Lyapunov Neural Network with Region of Attraction Search

Zili Wang, Sean B. Andersson, and Roberto Tron

Abstract— **Deep learning methods have been widely used in robotic applications, making learning-enabled control design for complex nonlinear systems a promising direction. Although deep reinforcement learning methods have demonstrated impressive empirical performance, they lack the stability guarantees that are important in safety-critical situations. One way to provide these guarantees is to learn Lyapunov certificates alongside control policies. There are three related problems: *1)* verify that a given Lyapunov function candidate satisfies the conditions for a given controller on a region, *2)* find a valid Lyapunov function and controller on a given region, and *3)* find a valid Lyapunov function and a controller such that the region of attraction is as large as possible. Previous work has shown that if the dynamics are piecewise linear, it is possible to solve problems 1) and 2) by solving a Mixed-Integer Linear Program (MILP). In this work, we build upon this method by proposing a Lyapunov neural network that considers monotonicity over half spaces in different directions. We *1)* propose a specific choice of Lyapunov function architecture that ensures non-negativity and a unique global minimum by construction, and *2)* show that this can be leveraged to find the controller and Lyapunov certificates faster and with a larger valid region by maximizing the size of a square inscribed in a given level set. We apply our method to a 2D inverted pendulum, unicycle path following, a 3-D feedback system, and a 4-D cart pole system, and demonstrate it can shorten the training time by half compared to the baseline, as well as find a larger ROA.**

I. Introduction

When designing controllers for dynamical systems, there are a variety of properties which are important to ensure, including stability, safety, and robustness. These properties can be proven via certificate functions. In this work, we focus on stability, although the methods we develop can apply to other properties as well. A powerful technique for certifying stability of nonlinear systems is through the use of Lyapunov functions [1]. A system's equilibrium point is asymptoically stable in the sense of Lyapunov if, starting from any state within a region (called the region of attraction (ROA)), the future states of the system eventually remain close to the equilibrium. The main challenge is that finding Lyapunov functions for nonlinear systems is difficult in general and requires intuition/manual verification.

In this paper, we are interested in automated controller synthesis with Lyapunov stability guarantees. A simple method is to linearize the system around the equilibrium point and then use LQR control, which gives a local stability guarantee [2]. Alternatively, one can use nonlinear polynomial approximation of the dynamics, and use semidefinite programming (SDP) to find a Lyapunov function as a sum-of-square (SOS) polynomial [3], [4]. Recently, with the development and wide application of deep learning in robotics, many approaches proposed neural network representations of more complex Lyapunov functions [5] to synthesize controllers, from a simple nonlinear representation [6], to more generalized fully connected neural networks [7], [8].

In order to synthesize a controller with Lyapunov guarantees, one needs to verify whether the Lyapunov conditions are satisfied for all the states within a region [9]. Given a bounded input, the two principal techniques used to verify an input-output relationship of a network are: *1)* exact verification using either Satisfiability Modulo Theories (SMT) [6] (where quadratic and $\tanh$ activation functions are used), or applying mixed-integer programs (MIP) solvers [7], [10] (where (leaky) ReLU activation is used), and *2)* inexact verification via convex relaxation [11].

In this paper, we use ReLU neural networks and a MILP verifier to give exact Lyapunov stability guarantees. Typically, the MILP is formulated to verify certain properties of ReLU neural networks such that if the optimal objective value is zero, certain conditions can be certified; otherwise a counter-example will be generated and used to reduce the violation [12]. There are two approaches to train networks to satisfy certain properties: *1)* counter-example guided training that adds counter-examples to the training set and minimizes a surrogate loss function of the Lyapunov condition violation on a training set [6], and *2)* bi-level optimization that minimizes the maximum violation of the Lyapunov conditions via gradient descent [7]. We use a bi-level optimization training process similar to [7] due to its observed better performance in higher dimensional systems.

In safety critical systems, it is particularly desirable to understand and find the largest ROA. There are several approaches to maximize the ROA. In counter-example guided training, a cost term between the L_2 norm of the state and the Lyapunov function can be added to the surrogate Lyapunov loss function to regulate how quickly the Lyapunov function value increases with respect to the radius of the level sets [6], [13]; or the Lyapunov function and the sub-level set (inner approximation of the ROA) can be searched simultaneously [14], [15]. However, there is currently no existing work based on the bi-level optimization training procedure to find the largest ROA.

Paper contributions. Our paper contains several contributions on the theoretical side:

- While there are existing methods to construct Lyapunov

Z. Wang is with the Division of Systems Engineering, S.B. Andersson and R. Tron are with the Division of Systems Engineering and the Department of Mechanical Engineering, Boston University, Boston, MA 02215, USA. {zw2445,sanderss,tron}@bu.edu.

This work was supported in part by NSF FRR-2212051 and the Center for Information and Systems Engineering at Boston University.

neural networks that give positive definiteness [14]–[16], they are in quadratic form and cannot be verified by MILPs. Instead, we provide a Lyapunov ReLU network architecture that is composed of monotonic functions over half-spaces in different directions. The architecture is positive definite and has a unique global minimum.

- The above architecture of the Lyapunov neural network enables a novel approach for finding a controller and its corresponding Lyapunov function with largest possible ROA. Since our architecture has nested level sets, we can enlarge the ROA by enlarging an inscribed cube as a MILP. The maximization of the ROA is done using gradient descent, with the gradient from MILP.

Through simulations, we provide the following key results:

- Compared to [7], we shorten the training time by half. We attribute this to our model structure, which effectively reduces the search space for the Lyapunov function.
- We demonstrate the strength of our approach in expanding the region of attraction in complex 2D to 4D systems. Our result shows larger/more predictable ROA than previous work.

II. Preliminaries

A. Lyapunov's Direct Method for Stability

We briefly review Lyapunov stability theory [2].

Definition 1 (Control Lyapunov Functions): Consider the nonlinear system:

$$x_{t+1} = f(x_t, u_t), \tag{1}$$

where x_t is a state in a domain $\mathcal{X} \subset \mathbb{R}^{n_x}$ and u_t is the control in a input space $\mathcal{U} \subset \mathbb{R}^{n_u}$. Let $\pi(\cdot) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ denote a control policy with $u_t = \pi(x_t)$. An exponentially-stable control-Lyapunov function (CLF) is a function $V : \mathbb{R}^{n_x} \to \mathbb{R}$ that is continuously-differentiable and satisfies

$$V(0) = 0, \tag{2a}$$

$$V(x) > 0, \ \forall x \in \mathcal{R} \setminus \{0\}, \tag{2b}$$

$$V\Big(f\big(x, \pi(x)\big)\Big) - V(x) \leq -\epsilon V(x), \ \forall x \in \mathcal{R} \setminus \{0\}, \tag{2c}$$

over a subset $\mathcal{R} \subset \mathcal{X}$.

B. ReLUs, MILP, Verification, and Learning

We review here how to represent a ReLU network and perform verfication over a MILP.

A Leaky ReLU (Rectified Linear Unit) function has the form

$$\sigma(y) = \max(0, cy), \tag{3}$$

where $0 \leq c < 1$ is a given positive scalar. For a fully-connected neural network with leaky ReLU activation, the input/output relationship can be represented as

$$\begin{aligned} \phi(x;\theta) &= \hat{z}_{k-1} \\ \hat{z}_i &= W_i z_{i-1} + b_i, \quad i = 1, 2, \ldots, k-1, \\ z_i &= \sigma(\hat{z}_i) = \max(\hat{z}_i, c\hat{z}_i), \quad i = 1, 2, \ldots, k-1, \\ z_0 &= x, \end{aligned} \tag{4}$$

where k is the number of layers and θ contains all the network parameters (i.e. all W_i and b_i).

The MILP formulation stems from the only nonlinear part in the network, namely, the ReLU activation function.

Remark 1: If the input to $\sigma(\cdot)$ is bounded ($\hat{z}_{i,lo} \leq \hat{z}_i \leq \hat{z}_{i,up}$), the input/output relationship through ReLU activation can be encoded as mixed-integer linear constraints

$$\begin{aligned} z_i \geq \hat{z}_i, \ z_i \geq c\hat{z}_i, \\ z_i \leq c\hat{z}_i - (c-1)\hat{z}_{i,up}\beta, \\ z_i \leq \hat{z}_i - (c-1)\hat{z}_{i,lo}(\beta - 1), \\ \beta \in \{0, 1\}, \end{aligned} \tag{5}$$

where the binary variable β is active when $\hat{z}_i \geq 0$.

The ReLU network (4) can be represented as a MILP by replacing the activation units with (5). In a multiple-layer network, the continuous variables in the MILP correspond to the input and each layer's output. With a bounded neural network input, the bound of each ReLU neuron input can be computed from the bounds of the connected nodes from the previous layers; these can be obtained by using Interval Arithmetic [12], solving a Linear Programming [17], or solving a MILP problem [18]. The relationship between network input and output is therefore fully captured in mixed-integer linear constraints. Mixed Integer Programs have been widely applied in domains including path planning [19], temporal logical [20], and controller synthesis [7], [21] to solve different problems.

III. Problem Statement

We consider a discrete-time nonlinear dynamical system (1) with bounded state space $\mathcal{D} \subset \mathcal{X}$ and control space $\mathcal{U}$. We assume that the dynamical system is represented by a piecewise linear function (however, as will be shown in Sec. VI, in practice, our techniques are also applicable to ReLU neural network approximation of general systems [22]). In the following, we consider three problems:

Problem I. Verify the satisfaction of the Lyapunov conditions for a given controller $\pi(\cdot)$ and Lyapunov function over a region $\mathcal{D}$.

Problem II. Find an exponentially stable controller $\pi(\cdot)$ and Lyapunov function on $\mathcal{D}$.

Problem III. Find a controller $\pi(\cdot)$ and Lyapunov function such that the ROA is as large as possible.

We use the Lyapunov conditions (2) to imply the system stability. However, the Lyapunov conditions themselves do not directly ensure safety, since the system can deviate arbitrarily far before eventually settling to the stable equilibrium. The region of attraction, however, defines a forward invariant set that is guaranteed to contain all possible trajectories of the system, and therefore establishes safety. A typical under-approximation of the ROA can be found by verifying the Lyapunov conditions on a sub-level set of the Lyapunov function $\mathcal{R}_\rho = \{x | V(x) \leq \rho\}$ for some value of the parameter $\rho > 0$.

In our setting, the dynamic system is represented by a fixed neural network ϕ_{dyn} with leaky ReLU activation functions,

$$f(x, u) = \phi_{dyn}(x, u) - \phi_{dyn}(x_{eq}, u_{eq}) + x_{eq}, \tag{6}$$

where x_{eq}, u_{eq} are the state and control at the equilibrium point. The equation guarantees that at the equilibrium state and control, the next state remains at the equilibrium point.

The control policy is represented by a feedforward, fully connected neural network ϕ_π with leaky ReLU activation,

$$\pi(x) = \text{clamp}(\phi_\pi(x) - \phi_\pi(x_{eq}) + u_{eq}, u_{min}, u_{max}), \quad (7)$$

where u_{min}, u_{max} are the lower and upper control limit, and clamp$(\cdot)$ clamps (element-wise) all the values within the control limits.

The Lyapunov function is represented by another neural network $\varphi_v(x)$,

$$V(x) = \phi_v(x) + \lambda |R(x - x_{eq})|_1 \quad (8)$$

where R is a full rank matrix and λ is a hyperparameter that balances $\phi_v(x)$ and the term containing R. The term containing R is strictly positive everywhere except at x_{eq}. In practice, adding this term facilitates learning, but it is not strictly necessary for our guarantees on $V(\cdot)$.

IV. Lyapunov Neural Network

In the section, we introduce a Lyapunov Neural Network that can be built from ReLU units and has a single global minimum and no other global minima *by construction*. With this architecture, some Lyapunov conditions are automatically satisfied, allowing us to formulate a simpler optimization problem to verify remaining conditions to synthesize the controller and Lyapunov function.

A. Monotonic Layers

The Lyapunov Neural Network is built from *units* that represent monotonically increasing functions; and *layers* that define *star convex* functions and are guaranteed to have a unique global minimum.

Definition 2 (Monotone functions): A function $m(\cdot) : \mathbb{R} \to \mathbb{R}$ is of class $\mathcal{M}$ if it has the following properties:

- $m(\cdot)$ is continuous,
- $m(0) = 0$,
- $m(\cdot)$ is strictly increasing everywhere.

If the last condition holds only for non-negative arguments, $m(\cdot)$ is of class $\mathcal{M}_+$.

A function $m(\cdot)$ is of class $\mathcal{M}^{[p,\varepsilon]}$ if it is of class $\mathcal{M}$ and:

- it is piecewise linear,
- its derivative has $p \in \mathbb{N}$ discontinuity points,
- all defined derivatives $m'(y)$ are greater than ε, i.e., ε is a lower bound on the slope of m.

If the function is of class $\mathcal{M}_+$ and the above conditions hold, then it is of class $\mathcal{M}_+^{[p,\varepsilon]}$. Note that functions of class $\mathcal{M}$ and $\mathcal{M}_+$ are of class $\mathcal{K}$ [2] when restricted on $[0, \infty)$.

Definition 3 (Monotonic unit): A *monotonic unit* $m^{[p,\varepsilon]}$ of order p is defined as

$$m(y) = a^{\mathrm{T}} \,\mathrm{ReLU}(y\mathbf{1} - b), \quad (9)$$

where $a, b \in \mathbb{R}^p$, $\mathbf{1} \in \mathbb{R}^p$ is the vector of all ones, and ReLU is the element-wise Rectifier Linear Unit operator (3).

Proposition 1: A monotonic unit m is of class $\mathcal{M}^{[p,\varepsilon]}$ if:

1) $b \in \mathbb{R}^p$ is a vector of *biases* satisfying $[b]_{i+1} < [b]_i$ where $[\cdot]_j$ is the j-th entry of a vector.
2) $a \in \mathbb{R}^p$ is a vector of *slopes* satisfying
   (a) $[a]_1 > 0$,
   (b) $[a]_{i+1} \geq -\sum_{i'=1}^{i-1} a_{i'} + \varepsilon$.

A monotonic unit m is of class $\mathcal{M}_+^{[p,\varepsilon]}$ if the additive condition $[b]_1 = 0$ is satisfied.

Proof: Expanding the inner product in (9), and using the fact that $\mathrm{ReLU}(y - b_i) = 0$ if $b_i \geq y$, we have

$$m(y) = \sum_{i=1}^{i'} a_i y - \sum_{i}^{i'} a_i b_i, \quad (10)$$

where i' is the largest i such that $b_i < y$, that is, m is piecewise linear in y. Condition (b) implies $\sum_{i=1}^{i''+1} a_i \geq \varepsilon$ for any i''; combined with (a) implies the slope in (10) is always no less than ε, i.e., the function is always monotonically increasing and hence of class $\mathcal{M}^{[p,\varepsilon]}$. If $[b]_1 = 0$, then $m(y) = 0$ for all $y \leq 0$, and the function is of class $\mathcal{M}_+^{[p,\varepsilon]}$. ■

Definition 4 (Monotonic Layer): A *monotonic layer* $M : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ is defined as a combination of monotonic units:

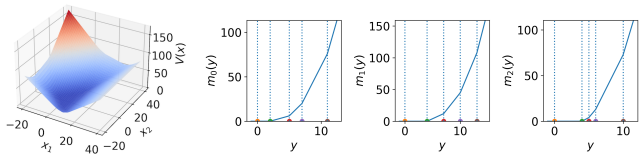
$$[M]_j(x) = \sum_{i=1}^{n_M} c_i m_i(v_i^{\mathrm{T}} x) \quad (11)$$

where $x \in \mathbb{R}^{n_x}$ is the input, $v_i \in \mathbb{R}^{n_x}$ is a direction and $v_i \neq 0$, $c_i > 0$, and $n_M \geq d + 1$.

Lemma 1: For a given direction v_i, the function $m_i(v_i^{\mathrm{T}} \cdot) : \mathbb{R}^{n_x} \to \mathbb{R}$ is:

1) monotonically increasing in the direction v_i,
2) constant along any direction orthogonal to v_i.

The proof is omitted to save space. A monotonic layer is monotonically increasing along radial lines $x(t) = vt$, where $v \in \mathbb{R}^{n_x}$. An example of a monotonic layer in $\mathcal{M}_+^{[p,\varepsilon]}$ is shown in Fig. 1.



(a) Layer

(b) Monotonic units in $\mathcal{M}_+^{[p,\varepsilon]}$

Fig. 1: Illustration of Monotonic Layer $V(x) = [M]_1 = \sum_{i=1}^{3} m_i(v_i^T x)$ with $v_i \in \{[1;0], [-1;1], [-1;-1]\}$, each $m_i(\cdot)$ is shown in (b).

B. Proposed Lyapunov Network

Definition 5 (Lyapunov Neural Network): An n-layer *Lyapunov Neural Network* $\phi_v : \mathbb{R}^{n_x} \to \mathbb{R}$ in (8) is defined as the composition of monotonic layers:

$$\phi_v(x) = (M_n \circ M_{n-1} \circ \ldots \circ M_1)(x)$$

Proposition 2: Assuming $\mathbf{0}$ is in the convex hull of a set of directions $\{v_i^M\}$ for M_1, $\phi_v(x)$ has the following properties:

- $\phi_v(x) \geq 0$, $V(0) = 0 \iff x = 0$,
- $\phi_v(x)$ is monotonically increasing along radial lines,
- $\phi_v(x)$ is radially unbounded, has a unique global minimum at $x = 0$, and no other local minima.

Proof: By definition, M_i is a function that is monotonically increasing along radial lines originating from zero. The composition of multiple such functions inherits this property. Since the monotonic units are of class $\mathcal{M}^{[p,\varepsilon]}$ and $\mathbf{0} \in \text{co}(\{v_i^M\})$ ($\text{co}(\cdot)$ denotes the convex hull of a set), $\phi_v(x)$ is increasing everywhere from zero and radially unbounded. Therefore, it has a unique global minimum at $\phi_v(0) = 0$. ■

Remark 2: The R term in (8)can be written in the form (11). Under this representation, the directions $\{v_i^R\}$ for the R term satisfy the condition $\mathbf{0} \in \text{co}(\{v_i^R\})$. Hence, with the R term, $V(x)$ can be proved to possess the same properties as $\phi_v(x)$, but without requiring that $\mathbf{0} \in \text{co}(\{v_i^M\})$.

Corollary 1: $\phi_v(x)$ is star convex with respect to the origin and has nesting, compact, simply connected, star-convex (with respect to the origin) level sets (i.e., $V^{-1}(v_1) \subset V^{-1}(v_2)$ for any $v_1 < v_2$).

Proof: The function from the origin to any point x is monotonically increasing, hence every point between 0 and x is inside any level set containing x. The claims follow. ■

Remark 3: The bias and slope vectors in the monotonic units can be converted to ReLU network parameters, in a way similar to [23]. To satisfy the monotonic unit properties in Proposition 1, the neural network parameters are thresholded to be positive/non-negative during training.

C. Verification of Lyapunov Conditions

With the proposed structure on $V(x)$, Lyapunov conditions (2a) and (2b) are satisfied by construction.

To ensure the stability of a given controller, one needs to verify the Lyapunov condition (2c) in a given bounded polytope $\mathcal{D}$ around the equilibrium. The verification problem (Problem I) can be formulated as

$$\max_{x \in \mathcal{D}} V\Big(f\big(x, \pi(x)\big)\Big) - V(x) + \epsilon V(x). \tag{12}$$

In our setting, $V(x)$, $f(x, \pi(x))$ and $V(f(x, \pi(x)))$ are piecewise-affine functions of x; therefore, the problem can be solved through MILPs as in Sec. II-B. If the optimal value is zero, the system is certified; otherwise the optimal value is greater than 0 and the optimal point is an counter-example.

D. Learning Procedure

With the contents in Sec. II, the problem of finding a valid controller and Lyapunov function (Problem II) becomes the problem of minimizing the maximum violation of Lyapunov condition (2c). This can be formulated as the bi-level min-max optimization:

$$\min_{\theta} \left(\max_{x \in \mathcal{D}} V(f(x, \pi(x))) - V(x) + \epsilon V(x) \right), \tag{13}$$

where θ contains the trainable controller and Lyapunov network parameters.

Such a problem can be solved in an iterative procedure [12] as shown in Algorithm. 1. At each iteration, with fixed network parameters, the inner maximization problem can be solved using MILP solvers. A general MILP formulation is

$$\begin{aligned} \gamma(\theta) = &\max_{s,\beta} a_\theta^T s + b_\theta \beta \\ &\text{s.t } A_\theta s + B_\theta \beta \leq c_\theta, \end{aligned} \tag{14}$$

where $a_\theta, b_\theta, A_\theta, B_\theta, c_\theta$ are explicit functions of θ, β are binary variables, and s (slack variables and x) are all the continuous variables in the MILP related to (5).

After solving the MILP to optimality, with optimal s^* and β^*, the active linear constraints at the solutions are $A_\theta^{act} s^* + B_\theta^{act} \beta^* = c_\theta^{act}$ [7]. Assuming an infinitesimal change of θ does not change the binary variable solution nor the indices of active constraint, and letting $(A_\theta^{act})^\dagger$ be the pseudo-inverse of A_θ^{act}, then the gradient of the MILP optimal objective with respect to the network parameters, given by

$$\gamma(\theta) = a_\theta^T (A_\theta^{act})^\dagger (c_\theta^{act} - B_\theta^{act} \beta^*) + b_\theta^T \beta^*, \tag{15}$$

can be used to update the network. In the case where the assumption does not hold, the problem is degenerate, but the gradient in (15) can still be used to improve the network. For further discussion, see [7].

The procedure terminates when the inner maximization objective gives a zero value, yielding a valid controller and a corresponding Lyapunov function that solves Problem II.

Algorithm 1 Train controller/Lyapunov function through min-max optimization

```
Given: a candidate control policy π and a candidate Lyapunov function V.
while not converged do
    Solve MILP max_{x∈D} V(f(x, π(x))) − V(x) + εV(x)
    if The MILP has maximal objective > 0 then
        Compute the gradient of the MILP objectives w.r.t network parameters θ as ∂γ/∂θ.
        θ = θ − StepSize × ∂γ/∂θ
    end if
end while
```

V. Region of Attraction Expansion

One of the main benefit of our proposed network is that, by leveraging the special monotonicity structure, we can offer a method to enlarge the ROA. We first introduce a bi-level optimization simliar to (13) but where $\mathcal{D}$ is a fixed sub-level set (used to represent the ROA). We then expand the level set by defining an inscribed square, and maximizing its area. If the expanded level set satisfies all the Lyapunov conditions, then it becomes the new ROA. This iterative procedure addressing Problem III is summarized in Algorithm 2.

A. Training over a Sub-level Set

Instead of finding the maximum violation over a fixed pre-defined bounded region $\mathcal{D}$ as in Sec. IV-C, we are interested in a compact sub-level set $\mathcal{R}_r = \{x | V(x) \leq r\}$, where r is pre-defined and the region of R_r depends on θ. The problem is a modified version of Problem II, where the region is given

Algorithm 2 Expand Region of Attraction with Lyapunov

1: Given: a candidate control policy π and a candidate Lyapunov function V.
2: **while** $\mathcal{R}_r \in \mathcal{D}$ **do**
3: Find a valid controller in $\mathcal{R}_r$ using modified Algorithm 1 (Sec. V-A)
4: **while** less than maximum trials **do**
5: Solve MILP problem (19) to find the inscribed square (Sec. V-B)
6: Compute the gradient of the $l^*(\theta)$ w.r.t. network parameters using $\frac{\partial l^*}{\partial \theta}$
7: Select a moving direction from (25) and implement gradient descent (Sec. V-C)
8: **end while**
9: **end while**

as a level set of the Lyapunov function itself, and a simple modification of Algorithm. 1 can be used to solve it. The problem of finding a valid controller and Lyapunov function over a level set then becomes

$$\min_{\theta} \left(\max_{x \in \mathcal{B}_r, V(x) \leq r} V(f(x, \pi(x))) - V(x) + \epsilon V(x) \right) \quad (16)$$

One difficulty is that, as stated in Remark 1, in order to provide the MILP formulation of the problem, one needs to know the bounded region $\mathcal{B}_r$ that contains R_r (i.e. $R_r \subset \mathcal{B}_r$). To find $\mathcal{B}_r$, we make one additional assumption.

Assumption 1: We assume the monotonic direction set $\{v_i\}$ contains all the axes (with indices set $\mathcal{I}_{axes}$).

Proposition 3: Given a network $\phi_v(x)$ represented as a combination of monotonic functions as $\sum_{i=1}^{n_M} c_i a_i^T \text{ReLU}(v_i^T x - b_i)$, the bounded domain $\mathcal{B}_r$ containing the sub-level set (i.e. $\mathcal{R}_r \subseteq \mathcal{B}_r$) can be found as

$$\begin{aligned} p_i &= m_i^{-1}(r), \forall i \in \mathcal{I}_{axes} \\ \mathbf{p} &= \text{stack}\{p_i\} \\ [\mathcal{B}_r]_j &= |||[\mathbf{p}]_j||_\infty, \ j \leq n_x \end{aligned} \quad (17)$$

where $\mathbf{p} \in \mathbb{R}^{n_M \times n_x}$ and $[\mathcal{B}_r]_j$ denotes the largest value of $\mathbf{p}$ in j_{th} dimension.

Proof: Without loss of generality, choose axis $i = 1$, and define p_1 as the point where $m_1(v_1^T p_1) = r$. Since $V(x) = \sum m_{i'}(x)$ and $m_{i'}(x) \geq 0$ for any i', p_1 is outside of the sub-level set $V^{-1}(r)$, i.e. $V(p_1) > r$. Furthermore, $m_1(v_1^T x)$ is constant along hyperplanes normal to v_1. Hence, any point on the plane $x_1 = [p_1]_1$ has value no smaller than r, and the plane is also outside of $V^{-1}(r)$ (see Fig. 2). Repeating the same process for all directions i, we have that the sub-level set is contained by the set surrounded by all the hyperplanes, i.e., the bounding box that contains $\mathcal{R}_r$. ■

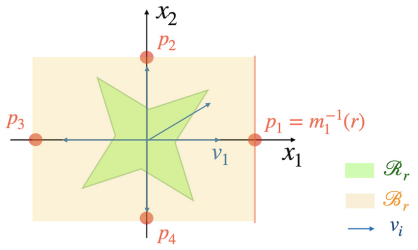


Fig. 2: Visualization for the proof of proposition 3.

B. Approximating a Sub-level Set

The problem of finding the inscribed cube can be formulated as a MILP:

$$\min_{l} l \quad \text{s.t.} \quad V(x) \leq l||x - x_{eq}||_\infty, \ \forall x \in \mathcal{R}_r. \quad (18)$$

The l_∞ norm term can be formulated as mixed-integer linear constraints. Letting

$$g(l) = \max_{x \in \mathcal{R}_r, \ ||x - x_{eq}||_\infty \geq \varepsilon} V(x) - l||x - x_{eq}||_\infty, \quad (19)$$

where ε is a small constant, (18) becomes

$$\min_{l} \quad l \quad \text{s.t.} \quad g(l) \leq 0 \quad (20)$$

From nonlinear optimization theory [24], the optimal solution of (20) is achieved when the constraints are active (i.e., $g(l^*) = 0$); if not, l^* would be unbounded and we could reach any arbitrarily low value. Therefore, the optimal objective l^* can be found by finding the root of (19) using a numerical method such as bisection search [25].

C. Selecting the ROA Expansion Direction

As illustrated in Fig. 3, the sub-level set $\mathcal{R}_r$ can be enlarged by minimizing l^* (which is a function of θ). Specifically, the relationship between l^* and the neural network parameters can be obtained after solving (19) to optimality (i.e. $g(l) = 0$). By fixing all the binary variables to their optimal solutions, and keeping only the active linear constraints, (19) becomes

$$\xi(\theta) = \ c_{\theta,v}^T s_v + d_{\theta,v} - l_\theta^* s_{l_{inf}} \quad (21)$$

where s_v and $s_{l_{inf}}$ need to satisfy the equations

$$A_{\theta,v} s_v = b_{\theta,v}, \quad A_{\theta,l_{inf}} s_{l_{inf}} = b_{\theta,l_{inf}}. \quad (22)$$

Here, s contains all the continuous variables in the MILP (including x and other slack variables), and the subscript v (l_{inf}) indicates the variable is related to $V(x)$ (l_∞). From (21) we have

$$l^* = \frac{c_{\theta,v} s_v + d_{\theta,v}}{s_{l_{inf}}} = \frac{c_{\theta,v}(A_{\theta,v}^{-1} b_{\theta,v}) + d_{\theta,v}}{(A_{\theta,l_{inf}}{}^{-1} b_{\theta,l_{inf}})}. \quad (23)$$

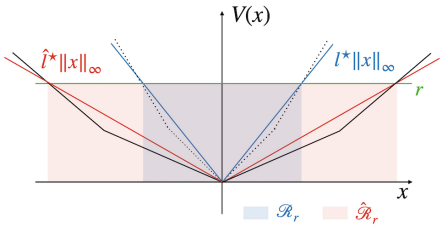


Fig. 3: Illustration of level set expansion on 1D Lyapunov functions: the blue box (identified with $l^\star$) is the original level set and the red one (identified with $\hat{l}^\star$) is after expansion.

We can use gradient descent on l^* with respect to the network parameters θ to enlarge the area of the inscribed square, and hence the area of $\mathcal{R}_r$. However, moving directly in the direction of the gradient might lead to a new $\mathcal{R}_r$ that severely violates the Lyapunov conditions. Therefore, a gradient descent direction w should satisfy

$$\nabla_\theta \gamma(\theta) w \geq 0,\ \nabla_\theta l^*(\theta) w \geq 0. \tag{24}$$

An optimal selection can be formulated as an LP problem

$$\begin{aligned} \max_{w}\ & \nabla_\theta l^*(\theta) w + s \\ \text{s.t. } & \nabla_\theta \gamma(\theta) w \geq s,\ \nabla_\theta l^*(\theta) w \geq 0, \\ & ||w|| \leq 1,\ s \geq 0. \end{aligned} \tag{25}$$

VI. Simulation

In this section, we demonstrate *1)* our proposed monotonic network by comparing the convergence time with a previous approach that used simple ReLU networks to represent the Lyapunov function, *2)* our proposed searching method to find the largest possible ROA, by applying it to four examples: an inverted pendulum, a unicycle path following system, a third-order strict-feedback system, and a cart pole system.

The dynamic model used for learning in each case is a fixed network that was pre-trained on a sufficiently large state space $\mathcal{X}$ to approximate the real dynamical system. Both the controller and Lyapunov networks were initialized by training them to approximate the LQR solution. The initial R term in (8) was composed of slight deviations of the eigenvectors of the solution to the Riccati equation of the LQR problem while ensuring the full rank property. All networks were trained following the approach in [7]. The directions defining the monotonic units in the Lyapunov Network were fixed as a combination of the axis vectors and selected eigenvectors of the solution to the Riccati equation. To verify a large region, we initially verified a small region and gradually enlarged that region. This gave more stable convergence of the algorithms in practice. Note that for running the actual simulations in the results below, the full dynamic model of each system, not its network-based approximation, was used.

A. Systems

System I: Stationary Inverted Pendulum. A second-order nonlinear system with dynamics

$$\ddot{\theta} = \frac{u - mgl\sin(\theta) - b\dot{\theta}}{ml^2}, \tag{26}$$

where $\theta = \pi$ is the upward position of the pendulum and θ is positive in the counter-clockwise direction. The stabilizing equilibrium state is $\theta = \pi$, $\dot{\theta} = 0$. We set the mass term to $m = 1$ and the damping coefficient to $b = 0.1$. The states are $(\theta, \dot{\theta})$ and the state space is

$$\mathcal{D} = \{(\theta, \dot{\theta}) | 0 \leq \theta \leq 2\pi, |\dot{\theta}| \leq 5\}.$$

The control space is $\{u | |u| \leq 10\}$.

System II: Wheeled Vehicle Path Following on a Unit Circle. A curvature-dependent unicycle system with kinematics

$$\begin{pmatrix} \dot{d_e} \\ \dot{\theta_e} \end{pmatrix} = \begin{pmatrix} v\sin\theta_e \\ u - \frac{v\kappa(s)\cos\theta_e}{1 - d_e\kappa(s)}. \end{pmatrix} \tag{27}$$

We set the curvature to $\kappa(s) = 1$ (defining a unit circle), and fix $v = 6$. The stabilizing equilibrium state is $d_e = 0$, $\theta_e = 0$, at which the system has no tracking error. The states are (d_e, θ_e) and the state space is

$$\mathcal{D} = \{(\theta_e, d_e) | |\theta_e| \leq 0.8, |d_e| \leq 0.8\}.$$

The control space is $\{u | |u| \leq 10\}$.

System III: Third-Order Strict Feedback Form. A three-dimensional system with the form

$$\dot{x_1} = e_1 x_2,\ \dot{x_2} = e_2 x_3,\ \dot{x_3} = e_3 x_1^2 + e_4 u, \tag{28}$$

where the states are (x_1, x_2, x_3) and the state space is

$$\mathcal{D} = \{(x_1, x_2, x_3) | |x_1| \leq 1.5, |x_2| \leq 1.5, |x_3| \leq 2\}.$$

The control space is $\{u | |u| \leq 30\}$.

System IV: Cart Pole. A system with dynamics

$$\begin{aligned} &(M+m)\ddot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta = u, \\ &ml^2\ddot{\theta} - mgl\sin\theta = ml\ddot{x}\cos\theta. \end{aligned} \tag{29}$$

The states are $(x, \theta, \dot{x}, \dot{\theta})$, where $\theta = 0$ is the upward position of the pole, and θ is positive in the counter-clockwise direction. The state space is

$$\mathcal{D} = \{(x, \theta, \dot{x}, \dot{\theta}) | |x| \leq 1, |\theta| \leq \pi/6, |\dot{x}| \leq 1, |\dot{\theta}| \leq 1\}.$$

The control space is $\{u | |u| \leq 30\}$. Previous approaches have found it difficult to find a controller for this system.

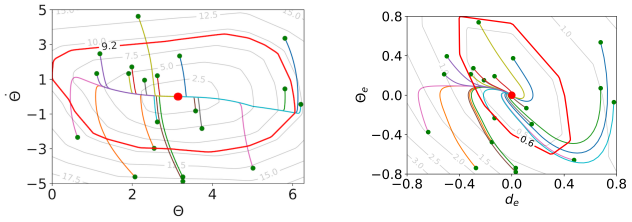
B. Comparison with Baseline

We first demonstrate the efficacy and efficiency of our monotonic network in terms of training convergence time over a fixed bounded state space $\mathcal{D}$ using Algorithm 1. As a baseline for comparison, we use a simple ReLU network [7] (which requires verification of all Lyapunov conditions 2a-2c during training). We apply both approaches to the four systems described in Sec. VI-A and compare them in terms of training time. The network structure for the baseline was selected on a trial-and-error basis for good performance, and our network structured is chosen accordingly to use a comparable number of ReLU units (therefore, the same number of binary variables). We use the same network structure for the controller in each. The network structure and the total training time is presented in the first section of Table. I. These results show a significant improvement in training time, ranging from a reduction of 24% for the 4-D cart-pole system, up to 90% for the 2D path following case.

Fig. 4 shows two examples of the resulting Lyapunov function and controlled trajectories, focusing on the 2D systems for easy visualization.

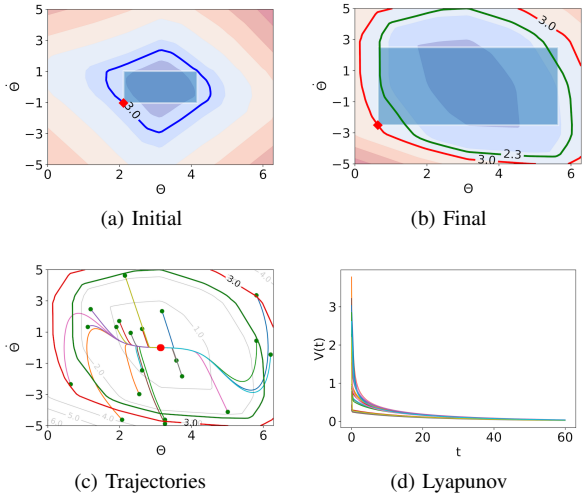
TABLE I: Network Structure and Learning Time: numbers in brackets in the baseline network represent the number of neurons in each layer. Our network has a number of directions (dirs) that define the monotonic functions. Each monotonic function is a piecewise linear function with a number of pieces (pcs). The unit for training time is in minutes.

	Baseline Network	Training Time	Our Network	Training Time	ROA Training Time
Inverted Pendulum	$[8, 8, 6, 1]$	97.2	[5 dirs/4 pcs]	24.3	250.0
Path Following	$[8, 8, 6, 1]$	45.1	[6 dirs/4 pcs]	4.8	32.4
Third-Order Strict	$[8, 8, 6, 1]$	21.4	[7 dirs/4 pcs]	10.1	76.9
Cart Pole	$[8, 7, 5, 1]$	74.8	[7 dirs/4 pcs]	57.2	69.3



(a) Pendulum (b) Path Following

Fig. 4: Final Lyapunov function and example controlled trajectories after applying Algo. 1. (bold red curve) ROA within the bounded domain. (red dot) Equilibrium point. (green dots) Randomly selected initial conditions, and the corresponding system trajectories under the learned controller.



(a) Initial (b) Final

(c) Trajectories (d) Lyapunov

Fig. 5: Illustration of ROA in the inverted pendulum example. (a) $\mathcal{R}_{3.0}$ before expansion is highlighted in blue. (b) $\mathcal{R}_{3.0}$ after expansion is in red; the l_∞ norm is shown as the inscribed blue box, and the intersection between l_∞ norm and $\mathcal{R}_{3.0}$ is shown as a red dot. The level set touching the state space bounds is shown in green ($\mathcal{R}_{2.3}$). (c) Trajectories. (d) Lyapunov values.

C. ROA Expansion

Algo. 2 combines learning the controller, certifying the Lyapunov function, and expanding the ROA.Fig. 5 shows results on System I (the inverted pendulum). Although the training time is longer than with Algorithm 1 (see the last column in Table 1), it can yield a significantly larger ROA. Figs 5-7 show the results for the four example systems.

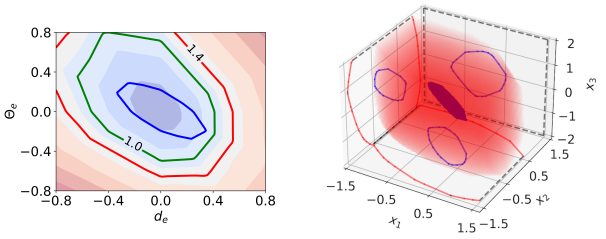
Fig. 5a shows the initial Lyapunov function, the defined sub-level set (chosen to yield a relatively small initial region to allow room for the algorithm to adjust the Lyapunov function and grow the ROA), and the inscribed square for System I. Fig. 5b shows the final result, with the level set shown in bold red. Note that the algorithm allows the ROA to extend beyond the domain $\mathcal{D}$. To guarantee the ROA, we then find the maximal level set that lies entirely within $\mathcal{D}$ (shown in green). The performance of the resulting controller is demonstrated in Fig. 5c-5d, which shows a collection of trajectories from randomly selected initial points (compare against the results from Algo. 1 in Fig. 4a) and the corresponding evolution of the Lyapunov function. Note that some of the selected initial conditions lay outside the ROA but are stabilized nevertheless.

The results of expansion for Systems II and III (Fig. 6) both show a significant increase in the final ROA over the initial one. To avoid overly cluttering the image, the results for Systems III (Fig. 6b) do not include the maximal level set inside $\mathcal{D}$. The results for System IV (the cart-pole) are shown in Figs. 7 and 8. Because this is a 4-D system, we have chosen to show four 3D slices of the ROAs, presenting clear improvement after expansion. It is also interesting to note that the final ROAs appear to be elongated in some dimensions; in future work we intend to consider expansions using inscribed zonotopes to further improve the result. The controlled evolution of the system from a randomly selected set of initial conditions shown in Fig. 8 clearly demonstrate that the system converges to the equilibrium point.

VII. Conclusions

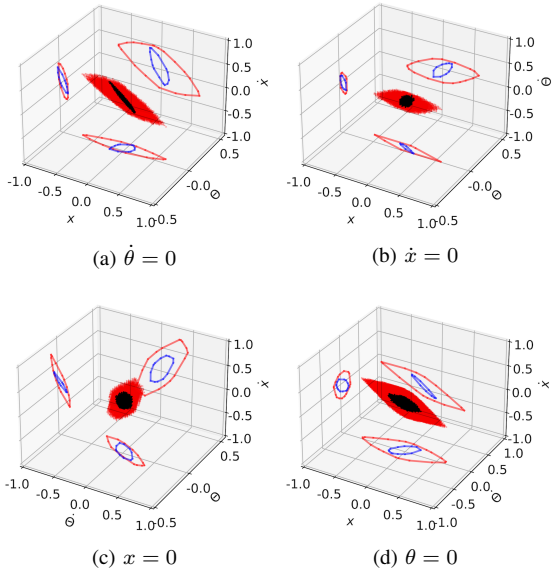
In this paper, we proposed a ReLU neural network that by construction satisfies the non-negativity property of a Lyapunov function. Our algorithm to learn both controller and verifying Lyapunov function for nonlinear systems requires significantly less training time than a baseline. Taking advantage of the inherent properties of our network structure, we developed an algorithm that concurrently expands the region of attraction for the system, ensuring both stability and safety over as large a domain in the state space as possible.

In this work we used only a single layer in the Lyapunov neural network, though the proposed structure is more general and in future work we intend to explore the advantage of the more general function representation, including adding additional layers. We also intend to consider modifications to our expansion algorithm that rely on counter-example guided



(a) Path Following (b) Third Order

Fig. 6: ROA Expansion. (blue) Initial level set. (red) Final level set. (a) System II, where (green) the maximal level set fully within the domain is also shown. (b) System III. Level sets are projected onto the corresponding planes. Dashed black line indicates the projection is outside of $\mathcal{D}$.



(a) $\dot{\theta} = 0$ (b) $\dot{x} = 0$

(c) $x = 0$ (d) $\theta = 0$

Fig. 7: ROA Expansion on System IV. Each plot fixes one of the states to visualize the corresponding 3D slice.

training, similar to [7], instead of the min-max approach we currently use. Finally, our proposed method does not consider any uncertainties in the model. In the future, we plan to analyze the neural network approximations with model errors, using a robust control Lyapunov function approach [16]. We also plan to apply our approach in real systems, combining it with control barrier functions to avoid obstacles.

References

[1] W. M. Haddad and V. Chellaboina, *Nonlinear dynamical systems and control: a Lyapunov-based approach*. Princeton university press, 2008.

[2] H. K. Khalil, *Nonlinear systems; 3rd ed.* Prentice-Hall, 2002.

[3] A. A. Ahmadi and A. Majumdar, "Some applications of polynomial optimization in operations research and real-time decision making," *Optimization Letters*, vol. 10, pp. 709–729, 2016.

[4] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control design along trajectories with sums of squares programming," *IEEE International Conference on Robotics and Automation*, pp. 4054–4061, 2012.

[5] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control," *IEEE Transactions on Robotics*, vol. 39, pp. 1749–1767, 2022.

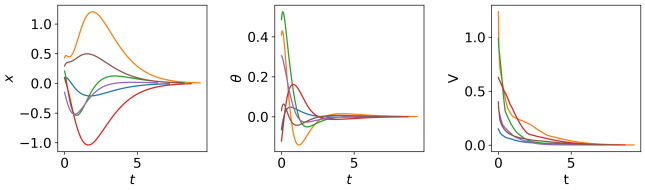


Fig. 8: Controlled trajectories on System IV.

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

[7] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, "Lyapunov-stable neural-network control," in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[8] J. Wu, A. Clark, Y. Kantaros, and Y. Vorobeychik, "Neural lyapunov control for discrete-time systems," *arXiv preprint arXiv:2305.06547*, 2023.

[9] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer, *et al.*, "Algorithms for verifying deep neural networks," *Foundations and Trends® in Optimization*, vol. 4, no. 3-4, pp. 244–404, 2021.

[10] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda, "A unified view of piecewise linear neural network verification," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[11] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *IEEE Transactions on Automatic Control*, vol. 67, no. 1, pp. 1–15, 2020.

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

[13] R. Zhou, T. Quartz, H. De Sterck, and J. Liu, "Neural lyapunov control of unknown nonlinear systems with stability guarantees," *Advances in Neural Information Processing Systems*, vol. 35, pp. 29 113–29 125, 2022.

[14] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Conference on Robot Learning*, 2018.

[15] S. Wei, P. Krishnamurthy, and F. Khorrami, "Neural lyapunov control for nonlinear systems with unstructured uncertainties," *2023 American Control Conference*, pp. 1901–1906, 2023.

[16] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado, "Learning region of attraction for nonlinear systems," in *IEEE Conference on Decision and Control*, 2021, pp. 6477–6484.

[17] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

[18] M. Fischetti and J. Jo, "Deep neural networks and mixed integer linear optimization," *Constraints*, vol. 23, no. 3, pp. 296–309, 2018.

[19] D. Ioan, I. Prodan, S. Olaru, F. Stoican, and S.-I. Niculescu, "Mixed-integer programming in motion planning," *Annual Reviews in Control*, vol. 51, pp. 65–87, 2021.

[20] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *IEEE Conference on Decision and Control*, 2014, pp. 81–87.

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

[22] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993.

[23] Z. Wang, S. B. Andersson, and R. Tron, "Bearing-based formation control with optimal motion trajectory," in *IEEE American Control Conference*, 2022, pp. 486–493.

[24] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[25] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical analysis*. Cengage learning, 2015.