LyaNet: A Lyapunov Framework for Training Neural ODEs

Ivan Dario Jimenez Rodriguez [1] **Aaron D. Ames** [1] **Yisong Yue** [1 2]

Abstract

We propose a method for training ordinary differential equations by using a control-theoretic Lyapunov condition for stability. Our approach, called LyaNet, is based on a novel Lyapunov loss formulation that encourages the inference dynamics to converge quickly to a correct prediction that minimizes the original supervised loss. Theoretically, we show that minimizing Lyapunov loss guarantees exponential convergence to a correct solution and enables a novel robustness guarantee. We also provide practical algorithms, including one that avoids the cost of backpropagating through a solver or using the adjoint method. Relative to standard Neural ODE training, we empirically find that LyaNet can offer improved prediction performance, faster convergence of inference dynamics, and improved adversarial robustness. Our code is available at `https://github.com/ivandariojr/LyapunovLearning`.

1. Introduction

The use of dynamical systems as a learnable function class has gained increasing attention, first sparked by the discovery of an alternative interpretation of ResNets as Ordinary Differential Equations (Haber & Ruthotto, 2017; E, 2017; Ruthotto & Haber, 2018; Lu et al., 2020), and further popularized with Neural ODEs (Chen et al., 2019). This fundamental insight holds the potential for a plethora of benefits including parameter efficiency, the ability to propagate probability distributions (Chen et al., 2019; Rozen et al., 2021; Song et al., 2020), accurate time-series modeling (Chen et al., 2021a), among others.

[1]Department of Computational and Mathematical Sciences, California Institute of Technology [2]Argo AI. Correspondence to: Ivan Dario Jimenez Rodriguez <ivan.jimenez@caltech.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

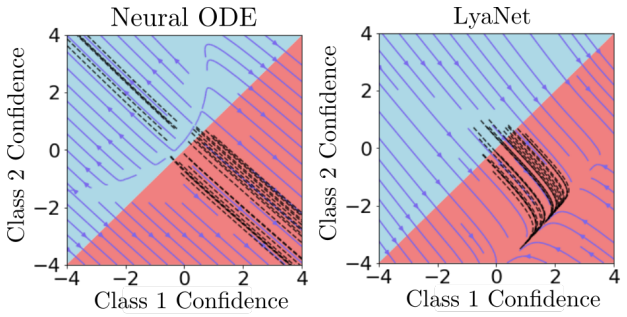

Figure 1: **Comparing learned dynamics** on a toy prediction task. The quiver lines show the dynamics (Equation (2)) of a 2-D state space, the dotted black lines show state trajectories from 100 initial conditions (Equations (1) and (2)), and the background coloring shows the class label from the output layer (Equation (3)), with red being the correct label. **Left:** a Neural ODE does not learn stable dynamics, where some dynamics point towards the incorrect prediction (blue region), and are especially sensitive around the initial conditions. **Right:** an identical model trained with LyaNet has much more stable dynamics that always smoothly point towards the correct prediction (red region).

Neural ODE Model Class. Inference or the "forward pass" uses a continuous-time ODE, parameterized by θ:

$$\eta(0) = \phi_\theta(x), \quad \text{(input layer)} \tag{1}$$

$$\frac{d\eta}{dt} = f_\theta(\eta, x, t), \quad \text{(continuum of hidden layers)} \tag{2}$$

$$\hat{y}(t) = \psi_\theta(\eta(t)). \quad \text{(output layer)} \tag{3}$$

Given input x, one makes a prediction (i.e., inference) by solving the ODE specified by Equations (1) & (2), and computing the output via Equation (3). Without loss of generality, we assume (2) evolves in the time interval $[0, 1]$, i.e., $\hat{y}(t)$ is typically computed at $t = 1$, although one could in principle compute $\hat{y}(t)$ at any $t \in [0, 1]$. We also assume that $x \in \mathbb{R}^n$, $\eta \in H \subset \mathbb{R}^k$, $\theta \in \Theta \subseteq \mathbb{R}^l$, and $y \in \mathbb{R}^m$. The state space H is assumed to be bounded and path connected.

Motivation. The standard learning approach is to differentiate through the ODE solution with techniques such as the adjoint method (Chen et al., 2019; Kidger et al., 2021), which typically does not impose desirable structure, such as stability, within the learned dynamics.[3] A such, the inference

[3]Some prior work has explored learning dynamical systems that are stable but not explicitly to a correct prediction (Ruthotto & Haber, 2018; Manek & Kolter, 2020; Bai et al., 2019; 2021), which can lead to a tension between stability and accuracy.

dynamics can exhibit unstable behavior that are difficult to integrate, leading to suboptimal prediction performance and solution fragility. There can also be high computational cost from generating roll-outs during training.

Consider Figure 1, which depicts the flow of two dynamical systems trained to solve a classification task (with almost identical nominal performance). In the left plot, the dynamics from standard Neural ODE training do not reliably converge toward the correct prediction (red region) throughout the state space. This instability leads to slower convergence and incorrect predictions given minor perturbations of the initial conditions (which is related to adversarial robustness). Our goal is to learn a stable dynamical system as shown in the right plot, which reliably and quickly converges to the correct prediction across a robust set of initial conditions.

Our Contribution. In this work, we study how to train dynamical systems using the control-theoretic principle of Lyapunov stability. Our contributions are as follows:

- We propose LyaNet, a framework for learning dynamical systems with stable inference dynamics. Our approach is based on a novel Lyapunov loss formulation that captures the degree of violation of the dynamics from satisfying the Lyapunov condition for exponential stability (Khalil, 2002; Ames et al., 2014).
- Theoretically, we show that minimizing Lyapunov loss will: 1) guarantee that the dynamical system satisfies a Lyapunov condition, and thus during inference or the "forward pass" has dynamics that exponentially converge, or stabilize, to a correct prediction; and 2) enable a novel adversarial robustness guarantee inspired by concepts in robust control.
- We develop practical algorithms for (approximately) optimizing Lyapunov loss, including a Monte Carlo approach that avoids the cost of backpropagating through an ODE solver. Our MC approach exploits the local-to-global structure of the Lyapunov condition where the exponential convergence guarantee arises from every state satisfying a local invariance.
- We evaluate LyaNet on a variety of computer vision datasets. Compared to standard Neural ODE training, we find that LyaNets enjoy competitive or superior prediction accuracy, and can offer improved adversarial robustness. We also show that the inference dynamics converge much faster, which implies early inference termination preserves prediction accuracy.

2. Preliminaries

2.1. Additional Details on Neural ODEs

To have unique solutions for all time, it is sufficient for an ODE to have an initial condition and a globally Lipschitz time derivative as shown in Equation (1) and Equation (2), respectively. We thus assume that the dynamics functions we learn are globally uniformly Lipschitz. This assumption is not overly onerous since most neural networks are compositions of globally Lipschitz preserving functions such as ReLUs, Convolutions, max or affine functions.

Equation (2) generalizes the original Neural ODE formulation by making f directly depend on x. This generalization is sometimes referred to as an Augmented Neural ODE (Dupont et al., 2019) or Data-controlled Neural ODE (Massaroli et al., 2020).

Connection to ResNet. One can think of the hidden layers in a ResNet architecture as a discrete-time Euler approximation to Equation (2) (Haber & Ruthotto, 2017; E, 2017), where each discretized hidden layer η_t is modeled as:

$$\eta_t = \eta_{t-\delta} + \delta f_\theta(\eta_{t-\delta}, x, t). \quad \text{(ResNet hidden layer)} \quad (4)$$

One obtains $\frac{\eta_t - \eta_{t-\delta}}{\delta} = f_\theta(\eta_{t-\delta}, x, t)$ by isolating f in Equation (4) . As $\delta \to 0$ we recover Equation (2) (assuming continuity of f). Although the original ResNet architecture does not have an explicit time step δ, each ResNet layer can be thought of as learning discrete-time dynamics with a fixed step size (i.e., an Euler approximation to the continuous-time dynamics in Equation (2)).

2.2. Supervised Learning as Inverse Control

We consider the standard supervised learning setup, where we are given a training set of input/output pairs, $(x, y) \sim D$, and the goal is to find a parameterization of our model that minimizes a supervised loss over the training data:

$$\underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{(x,y)\sim D} \mathcal{L}(\hat{y}_x(1), y), \quad (5)$$

where $\hat{y}_x(1)$ is shorthand for Equations (1) to (3).

As is typical in deep learning, the standard approach for training Neural ODEs is via backpropagation through Equation (5). The "end-to-end" training optimization problem is equivalent to the following finite-time optimal control problem (using just a single (x, y) for brevity):

$$\begin{aligned} \underset{\theta}{\operatorname{argmin}} \quad & \mathcal{L}(\hat{y}(1), y), \quad (6) \\ \text{s.t. } & \frac{\partial \eta}{\partial t} = f_\theta(\eta(t), x, t), \\ & \eta(0) = \phi_\theta(x), \\ & \hat{y}(1) = \psi_\theta(\eta(1)). \end{aligned}$$

Differentiating θ through (6) requires computing $\frac{\partial \eta}{\partial \theta}$, which was shown to be possible from rollouts of the dynamics using either backpropagation through the solver or the adjoint method (Chen et al., 2019; E, 2017).

Challenges. In Equation (6), there is no explicit penalty or regularization for intermediate states of the dynamical system. As such, even if (6) is optimized, the resulting dynamics $\eta(t)$ can exhibit problematic behavior. Indeed, one can observe such issues in Figure 1 where the Neural ODE dynamics are unstable, and in Figure 4 where the Neural ODE learns a fragile solution. Furthermore, training in this fashion can have high computational costs from generating roll-outs and the inherent numerical difficulty of integrating unstable dynamics. Our LyaNet approach addresses these limitations via a control-theoretic learning objective.

2.3. Lyapunov Conditions for Stability

In control theory, a stable dynamical system implies that all solutions in some region around an equilibrium point flow to that point. Lyapunov theory generalizes this concept by reasoning about convergence to states that minimize a potential function V. These potential functions are a special case of dynamic projections (Taylor et al., 2019).

Definition 1 (Dynamic Projection (Taylor et al., 2019)). *A continuously differentiable function $V : H \to \mathbb{R}$ is a dynamic projection if there exist constants $\underline{\sigma}, \overline{\sigma} > 0$ and an η^* in H satisfying:*

$$\forall \eta \in H : \ \underline{\sigma}\|\eta - \eta^*\| \leq V(\eta) \leq \overline{\sigma}\|\eta - \eta^*\|. \quad (7)$$

We can now define exponential stability with respect to V. As is common in many nonlinear convergence analyses, studying the behavior of a potential function is typically much easier than directly reasoning about the full dynamics.

Definition 2 (Exponential Stability). *We say that the ODE defined in Equations* (1) *and* (2) *is **exponentially stable** if there exists a positive definite dynamic projection potential function V and a constant $\kappa > 0$, such that all solution trajectories $\eta(t)$ of the ODE for all $t \in [0, 1]$ satisfy:*

$$V(\eta(t)) \leq V(\eta(0))e^{-\kappa t}. \quad (8)$$

Exponential stability implies that the dynamics converge exponentially fast to states with minimal V. Later, we will instantiate V using supervised loss.[4] Exponential stability is desirable because: 1) it guarantees fast convergence to desired states (as defined by V) after integrating for finite time (e.g., for $t \in [0, 1]$); and 2) it has implications for adversarial robustness, discussed later.

In order to guarantee exponential stability, we will impose additional structure, as described in the following theorem.

Theorem 1 (Exponentially Stabilizing Control Lyapunov Function (ES-CLF) Implies Exponential Stability (Ames et al. (2014))). *For the ODE in Equations* (1) *and* (2)*, a continuously differentiable dynamic projection V is an ES-CLF if there is a constant $\kappa > 0$ such that:*

$$\min_{\theta \in \Theta} \left[\frac{\partial V}{\partial \eta}\bigg|_{\eta}^{\top} f_\theta(\eta, x, t) + \kappa V(\eta) \right] \leq 0 \quad (9)$$

holds for all $\eta \in H$ and $t \in [0, 1]$. The existence of an ES-CLF implies that there is a $\theta \in \Theta$ that can achieve:

$$\frac{\partial V}{\partial \eta}\bigg|_{\eta}^{\top} f_\theta(\eta, x, t) + \kappa V(\eta) \leq 0, \quad (10)$$

and furthermore the ODE using θ is exponentially stable with respect to V (and constant κ).

In other words, we require the ODE to satisfy additional structure as specified by Equations (9) and (10), in order for the ODE to be exponentially stable w.r.t. V. In Section 3, we will develop a learning framework that encourages finding parameters θ that satisfy Equation (10).

Local-to-Global Contraction Structure. Equation (10) is essentially a contraction condition on V with respect to time (with κ controlling the rate of contraction), whereby the rate of decrease in V should be (at least) proportional to its current value. One can further interpret Equation (10) as a local invariance property: the condition only depends on the local state η rather than, say, the entire trajectory. In essence, Lyapunov theory exploits this local-to-global structure so that establishing a local contraction-based invariance implies global stability.

Control Lyapunov Functions. The potential function used in Theorem 1 is called a control Lyapunov function (CLF). The main difference between CLFs and classic Lyapunov functions is the additional minimization over θ. From the perspective of control theory, one can think of the parameters θ as a "controller" and Theorem 1 establishes conditions when there exists such a controller that can render the dynamics exponentially stable.[5] An exponentially stabilizing CLF (ES-CLF) is a CLF where κ in Equations (9) and (10) is strictly greater than zero.

Connection to Learnability. The minimization in Equation (9) can be interpreted as a statement about learnability or realizability. Satisfying Equation (9) equates to the family of parameters Θ realizing exponential stability with respect to V (i.e., some $\theta \in \Theta$ satisfies Equation (9)). If this potential function corresponds to a supervised loss, this means that supervised learning problem is learnable by this function class (i.e. some $\theta \in \Theta$ achieves low supervised loss). A natural way to prove that a V is an ES-CLF is to find (i.e., learn) a parameter θ that satisfies Equation (10). Note that stability is a much stronger condition than achieving low

[4] For instance, in Figure 1, we want $V \approx 0$ only within the red region. Our definition of V will also involve Equation (3).

[5] Conventional applications of Theorem 1 focus on designing controllers to stabilize a given physical system (Ames et al., 2014).

supervised loss, which implies that the set of stably fitting models is a subset of all models that fit the data.

Learning Exponentially Stable Systems. In this paper, we aim to shape a (highly overparameterized) system to satisfy a Lyapunov condition for stabilizing to predictions with minimal supervised loss. Prior work has explored learning dynamical systems that are stable but not explicitly to a correct prediction (Ruthotto & Haber, 2018; Manek & Kolter, 2020; Bai et al., 2019; 2021), which can lead to a tension between stability and accuracy. Compared to conventional work in Lyapunov analysis, our goal can be viewed as the dual of the more common goal of finding a Lyapunov function to associate with a pre-specified system.

3. LyaNet Framework

We now present LyaNet, our Lyapunov framework for training ODEs of the form specified by Equations (1) to (3). As alluded to in Section 2.3, our goal is to find parameters θ of the ODE to satisfy the Lyapunov exponential stability condition in Theorem 1 with respect to a potential function V. We develop the formulation in two steps:

1. Section 3.1: For a given supervised loss, we define an appropriate potential function V.
2. Section 3.2: For that V, we define the Lyapunov loss which captures the degree of violation from satisfying the contraction condition in Equation (10) that implies exponential stability.

Theoretically, we show that optimizing the Lyapunov loss implies the learned ODE exponentially stabilizes to predictions with minimal supervised loss (Theorem 2), which in turn implies a novel adversarial robustness guarantee (Theorem 3).[6] In other words, if we find a $\theta \in \Theta$ that achieves zero Lyapunov loss, then Equation (10) will be satisfied. We present practical learning algorithms in Section 4.

3.1. Potential Function for Supervised Loss

For a given input-output pair (x, y) and supervised loss $\mathcal{L}$, define the following potential function:

$$V_y(\cdot) := \mathcal{L}(\psi_\theta(\cdot), y), \tag{11}$$

where the input to V_y depends on x, and ψ_θ is the output layer from Equation (3). Typically, one would input the hidden state at some time t, $\eta_\theta(t)$, where the θ subscript denotes the parameters in Equations (1) and (2). An additional technical requirement is the continuous-differentiability of ψ_θ, which is satisfied by typical network architectures.

[6] Our results assume the function class is expressive enough to minimize the Lyapunov loss, which we expect from overparameterized dynamical systems.

Implications. Recall that the model's prediction is defined as $\hat{y}(t) = \psi_\theta(\eta(t))$. Therefore, V in Equation (11) is minimized when the dynamical system converges to states η with zero supervised loss i.e. $(V_y(\eta) = \mathcal{L}(\hat{y}, y)) = 0)$.[7] In the following, we further develop the formulation so that we can interpret V as an exponentially stabilizing control Lyapunov function, and thus invoke Theorem 1 to prove exponential stability of the inference dynamics to a loss-minimizing prediction.

Truncated Cross Entropy Loss. In this paper, we focus on the standard cross entropy loss that is widely used for classification. As a theoretical technical detail, our formulation and analysis use the truncated cross entropy loss defined as: $\mathcal{L}(\cdot) := \max\{0, \mathcal{L}_{ce}(\cdot) - \gamma\}$, where $\mathcal{L}_{ce}$ is cross entropy, and $\gamma > 0$ is a small constant. The main difference is that the truncated version attains $\mathcal{L} = 0$ at a finite point (whereas cross entropy only attains 0 at infinity), which simplifies the stability analysis. In practice, we can choose γ to be machine precision, and ignore it in the learning algorithms.

Dynamic Projections. Following the arguments in Section 2.3, the first step is showing that V in Equation (11) is a dynamic projection, which is analyzed in the following for truncated cross entropy. See Appendix A.2 for proof.

Lemma 1. *For the dynamical system described in Equations* (1) *to* (3)*, $V_y(\eta)$ in Equation* (11) *with $\mathcal{L}$ as truncated cross entropy loss is a dynamic projection.*

3.2. Lyapunov Loss

In order to use the potential function V from Section 3.1 to impose stability on ODE training, the remaining step is to satisfy the condition in Equation (10). To do so, we define the Lyapunov loss, starting with the point-wise version.

Definition 3 (Point-wise Lyapunov Loss). *For the dynamical system defined in Equations* (1) *to* (3)*, a single input-output pair (x, y), and dynamic projection $V_y : H \to R_{\geq 0}$ from Equation* (11)*, the point-wise Lyapunov loss is:*

$$\mathscr{V}(x, y, \eta, t) := \max\left\{0, \frac{\partial V_y}{\partial \eta}^\top f_\theta(\eta, x, t) + \kappa V_y(\eta)\right\}. \tag{12}$$

Note that the point-wise Lyapunov loss $\mathscr{V}$ is exactly the violation of the local invariance property in Equation (10). Intuitively, for an input-output (x, y), if $\mathscr{V}$ is zero for all $\eta \in H$ and $t \in [0, 1]$, then Equation (9) holds everywhere, and thus Theorem 1 implies that the inference dynamics converge exponentially to a loss minimizing prediction. Figure 2 provides a depiction of this intuition, and Theorem 2 formalizes it. The Lyapunov loss then applies the point-wise Lyapunov loss to all training examples and possible states.

[7] We later show in Figure 3 the convergence rate of inference, which we can exactly interpret as V for cross entropy loss.

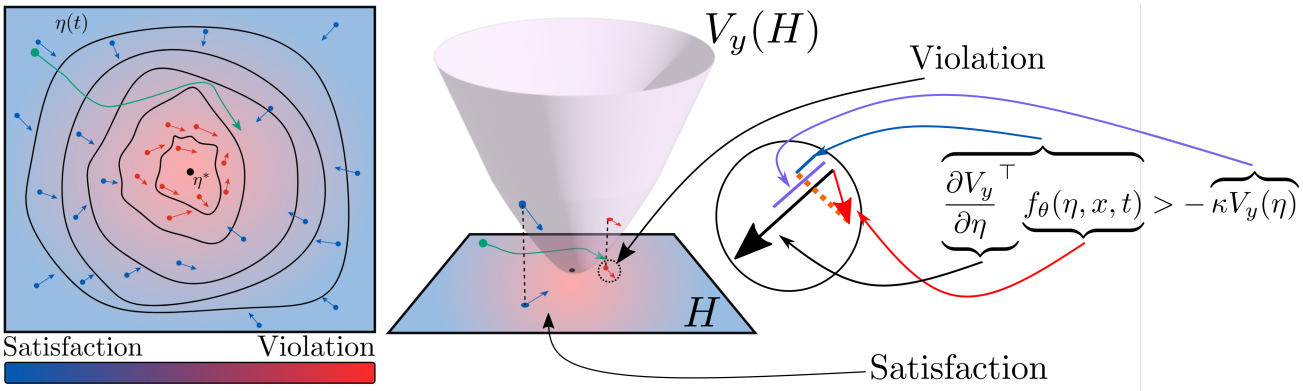

Figure 2: **Left:** a phase space plot a dynamical system (Equation (2)), along with level sets of the potential function V_y (Equation (11)) which in this example is minimized at η^*. The blue arrows represent flows that locally satisfy the Lyapunov exponential stability condition Equation (10)), while red arrows violate it. The background coloring indicates a local measure of this violation, as captured in the point-wise Lyapunov loss (Equation (12)). The green line denotes an example trajectory $\eta(t)$ (Equation (2)), which in this case does not stabilize to η^* which has minimal V_y. **Right:** depicting the geometric correspondence between the point-wise Lyapunov loss and the 1-D projected dynamics of V_y, where the inequality is a re-arrangement of terms. At any state η, if the point-wise Lyapunov loss is positive (i.e., the depicted inequality is satisfied) then the 1-D projected dynamics at $V_y(\eta)$ is not guaranteed to be exponentially stabilizing to 0. Conversely, choosing a θ to break the depicted inequality (and thus achieve zero Lyapunov loss) will guarantee exponential stability.

Definition 4 (Lyapunov Loss). *For the dynamical system defined in Equations* (1) *to* (3)*, a dataset of input-output pairs* $(x, y) \sim D$*, dynamic projection* $V_y : H \to R_{\geq 0}$*, and* $\mathscr{V}$ *from Equation* (12)*, the Lyapunov loss is:*

$$\mathscr{L}(\theta) \coloneqq \mathop{\mathbb{E}}_{(x,y)\sim D}\left[\int_0^1 \mathscr{V}(x, y, \eta_\theta(\tau), \tau) d\mu(\tau)\right]. \quad (13)$$

Theorem 2. *Consider the setting in Definition 4. If there exists a parameter* $\theta^* \in \Theta$ *of the dynamical system that attains* $\mathscr{L}(\theta^*) = 0$*, then for each* $(x, y) \sim D$*:*

1. *The potential function* $V_y(\cdot)$ *in Equation* (11) *is an exponentially stabilizing control Lyapunov function with* θ^* *satisfying Equation* (10)*.*

2. *For* $t \in [0, 1]$*, the inference dynamics satisfy the following convergence rate w.r.t. the supervised loss* $\mathcal{L}$*:*

$$\mathcal{L}(\hat{y}_\theta(t), y) \leq \mathcal{L}(\hat{y}_\theta(0), y) e^{-\kappa t}, \quad (14)$$

where $\hat{y}_\theta$ *is the output of Equation* (3) *with subscript* θ *denoting all parameters in Equations* (1) *to* (3)*.*

Theorem 2 is essentially a consistency result between minimizing Lyapunov loss and guaranteeing exponential stability on converging to states η that have low supervised loss on the training examples. See Appendix A.3 for proof. Future directions include analyzing when the Lyapunov loss is only approximately minimized, as well as generalization.

Choosing κ. The hyperparameter κ corresponds to the convergence rate of the loss dynamics. A larger κ enables faster convergence. However, as will come up in Section 4, larger κ can also make the learning problem more challenging, since the dynamics will have larger magnitude.

3.3. Adversarial Robustness

One attractive aspect of connecting learnable ODEs to control theory is the ability to leverage concepts in robust control. Fundamentally, robust control provides guarantees that a system will remain stable under perturbations, which is the same kind of guarantee studied in adversarially robust machine learning (Wong & Kolter, 2018; Raghunathan et al., 2018; Cohen et al., 2019; Robey et al., 2021). One interesting contrast with prior work on adversarially robust learning is that LyaNet does not explicitly optimize for adversarial robustness, but rather the robustness guarantee presented here directly follows from optimizing Lyapunov loss.

Definition 5 (δ-Stable Inference Dynamics for (x, y)). *A dynamical system defined in Equations* (1) *to* (3)*, with global Lipschitz constant* L *on* $\eta(t)$ *and associated dynamic projections* V_y *(Equation* (11)*) has* δ*-stable inference dynamics for example* (x, y) *if it satisfies the following conditions:*

1. ***Correct Classification:***

$$\operatorname*{argmax}_{i\in\{1\ldots m\}} \psi_\theta(\eta(1))_i = \operatorname*{argmax}_{i\in\{1\ldots m\}} y_i.$$

2. ***Exponential Stability:*** *$V_y(\eta(t))$ satisfies the condition in Equation* (10)*.*

3. ***δ-Final Loss:***

$$V_y(\eta(1)) \leq e^{-\kappa} V_y(\eta(0)) \leq \delta.$$

Definition 5 captures the general properties needed to analyze adversarial robustness. First, the learned model must correctly classify the unperturbed example (implied by minimizing V_y). Second, the learned dynamics must be exponentially stable w.r.t. V_y (implied by minimizing Lyapunov

loss). Third, at termination of inference ($t = 1$), the classification loss encoded in V_y is within an additive constant $\delta \geq 0$ from optimal (implied by exponential stability).

Theorem 3 (Adversarial Robustness of LyaNet). *Consider an ODE defined in Equations* (1) *to* (3) *that has δ-stable inference dynamics for input-output pair (x, y). Then the system given a perturbed input $x + \epsilon$ with $\|\epsilon\|_\infty \leq \bar{\epsilon}$ will produce a correct classification (Definition 5) so long as:*

$$\delta \leq \log(2) - \frac{L\bar{\epsilon}}{\kappa}\left(1 - e^{-\kappa}\right). \tag{15}$$

See Appendix A.4 for proof. Essentially, if a system is exponentially stable, then under perturbation, the system will be exponentially stable to a relaxed set of states (the radius of the relaxation is proportional to the perturbation magnitude). So long as that relaxation is contained in the part of the state space that outputs the correct classification, the final prediction is also adversarially robust. Another consequence of Theorem 3 is that the guarantee is stronger the more accurate the learned model is, since δ is exactly the nominal cross entropy loss of the prediction.

4. Learning Algorithms

We now present two algorithms for (approximately) optimizing Lyapunov loss (Equation (13)). The first approach is based on Monte Carlo sampling, and is suitable when the dimension of η is small-to-moderate (e.g., tens to hundreds). The second approach is based on discretized path integrals, and is suitable when the dimension of η is very large (e.g., hundreds of thousands). The main benefit of the MC approach is that it is easily parallelized (since it does not require solving an ODE during training), but may require too many samples to be practical in high dimensions.

The difficulty in optimizing Lyapunov loss is that the distributions for η and t are coupled, due to the inner integral in Equation (13). If one can effectively sample from this joint (η, t) distribution, then one can minimize the (expected) Lyapunov loss by optimizing the parameters θ w.r.t. the point-wise Lyapunov loss (Equation (12)) at those samples.

Restriction on Initial State. To simplify algorithm design, we restrict to learning ODEs that always initialize at $\eta_0 = 0$, i.e., Equation (1) is a constant function. The trained ODEs still perform well in practice.

Monte Carlo Training (Algorithm 1). The key idea is to sample η and t independently from measures μ_H and $\mu_{[0,1]}$, which is efficient so long as sampling from these measures is efficient. In practice, we choose uniform for $\mu_{[0,1]}$ and discuss μ_H next. The resulting (implicit) learning objective approximates the Lyapunov loss (Appendix B) so that zero learning objective implies the Lyapunov Loss is zero.

Algorithm 1 Monte Carlo LyaNet Training

hyperparameters:
  $\mu_H, \mu_{[0,1]}$ ▷ Measures on η and t
  Γ ▷ Number of MC samples
  α, M ▷ Learning rate and max iterations
initialize θ ▷ Any standard NN initialization
for $i = 1 : M$ **do**
  $(x, y) \sim \mathcal{D}$ ▷ Sample training data
  $\{(\eta_j, t_j)\}_{j=1}^{\Gamma} \sim \mu_H \times \mu_{[0,1]}$ ▷ Sample Γ (η, t) pairs
  $\theta \leftarrow \theta - \alpha \nabla_\theta \sum_j \mathscr{V}(x, y, \eta_j, t_j)$ ▷ From Equation (12)
end for
return θ

Algorithm 2 Path Integral LyaNet Training

hyperparameters:
  Γ ▷ Time discretization resolution
  α, M ▷ Learning rate and max iterations
initialize θ ▷ Any standard NN initialization
define $t_0, t_1, \ldots, t_\Gamma$ ▷ Evenly spaced from 0 to 1
for $i = 1 : M$ **do**
  $(x, y) \sim \mathcal{D}$ ▷ Sample training data
  $\eta(t_j) \leftarrow \int_{t_{j-1}}^{t_j} f(\eta(\tau), x, \tau) d\tau + \eta(t_{j-1}) \quad \forall_{1 \leq j \leq \Gamma}$ ▷ Generate path integral discretization
  $\theta \leftarrow \theta - \alpha \nabla_\theta \sum_j \mathtt{V}(x, y, \eta(t_j), t_j)$ ▷ See Equation (16)
end for
return θ

One subtlety in defining μ_H is choosing the support set H of the state space. Earlier, H had been used solely as a theoretical object, but now must be made explicit. The key requirement is that H covers the actual states visited by the ODE during inference. Since the ODE function class is globally Lipschitz, we can bound how far states can evolve from the origin. In practice, we choose μ_H to be a either a uniform on a hypercube or on a hypersphere biased towards the origin, with radius of H being the distance from the initial condition to states corresponding to the corners of the hypercube that achieve losses of $e^{\pm\kappa}$ (Appendix B).

Due to the parallel nature of random sampling, this method often outperforms conventional Neural ODE training w.r.t. compute time, since it avoids the sequential integration steps needed for standard backpropagation.[8] For instance, we found that even for 100-dimensional state spaces, only $\Gamma = 500$ samples were required to reliably approximate the integral, which can be very efficient with modern GPUs.

Path Integral Training (Algorithm 2). For systems with extremely high dimensional state spaces (e.g., hundreds of thousands), or for systems that are better approximated in discrete time, we consider an alternative approach based on collecting roll-outs of the original dynamics.

The basic idea is to approximate the inner integral in the

[8] The computational gains remain even after significantly reducing the precision of the ODE solver.

Lyapunov loss using Euler integration, which uniformly discretizes the integration into Γ segments. We can also define a discrete-time version of the point-wise Lyapunov loss:[9]

$$\mathtt{V}(x, y, \eta, t_j, t_{j-1}) = \max\left\{0, V_y(\eta(t_i)) + (\kappa - 1)V_y(\eta(t_{j-1}))\right\}. \tag{16}$$

The resulting discrete-time Lyapunov loss is then:

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\sum_{i=1}^{\Gamma} \mathtt{V}(x, y, \eta, t_i, t_{i-1})\right]. \tag{17}$$

Optimizing Equation (17) has the advantage of being more computationally efficient than Monte Carlo training in high-dimensional systems such as ResNet-inspired Continuous-in-Depth architectures (Queiruga et al., 2020). This is due to the Monte Carlo method placing very little density on the states actually traversed by the ODE, thus requiring many samples to learn reliably. Notice, however, that the underlying inference dynamics remain continuous, we simply apply a discrete-time condition to it. Furthermore, the integral in Algorithm 2 can be replaced with discrete-time dynamics for application in discrete systems.

5. Experiments

In our experiments we address the following questions:

- How well do LyaNets perform compared to their baseline counterparts, including under adversarial attacks?
- How quickly can LyaNet inference converge (i.e., does exponential stability hold), thus allowing early inference termination?
- How does the decision boundary of LyaNet compare with that of other methods?
- How does the computational cost of training LyaNets compare with regular backpropagation?
- How does varying the paremter κ affect model generalization?

5.1. Experiment Setup

We compare with three model classes:

- ResNet-18 (He et al., 2015), which can be viewed as an Euler integrated dynamical system.
- A Continuous-in-Depth Network, as presented by Queiruga et al. (2020), trained with both the adjoint method (labeled Continuous) and Path-Integral LyaNet (labeled Continuous LyaNet).

[9]There also exist analogous results to Theorem 1 for discrete-time systems (Zhang et al., 2009), a thorough treatment of which is beyond the scope of this work.

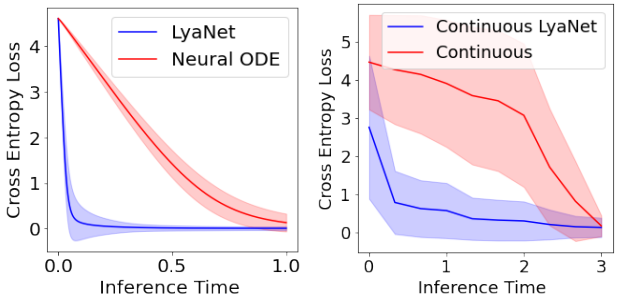

Figure 3: Plotting inference time vs prediction loss (if we stopped inference early and made a classification) on 512 correctly classified test examples from CIFAR-100. We see across two model classes that the LyaNet inference dynamics converge much faster.

- A Data-Controlled Neural ODE where ResNet-18 is used to learn a parameterization of the dynamics trained with both the adjoint method (labeled Neural ODE) and Monte Carlo LyaNet (labeled LyaNet). In both cases the number of hidden dimensions correspond to the number of classes.

We evaluate primarily on three computer vision datasets: FashionMNIST, CIFAR-10 and CIFAR-100. More details on the training can be found in Appendix C

5.2. Benchmark Experiments

Table 1 shows the primary quantitative results for both standard test prediction error as well as prediction error under PGD bounded adversarial attacks. For standard or nominal test error, we see that our LyaNet variants achieve competitive or superior performance compared to their counterparts trained via direct backpropagation.

For our robustness results, LyaNet trained using the Monte Carlo methods exhibits improved robustness for CIFAR-10 and FashionMNIST, despite not being explicitly trained to handle adversarial attacks. CIFAR-100 has a significantly larger hidden state dimensionality, and this larger hidden state also hurt nominal performance. These results are consistent with Theorem 3, which requires a low nominal error (δ) relative to perturbation size (ϵ) to guarantee adversarial robustness.

We finally note that the Path Integral method used to train the continuous-in-depth network was not able to improve adversarial robustness. This may be due to the coarseness of the path integration, which breaks the continuous exponential stability condition needed to guarantee robustness in Theorem 3 (since we only proved exponential stability for continuous-time integration and not discrete-time).

These results suggest that the LyaNet framework offers promising potential to improve the reliability of Neural ODE training, and more gains may be possible with improved training algorithms for optimizing Lyapunov loss.

Dataset	Method	Nominal	$\ell_\infty(\epsilon = 8/255)$	$\ell_2(\epsilon = 127/255)$
FashionMNIST	ResNet18	8.17	84.55	33.69
	Continuous	7.67	91.85	45.87
	Continuous LyaNet	**7.18**	**93.96**	**46.54**
	Neural ODE	8.66	77.9	29.85
	LyaNet	**7.9**	**46.33**	**25.14**
CIFAR10	ResNet18	19.73	76.44	40.06
	Continuous	13.12	88.79	41.53
	Continuous LyaNet	**12.99**	**91.03**	**39.82**
	Neural ODE	18.5	75.7	38.91
	LyaNet	**17.22**	**58.97**	**38.25**
CIFAR100	ResNet18	41.07	91.08	83.72
	Continuous	35.96	97.12	80.34
	Continuous LyaNet	**35.81**	**97.77**	**78.68**
	Neural ODE	41.57	92.77	81.7
	LyaNet	**41.07**	**91.08**	**81.91**

Table 1: Test error percentage comparison for networks trained with our approach and the equivalent network trained with other methods. Continuous LyaNet uses the Continuous Net architecture, and LyaNet uses the Neural ODE architecture. Adversarial robustness results are trained with PGD. LyaNet trained models have similar performance to their counterparts trained by back-propagating through solutions. We note that LyaNet trained with the Monte Carlo approach has a significant robustness enhancement in low-dimensional datasets.

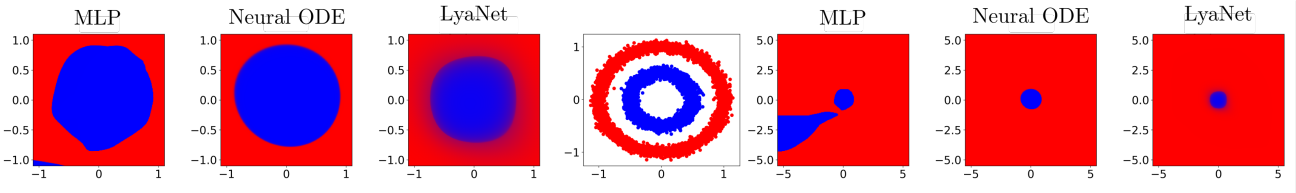

Figure 4: **Comparing softmax outputs** of learned dynamical systems. **Center:** depicting the training data with two classes (red and blue) in a 2-D input space. **Left 3 Plots:** comparing the softmax outputs, where we see that MPL and Neural ODE have sharp transitions between the two classes, while LyaNet has much smoother transitions. **Right 3 Plots:** showing a zoomed out version of left 3 plots.

5.3. Inference Convergence Speed

Early inference termination refers to stopping the dynamics before $t = 1$, and output $\hat{y}(t)$. Intuitively, one might expect LyaNets to perform well under early inference termination.

Figure 3 shows the results for the CIFAR-100 dataset. We see that both the continuous-in-depth models and the data-controlled models show significantly better performance during early termination when trained with LyaNet compared to standard backpropagation. These results also demonstrate that the internal dynamics of the learned models are fundamentally different when trained with LyaNet versus standard backpropagation.

5.4. Inspecting Decision Boundaries

To provide some intuition on the the type of decision boundaries being learned, we run an experiment on a low-dimensional dataset as shown in Figure 4. All models were trained with the same number of data-points and tested on a uniform grid. The Neural ODE, MLP and (Monte Carlo) LyaNet have the same number of parameters and the models are trained to similar accuracy. We note that because of the low-dimensional setting, this is a best-case scenario for the LyaNet approach. The MLP exhibits the fragility we expect from conventional neural networks. Although the Neural ODE captures the data well, its decision boundary is very sharp and biased towards the outer circle. Meanwhile, the LyaNet is able to capture the uncertainty between the inner and outer circles, and this uncertain boundary can be interpreted as related to adversarial robustness.

5.5. Computation Time

For all models we compared training times for one epoch using the CIFAR-100 dataset. As you can see in Table 2, training with LyaNet can cause a marginal increase in training time for both models. This is expected since we have to evaluate the loss at all the evaluation times for the path integral method and over a large number of samples for the Monte Carlo approach. Still, the difference in training time is about 1 minute per epoch for the Monte Carlo approach and less than 10 seconds for Path Integral approach.

Model	Batch Size 64	Batch Size 128
ResNet18	44.29	23.23
Neural ODE	183.82	104.95
LyaNet	151.08	170.64
Continuous	109.41	57.33
Continuous LyaNet	112.98	62.37

Table 2: Amount of time in seconds needed to complete an epoch of training for each method on the CIFAR100 dataset.

5.6. κ Ablation Study

The parameter κ corresponds to the exponential rate of convergence in Equation (9). We trained models with values of κ different to those used in Table 1 to observe how changing κ affects the performance of the model. Recall that in the continuous setting (LyaNet) $\kappa > 0$ is the only constraint on the rate of convergence while in the discrete-time setting (Continuous LyaNet) $0 < \kappa < 1$ must be satisfied.

As you can see in Table 3, the LyaNet experiments have a

κ	LyaNet	κ	Continuous LyaNet
0.1	43.18	0.9	37.12
1.0	39.83	0.09	39.85
10.0	40.46	0.009	47.73

Table 3: Test error percentage for models trained with different exponential convergence parameter κ on the CIFAR-100 dataset.

minimum at $\kappa = 1.0$. Recall that the Continuous LyaNet architecture has continuous dynamics but we apply the discrete-time Lyapunov condition. In this case, progressively larger κ appear to improve the models performance. For reference, in Table 1 LyaNet is using $\kappa = 5$ and Continuous LyaNet is using $\kappa = 0.999955$.

In the continuous case, κ is a form of regularization for the dynamics of the model. As you can see in the LyaNet experiment, increased stability can come at the cost of expressiveness. This trade-off between expressiveness and stability is expected given Theorem 3(which that implies a larger κ makes the model more robust) and the fact that robustness decreases model expressiveness.

6. Related Work

Prior Work in Learning Dynamical Systems. Most prior work has focused on using the adjoint method to infer dynamics (Chen et al., 2019; Antonelo et al., 2021). The proposal by E (2017) discusses properties like controllability but ultimately frames inference as a standard supervised learning, or optimal control, problem. Although optimal control is a powerful framework, for ODE training it may be challenging to avoid fragile solutions and attain strong stability guarantees.

Prior work have also studied how to impose stability in various forms, including guaranteeing stability around an equilibrium point (Manek & Kolter, 2020; Bai et al., 2021; 2019; Schlaginhaufen et al., 2021), and designing architectures that stable by construction Haber & Ruthotto (2017). Drawbacks of these approaches are that they do not guarantee exponential stability everywhere in the state space, or they can create a tension between accuracy and stability by using less expressive models.

Prior Work in Learning and Control. There have been many other intersection points between learning and control theory, although none have studied using control theory to influence the inference procedure of a learned model as we do. Wilson et al. (2018) use Lyapunov theory to analyze the dynamical system implicit in the momentum updates of stochastic gradient descent, Amos et al. (2018); Williams et al. (2017) differentiate through controllers such as MPC, and Peng et al. (2020) learn control policies directly for a real system. Richards et al. (2018b) safely learn physical dynamics by taking into account Lyapunov-like conditons during training. Chang et al. (2019) use an adversarial approach with a similar loss condition to learn controllers using Lyapunov functions. Cheng et al. (2019) uses a control prior to regularize reinforcement learning, in some cases with robustness properties of the learned policy. Chow et al. (2018) uses a Lyapunov condition to improve safety conditions in reinforcement learning. Dean et al. (2020) study the sample complexity of learning controllers such as for LQR. Rosolia & Borrelli (2017) study how to learn cost-to-go value functions for MPC under safety constraints. Learning Lyapunov functions for controlling hybrid dynamical systems Chen et al. (2021b). Learning Lyapunov stability certificate for a black-box neural network policy Richards et al. (2018a). Neural networks have also been used to learn contraction metrics (a similar concept to Lyapunov) for the control and estimation of stochastic systems Tsukamoto et al. (2020).

7. Discussion & Future Work

This work studies a new connection between learning and control in the context of training network architectures built using dynamical systems. In contrast to studying physical systems where the state description has a real physical interpretation that cannot be modified, the internal state of Neural ODEs is highly controllable (or modifiable) due to overparameterization. As a consequence, the "control" problem here can rely on relatively simple control theoretic tools due being able to design (or train) the entire system, rather controlling only a part of the system to force the whole system to be stable (e.g., controlling the motor of a seqway so that it stands upright). The tight integration of the Lyapunov condition via optimizing Lyapunov loss also automatically confers many benefits such as adversarial robustness.

There are many interesting directions for future work. For instance, our learning algorithms are the natural starting points for optimizing Lyapunov loss, but can be significantly improved. It may also be interesting to combine LyaNet with other approaches for improving adversarial robustness. One can also study other concepts in control theory, such as barrier conditions (Ames et al., 2016) instead of Lyapunov conditions, to guarantee set invariance rather than stability. Finally, the fast inference dynamics of LyaNet may have applicability in time-critical inference settings.

Acknowledgements

We would like to thank Andrew Taylor, Jeremy Bernstein, Yujia Huang and Matt Levine for the insightful discussions. This project was funded in part by AeroVironment.

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.

Ames, A. D., Xu, X., Grizzle, J. W., and Tabuada, P. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2016.

Amos, B., Rodriguez, I. D. J., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable MPC for end-to-end planning and control. *CoRR*, 2018. URL http://arxiv.org/abs/1810.13400.

Antonelo, E. A., Camponogara, E., Seman, L. O., de Souza, E. R., Jordanou, J. P., and Hubner, J. F. Physics-informed neural nets for control of dynamical systems, 2021.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models, 2019.

Bai, S., Koltun, V., and Kolter, J. Z. Stabilizing equilibrium models by jacobian regularization, 2021.

Chang, Y.-C., Roohi, N., and Gao, S. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *Neural Information Processing Systems (NeurIPS)*, 2019. URL http://arxiv.org/abs/1806.07366.

Chen, R. T. Q., Amos, B., and Nickel, M. Learning neural event functions for ordinary differential equations. In *International Conference on Learning Representations (ICLR)*, 2021a. URL http://arxiv.org/abs/2011.03902.

Chen, S., Fazlyab, M., Morari, M., Pappas, G. J., and Preciado, V. M. Learning lyapunov functions for hybrid systems. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pp. 1–11, 2021b.

Cheng, R., Verma, A., Orosz, G., Chaudhuri, S., Yue, Y., and Burdick, J. Control regularization for reduced variance reinforcement learning. In *International Conference on Machine Learning*, pp. 1141–1150. PMLR, 2019.

Chow, Y., Nachum, O., Duéñez-Guzmán, E. A., and Ghavamzadeh, M. A lyapunov-based approach to safe reinforcement learning. In *NeurIPS*, 2018.

Cohen, J., Rosenfeld, E., and Kolter, Z. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pp. 1310–1320. PMLR, 2019.

Dean, S., Mania, H., Matni, N., Recht, B., and Tu, S. On the sample complexity of the linear quadratic regulator. *Foundations of Computational Mathematics*, 20(4):633–679, 2020.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes, 2019.

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

Haber, E. and Ruthotto, L. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, dec 2017. doi: 10.1088/1361-6420/aa9a90. URL https://doi.org/10.1088/1361-6420/aa9a90.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Khalil, H. Nonlinear systems third edition. *Patience Hall*, 2002.

Kidger, P., Chen, R. T. Q., and Lyons, T. "hey, that's not an ode": Faster ode adjoints via seminorms, 2021.

Kim, H. Torchattacks: A pytorch repository for adversarial attacks. *arXiv preprint arXiv:2010.01950*, 2020.

Liu, Y., Bernstein, J., Meister, M., and Yue, Y. Learning by turning: Neural architecture aware optimisation, 2021.

Lu, Y., Zhong, A., Li, Q., and Dong, B. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations, 2020.

Manek, G. and Kolter, J. Z. Learning stable deep dynamics models, 2020.

Massaroli, S., Poli, M., Park, J., Yamashita, A., and Asama, H. Dissecting neural odes. *arXiv preprint arXiv:2002.08071*, 2020.

Muller, M. E. A note on a method for generating points uniformly on n-dimensional spheres. *Commun. ACM*, 2(4):19–20, April 1959. ISSN 0001-0782. doi: 10.1145/377939.377946. URL https://doi.org/10.1145/377939.377946.

Peng, X. B., Coumans, E., Zhang, T., Lee, T.-W., Tan, J., and Levine, S. Learning agile robotic locomotion skills by imitating animals, 2020.

Queiruga, A. F., Erichson, N. B., Taylor, D., and Mahoney, M. W. Continuous-in-depth neural networks, 2020.

Raghunathan, A., Steinhardt, J., and Liang, P. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

Richards, S. M., Berkenkamp, F., and Krause, A. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Conference on Robot Learning*, pp. 466–476. PMLR, 2018a.

Richards, S. M., Berkenkamp, F., and Krause, A. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems, 2018b.

Robey, A., Chamon, L., Pappas, G., Hassani, H., and Ribeiro, A. Adversarial robustness with semi-infinite constrained learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Rosolia, U. and Borrelli, F. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, 2017.

Rozen, N., Grover, A., Nickel, M., and Lipman, Y. Moser flow: Divergence-based generative modeling on manifolds. *Advances in Neural Information Processing Systems*, 34, 2021.

Ruthotto, L. and Haber, E. Deep neural networks motivated by partial differential equations, 2018.

Schlaginhaufen, A., Wenk, P., Krause, A., and Dorfler, F. Learning stable deep dynamics models for partially observed or delayed dynamical systems. *Advances in Neural Information Processing Systems*, 34, 2021.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2020.

Sontag, E. D. and Wang, Y. On characterizations of the input-to-state stability property. *Systems & Control Letters*, 24(5):351–359, 1995.

Taylor, A. J., Dorobantu, V. D., Krishnamoorthy, M., Le, H. M., Yue, Y., and Ames, A. D. A control lyapunov perspective on episodic learning via projection to state stability. *2019 IEEE 58th Conference on Decision and Control (CDC)*, Dec 2019. doi: 10.1109/cdc40024.2019.9029226. URL http://dx.doi.org/10.1109/CDC40024.2019.9029226.

Tsukamoto, H., Chung, S.-J., and Slotine, J.-J. E. Neural stochastic contraction metrics for learning-based control and estimation. *IEEE Control Systems Letters*, 5(5):1825–1830, 2020.

Williams, G., Wagener, N., Goldfain, B., Drews, P., Rehg, J. M., Boots, B., and Theodorou, E. A. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721, 2017. doi: 10.1109/ICRA.2017.7989202.

Wilson, A. C., Recht, B., and Jordan, M. I. A lyapunov analysis of momentum methods in optimization, 2018.

Wong, E. and Kolter, Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5286–5295. PMLR, 2018.

Zhang, W., Abate, A., Hu, J., and Vitus, M. P. Exponential stabilization of discrete-time switched linear systems. *Automatica*, 45(11):2526–2536, 2009.

A. Proofs

A.1. Auxiliary Facts

Definition 6 (Dynamic Projection). *A continuously differentiable function $\psi : \mathbb{R}^k \to \mathbb{R}^m$ is a dynamic projection if there exist $\underline{\sigma}, \overline{\sigma} \in \mathbb{R}_{>0}$ so that for all $\eta \in H \subseteq \mathbb{R}^k$:*

$$\underline{\sigma}\|\eta\| \leq \|\psi(\eta)\| \leq \overline{\sigma}\|\eta\| \tag{18}$$

Lemma 2 (Composition of Dynamic Projections). *The composition of dynamic projections is a dynamic projection.*

Proof. Let ψ and V both be dynamic projections with corresponding constants $\underline{\sigma}^{\psi}, \underline{\sigma}^{V}, \overline{\sigma}^{\psi}, \overline{\sigma}^{V}$. Then $V \circ \psi$ satisfies the following inequalities:

$$\underline{\sigma}^{V}\|\psi(\eta(t))\| \leq \|V(\psi(\eta(t)))\| \leq \overline{\sigma}^{V}\|\psi(\eta(t))\| \tag{19}$$

$$\underline{\sigma}^{\psi}\underline{\sigma}^{V}\|\eta(t)\| \leq \|V(\psi(\eta(t)))\| \leq \overline{\sigma}^{V}\overline{\sigma}^{\psi}\|\eta(t)\| \tag{20}$$

$$\tag{21}$$

Therefore $V \circ \psi$ is a dynamic projection with constants $\underline{\sigma}^{\psi}\underline{\sigma}^{V}$ and $\overline{\sigma}^{V}\overline{\sigma}^{\psi}$.

□

A.2. Proof of Lemma 1

Proof. The cross entropy loss function is the composition of the negative-log-likelihood loss function, softmax operation and the output coordinates ψ_θ. Using Lemma 2 all we need to prove is that for the set of admissible times where $\eta(t)$ evolves, the negative log-likelihood and the softmax operation are dynamic projections. Note that ψ_θ is assumed to be a dynamic projection since it is usually a globally Lipschitz neural network.

Since we assume f is globally Lipschitz we know $\eta(t)$ exists and is unique for all time. Since t is in the compact set $[0, 1]$, we can conclude that $\mathcal{H} = \{\eta(t) : t \in [0, 1]\}$ is also compact. We define $\eta^* = \operatorname{argmin}_{h\in\mathcal{H}} \mathcal{L}(\psi_\theta(\eta), y)$, in other words the closest point in $\mathcal{H}$ to the correct classification under the cross-entropy loss function. Since $\mathcal{H}$ is compact we know that supremums and infimums with respect to the objective $\|\eta - \eta^*\|$ are attained within the set. Let $\overline{h} = \operatorname{argmax}_{\eta\in\mathcal{H}}\|\eta - \eta^*\|$ and $\underline{h} = \operatorname{argmin}_{\eta\in\mathcal{H}}\|\eta - \eta^*\|$.

Therefore, we can conclude that the Softmax operation is a dynamic projection with the following bounds:

$$\frac{\sigma^{\psi}}{\|\overline{h}\|}\|\eta(t) - \eta^*\| \leq \|\text{Softmax}(\psi(\eta(t)))\| \leq \frac{\sigma^{\psi}}{\|\underline{h}\|}\|\eta(t) - \eta^*\| \tag{22}$$

Since $\psi(\eta(t))$ is bounded we know that all entries i of the output of the Softmax operation satisfy the following inequality for all $t \in [0, 1]$:

$$0 < \text{Softmax}(\psi(\eta(t)))_i < 1 \tag{23}$$

Let $\underline{s} = \min_{i\in m, t\in[0,1]} \text{Softmax}(\psi(\eta(t)))_i$ and $\overline{s} = \max_{i\in m, t\in[0,1]} \text{Softmax}(\psi(\eta(t)))_i$. Then the negative log likelihood for one-hot-encoded label y, V_y satisfies the dynamic projection constrains:

$$\frac{\sigma^{\psi}}{-\log(\overline{s})\|\overline{h}\|}\|\eta(t) - \eta^*\| \leq V_y(\text{Softmax}(\psi(\eta(t)))) \leq \frac{\sigma^{\psi}}{-\log(\underline{s})\|\underline{h}\|}\|\eta(t) - \eta^*\| \tag{24}$$

□

A.3. Proof of Theorem 2

Proof. We begin by rearranging the terms of the integral:

$$\mathscr{L}(\theta) = E_{(x,y)\sim D}\left[\int_0^1 \mathscr{V}(x,y,\eta_\theta(\tau),\tau)d\mu(\tau)\right] = \int_D \int_0^1 \mathscr{V}(x,y,\eta_\theta(\tau),\tau)d\mu(\tau)dD((x,y)) \tag{25}$$

$$= \int_{D\times[0,1]} \mathscr{V}(x,y,\eta_\theta(\tau),\tau)d\mu \times D(\tau,(x,y)) \tag{26}$$

$$\tag{27}$$

Recall that

$$\mathscr{V}(x,y,\eta,t) = \max\left\{0, \frac{\partial V_y}{\partial \eta}^\top f_\theta(\eta,x,t) + \kappa V_y(\eta)\right\} \tag{28}$$

We note that that $\mathscr{V}$ is being integrated over a bounded domain and it satisfies the following properties:

1. $\mathscr{V}(x,y,\eta(t),t) \geq 0$ for all values of x, y, η and t.
2. $\mathscr{V}(x,y,\eta(t),t)$ is continuous since it is the maximum of two continuous functions: the 0 function and $\frac{\partial V_y}{\partial \eta}^\top f_\theta(\eta,x,t) + \kappa V_y(\eta)$ which is continuous since it is differentiable.

From these two facts we can conclude that $\mathscr{L}(\theta) = 0$ implies that for all $t, (x,y) \in [0,1]\times D$ the function $\mathscr{V}(x,y,\eta(t),t) = 0$. This follows from the standard calculus argument that if the function weren't zero at a point there would be an ϵ region surrounding that point that would integrate to a strictly positive value. Since $\mathscr{V} \geq 0$, integrating it over any region must also be non negative which would be a contradiction given that we assumed the integral is non-negative. □

A.4. Proof of Theorem 3

Theorem 3 is inspired by concepts underlying Input-to-State and Projection-to-State stability (Sontag & Wang, 1995). The key technical detail is leveraging the Comparison Lemma in a similar fashion as Taylor et al. (2019).

Proof. We will use the following notation for the time derivative of V_y:

$$\dot{V}_y(\eta(t),x,t) = \frac{d}{dt}V_y(\eta(t)) = \left.\frac{\partial V_y}{\partial \eta}\right|^\top_{\eta=\eta(t)} f(\eta(t),x,t) \tag{29}$$

We now begin the derivation.[10]

$$\dot{V}_y(\eta, x+\epsilon, t) \tag{30}$$

$$= \dot{V}_y(\eta,x,t) + \dot{V}_y(\eta,x+\epsilon,t) - \dot{V}_y(\eta,x,t) \qquad \text{Add 0} \tag{31}$$

$$\leq \dot{V}_y(\eta,x,t) + |\dot{V}_y(\eta,x+\epsilon,t) - \dot{V}_y(\eta,x,t)| \tag{32}$$

$$\leq \dot{V}_y(\eta,x,t) + \left|\frac{\partial V_y}{\partial \eta}^\top f(\eta,x+\epsilon,t) - \frac{\partial V_y}{\partial \eta}^\top f(\eta,x,t)\right| \tag{33}$$

$$\leq \dot{V}_y(\eta,x,t) + \underbrace{L_V L_f}_{L}\|\epsilon\| \qquad \text{From Global Uniform Lipschitz property of } V \text{ and } f \tag{34}$$

$$\leq \dot{V}_y(\eta,x,t) + L\bar{\epsilon} \qquad \text{Adversarial disturbance bound } \|\epsilon\| \leq \bar{\epsilon} \tag{35}$$

$$\leq -\kappa V_y(\psi(\eta)) + L\bar{\epsilon} \qquad \text{From Lyapunov Exponential Stability condition in Theorem 1} \tag{36}$$

[10] Note that the choice of norm is arbitrary given equivalence of norms in finite dimensional spaces

We now consider the dynamical system that achieves the upper bound Equation (36) in preparation to apply the comparison lemma:

$$\dot{\gamma} = -\kappa\gamma(t) + L\bar{\epsilon} \tag{37}$$

This dynamical system is linear and therefore has solutions of the following form:

$$\gamma(t) = e^{-\kappa t}c + \frac{L\bar{\epsilon}}{\kappa} \tag{38}$$

Now we solve for c in terms of the initial condition of this system:

$$\gamma(0) = c + \frac{L\bar{\epsilon}}{\kappa} \tag{39}$$

$$\rightarrow c = \gamma(0) - \frac{L\bar{\epsilon}}{\kappa} \tag{40}$$

$$\rightarrow \gamma(t) = e^{-\kappa t}\gamma(0) + \frac{L\bar{\epsilon}}{\kappa}(1 - e^{-\kappa t}) \tag{41}$$

Where the last line comes from plugging the resulting value of c back into the solution $\gamma(t)$.

To apply the Comparison Lemma, we need a choice of $\gamma(0)$ that satisfies $V_y(\eta(0)) \leq \gamma(0)$. Recall from the δ-Final Loss condition of Definition 5 that $e^{-\kappa}V_y(\eta(0)) \leq \delta$. This implies that $V_y(\eta(0)) \leq \delta e^{\kappa}$ so that we can pick $\gamma(0) = \delta e^{\kappa}$ to satisfy the initial condition inequality of the comparison lemma. Plugging this value into our form of the solution for $\gamma(t)$ results in the following:

$$\gamma(t) = e^{\kappa(1-t)}\delta + \frac{L\bar{\epsilon}}{\kappa}(1 - e^{-\kappa t}) \tag{42}$$

Therefore, since the following conditions hold:

1. $\dot{V}_y(\eta(t))$ and $\dot{\gamma}(\gamma(t))$ are continuous functions of time and state.
2. $\dot{V}_y(\eta(t)) \leq \dot{\gamma}(\gamma(t))$
3. $V_y(\eta(0)) \leq \gamma(0)$

We can conclude from the comparison lemma that $V_y(\eta(t)) \leq \gamma(t)$.

Recall from the definition of V_y as the cross entropy loss, that $V_y(\eta(t)) \leq -\log(\frac{1}{2}) = \log(2)$ is sufficient for to guarantee Correct Classification as defined in Definition 5. Therefore, it suffices to show that $\gamma(t) \leq \log(2)$. Plugging in the definition of $\gamma(t)$ evaluated at the end of the integration time $t = 1$ we obtain:

$$\delta + \frac{L\bar{\epsilon}}{\kappa}(1 - e^{-\kappa}) \leq \log(2) \tag{43}$$

$$\delta \leq \log(2) - \frac{L\bar{\epsilon}}{\kappa}(1 - e^{-\kappa}) \tag{44}$$

Which results in the desired bound.

□

B. Details on Learning Algorithms

Monte Carlo Method. We can formulate the objective of the training optimization problem as a Monte Carlo integral as follows:

$$\min_{\theta\in\Theta}\mathscr{L}(\theta) \approx \min_{\theta\in\Theta} \mathop{\mathbb{E}}_{\substack{(x,y)\sim\mathcal{D}\\ \eta\sim\mu_H\\ t\sim\mu_{(0,1)}}} [\mathscr{V}(x,y,\eta,t)] \tag{45}$$

$$\approx \min_{\theta\in\Theta} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\int_{H\times(0,1)} \mathscr{V}(x,y,\eta,t) d\mu_{H\times(0,1)}(\eta,t)\right]. \tag{46}$$

The expectation in Equation (45) has the property that 0 loss implies the Lyapunov Loss ,$\mathscr{L}$, is also 0 provided the PDF implied by μ is non-zero everywhere in H. This is a straight forward implication from the continuity of $\mathscr{V}$ with respect to η.

Some of the stated assumptions require justification. First, we know that perfect classifications using the cross entropy loss function require infinite inputs in one coordinate. Exponential stability to a correct classification under the cross entropy lyapunov dynamics would thus imply exponential growth for the hidden state. To avoid numerical stability issues we bound the state-space where the dynamics can evolve through our choice κ and by noting that most neural network architectures are globally Lipschitz. This implies uniqueness and existence of solutions for all time. When taking into account that we only compute a finite time integral, we can conclude that the image of a bounded set of initial conditions through the dynamics's evolution will remain be bounded. Therefore, we only need to enforce a bounded set of initial condition to have a bounded reachable set. This means our predictions can only have a maximum confidence but with a sufficiently a large bounded region where the dynamics can evolve, this error can become arbitrarily low. In practice, we initialized the dynamics with the constant zero function.

The choice of measure for both η and t has the potential to significantly impact the convergence integral; however, in practice we found that a uniform distribution over the n-cube performed well for most problems and in high-dimensional settings a slightly different distribution can be used to mitigate the curse of dimensionality.

Picking H to sample from:

We will sample from a k-dimensional hypercube with corners at $(\pm s, \pm s, \ldots, \pm s)$. We now have to pick s so that η does not evolve outside of the set H. In principle we could instead select a k-hypersphere but it is well known that most of the volume of the hypersphere is concentrated towards the surface in higher dimensions. In reality, the dynamics will spend most of the time evolving in the interior since our prediction dynamics are initialized at the origin. Although the hypercube also has most of its volume near the corners, we expect systems to evolve near the corners that correspond to a classification. We note that this choice of H is unlikely to be optimal.

To select and s, begin by considering the bound we are trying to enforce on the dynamics:

$$\dot{V}_y(\eta,x,t) \leq -kV(\eta) \tag{47}$$

In a comparison lemma style analysis consider the system that achieves the upper bound:

$$\dot{\gamma} = -k\gamma \tag{48}$$

We hypothesise that in the case of an incorrect classification, the system evolves as if lower bounded by $\underline{\gamma} = k\underline{\gamma}$. This assumption is based on the supposition that the dynamical system will evolve to a corner of the hypercube corresponding to an admissible but incorrect classification. Therefore we only have to solve for an s that satisfies $V_y(\eta^*) = e^{-k}$ where

$$\eta_i^* = \begin{cases} s & \text{if } 1 \leq i \leq k \text{ is the correct class.} \\ -s & \text{otherwise} \end{cases} \tag{49}$$

Although this choice of H worked in practice, this choice of H is not optimal and could be improved by better exploiting the structure of the dynamics and the classification problem more generally.

Sampling on Hypersphere In high-dimensional spaces sampling in the hypercube is highly inefficient for retrieving samples around the origin. Our models are also all initialized at the origin and should end close to a vertex of the simplex that corresponds to a classification. It's thus imperative to bias sampling towards the origin in higher dimensions. To do this we use the following procedure to generate samples:

Algorithm 3 Biased Sampling on Hypersphere

Require: A hyper sphere of radius r_{max} inscribes the hypercube for H

$r \sim U(0, r_{max})$ ▷ sample a radius

$\hat{\eta} \sim \mathcal{N}(0, 1)^k$ ▷ sample k dimensional normally distributed random vector

return $r \frac{\hat{\eta}}{\|\hat{\eta}\|_2}$ ▷ Method for generating point on surface of unit sphere Muller (1959).

Although this method significantly biases the samples towards zero, we note that that the region near zero is the area that implies the smallest prior for a classification. If we are very far away from the origin we would expect to keep moving in the direction opposite of the origin (since we have a strong prior). The dynamics should learn to exploit this by having relatively simple dynamics in points far away from the origin that get more complex as they approach the point with lowest prior (zero).

C. Experimental Details

To simplify tuning, we trained our models using Nero (Liu et al., 2021) with a learning rate of 0.01 with a batch size of 64 for models trained with LyaNet and 128 for models trained with regular backpropagation. We found this by performing a grid search on learning rates and batch sizes over $(0.1, 0.001, 0.001) \times (32, 64, 128)$, validated on a held out set of 10% of training data. All models were trained for a total of 120 epochs. For our adversarial attack we used PGD as implemented by Kim (2020) for 10 iterations with a step size $\alpha = \frac{2}{255}$. Our experiments ran on a cluster 6 GPUs: 4 GeForce 1080 GPUs, 1 Titan X and Titan RTX. All experiments were able to run on less than 10GB of VRAM.