. PMLR, 2022.

[29] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[30] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

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

[32] Ricky T. Q. Chen. torchdiffeq, 2018.

Appendix

4.1 Neural ODEs

Definition 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*

$$\begin{cases} \dot{z}(t) = f(t, z(t), \theta) \\ z(t_0) = h_x(x) \\ \hat{y}(t) = h_y(z(t)) \end{cases} \qquad t \in \mathcal{S} \tag{4}$$

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

For Vanilla Neural ODEs, the loss function is designed as

$$\ell := \Phi\left(z\left(t_f\right)\right) \tag{5}$$

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

$$\begin{aligned} &\min_{\theta \in U} \quad \ell \\ s.t. \quad &\dot{z}(t) = f(t, z(t), \theta), t \in \mathcal{S} \\ &z(t_0) = h_x(x), \hat{y} = h_y\left(z\left(t_f\right)\right) \end{aligned} \tag{6}$$

which is a *Mayer* 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.

In our paper, the loss function can be designed as

$$\ell := \Phi\left(z\left(t_f\right)\right) + \int_{t_0}^{t_f} L(z(t), \theta, t)\, dt \tag{7}$$

Then problem (6) is a *Bolza* optimal control problem.

Proposition 1. *Consider the problem* (4)*,* (6) *and* (7)*, the gradient of loss ℓ with respect to parameter θ is*

$$\nabla_\theta \ell = \mu(t_0) \tag{8}$$

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

$$\begin{aligned} &\dot{z}(t) = f,\ z(t_0) = z_0 \\ &\dot{p}(t) = -p(t)\frac{\partial f}{\partial z} - \frac{\partial L}{\partial z},\ p\left(t_f\right) = \frac{\partial \Phi}{\partial z\left(t_f\right)} \\ &\dot{\mu}(t) = -p(t)\frac{\partial f}{\partial \theta} - \frac{\partial L}{\partial \theta},\ \mu\left(t_f\right) = \mathbb{0}_{n_\theta} \end{aligned} \tag{9}$$

4.2 Lyapunov Conditions for Convergence

For the ODE described in (4), 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:

$$\min_\theta \left[\left.\frac{\partial V}{\partial z}\right|_z f(t, z, \theta) + \kappa V(z) \right] \leq 0 \tag{10}$$

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

$$\left.\frac{\partial V}{\partial z}\right|_z f(t, z, \theta) + \kappa V(z) \leq 0 \tag{11}$$

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

$$V(z(t)) \leq V(z(t_0))\, e^{-\kappa t} \tag{12}$$

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.

Theorem 1 ([28])**.** *Consider the Lyapunov loss l 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:*

$$\Phi(z(t)) \leq \Phi(z(t_0))\, e^{-\kappa t} \tag{13}$$

5 Algorithms

The algorithms mentioned in the paper are shown as follows.

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**
  $\mathbf{z} \leftarrow ODESolver(f(t,z,\theta), z(t_0), t_0, t_f)$
  $\theta \leftarrow Optimizer(\nabla_\theta \ell, \theta)$ ▷ Update neural network parameters
  $t_f \leftarrow Optimizer(\nabla_{t_f} \ell, t_f)$ ▷ Update terminal time
end for

Algorithm 2 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**
  $\mathbf{z} \leftarrow ODESolver(f(t,z,\theta), z(t_0), t_0, t_f)$
  $\theta \leftarrow Optimizer(\nabla_\theta \ell, \theta)$ ▷ Update neural network parameters
end for
while $\Phi(t_f) > \varepsilon$ **do**
  $t_f \leftarrow t_f + \gamma$ ▷ Update terminal time
  $\mathbf{z} \leftarrow ODESolver(f(t,z,\theta), z(t_0), t_0, t_f)$
  $\theta \leftarrow Optimizer(\nabla_\theta \ell, \theta)$
end while

6 Experiments Details

6.1 Concentric Annuli

The Concentric Annuli is shown as Figure 4. The experiments are run on Apple M1 Pro chip. The

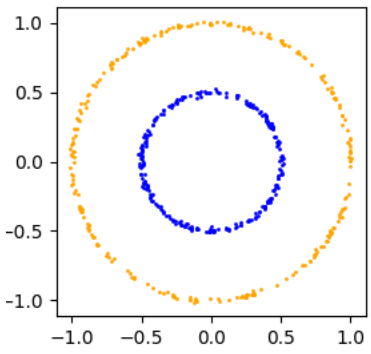
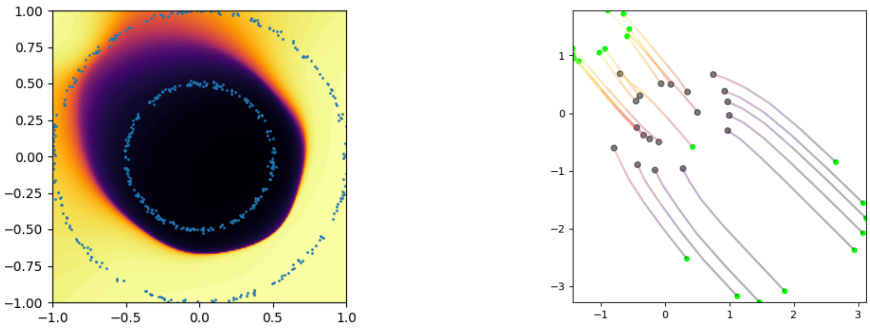


Figure 4: Concentric Annuli

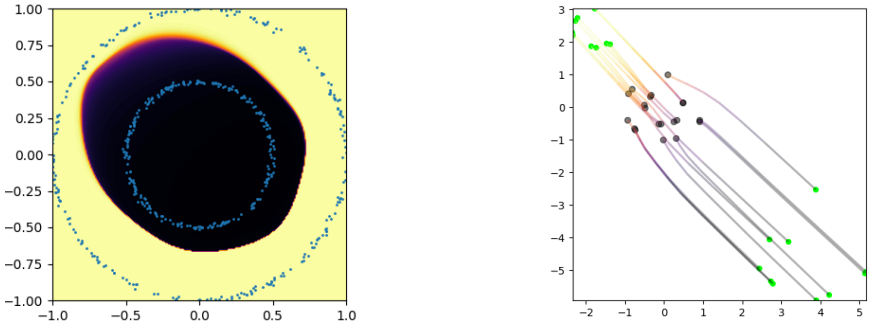
basic sturcture is chosen as Augmented Neural ODEs to address the problem of intersection of integral trajectory. We fit a three–layer network of hidden dimensions 32. ODE-solver is chosen as RK4 with step size 0.01. Optimizer is Adam with learning rate as 0.01, scheduler as ExponentialLR.

Especially, for Lyapunov method, which is pre-trained on $[0, 0.05]$, the pre-trained results are shown in Figure 5. Then, the threshold of loss is set as 0.01, the results after finishing the iteration to find the safe terminal time are shown in Figure 6.



(a) Boundary Decision (b) State Trajectories

Figure 5: Performance of pre-trained model ($[0, 0.05]$) on 20 correctly classified examples



(a) Boundary Decision (b) State Trajectories

Figure 6: Performance of final model ($[0, 0.09]$) on 20 correctly classified examples

6.2 Image Classification

The experiments are run on NVIDIA Tesla V100 GPU. The network sturcture is based on the code from [32]. 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.