Lyapunov Guarantees for Learned Policies

Kyle Crandall
US Naval Research Laboratory
Washington, DC, United States of America
kyle.l.crandall2.civ@us.navy.mil

Connor Yates
US Naval Research Laboratory
Washington, DC, United States of America
connor.l.yates.civ@us.navy.mil

Corbin Wilhelmi
US Naval Research Laboratory
Washington, DC, United States of America
corbin.wilhelmi.civ@us.navy.mil

ABSTRACT

Reinforcement Learned policies are notorious in their lack of supporting proofs and guarantees. One approach to providing such guarantees is learning a Domain of Attraction that proves stability based on the Lyapunov Stability Criterion. We build on this approach to improve performance and ease of implementation, and present a highly parallelizable algorithm that produces a uniform grid that tessellates the desired region of the state space. By discretizing the state space, we take advantage of the Lipschitz nature of the problem to prove that not only is a sample point stable, but so is a neighborhood of it. This discretization is then combined with existing algorithms to learn a neural network that can be used as a Lyapunov candidate. We present our proposed algorithm, and demonstrate it on a torque limited inverted pendulum, as well as highlight effects of our improvements in experimental results.

KEYWORDS

Reinforcement Learning, Lyapunov Stability, Stability Guarantees

1 INTRODUCTION

Reinforcement Learning (RL) has been a popular method for determining a policy to drive a Markov system to a desired behavior. One of the major challenges with these methods has been demonstrating that a learned policy will work beyond the tested data points. While it is generally assumed that a learned policy will work in a general area that is well represented in the sample data, what qualifies as "well represented," as well as a prediction of how far outside the sampled region the policy is valid, is a major open question. Previous work has sought to learn a such a certificate based on a set of trajectories [4], however this method still relies on sampled trajectories, and does not exploit knowledge of the underlying dynamics of the system to extrapolate beyond the sampled set.

The Lyapunov Stability Criterion (LSC) is the standard basis for proving stability of a closed loop system in the field of control theory. This approach relies on finding a positive definite function known as the Lyapunov candidate, and showing that the closed loop dynamics of a system result in the candidate's time derivative being negative definite on some region, thus proving that the closed loop dynamics are stable, and must converge asymptotically to the origin if the initial condition resides in a calculated Domain of Attraction (DoA) [8]. This concept can also be applied to RL, which seeks to solve a larger class of problem beyond just dynamical systems. Using the LSC can provide a stability certificate for a learned policy, especially in cases with continuous state and action spaces by exploiting knowledge of the dynamics to expand the DoA beyond just the training points.

The idea of using the LSC to validate a learned policy is not itself new. Prior work has proposed learning a Lyapunov candidate in an iterative manner, starting with an Sum of Squares (SoS) function that is guaranteed to have at least a small DoA for a stable system, then building on it and expanding it as far as is needed [9]. Other methods propose using the value function that is the result of a RL process as a Lyapunov function, provided the reward is appropriately shaped [1]. In some cases, the learned Lyapunov candidate may be utilized to inform a new round of learning thus integrating the LSC into the Reinforcement Learning process itself [1, 5].

One of the key steps to this problem is proving that the LSC is satisfied on a set, not just on single points. Methods using Satisfiability Modulo Theory (SMT) have been proposed for general neural networks [6, 7] that, given a classification problem, consider the minimal amount an input point can be varied before the class changes, resulting in a radius around the point that forms a set on which the classification is consistent. Other work utilize neural networks as Lyapunov candidates, called Lyapunov Neural Networks (LNNs), and train them by similarly framing the Lyapunov problem as a classification problem [9], or utilizing the value function that is a result of the RL processes [1].

We present a more efficient LNN algorithm that uses a uniform discretization of the state space. Our uniform discretization better integrates with the overall training paradigm of the LNN, and parallelizes training better than multi-resolution methods produced by previous work [9]. By basing our algorithm off [9] and [2], we are able to demonstrate that our method learns a stability certificate that provides a well defined DoA on a torque-limited inverted pendulum. We also present an analysis on the construction and training of the LNN, specifically, how the numerical Lyapunov analysis can be integrated into to the training, and the effects of various hyper-parameters on the overall performance.

2 LYAPUNOV STABILITY ANALYSIS WITH NEURAL NETWORKS

Consider a discrete time dynamical systems with a state space $\mathcal{X}$ and action space $\mathcal{U}$. The function $f_{ol} : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ specify the open loop dynamics $x_{k+1} = f_{ol}(x_k, u_k)$ where $x_k \in \mathcal{X}$ is the state of the system, and $u_k \in \mathcal{U}$ the action taken at time k. If the action is determined by some policy $\pi : \mathcal{X} \to \mathcal{U}$, the closed-loop autonomous dynamics of this system are expressed as $f(x) = f_{ol}(x, \pi(x))$ We assume that the origin is an equilibrium of the closed loop dynamics, i.e. $f(0) = 0$ (Note if this is not true, it can me made true with a change of variables). The Lyapunov Stability Criterion demonstrates that the policy will drive the state to the

Proc. of the Adaptive and Learning Agents Workshop (ALA 2024), Avalos, Müller, Wang, Yates (eds.), May 6-7, 2024, Online, https://ala2024.github.io/. 2024.

origin asymptotically if the initial state falls within some DoA $\mathcal{D}_c \subset \mathcal{X}$ that is a neighborhood of the origin. [3, 8].

Theorem 1 (Lyapunov Stability Criterion for Discrete Time Systems). *If there exists some function $v : \mathcal{X} \to \mathbb{R}$ that is positive definite, and if the value $\Delta v(x) = v(f(x)) - v(x)$ is negative definite on a sub-level set $\mathcal{V}(c) = \{x \mid v(x) \leq c\}$, and the functions f and v are Lipschitz on $\mathcal{V}(c)$, then the system is asymptotically stable, with a DoA $\mathcal{V}(c)$*

The typical structure of a Lyapunov analysis starts with a candidate function v, known as the Lyapunov candidate, that is positive definite. We then analyze the value Δv to determine where it is negative, and finally identify the largest sub-level set $\mathcal{V}(c)$ that fits inside that area [8]. If we are to train a LNN as the Lyapunov candidate $v(\cdot)$, there are two things we need to accomplish: first we must determine the DoA for a given Lyapunov candidate (section 2.1), second, we will update the LNN based on this information such that the area of the DoA increases during training(section 2.2).

2.1 State Space Discretization

Given a closed-loop system, per theorem 1, the DoA is a sub-level set of v on which the value Δv, which we will call the Lyapunov difference, is negative. We will refer to the sign of this value as the Lyapunov direction. Bobiti and Lazar propose a method of multi-resolution sampling to determining such a space [2], however this sampling would need to be redone from scratch for each step of training, and the resolutions recalculated, which represents a significant amount of redundant work. Instead, we will take inspiration from this multi-sampling approach, but apply it to a uniform discretization of the state space that can be reused in the next iteration. We can take advantage of the Lipschitz nature of both f and v [11] to extend the Lyapunov directionality of a point to a neighborhood of that point.

Lemma 1. *If the Lyapunov direction of a point $x \in \mathcal{X}$, is negative, then there exists some neighborhood $\mathcal{N}_x = \{\hat{x} \in \mathcal{X} \mid \|x - \hat{x}\| < \epsilon\}$, where $\epsilon = \frac{|\Delta v(x)|}{L_v(1+L_f)}$, on which all points have a negative Lyapunov direction.*

Proof. Let L_f be the local Lipschitz constant of the dynamics function f, and L_v be a Lipschitz constant of the Lyapunov function v (using the same distance metric $\|\cdot\|$) on some domain $\mathcal{D}_x \in \mathcal{X}$ where $x \in \mathcal{D}_x$. Let $\Delta x = \hat{x} - x$ for some arbitrary $\hat{x} \in D_x$. By the Lipschitz continuity definition

$$\begin{aligned}\|\Delta v(x) - \Delta v(\hat{x})\| &= \|v(f(x)) - v(x) - v(f(\hat{x})) + v(x)\| \\ &\leq \|v(f(x)) - v(f(\hat{x}))\| + \|v(x) - v(\hat{x})\| \\ &\leq L_v\left(1 + L_f\right)\|x - \hat{x}\| \\ &\leq L_v\left(1 + L_f\right)\|\Delta x\|\end{aligned}$$

Therefore we can say that $\Delta v(\hat{x})$ is bounded by

$$\Delta v(x) - L_v\left(1 + L_f\right)\|\Delta x\| \leq \Delta v(\hat{x}) \leq \Delta v(x) + L_v\left(1 + L_f\right)\|\Delta x\|$$

If $\Delta v(x) < 0$, then any point where $\Delta x \leq \epsilon$, and thus is in the set $\mathcal{N}_X$, will also have a negative Lyapunov direction. □

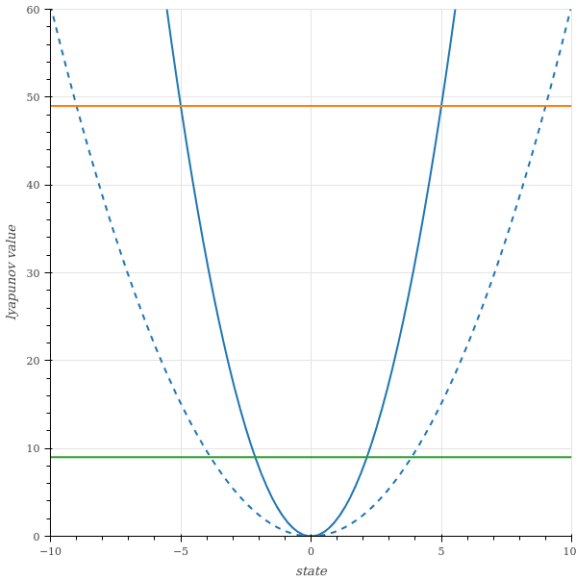

Figure 1: Example Lyapunov curves demonstrating the Lyapunov function near the origin for a Lyapunov function $v(x) = ax^2$ for values of $a = 1.9$ for the solid blue line, and $a = 0.6$ for the dashed blue line. The orange line represents the level set, and the green line represents the resolution limit ϵ from algorithm 1.

Corollary 1.1. *Similarly, if the point x has a positive Lyapunov direction then the all points $\hat{x} \in \mathcal{N}_x$ will also have a positive Lyapunov direction.*

Consider a rectilinear tile of the state space with a radius τ: $T_x^\tau = \{\hat{x} \in \mathcal{X} \mid \|x - \hat{x}\|_\infty \leq \tau\}$. Let L_f and L_v be the Lipschitz constants of f and v respectively on T_x^τ under the $\|\cdot\|_\infty$ distance metric.

- If $\Delta v(x) > L_v\left(1 + L_f\right)\tau$, then the tile T_x^τ is considered ascendant.
- If $\Delta v(x) < -L_v\left(1 + L_f\right)\tau$, then the tile T_x^τ is considered descendant.
- otherwise, the tile may contain ascendant and descendant points, and is considered mixed.

We tessellate a domain of interest with such tiles that can be analyzed independently. If a tile is found to be mixed, we can use a similar recursive refinement as suggested in [2], i.e. that tile can be subdivided into smaller tiles that can be similarly analyzed. However, we will go a step further and say that, if all of these sub-tiles are found to be ascendant or descendant, then the parent tile can be considered so as well, but if one sub-tile is found to be ascendant while another is found to be descendant, then the parent tile must be mixed. This will result in a uniform tiling of the state space, which is more useful when using this discretization to learn a Lyapunov function, as we will demonstrate in the next section. The pseudocode for this algorithm is presented in algorithm 1.

Illustrative sample results are given in figure 1 for a single state system with a Lyapunov candidate $v(x) = ax^2$ represented as the blue lines. The orange line represent the level c of the estimated

Algorithm 1: Algorithm to determine regions of uniform Lyapunov directionality

Data: Tile centers $C \in \mathcal{X}$, tile radius $\tau \in \mathbb{R}$, minimum resolution $\tau_m \in \mathbb{R}$, sub-sample rate $n \in \mathbb{N}_{\geq 2}$
Result: Lyapunov direction for each tile D

$L \leftarrow \emptyset$;
$d_x \leftarrow 0 \; \forall \; x \in C$;
$T \leftarrow \{\{\text{center: } x \text{ parent: } x \text{ dir: } 0\} \; \forall \; x \in C\}$;
while $\tau \geq \tau_m$ **do**
  $D_x \leftarrow \emptyset \; \forall x \in C$;
  for $t \in T$ **do**
    $\Delta v \leftarrow v\left(f\left(t.\text{center}\right)\right) - v\left(t.\text{center}\right)$;
    $\epsilon \leftarrow L^{\tau}_{v,t.\text{center}} \left(1 + L^{\tau}_{f,t.\text{center}}\right) \tau$;
    $t.\text{dir} \leftarrow \text{sgn}\left(\min\left(\Delta v + \epsilon, 0\right) + \max\left(\Delta v - \epsilon, 0\right)\right)$;
    $D_{t.\text{parent}} \leftarrow D_{t.\text{parent}} \cup \{t.\text{dir}\}$;
  for $x \in C \setminus L$ **do**
    $d^+ \leftarrow 1$ if $d = 1$ for any $d \in D_x$;
    $d^- \leftarrow 1$ if $d = -1$ for any $d \in D_x$;
    if *(d^+ and d^-)* **then**
      $L \leftarrow L \cup x$;
    else if $d = 1 \; \forall \; d \in D_x$ **then**
      $d_x = 1$;
      $L \leftarrow L \cup x$;
    else if $d = -1 \; \forall \; d \in D_x$ **then**
      $d_x = -1$;
      $L \leftarrow L \cup x$;
  $\tau \leftarrow \tau / n$;
  $\hat{T} \leftarrow \emptyset$;
  for $t \in T$ **do**
    if $t.dir = 0$ *and* $t.parent \notin L$ **then**
      Sample new points X on grid of size $2n$ centered at t.center with spacing τ;
      $\hat{T} \leftarrow \hat{T} \cup \{\{\text{center: } x \text{ parent: } t.\text{parent dir: } 0\} \; \forall \; x \in X\}$;
  $T \leftarrow \hat{T}$;

DoA, while the green line in the figure represents the resolution limit ϵ. Note that there is a region near the origin, no matter what valid Lyapunov candidate is used, that cannot satisfy the conditions given in Lemma 1 for a fixed value of ϵ, and a finite sampling resolution because $v\left(x\right) < \epsilon$ for those points. Because of this, an analytical solution such as the Lyapunov Indirect Method [8], or a previous Lyapunov analysis, must serve as a starting point to show that this area is in fact part of the DoA. Therefore, unlike the multi-resolution method proposed previously [2], this method can naturally build upon the results of previous analysis. This will make it particularly useful when we use it to train a LNN.

Once the Lyapunov directionality of the tiles has been found, the DoA is estimated per theorem 1 as the sub-level set of $v\left(\cdot\right)$ such that the Lyapunov directionality of all points in that set is -1. This can be estimated form the tiles by first identifying the tiles on the boundary of the area where the directionality is -1:

$$\mathcal{B} = \left\{x \text{ if } d_x = -1 \text{ and } d_n \neq -1 \; \forall \; d_n \in \mathcal{A}^+_x \; \forall \; x \in C\right\}$$

where $\mathcal{A}^+_x$ is the set of neighboring tiles distal from the origin. We then identify the minimal value of $v\left(\cdot\right)$ on this boundary set $c = \min_{x \in \mathcal{B}} v\left(x\right)$, that will be the sub-level set that forms the DoA $\mathcal{D} = \mathcal{V}\left(c\right)$. Note that this is a conservative estimate of the DoA as it just uses the value at the center, however this is sufficient since $v\left(x\right) \leq \max_{\hat{x} \; | \; \|x - \hat{x}\| \leq \tau} v\left(\hat{x}\right)$.

As a modification to this method, the centers can be simulated forward in time multiple steps [10]. Let x_0 be the center point of a tile, and x_m be the end of an m step trajectory segment. If we consider $\Delta v\left(x_0\right) = v\left(x_m\right) - v\left(x_0\right)$, then $\epsilon = L_v \left(1 + L^m_f\right)$ per lemma 1. This has a twofold effect on the performance of this algorithm, first, if $L_f < 1$, then $\epsilon \rightarrow L_v$ as $m \rightarrow \infty$. Secondly, a larger trajectory near a stable equilibrium will have a larger Δv. Both of these facts will mean that that it will be easier to classify tile as ascendant or descendant, and decrease the number of minimum-resolution tiles that must be analyzed.

2.2 Lyapunov Neural Networks

The key requirement of a Lyapunov candidate is that it must be positive definite, i.e. $v\left(0\right) = 0$ and $v\left(x\right) > 0 \; \forall \; x \in \mathcal{X} \setminus 0$. We will call a neural network used as a Lyapunov function a Lyapunov Neural Network. The LNN will be represented as a function of the state parameterized by the weights of the neural network θ, thus our Lyapunov candidate is $v\left(x|\theta\right)$. It is important to note that a traditional MultiLayered Perceptron (MLP) style layered neural network does not satisfy these conditions, and so we use the structure proposed by Richards et. al. [9].

In addition to constructing the LNN in this manner, the methods proposed in [9] pre-train the network using a simple SoS to give the network a starting point. We propose to instead pretrain the network based on a Lyapunov indirect analysis of the system [8]. In this analysis, we first linearize the dynamics about the origin.

$$f\left(x\right) \approx \left.\frac{\partial f}{\partial x}\right|_{x=0} x = F_x x$$

We can then use the quadratic Lyapunov function $v\left(x\right) = x^T P x$ where P is the solution to the discrete-time Lyapunov equation $F_x P F_x^T - P + Q = 0$, and Q is an arbitrary, positive-definite matrix. This Lyapunov function will provide a larger initial DoA than an SoS Lyapunov function, and can be used just as easily for pre-training by exploiting a-prori knowledge of the system that is simple to calculate.

2.3 Learning Lyapunov Functions

To train a Neural Network to act as a Lyapunov candidate, a LNN, we present a novel approach in Algorithm 1, adapted from [9]. The algorithm can be broken into 4 steps that are repeated until convergence is reached:

(1) Determine the sub-level set DoA $\mathcal{V}\left(c\right)$ per the LSC given the LNN as a Lyapunov candidate.
(2) Sample a batch of points $X_b \subset \mathcal{V}\left(\alpha_1 c\right)$
(3) Determine labels by simulating points X_b forward N steps $Y_b = \{1 \text{ if } v\left(x_T\right) < c \text{ else } -1\}$

(4) Train the LNN on the labeled batch (X_b, Y_b)

Algorithm 1 is novel in two key ways, first we use the discretization method presented in section 2.1 to estimate the DoA in step 1, and also to perform the batch sampling in step 2. Because each tile requires calculating the Lyapunov value of the center and the Lipschitz constant of the Lyapunov candidate on that tile, the tiles that are within the level set $\mathcal{V}(\alpha c)$ can be identified, and samples can be drawn from these tiles.

Further, we can use the results of the previous DoA calculation as a starting point for the next one. Since the DoA is only a known subset of the true DoA, we can set the Lyapunov direction of all tiles that are fully with the DoA to -1, and add them to the lockout list L. This means that the resulting DoA will be at worst the largest sub-level set of the new Lyapunov function that fits within the previously calculated DoA. This use of the discretization we present is due to the fact that while we use a multi-resolution analysis, the results are ultimately presented as a uniform grid which does not shift as the Lyapunov candidate is adapted. In addition, we will also train the LNN on the batch multiple time, specifically, we will choose M mini-epochs, meaning the training will go through the batch M times.

The second improvement we make to the algorithm is in the loss function

$$l = \max\left\{0, -y * \left((c^* - v(x)\right)\right\} + \lambda \frac{y+1}{2} \max\{0, \Delta v(x)\}$$

where x is the test point and y is the predicted label. The value c^* is a hyperparemter that represents the target level that defines the DoA. A consequence of this however, is that if c^* is too small, the Lyapunov function flattens near the origin, and the neighborhood of the origin that cannot be proven to be descendant by Algorithm 1 will grow. This can be seen in figure 1 by comparing the two different Lyapunov candidates depicted. We adjust the parameter a so the Lyapunov candidate shifts from the solid blue line to to the dashed blue line, to increase the size of the DoA, using the same target c^*, While this increases the size of the DoA, it also increases the size of every other level set, including the one given by ϵ (represented by the green line). In the original training algorithm presented in [9], this growth can eventually outpace the growth of the estimated region of attraction, and will result in a collapse of that estimate if there are points in this neighborhood of the origin that have not been previously established to be stable. To rectify this, we propose to grow this target along with the growth of the sample region in step 2 with a factor $\alpha_2 \geq 1$ to address this issue. By choosing appropriate values for α_1 and α_2, this flattening issue can be postponed, or stopped directly. Using this growth of c^* allows us to use larger values for α_1, which means we are able to grow the estimated DoA faster, and thus speed up the required training times.

The resulting algorithm with our proposed modifications is presented Algorithm 2. Algorithm 2 incorporates the discretization results from Algorithm 1 in Line 4, as well as the c^* growth in Line 7.

Algorithm 2: Training a Lyapunov Neural Network

Data: Closed loop dynamics $f : \mathcal{X} \to \mathcal{X}$, Initialized, parameterized LNN $v : \mathcal{X} \to \mathbb{R}^+$, Level set expansion multiplier α, Forward simulation horizon N, Number of mini-epochs to use per epoch M, initial target c_0^*

Result: Trained LNN v

1 Discretize state space with rectilinear tiles with centers $\mathcal{X}_c$;
2 $c^* \leftarrow c_0^*$;
3 **repeat**
4 Calculate Lyapunov direction $d_x \; \forall \; x \in \mathcal{X}_c$ with algorithm 1;
5 $c \leftarrow max_{x \in \mathcal{X}_c} \{v(x) \,|d_{x+\tau\delta} \neq -1 \forall \delta \in \text{Basis}(\mathcal{X})\}$;
6 Sample Batch $\mathcal{X}_b$ from the set $\{x \mid v(x) \leq \alpha_1 c \; \forall \; x \in \mathcal{X}_c\}$;
7 $c^* \leftarrow \alpha_2 c^*$; `// This expansion of c* is critical do prevent leveling in the DoA`
8 Label Batch $\mathcal{Y}_b = \{1$ if $v(x_N) \leq c$, -1 otherwise $\forall \; x_0 \in \mathcal{X}_b\}$;
9 Lock all tiles $x \in \mathcal{X}_c \mid v(x) < \frac{c}{L_v(x,\tau)}$ as stable;
10 Train $v(\cdot)$ on batch $(\mathcal{X}_b, \mathcal{Y}_b)$ for M mini-epochs to target c^*;
11 **until** *convergence*;

3 EXAMPLE

We demonstrate training an LNN on a torque limited inverted pendulum governed by the following dynamics:

$$\ddot{\theta} + \alpha_1 \dot{\theta} - \alpha_2 \sin\theta = \alpha_3 u \tag{1}$$

where $|u| \leq u_{max} < \frac{\alpha_2}{\alpha_3}$ is the input (limited) torque. We use values of $\alpha_1 = 1.5$, $\alpha_2 = 15$, and $\alpha_3 = 3$ for our model and $u_{max} = 4$ for the torque limiting. We design a simple Linear Quadratic Regulator (LQR) control law $u = -Kx$ where the state $x = [\theta, \omega]^T$ and $\omega = \dot{\theta}$. Due to the torque limiting, the control law is saturated at $\pm u_{max}$. We use semi-implicit Euler integration to discretize this system in time to get the closed loop dynamics.

Figure 2 shows estimates of the DoA yielded by three different methods in the phase space of the plant given in (1). Along with these estimates, the phase trajectories are included to give some context for the DoA. The red trajectories are generated by picking points on the stable sides of the two equilibria at $\theta = \pm \operatorname{asin}\left(\frac{\alpha_3 u}{\alpha_2}\right)$ and $\dot{\theta} = 0$. These trajectories give us an better idea of the true DoA.

The first and smallest DoA, given in figure 2 by the darkest green contour, is the product of a Lyapunov indirect analysis that verifies that this system is controllable, that the closed loop system is stable. This initial DoA is built on using the discretization method outlined in algorithm 1 with a quadratic Lyapunov candidate $v(x) = x^T P x$ where P is the solution to the Lyapunov equation discussed previously (using $Q = I$). This analysis yields a less conservative estimate of the DoA, given in figure 2 by the next shade out.

Figure 3 illustrates the resulting Lyapunov direction analysis based on the discretization method proposed. The phase trajectories are included in this figure as well to help map the points in the phase space to figure 2. Note that there are descendent tiles that are clearly in the unstable region, as well as ascendant tiles that are clearly in the stable region. This is due to the fact that a level set of the quadratic Lyapunov function cannot have the shape necessary to fully encompass the true DoA, and here is where the use of LNNs can be useful.

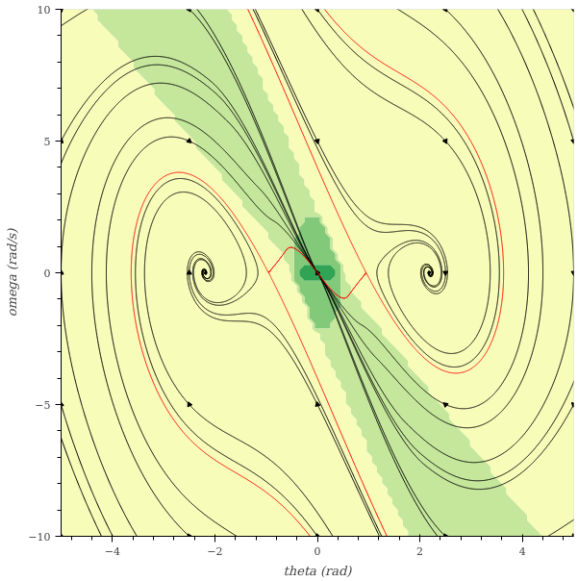

Figure 2: Estimated DoA in the phase space of system based on (moving from green to yellow) the Lyapunov indirect method, quadradic Lyapunov function, and an LNN. The red trajectories serve as a far less conservative approximation of the overall DoA. In addition, sample trajectories in the phase space are plotted, the red trajectories are chosen to outline the true DoA.

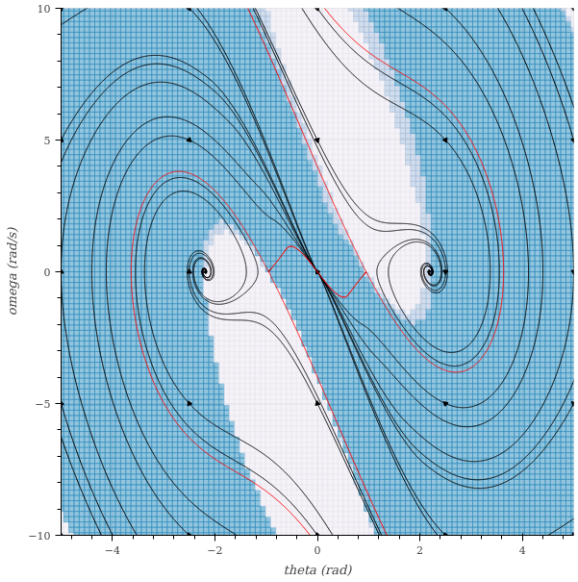

Figure 3: Lyapunov direction based on proposed discretization method and the quadratic Lyapunov candidate. Dark tiles are descendant, white tiles are ascendant, and the light blue tiles are mixed

Using the modified algorithm from [9] outlined in algorithm 2, we can train an LNN to give us the the largest estimate of the DoA given in figure 2 by the lightest green contour. For this method, we utilized an LNN structured as proposed in [9], pretrained on the quadratic Lyapunov function, and trained for 10 total epochs. The growth of the DoA can be seen in figure 4.

4 RESULTS

The main experimental result we present demonstrates the potential for improvement to the learning that we get by growing our target level set value c^* as discussed in section 2.3. Figure 5 shows the percentage of tiles in the discretization that are known to be stable at a given epoch for three different trials, each with different values for α_1 and α_2. The flattening problem discussed in section 2.3 is apparent in these plots as a dramatic decrease in the percentage of the area that is provably stable. This is due to the fact that the new level set will now be constrained to be inside, the flat area near the origin. When we increase α_1 from 1.5 to 2.0, but let $\alpha_2 = 1$, we get a slight improvement in learning speed, but we also see the point of collapse happen sooner as well, and getting similar overall peak performance. However, when we let $\alpha_2 = 2.0$, we see a general improvement in learning speed, as well as postponing the point of collapse. The final result is an overall improvement to peak performance.

It is important to note that α_1 and α_2 are hyperparameters, and suffer from some of the issues common in machine learning in that it can be difficult to predict exactly how they will effect the overall performance beyond some basic rules of thumb. However, these result do show the potential for the effects of α_1 and α_2 to amplify each other and result in better overall performance.

5 CONCLUSION

In this paper, we propose a method for discretizing a state space and performing a Lyapunov analysis to estimate a Domain of Attraction. This method is then integrated with method for training a Neural Network to serve as a Lyapunov function, and we highlight the key ways that our proposed discretization method augments this training. In addition, we propose some other modification to improve the overall performance of the training by expanding the expected DoA, and the use of mini-epochs. This results in an algorithm that can be used to learn a DoA for an arbitrary policy.

There are several areas for future work on this topic to address The method we present for addressing the flattening issue discussed in section 2.3 works, but is still reliant on choosing correct hyper parameters. Future work will seek to better understand these hyper-parameters, and their effects on the overall training. Additionally, we also plan to explore improved methods, for example, adjusting the loss term to account for this phenomenon more directly, or utilizing a more strategic sampling method.

In addition to improving on this method, we plan to utilize the results of this method to build methods of reinforcement learning that are conscious of their proposed policies abilities to achieve goals. By establishing a DoA for a policy, a learner will be able to identify areas the need focus in the next learning cycle, rather than spending learning time improving areas on which the policy already works. This will result in a more efficient overall learning,

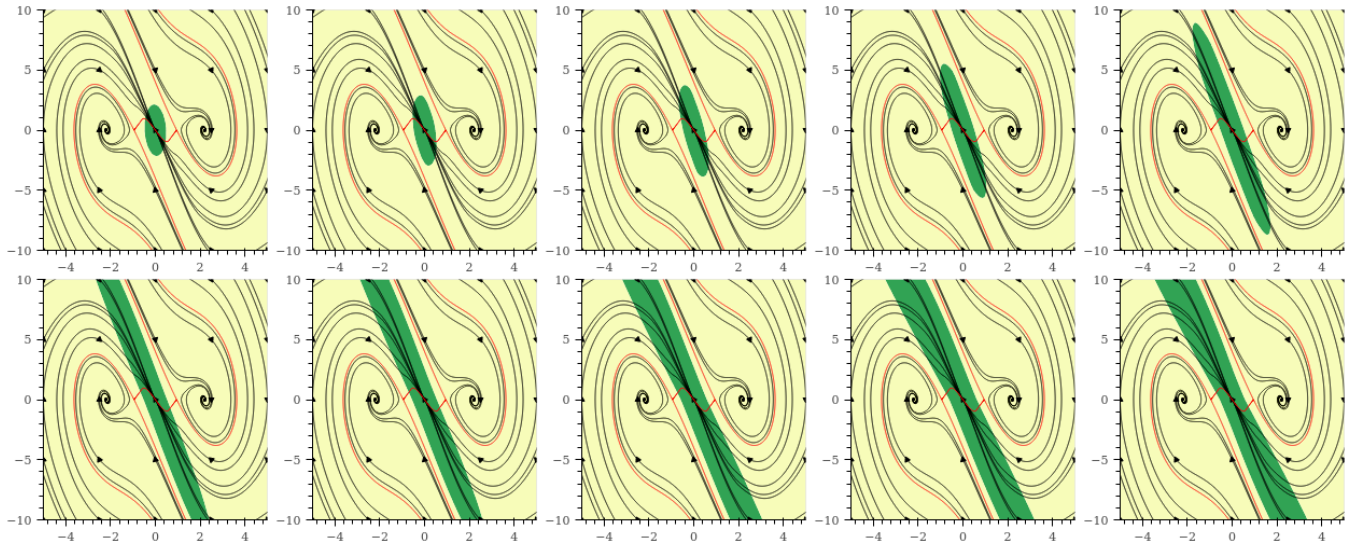

Figure 4: Estimated DoA throughout the training of an LNN starting after pretraining with epoch 1 at the top left, to epoch 10 in the bottom right.

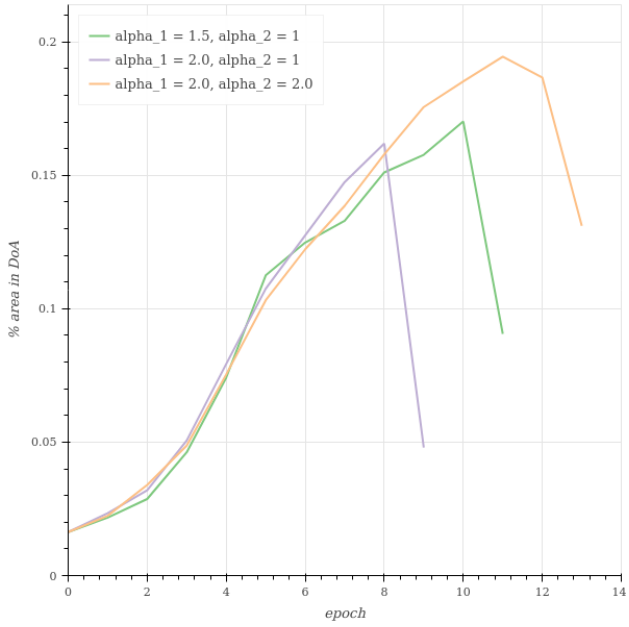

Figure 5: Training comparison for different values of α_1 and α_2, illustrating the percentage of the area of interest that is determined to be stable at each epoch of training.

and provide stability certificates for results that guarantee that the policy will work under the assumed dynamics. In addition, an established Lyapunov analysis may also allow for additional analysis such as examining stability margins to determine how wrong assumptions about the dynamics can be for the policy to still work.

ACKNOWLEDGMENTS

This work was funded by the US Office of Naval Research.

REFERENCES

[1] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. 2017. Safe model-based reinforcement learning with stability guarantees. *Advances in neural information processing systems* 30 (2017).

[2] Ruxandra Bobiti and Mircea Lazar. 2016. A sampling approach to finding Lyapunov functions for nonlinear discrete-time systems. In *2016 European Control Conference (ECC)*. IEEE, 561–566.

[3] Nicoletta Bof, Ruggero Carli, and Luca Schenato. 2018. Lyapunov theory for discrete time systems. *arXiv preprint arXiv:1809.05289* (2018).

[4] Nicholas Boffi, Stephen Tu, Nikolai Matni, Jean-Jacques Slotine, and Vikas Sindhwani. 2021. Learning stability certificates from data. In *Conference on Robot Learning*. PMLR, 1341–1350.

[5] Ya-Chien Chang, Nima Roohi, and Sicun Gao. 2019. Neural lyapunov control. *Advances in neural information processing systems* 32 (2019).

[6] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 3–29.

[7] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 97–117.

[8] Hassan K Khalil. 2002. *Nonlinear systems* (third ed.). Prentice Hall.

[9] Spencer M Richards, Felix Berkenkamp, and Andreas Krause. 2018. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Conference on Robot Learning*. PMLR, 466–476.

[10] Roy Siegelmann, Yue Shen, Fernando Paganini, and Enrique Mallada. 2023. A Recurrence-based Direct Method for Stability Analysis and GPU-based Verification of Non-monotonic Lyapunov Functions. In *62nd IEEE Conference on Decision and Control (CDC)*.

[11] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.