Lyapunov-stable neural-network control

Hongkai Dai *, Benoit Landry†, Lujie Yang‡, Marco Pavone† and Russ Tedrake*‡
* Toyota Research Institute
†Stanford University
‡Massachusetts Institute of Technology
Email: hongkai.dai@tri.global, blandry@stanford.edu lujie@mit.edu pavone@stanford.edu russ.tedrake@tri.global

***Abstract*—Deep learning has had a far reaching impact in robotics. Specifically, deep reinforcement learning algorithms have been highly effective in synthesizing neural-network controllers for a wide range of tasks. However, despite this empirical success, these controllers still lack theoretical guarantees on their performance, such as Lyapunov stability (i.e., all trajectories of the closed-loop system are guaranteed to converge to a goal state under the control policy). This is in stark contrast to traditional model-based controller design, where principled approaches (like LQR) can synthesize stable controllers with provable guarantees. To address this gap, we propose a generic method to synthesize a Lyapunov-stable neural-network controller, together with a neural-network Lyapunov function to simultaneously certify its stability. Our approach formulates the Lyapunov condition verification as a mixed-integer linear program (MIP). Our MIP verifier either certifies the Lyapunov condition, or generates counter examples that can help improve the candidate controller and the Lyapunov function. We also present an optimization program to compute an inner approximation of the region of attraction for the closed-loop system. We apply our approach to robots including an inverted pendulum, a 2D and a 3D quadrotor, and showcase that our neural-network controller outperforms a baseline LQR controller. The code is open sourced at https://github.com/StanfordASL/neural-network-lyapunov.**

I. Introduction

The last few years have seen sweeping popularity of applying neural networks to a wide range of robotics problems [48], such as perception [30, 40, 19], reasoning [16] and planning [25]. In particular, researchers have had great success training control policies with neural networks on different robot platforms [32, 50, 23, 27]. Typically these control policies are obtained through reinforcement learning (RL) algorithms [49, 44, 22]. Although immensely successful, these neural-network controllers still generally lack theoretical guarantees on their performance, which could hinder their adoption in many safety-critical applications.

A crucial guarantee currently missing for neural-network controllers is the stability of the closed-loop system, especially Lyapunov stability. A system is regionally stable in the sense of Lyapunov if starting from any states within a region, the system eventually converges to an equilibrium. This region is called the region of attraction (ROA) [46]. Lyapunov stability provides a strong guarantee on the asymptotic behavior of the system for any state within the region of attraction. It is well known that a system is Lyapunov stable if and only if there exists a *Lyapunov function* [46] that is strictly positive definite and strictly decreasing everywhere except at the goal equilibrium state. Therefore, our goal is to synthesize a pair: a neural-network controller to stabilize the system, and a Lyapunov function to certify its stability.

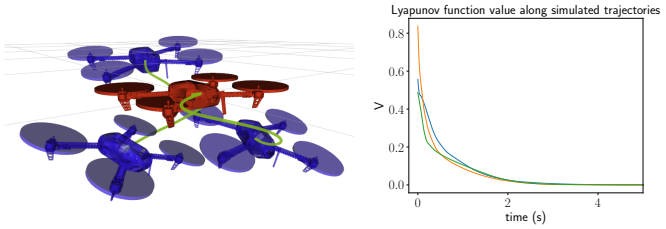
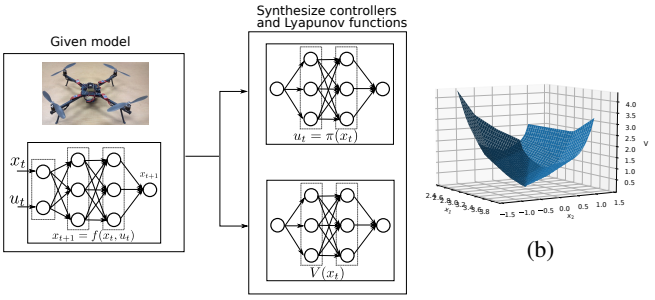


Fig. 1: (left) Snapshots of stabilizing a 3D quadrotor with our neural-network controller to the hovering position at the origin (red snapshot) from different initial states. The green curves are the paths of the quadrotor center. (right) value of the neural-network Lyapunov functions along the simulated trajectories. The Lyapunov function has positive values, and decreases along the trajectories.

In the absence of neural networks in the loop, a significant body of work from the control community provides tools to synthesize Lyapunov-stable controllers [46, 9]. For example, for a linear dynamical system, one can synthesize a linear LQR controller to achieve Lyapunov stability (with the quadratic Lyapunov function solved through the Riccati equation). For a control-affine system with polynomial dynamics, Javis-Wloszek et al. [26] and Majumdar et al. [35] have demonstrated that a Lyapunov-stable controller together with a Lyapunov function, both polynomial functions of the state, can be obtained by solving a sum-of-squares (SOS) program. Recently, for more complicated systems, researchers have started to represent Lyapunov functions (but not their associated controllers) using neural networks. For example, Chang et al. synthesized linear controllers and neural network Lyapunov functions for simple nonlinear systems [11]. In a similar spirit, there is growing interest to approximate the system dynamics with neural networks, such as for racing cars [56], actuators with friction/stiction [24], perceptual measurement like keypoints [36], system with contacts [41], and soft robots [21], where an accurate Lagrangian dynamics model is hard to obtain, while the neural-network dynamics model can be extracted from rich measurement data. Hence we are interested in systems whose dynamics are given as a neural network.

Unlike previous work which is restricted to linear [9, 11]



(a)

(b)

Fig. 2: (Left) The forward system[1](which contains a neural network) is given, and we aim at finding the controller and a Lyapunov function to prove Lyapunov stability of the closed-loop system. Both the controller and Lyapunov function contain neural networks. (right) Visualization of a Lyapunov function for a 2-dimensional system. The Lyapunov function is usually a bowl-shaped function that is strictly positive except at the goal state.

or polynomial controllers [26, 35], our paper provides a novel approach to synthesize a stable neural-network controller, together with a neural-network Lyapunov function, for a given dynamical system whose forward dynamics is approximated by another neural network. The overall picture together with a Lyapunov function is visualized in Fig. 2.

In order to synthesize neural-network controllers and Lyapunov functions, one has to first be able to verify that the neural-network functions satisfy the Lyapunov condition for *all* states within a region. There are several techniques to verify certain properties of neural network outputs for all inputs within a range. These techniques can be categorized by whether the verification is exact, e.g., using Satisfiability Modulo Theories (SMT) solvers [29, 11, 1] or mixed-integer programs (MIP) solvers [10, 52, 14, 12], versus inexact verification by solving a relaxed convex problem [7, 17, 57, 45]. Another important distinction among these techniques is the activation functions used in the neural networks. For example, Abate et al. [1] and Chang et al. [11] learn neural-network Lyapunov functions with quadratic and *tanh* activation functions respectively. On the other hand, the piecewise linear nature of (leaky) ReLU activation implies that the input and output of a (leaky) ReLU network satisfy mixed-integer linear constraints, and hence network properties can be exactly verified by MIP solvers [52, 14]. In this work, due to its widespread use, we choose the (leaky) ReLU unit for all neural networks. This enables us to perform exact verification of the Lyapunov condition without relaxation for safety-critical robot missions.

The verifiers (both SMT and MIP solvers) can either definitively certify that a given candidate function satisfies the Lyapunov condition everywhere in the region, or generate counter examples violating the Lyapunov condition. In this work, we solve MIPs to find the most adversarial counter examples, namely the states with the maximal violation of the Lyapunov condition. Then, in order to improve the satisfaction of the Lyapunov conditions, we propose two approaches to jointly train the controller and the Lyapunov function. The first approach is a standard procedure in counter-example guided training, where we add the counter examples to the training set and minimize a surrogate loss function of the Lyapunov condition violation on this training set [1, 11, 42]. The second approach is inspired by the bi-level optimization community [6, 31, 14], where we directly minimize the maximal violation as a min-max problem through gradient descent.

Our contributions include: 1) we synthesize a Lyapunov-stable neural-network controller together with a neural-network Lyapunov function. To the best of our knowledge, this is the first work capable of doing this. 2) We compute an inner approximation of the region of attraction for the closed-loop system. 3) We present two approaches to improve the networks based on the counter examples found by the MIP verifier. 4) We demonstrate that our approach can successfully synthesize Lyapunov-stable neural-network controllers for systems including inverted pendulums, 2D and 3D quadrotors, and that they outperform a baseline LQR controller.

II. Problem Statement

We consider a discrete-time system whose forward dynamics is

$$x_{t+1} = f(x_t, u_t) = \phi_{\text{dyn}}(x_t, u_t) - \phi_{\text{dyn}}(x^*, u^*) + x^* \quad \text{(1a)}$$

$$u_{\min} \le u_t \le u_{\max} \quad \text{(1b)}$$

where $x_t \in \mathbb{R}^{n_x}, u_t \in \mathbb{R}^{n_u}$, $u_{\min}$ and $u_{\max}$ are the lower/upper input limits. ϕ_{dyn} is a feedforward fully connected neural network with leaky ReLU activation functions [2] [3]. x^*/u^* are the state/control at the goal equilibrium. By definition the dynamics equation (1a) guarantees that at the equilibrium state/control $x_t = x^*, u_t = u^*$, the next state x_{t+1} remains the equilibrium state $x_{t+1} = x^*$. Due to the universal approximation theorem [33], we can approximate an arbitrary smooth dynamical system written as (1a) with a neural network. Our goal is to find a control policy $u_t = \pi(x_t)$ and a Lyapunov function $V(x_t) : \mathbb{R}^{n_x} \to \mathbb{R}$, such that the following Lyapunov conditions are satisfied:

$$V(x_t) > 0 \; \forall x_t \in \mathcal{S}, x_t \neq x^* \quad \text{(2a)}$$

$$V(x_{t+1}) - V(x_t) \le -\epsilon_2 V(x_t) \; \forall x_t \in \mathcal{S}, x_t \neq x^* \quad \text{(2b)}$$

$$V(x^*) = 0 \quad \text{(2c)}$$

where $\mathcal{S}$ is a compact sub-level set $\mathcal{S} = \{x_t | V(x_t) \le \rho\}$, and $\epsilon_2 > 0$ is a given positive scalar. The Lyapunov conditions in (2) guarantee that starting from any state inside $\mathcal{S}$, the state converges exponentially to the equilibrium state x^*, and $\mathcal{S}$ is a region of attraction of the closed-loop system.

[1]The quadrotor picture is taken from [8].

[2]Since ReLU can be regarded as a special case of leaky ReLU, we present our work with leaky ReLU for generality.

[3]Our approach can also handle other architectures such as convolution. For simplicity of presentation we don't discuss them in this paper.

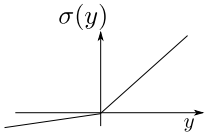


Fig. 3: A leaky ReLU activation function.

In addition to the control policy and the Lyapunov function, we will find an inner approximation of the region of attraction. Note that condition (2b) is a constraint on the Lyapunov function $V(\cdot)$ as well as the control policy $\pi(\cdot)$, since $V(x_{t+1}) = V(f(x_t, \pi(x_t)))$ depends on both the control policy to compute x_{t+1} together with the Lyapunov function $V(\cdot)$.

III. Background on ReLU and MIP

In this section we give a brief overview of the mixed-integer linear formulation which encodes the input/output relationship of a neural network with leaky ReLU activation. This MIP formulation arises from the network output being a piecewise-affine function of the input, hence intuitively one can use linear constraints for each affine piece, and binary variables for the activated piece. Previously researchers have solved mixed-integer programs (MIP) to verify certain properties of the feedforward neural network in machine learning applications such as verifying image classifiers [10, 52].

For a general fully-connected neural network, the input/output relationship in each layer is

$$z_i = \sigma(W_i z_{i-1} + b_i), i = 1, \ldots, n-1 \tag{3a}$$
$$z_n = W_n z_{n-1} + b_n, z_0 = x, \tag{3b}$$

where W_i, b_i are the weights/biases of the i'th layer. The activation function $\sigma(\cdot)$ is the leaky ReLU function shown in Fig.3, as a piecewise linear function $\sigma(y) = \max(y, cy)$ where $0 \le c < 1$ is a given scalar. If we suppose that for one leaky ReLU neuron, the input $y \in \mathbb{R}$ is bounded in the range $y_{\text{lo}} \le y \le y_{\text{up}}$ (where $y_{\text{lo}} < 0$ and $y_{\text{up}} > 0$), then we can use the big-M technique to write out the input/output relationship of a leaky ReLU unit $w = \sigma(y)$ as the following mixed-integer linear constraints

$$w \ge y, \quad w \ge cy \tag{4a}$$
$$w \le cy - (c-1)y_{\text{up}}\beta, \quad w \le y - (c-1)y_{\text{lo}}(\beta - 1) \tag{4b}$$
$$\beta \in \{0, 1\}, \tag{4c}$$

where the binary variable β is active when $y \ge 0$. Since the only nonlinearity in the neural network (3) is the leaky ReLU unit $\sigma(\cdot)$, by replacing it with constraints (4), the relationship between the network output z_n and input x is fully captured by mixed-integer linear constraints.

We expect *bounded* input to the neural networks since we care about states within a neighbourhood of the equilibrium so as to prove regional Lyapunov stability, and the system input u_t is restricted within the input limits (Eq. (1b)). With a bounded neural network input, the bound of each ReLU neuron input can be computed by either *Interval Arithmetic* [57], by solving a linear programming (LP) problem [52], or by solving a mixed-integer linear programming (MILP) problem [13, 18].

After formulating neural network verification as a mixed-integer program (MIP), we can efficiently solve MIPs to global optimality with off-the-shelf solvers, such as Gurobi [39] and CBC [20] via branch-and-cut method.

IV. Approach

In this section we present our approach to finding a pair of neural networks as controller and the Lyapunov function. We will first use the technique described in the previous section III, and demonstrate that one can verify the Lyapunov condition (2) through solving MIPs. Then we will present two approaches to reduce the Lyapunov condition violation using the MIP results. Finally we explain how to compute an inner-approximation of the region of attraction.

A. Verify Lyapunov condition via solving MIPs

We represent the Lyapunov function with a neural network $\phi_V : \mathbb{R}^{n_x} \to \mathbb{R}$ as

$$V(x_t) = \phi_V(x_t) - \phi_V(x^*) + |R(x_t - x^*)|_1, \tag{5}$$

where R is a matrix with full column rank. $|R(x_t - x^*)|_1$ is the 1-norm of the vector $R(x_t - x^*)$. Eq. (5) guarantees $V(x^*) = 0$, hence condition (2c) is trivially satisfied. Notice that even without the term $|R(x_t - x^*)|_1$ in (5), the Lyapunov function would still satisfy $V(x^*) = 0$, but adding this 1-norm term assists $V(\cdot)$ in satisfying the Lyapunov condition $V(x_t) > 0$. As visualized in Fig 4, $\phi_V(x_t) - \phi_V(x^*)$ is a piecewise-affine function of x_t passing through the point $(x^*, 0)$. Most likely $(x^*, 0)$ is in the interior of one of the linear pieces, instead of on the boundary of a piece; hence locally around x^*, the term $\phi_V(x_t) - \phi_V(x^*)$ is a linear function of x_t, which will become negative away from x^*, violating the positivity condition $V(x_t) > 0$ ((2a)). To remedy this, we add the term $|R(x_t - x^*)|_1$ to the Lyapunov function. Due to R being full-rank, this 1-norm is strictly positive everywhere except at x^*. With sufficiently large R, we guarantee that at least locally around x^* the Lyapunov function is positive. Notice that $V(x_t)$ is a piecewise-affine function of x_t.

Our approach will entail searching for both the neural network ϕ_V and the full column-rank matrix R in (5). To guarantee R being full column-rank, we parameterize it as

$$R = U\left(\Sigma + \text{diag}(r_1^2, \ldots, r_{n_x}^2)\right) V^T, \tag{6}$$

where U, V are given orthonormal matrices, Σ is a given diagonal matrix with strictly positive diagonal entries, and scalars $r_1, \ldots, r_{n_x}$ are free variables. The parameterization (6) guarantees R being full column-rank since the minimal singular value of R is strictly positive.

We represent the control policy using a neural network $\phi_\pi : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ as

$$u_t = \pi(x_t) = \text{clamp}\left(\phi_\pi(x_t) - \phi_\pi(x^*) + u^*, u_{\min}, u_{\max}\right), \tag{7}$$

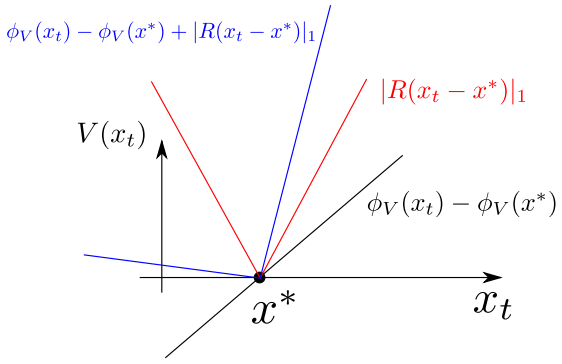


Fig. 4: The term $\phi_V(x_t) - \phi_V(x^*)$ is a piecewise-affine function that passes through $(x_t, V(x_t)) = (x^*, 0)$. Most likely x^* is within the interior of a linear piece, but not at the boundary between pieces. This linear piece will go negative in the neighbourhood of x^*. By adding the 1-norm $|R(x_t - x^*)|_1$ (red lines), the Lyapunov function (blue lines) is at least locally positive around x^*.

where clamp$(\cdot)$ clamps the value $\phi_\pi(x_t) - \phi_\pi(x^*) + u^*$ elementwisely within the input limits $[u_{\min}, u_{\max}]$, namely

$$\text{clamp}(\alpha, \text{lo}, \text{up}) = \begin{cases} \text{up if } \alpha > \text{up} \\ \alpha \text{ if lo} \le \alpha \le \text{up} \\ \text{lo if } \alpha < \text{lo} \end{cases}. \tag{8}$$

The control policy (7) is a piecewise-affine function of the state x_t. Notice that (7) guarantees that at the equilibrium state $x_t = x^*$, the control action is $u_t = u^*$.

It is worth noting that our approach is only applicable to systems that can be stabilized by *regular* (e.g., locally Lipschitz bounded) controllers. Some dynamical systems, for example a unicycle, require non-regular controllers for stabilization, where our approach would fail. The readers can refer to [47] for more background on regular controllers.

The Lyapunov condition (2), in particular, (2a), is a strict inequality. To verify this through MIP which only handles non-strict inequalities constraints $\ge$ and $\le$, we change condition (2a) to the following condition with $\ge$

$$V(x_t) \ge \epsilon_1 |R(x_t - x^*)|_1 \; \forall x \in \mathcal{S}, \tag{9}$$

where $0 < \epsilon_1 < 1$ is a given positive scalar. Since R is full column-rank, the right-hand side is 0 only when $x_t = x^*$. Hence the non-strict inequality constraint (9) is a sufficient condition for the strict inequality constraint (2a). In Appendix VII-C we prove that it is also a necessary condition.

In order to verify the Lyapunov condition (2) for a candidate Lyapunov function and a controller, we consider verifying the condition (9) and (2b) for a given bounded polytope $\mathcal{B}$ around the equilibrium state. The verifier solves the following optimization problems

$$\max_{x_t \in \mathcal{B}} \epsilon_1 |R(x_t - x^*)|_1 - V(x_t) \tag{10a}$$

$$\max_{x_t \in \mathcal{B}} V(x_{t+1}) - V(x_t) + \epsilon_2 V(x_t), \tag{10b}$$

where the objectives are the violation of condition (9) and (2b) respectively. If the optimal values of both problems are 0 (attained at $x_t = x^*$), then we certify the Lyapunov condition (2). The objective in (10a) is a piecewise-affine function of the variable x_t since both $V(x_t)$ and $|R(x_t - x^*)|_1$ are piecewise-affine. Likewise in optimization problem (10b), since the control policy (7) is a piecewise-affine functions of x_t, and the forward dynamics (1a) is a piecewise-affine function of x_t and u_t, the next state $x_{t+1} = f(x_t, \pi(x_t))$, its Lyapunov value $V(x_{t+1})$ and eventually the objective function in (10b) are all piecewise-affine functions of x_t. It is well known in the optimization community that one can maximize a piecewise-affine function within a bounded domain ($\mathcal{B}$ in this case) through solving an MIP [55]. In section III we have shown the MIP formulation on neural networks with leaky ReLU units; in Appendix VII-B we present the MIP formulation for the 1-norm in $|R(x_t - x^*)|_1$ and the clamp function in the control policy.

By solving the mixed-integer programs in (10), we either verify that the candidate controller is Lyapunov-stable with the candidate Lyapunov function $V(x_t)$ as a stability certificate; or we generate counter examples of x_t, where the objective values are positive, hence falsify the candidates. By maximizing the Lyapunov condition violation in the MIP (10), we find not only *a* counter example if one exists, but the *worst* counter example with the largest violation. Moreover, since the MIP solver traverses a binary tree during branch-and-cut, where each node of the tree might find a counter example, the solver finds a list of counter examples during the solving process. In the next subsection, we use both the worst counter example and the list of all counter examples to reduce the Lyapunov violation.

B. Trainer

After the MIP verifier generates counter examples violating Lyapunov conditions, to reduce the violation, we use these counter examples to improve the candidate control policy and the candidate Lyapunov function. We present two iterative approaches. The first one minimizes a surrogate function on a training set, and the counter examples are appended to the training set in each iteration. This technique is widely used in the counter-example guided training [11, 1, 12, 28]. The second approach minimizes the maximal Lyapunov condition violation directly by solving a min-max problem through gradient descent. In both approaches, we denote the parameters we search for as θ, including

- The weights/biases in the controller network ϕ_π;
- The weights/biases in the Lyapunov network ϕ_V;
- $r_1, \ldots, r_{n_x}$ in the full column-rank matrix R (Eq. (6)).

namely we optimize both the control policy and the Lyapunov function simultaneously, so as to satisfy the Lyapunov condition on the closed-loop system.

1) Approach 1, growing training set with counter examples: A necessary condition for satisfying the Lyapunov condition for any state in $\mathcal{B}$, is that the Lyapunov condition holds for many sampled states within $\mathcal{B}$. Hence we could reduce a

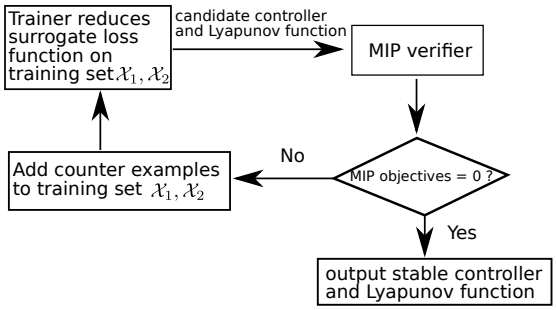


Fig. 5: Flow chart of Algorithm 1.

surrogate loss function on a training set $\mathcal{X}$ containing sampled states. The training set $\mathcal{X}$ grows after each MIP solve by appending the counter examples generated from the MIP solve. Since the MIP (10a) and the MIP (10b) generate different counter examples, we keep two separate training sets $\mathcal{X}_1$ and $\mathcal{X}_2$ for MIP (10a) and MIP (10b) respectively.

We design a surrogate loss function for $\mathcal{X}_1, \mathcal{X}_2$ to measure the violation of the Lyapunov condition on the training set. We denote the violation of condition (9) on a sample state $x_1^i \in \mathcal{X}_1$ as $\eta_1(x_1^i)$, and the violation of condition (2b) on a sample state $x_2^i \in \mathcal{X}_2$ as $\eta_2(x_2^i)$, defined as

$$\eta_1(x_1^i) = \max(\epsilon_1 |R(x_1^i - x^*)|_1 - V(x_1^i), 0) \tag{11a}$$

$$\eta_2(x_2^i) = \max(V(f(x_2^i, \pi(x_2^i))) - V(x_2^i) + \epsilon_2 V(x_2^i), 0), \tag{11b}$$

We denote $\eta_1(\mathcal{X}_1)$ and $\eta_2(\mathcal{X}_2)$ as the vectors whose i'th entry is the violation on the i'th sample $\eta_1(x_1^i)$ and $\eta_2(x_2^i)$ respectively, then our surrogate function is defined as

$$\text{loss}_\theta(\mathcal{X}_1, \mathcal{X}_2) = |\eta_1(\mathcal{X}_1)|_p + |\eta_2(\mathcal{X}_2)|_p, \tag{12}$$

where $|\cdot|_p$ denotes the p-norm of a vector, such as 1-norm (mean of the violation), $\infty-$norm (maximal of the violation) and $4-$norm (a smooth approximation of the $\infty-$ norm). The subscript θ in the loss function (12) emphasizes its dependency on θ, the parameters in both the controller and the Lyapunov function. We then minimize the surrogate loss function on the training set via standard batched gradient descent on θ. The flow chart of this approach is depicted in Fig. 5. Algorithm 1 presents the pseudo-code.

Since the surrogate loss function is the Lyapunov condition violation on just the sampled states, the batched gradient descent will overfit to the training set, and potentially cause large violation away from the sampled states. To avoid this overfitting problem, we consider an alternative approach without constructing the training sets.

2) Approach 2, minimize the violation via min-max program: Instead of minimizing a surrogate loss function on a training set, we can minimize the Lyapunov condition violation

Algorithm 1 Train controller/Lyapunov function on training sets constructed from verifier

Start with a candidate neural-network controller π, a candidate Lyapunov function V, and training sets $\mathcal{X}_1, \mathcal{X}_2$.
while not converged **do**
  Solve MIPs (10a) and (10b).
  if MIP (10a) or MIP (10b) has maximal objective > 0 **then**
    if MIP (10a) maximal objective > 0 **then**
      Add the counter examples from MIP (10a) to $\mathcal{X}_1$.
    end if
    if MIP (10b) maximal objective > 0 **then**
      Add the counter examples from MIP (10b) to $\mathcal{X}_2$.
    end if
    Perform batched gradient descent on the parameters θ to reduce the loss function (12) on the training set $\mathcal{X}_1, \mathcal{X}_2$. Stop until either $\text{loss}_\theta(\mathcal{X}_1, \mathcal{X}_2) = 0$, or reaches a maximal epochs.
  else
    converged = true.
  end if
end while

directly through the following min-max problem

$$\min_\theta \left(\underbrace{\max_{x_t \in \mathcal{B}} \epsilon_1 |R(x_t - x^*)|_1 - V(x_t)}_{\text{MIP (10a)}} + \underbrace{\max_{x_t \in \mathcal{B}} V(x_{t+1}) - V(x_t) + \epsilon_2 V(x_t)}_{\text{MIP (10b)}} \right), \tag{13}$$

where θ are the parameters in the controller and the Lyapunov function, introduced at the beginning of this subsection IV-B. Unlike the traditional optimization problem, where the objective function is a closed-form expression of the decision variable θ, in our problem (13) the objective function is the result of other maximization problems, whose coefficients and bounds of the constraint/cost matrices depend on θ. In order to solve this min-max problem, we adopt an iterative procedure. In each iteration we first solve the inner maximization problem using MIP solvers, and then compute the gradient of the MIP optimal objective w.r.t the variables θ, finally we apply gradient descent along this gradient direction, so as to reduce the objective in the outer minimization problem.

To compute the gradient of the maximization problem objective w.r.t θ, after solving the inner MIP to optimality, we fix all the binary variables to their optimal solutions, and keep only the active linear constraints. The inner maximization

problem can then be simplified to

$$\gamma(\theta) = \max_s c_\theta^T s + d_\theta \tag{14a}$$

$$\text{s.t } A_\theta s = b_\theta, \tag{14b}$$

where the problem coefficients/bounds $c_\theta, d_\theta, A_\theta, b_\theta$ are all explicit functions of θ. s contains all the continuous variables in the MIP, including x_t and other slack variables. The optimal cost of (14) can be written in the closed form as $\gamma(\theta) = c_\theta^T A_\theta^{-1} b_\theta + d_\theta$, and then we can compute the gradient $\partial\gamma(\theta)/\partial\theta$ by back-propagating this closed-form expression. Note that this gradient is well defined if a tiny perturbation on θ changes only the optimal value of the continuous variables s, but not the set of active constraints or the optimal binary variable values (changing them would make the gradient ill-defined). This technique to differentiate the optimization objective w.r.t neural network parameters is becoming increasingly popular in the deep learning community. The interested readers can refer to [4, 2] for a more complete treatment on differentiating an optimization layer.

Algorithm 2 shows pseudo-code for this min-max optimization approach.

Algorithm 2 Train controller/Lyapunov function through min-max optimization

1: Given a candidate control policy π and a candidate Lyapunov function V.
2: **while** not converged **do**
3: Solve MIP (10a) and (10b).
4: **if** Either of MIP (10a) of (10b) has maximal objective > 0 **then**
5: Compute the gradient of the MIP objectives w.r.t θ, denote this gradient as $\partial\gamma/\partial\theta$.
6: $\theta = \theta - \text{StepSize} * \partial\gamma/\partial\theta$.
7: **end if**
8: **end while**

C. Computing region of attraction

After the training process in section IV-B converges to satisfy the Lyapunov condition for every state inside the bounded polytope $\mathcal{B}$, we compute an inner approximation of the region of attraction for the closed-loop system. (Notice that the verified region $\mathcal{B}$ is *not* a region of attraction, since it's not an invariant set, while the sub-level sets of V are guaranteed to be invariant). One valid inner approximation is the largest sub-level set $\mathcal{S} = \{x_t | V(x_t) \le \rho\}$ contained inside the verified region $\mathcal{B}$, as illustrated in Fig. 6. Since we already obtained the Lyapunov function $V(x_t)$ in the previous section, we only need to find the largest value of ρ such that $\mathcal{S} \subset \mathcal{B}$. Equivalently we can find ρ through the following optimization problem

$$\rho = \min_{x_t \in \partial\mathcal{B}} V(x_t), \tag{15}$$

where the compact set $\partial\mathcal{B}$ is the boundary of the polytopic region $\mathcal{B}$, and the constraint $x_t \in \partial\mathcal{B}$ can be formulated as mixed-integer linear constraints (with one binary variable for a face of the polytope $\mathcal{B}$). As explained previously, the Lyapunov function $V(x_t)$ is a piecewise-affine function of x_t, hence the optimization problem (15) is again an MIP, and can be solved efficiently by MIP solvers.

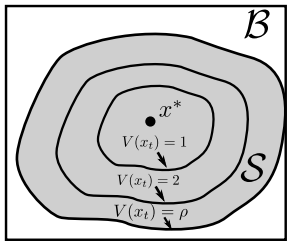


Fig. 6: An inner approximation of the region of attraction $\mathcal{S}$ is the largest sub-level set $V(x_t) \le \rho$ contained inside the verified region $\mathcal{B}$, where the Lyapunov function is positive definite and strictly decreasing.

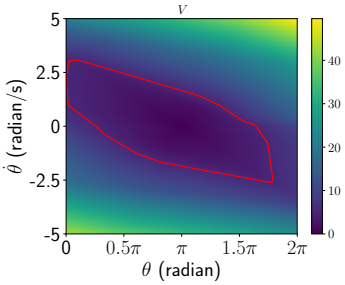


Fig. 7: Heatmap of the Lyapunov function for the inverted pendulum. The red contour is the boundary of the verified inner approximation of the region of attraction, as the largest sub-level set contained in the verified box region $0 \le \theta \le 2\pi, -5 \le \dot{\theta} \le 5$.

It is worth noting that the size of this inner approximation of the region of attraction can be small, as we fix the Lyapunov function and only search for its sub-level set. To verify a larger inner approximation, one possible future research direction is to search for the Lyapunov function and the sub-level set simultaneously, as in [43].

V. Results

We synthesize stable controllers and Lyapunov functions on pendulum, 2D and 3D quadrotors. We use Gurobi as the MIP solver. All code runs on an Intel Xeon CPU. The sizes of the neural networks are shown in Table III in Appendix VII-A.

A. Inverted pendulum

We first test our approach on an inverted pendulum. We approximate the pendulum Lagrangian dynamics using a neural network, by first simulating the system with many state/action pairs, and then approximating the simulation data through regression. To stabilize the pendulum at the top equilibrium $\theta = \pi, \dot{\theta} = 0$, we synthesize a neural-network controller and a Lyapunov function using both Algorithm 1 and 2. We verify the Lyapunov condition in the box region $0 \le \theta \le 2\pi, -5 \le \dot{\theta} \le 5$. The Lyapunov function V is shown in Fig. 7.

We simulate the synthesized controller with the original pendulum Lagrangian dynamics model (not the neural network dynamics ϕ_{dyn}). The result is shown in Fig. 8. Although the neural network dynamics ϕ_{dyn} has approximation error, the simulation results show that the neural-network controller swings up and stabilizes the pendulum for not only the approximated neural network dynamics, but also for the original Lagrangian dynamics. Moreover, starting from many states

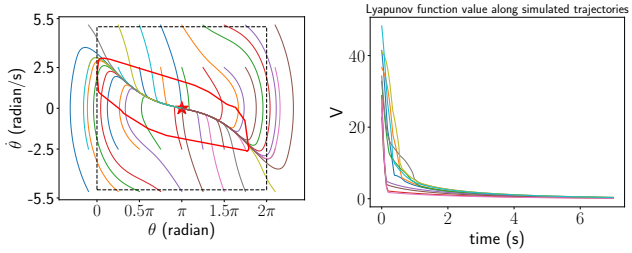


Fig. 8: (left) phase plot of simulating the pendulum Lagrangian dynamics with the neural-network controller. The red contour is the boundary of the verified region of attraction, as the largest level set within the verified box region $\mathcal{B}$ (black dashed box). All the simulated trajectories (even starting outside of the dashed box) converge to the goal state. (right) Lyapunov function value along the simulated trajectories. The Lyapunov function decreases monotonically along the trajectories.

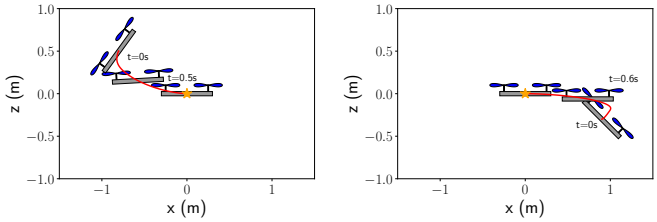


Fig. 9: Snapshots of the 2D quadrotor simulation (with the original Lagrangian dynamics) using our neural-network controller from different initial states. The red lines are the trajectories of the quadrotor body frame origin.

outside the verified region of attraction, and even outside our verified box region, the trajectories still converge to the equilibrium. This suggests that the controller generalizes well. The small verified region of attraction suggests that in the future we can improve its size by searching over the Lyapunov function and the sub-level set simultaneously.

We start with a small box region $0.8\pi \leq \theta \leq 1.2\pi, -1 \leq \dot{\theta} \leq 1$, and then gradually increase the verified region. We initialize the controller/Lyapunov network as the solution in the previous iteration on a smaller box region (at the first iteration, all parameters are initialized arbitrarily). For the smaller box $0.8\pi \leq \theta \leq 1.2\pi, -1 \leq \dot{\theta} \leq 1$, both algorihm 1 and 2 converge within a few minutes. For the larger box $0 \leq \theta \leq 2\pi, -5 \leq \dot{\theta} \leq 5$, both algorithms converge within 3 hours.

B. 2D quadrotor

We synthesize a stabilizing controller and a Lyapunov function for the 2D quadrotor model used in [51]. Again we first train a neural network ϕ_{dyn} to approximate the Lagrangian dynamics. Our goal is to steer the quadrotor to hover at the origin. In Fig.9 we visualize the snapshots of the quadrotor stabilized by our neural-network controller. We verified the Lyapunov conditions in the region $[-0.75, -0.75, -0.5\pi, -4, -4, -2.75] \leq [x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}] \leq [0.75, 0.75, 0.5\pi, 4, 4, 2.75]$.

	NN succeeds	NN fails
LQR succeeds	8078	0
LQR fails	1918	4

TABLE I: Number of success/failure for $10,000$ simulations of 2D quadrotor with the neural network (NN) controller and an LQR controller. The simulation uses the Lagrangian dynamics.

We sample 10000 initial states uniformly in the box $[-0.9, -0.9, -0.5\pi, -4.5, -4.5, -3] \leq [x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}] \leq [0.9, 0.9, 0.5\pi, 4.5, 4.5, 3]$. For each initial state we simulate the Lagrangian dynamics with the neural network and an LQR controller. We summarize the result in table I on whether the simulation converges to the goal state or not. More states can be stabilized by the neural-network controller than the LQR controller. Moreover, the off-diagonal entries in Table I demonstrates that the set of sampled states that are stabilized by the neural-network controller is a strict super-set of the set of states stabilized by the LQR controller. We believe there are two factors contributing to the advantage of our neural-network controller against an LQR: 1) the neural-network controller is piecewise linear while the LQR controller is linear; the latter can be a special case of the former. 2) the neural-network controller is aware of the input limits while the LQR controller is not.

We then focus on certain two dimensional slices of the state space, and sample many initial states on these slices. For each sampled initial state we simulate the Lagrangian dynamics using both the neural-network and the LQR controller. We visualize the simulation results in Fig. 10. Each dot represents a sampled initial state, and we color each initial state based on whether the neural-network (NN)/LQR controllers succeed in stabilizing that initial state to the goal

- Purple: NN succeeds but LQR fails.
- Green: both NN and LQR succeed
- Red: both NN and LQR fail.

Evidently the large purple region suggests that the region of attraction with the neural-network controller is a strict super-set of that with the LQR controller. We observe that for the initial states where LQR fails, the LQR controller requires thrusts beyond the input limits. If we increase the input limits then LQR can stabilize many of these states. Hence by taking input limits into consideration, the neural-network controller achieves better performance than the LQR.

Both algorithm 1 and 2 find the stabilizing controller. For a small box region $[-0.1, -0.1, -0.1\pi, -0.5, -0.5, -0.3] \leq [x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}] \leq [0.1, 0.1, 0.1\pi, 0.5, 0.5, 0.3]$, both algorithms converge in 20 minutes. For the larger box used in Table I, the algorithms converge in 1 day.

C. 3D quadrotor

We apply our approach to a 3D quadrotor model with 12 states [38]. Again, our goal is to steer the quadrotor to hover at the origin. As visualized in Fig.11, our neural-network controller can stabilize the system. Training this controller took 3 days.

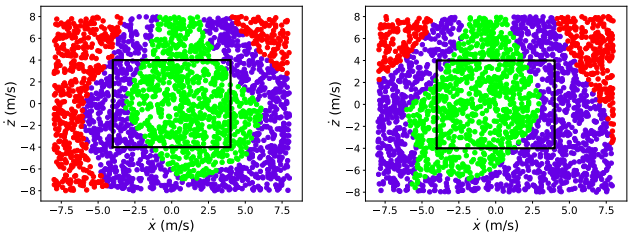


Fig. 10: We sample 2500 initial states within the box region $[-8,-8] \leq [\dot{x},\dot{z}] \leq [8,8]$, with $[x,z,\theta,\dot{\theta}]$ fixed to $[-0.75, 0.3, 0.3\pi, 2]$ (left), and $[0.75, 0.5, -0.4\pi, 2]$ (right). We simulate from each initial state with the neural-network controller (NN) and the LQR controller, and color each initial state based on whether the simulation converges to the goal. All red dots (where the NN controller fails) are outside of the black box region within which we verified the Lyapunov conditions.

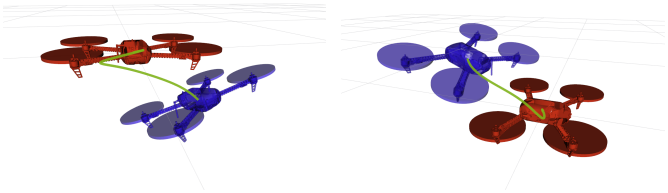


Fig. 11: Snapshots of simulating the quadrotor using our neural-network controller with the Lagrangian dynamics. The quadrotor converge to the hovering state at the origin (red).

The quadrotor Lagrangian dynamics is approximated by a neural network ϕ_{dyn} with comparatively large mean-squared error (MSE) around 10^{-4} (reducing MSE would require a neural network too large for our MIP solver), while other examples in this paper have MSE in the order of 10^{-6}. Hence there are noticeable discrepancies between the simulation with Lagrangian dynamics and with the neural network dynamics ϕ_{dyn}. In Fig 12 we select some results to highlight the discrepancy, that the Lyapunov function always decreases along the trajectories simulated with neural-network dynamics, while it could increase with Lagrangian dynamics. Nevertheless, the quadrotor eventually always converges to the goal state. We note that the same phenomenon would also happen if we took a linear approximation of the quadrotor dynamics and stabilized the quadrotor with an LQR controller. If we were to plot the quadratic Lyapunov function (which is valid for the LQR controller and the linearized dynamics), that Lyapunov function could also increase along the trajectories simulated with the nonlinear Lagrangian dynamics (see Fig 14 in the Appendix). Analogous to approximating the nonlinear dynamics with a linear one and stabilizing it with a linear LQR controller, our approach can be regarded as approximating the nonlinear dynamics with a neural network and stabilizing it with another neural-network controller.

Finally we compare the performance of Algorithm 1 which appends counter examples to training sets, against Algorithm 2 with min-max optimization. We take 10 runs of each algorithm, and report the average computation time for each algorithm in Table II [4]. For the small-sized task (pendulum with 2 states), Algorithm 1 is orders of magnitude faster than Algorithm 2, and they take about the same time on the medium-sized task (2D quadrotor with 6 states). On the large-sized task (3D quadrotor with 12 states), Algorithm 1 doesn't converge while Algorithm 2 can find the solution. We speculate this is because Algorithm 1 overfits to the training set. For a small-sized task the overfitting is not a severe problem as a small number of sampled states are sufficient to represent the state space; while for a large-sized task it would require a huge number of samples to cover the state space. With the limited number of counter examples Algorithm 1 overfits to these samples while causing large Lyapunov condition violation elsewhere. This is evident from the loss curve plot in Fig.13 for a 2D quadrotor task. Although both algorithms converge, the loss curve decreases steadily with Algorithm 2, while it fluctuates wildly with Algorithm 1. We believe that the fluctuation is caused by overfitting to the training set in the previous iteration. Nevertheless, this comparison is not yet conclusive, and we are working to improve the performance of Algorithm 1 on the large-size task.

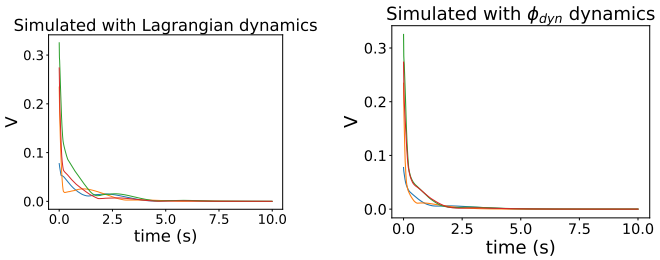


Fig. 12: 3D quadrotor Lyapunov function value along the simulated trajectories with our neural-network controller. The quadrotor is simulated with Lagrangian dynamics (left) vs the dynamics approximated by a neural network ϕ_{dyn} (right). In both left and right sub-plots, the initial states are the same.

	Algorithm 1	Algorithm 2
Pendulum	8.4s	224s
2D quadrotor	948.3s	1004.7s
3D quadrotor	Time out after 5 days	65.7hrs

TABLE II: Average computation time of 10 runs for both algorithms. To speed up the computation, the verified region $\mathcal{B}$ is relatively small.

VI. Conclusion and Future Work

In this paper, we demonstrate a method to synthesize a neural-network controller to stabilize a robot system, as well as a neural-network Lyapunov function to prove the resulting stability. We propose an MIP verifier to either certify Lyapunov stability, or generate counter examples that can be used to improve the candidate controller and the Lyapunov function.

[4]There are 2 failed runs with Algorithm 1 on the 2D quadrotor example, that they time out after 6 hours, and are not included in Table II. For the 3D quadrotor we only include 3 runs as they are too time-consuming

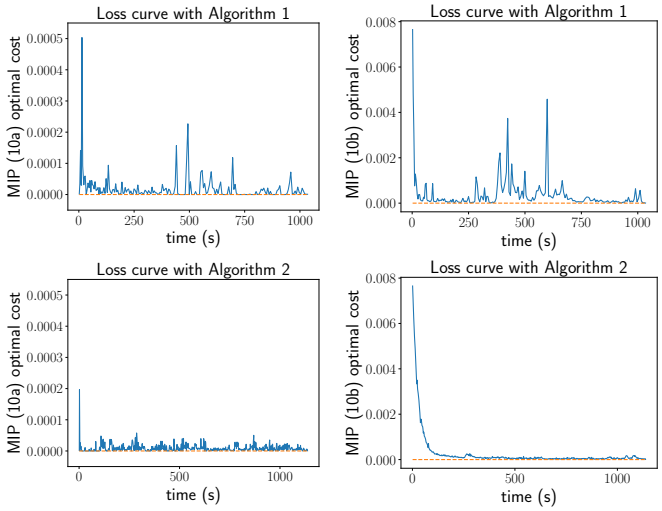


Fig. 13: Loss curves on the 2D quadrotor task for Algorithm 1 and 2

We present another MIP to compute an inner approximation of the region of attraction. We demonstrate our approach on an inverted pendulum, 2D and 3D quadrotors, and showcase that it can outperform a baseline LQR controller.

Currently, the biggest challenge of our approach is scalability. The speed bottleneck lies in solving MIPs, where the number of binary variables scales linearly with the total number of neurons in the networks. In the worst case, the complexity of solving an MIP scales exponentially with the number of binary variables, when the solver has to check every node of a binary tree. However, in practice, the branch-and-cut process significantly reduces the number of nodes to explore. Recently, with the growing interest from the machine learning community, many approaches were proposed to speed up verifying neural networks through MIP by tightening the formulation [5, 54]. We plan to explore these approaches in the future.

Our proposed MIP formulation works for discrete-time dynamical systems. For continuous-time dynamical systems, neural networks have been previously used either to approximate the system dynamics [34], or to synthesize optimal controllers [15]. We plan to extend our approach to continuous-time dynamical systems. Moreover, we can readily apply our approach to systems whose dynamics are approximated by piecewise-affine dynamical systems, such as soft robots [53] and hybrid systems with contact [37], since piecewise-affine dynamic constraints can easily be encoded into MIP.

Many safety-critical missions also require the robot to avoid unsafe regions. We can readily extend our framework to synthesize barrier functions [3] so that the robot certifiably stays within the safe region.

Acknowledgments

Benoit Landry is sponsored by the NASA University Leadership initiative (grant #80NSSC20M0163), and Lujie Yang is sponsored by Amazon Research Award #6943503. This article solely reflects the opinions and conclusions of its authors and not any funding agencies. We would like to thank Vincent Tjeng and Shen Shen for the valuable discussion.

References

[1] Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. Formal synthesis of lyapunov neural networks. *IEEE Control Systems Letters*, 5(3):773–778, 2020.

[2] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable convex optimization layers. In *Advances in neural information processing systems*, 2019.

[3] Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2016.

[4] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[5] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, pages 1–37, 2020.

[6] Jonathan F Bard. *Practical bilevel optimization: algorithms and applications*, volume 30. Springer Science & Business Media, 2013.

[7] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2621–2629, 2016.

[8] R Bonna and JF Camino. Trajectory tracking control of a quadrotor using feedback linearization. In *International Symposium on Dynamic Problems of Mechanics*, 2015.

[9] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *Linear matrix inequalities in system and control theory*. SIAM, 1994.

[10] Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, 2018.

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

[12] Shaoru Chen, Mahyar Fazlyab, Manfred Morari, George J Pappas, and Victor M Preciado. Learning lyapunov functions for hybrid systems. *arXiv preprint arXiv:2012.12015*, 2020.

[13] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017.

[14] Hongkai Dai, Benoit Landry, Marco Pavone, and Russ Tedrake. Counter-example guided synthesis of neural network lyapunov functions for piecewise linear systems. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1274–1281. IEEE, 2020.

[15] Farbod Farshidian, David Hoeller, and Marco Hutter. Deep value model predictive control. In *3rd Conference on Robot Learning (CoRL 2019)*, 2019.

[16] Nima Fazeli, Miquel Oller, Jiajun Wu, Zheng Wu, Joshua B Tenenbaum, and Alberto Rodriguez. See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. *Science Robotics*, 4(26), 2019.

[17] Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, 2020.

[18] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23 (3):296–309, 2018.

[19] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Conference on Robot Learning*, pages 373–385. PMLR, 2018.

[20] John Forrest and Robin Lougee-Heimer. Cbc user guide. In *Emerging theory, methods, and applications*, pages 257–277. INFORMS, 2005.

[21] Morgan T Gillespie, Charles M Best, Eric C Townsend, David Wingate, and Marc D Killpack. Learning nonlinear dynamic models of soft robots for model predictive control with neural networks. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 39–45. IEEE, 2018.

[22] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[23] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[24] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[25] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[26] Zachary Jarvis-Wloszek, Ryan Feeley, Weehong Tan, Kunpeng Sun, and Andrew Packard. Some controls applications of sum of squares programming. In *42nd IEEE international conference on decision and control (IEEE Cat. No. 03CH37475)*, volume 5, pages 4676–4681. IEEE, 2003.

[27] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[28] James Kapinski, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan, and Nikos Arechiga. Simulation-guided lyapunov analysis for hybrid dynamical systems. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 133–142, 2014.

[29] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[30] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *Proceedings of the IEEE international conference on computer vision*, pages 1521–1529, 2017.

[31] Benoit Landry, Zachary Manchester, and Marco Pavone. A differentiable augmented lagrangian method for bilevel nonlinear optimization. In *Proceedings of Robotics: Science and Systems*, 2019.

[32] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5 (47), 2020.

[33] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[34] Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2018.

[35] Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake. Control design along trajectories with sums of squares programming. In *2013 IEEE International Conference on Robotics and Automation*, pages 4054–4061. IEEE, 2013.

[36] Lucas Manuelli, Yunzhu Li, Pete Florence, and Russ Tedrake. Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning. *arXiv preprint arXiv:2009.05085*, 2020.

[37] Tobia Marcucci and Russ Tedrake. Mixed-integer formulations for optimal control of piecewise-affine systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 230–239, 2019.

[38] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[39] Gurobi Optimization. Inc.,"gurobi optimizer reference manual," 2015, 2014.

[40] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[41] Samuel Pfrommer, Mathew Halm, and Michael Posa. Contactnets: Learning of discontinuous contact dynamics with smooth, implicit representations. *arXiv preprint arXiv:2009.11193*, 2020.

[42] Hadi Ravanbakhsh and Sriram Sankaranarayanan. Learning control lyapunov functions from counterexamples and demonstrations. *Autonomous Robots*, 43(2):275–307, 2019.

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

[44] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[45] Shen Shen. *Convex Optimization and Machine Learning for Scalable Verification and Control*. PhD thesis, Massachusetts Institute of Technology, 2020.

[46] Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991.

[47] Eduardo D Sontag. Stability and stabilization: discontinuities and the effect of disturbances. In *Nonlinear analysis, differential equations and control*, pages 551–598. Springer, 1999.

[48] Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, et al. The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5):405–420, 2018.

[49] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[50] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[51] Russ Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines. *Course notes for MIT*, 6:832, 2009.

[52] Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2018.

[53] Sander Tonkens, Joseph Lorenzetti, and Marco Pavone. Soft robot optimal control via reduced order finite element models. *arXiv preprint arXiv:2011.02092*, 2020.

[54] Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained relu neural networks. *arXiv preprint arXiv:2102.04373*, 2021.

[55] Juan Pablo Vielma, Shabbir Ahmed, and George Nemhauser. Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Operations research*, 58(2):303–315, 2010.

[56] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[57] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

	forward dynamics network ϕ_{dyn}	controller network ϕ_π	Lyapunov network ϕ_V
Inverted pendulum	(5, 5)	(2, 2)	(8, 4, 4)
2D quadrotor	(7, 7)	(6, 4)	(10, 10, 4)
3D quadrotor	(10, 10)	(16, 8)	(16, 12, 8)

TABLE III: Size of the neural network hidden layers in each task.

VII. Appendix

A. Network structures in each example

For each task in the result section, we use three fully connected feedforward neural networks to represent forward dynamics, the control policy and the Lyapunov function respectively. All the networks use leaky ReLU activation function. The size of the network is summarized in Table III. Each entry represents the size of the hidden layers. For example $(3, 4)$ represents a neural network with 2 hidden layers, the first hidden layer has 3 neurons, and the second hidden layer has 4 neurons.

B. MIP formulation for l_1 norm and clamp function

For the l_1 norm constraint on $x \in \mathbb{R}^n$

$$y = |x|_1 \tag{16}$$

where x is bounded elementwisely as $l \leq x \leq u$, we convert this constraint (16) to the following mixed-integer linear constraint

$$y = z_1 + \ldots + z_n \tag{17a}$$
$$z_i \geq x_i,\ z_i \geq -x_i \tag{17b}$$
$$z_i \leq x_i + 2l_i(\alpha_i - 1),\ z_i \leq 2u_i\alpha_i - x_i \tag{17c}$$
$$\alpha \in \{0, 1\} \tag{17d}$$

where we assume $l < 0, u > 0$ (the case when $l \geq 0$ or $u \leq 0$ is trivial).

For a clamp function

$$y = \begin{cases} l, \text{ if } x \leq l \\ x, \text{ if } l \leq x \leq u \\ u, \text{ if } x \geq u \end{cases} \tag{18}$$

This clamp function can be rewritten in the following form using ReLU function

$$y = u - ReLU(u - (ReLU(x - l) + l)) \tag{19}$$

As explained in (4) in section III, we can convert ReLU function to mixed-integer linear constraints, hence we obtain the MIP formulation of (18).

C. Necessity of condition (9)

Lemma 7.1: There exists a piecewise-affine function $V(x) : \mathbb{R}^n \to \mathbb{R}$ satisfying

$$V(x^*) = 0, V(x) > 0 \; \forall x \neq x^* \tag{20}$$

if and only if for a given positive scalar $\epsilon > 0$ and a given full column-rank matrix R, there exists another piecewise-affine function $\bar{V}(x) : \mathbb{R}^n \to \mathbb{R}$ satisfying

$$\bar{V}(x^*) = 0, \bar{V}(x) \geq \epsilon |R(x - x^*)|_1 \; \forall x \neq x^* \tag{21}$$

Proof: The "only if" part is trivial, if such $\bar{V}(x)$ exists, then just setting $V(x) = \bar{V}(x)$ and (20) holds. To prove the "if" part, assume that $V(x)$ exists, and we will show that there exists a positive scalar, such that by scaling $V(\cdot)$ we get $\bar{V}(\cdot)$. Intuitively this scalar could be found as the smallest ratio between $V(x)$ and $\epsilon |R(x - x^*)|_1$. Formally, given a unit length vector $d \in \mathbb{R}^n$, we define a scalar function $\phi_d(t) = V(x^* + td)$, this scalar function $\phi(\cdot) : \mathbb{R} \to \mathbb{R}$ is also piecewise-affine, as it is just the value of $V(\cdot)$ along the direction d. Likewise we define another scalar function $\psi_d(t) = \epsilon |R(td)|_1$ which is just the value of the right-hand side of (21) along the direction d. Since both $\phi_d(t), \psi_d(t)$ are piecewise-affine and positive definite, the minimal ratio $\zeta(d) = \min_{t \neq 0} \phi_d(t)/\psi_d(t)$ is strictly positive. Moreover, consider the value $\kappa = \min_{d^T d = 1} \zeta(d)$, since the domain $\{d | d^T d = 1\}$ is a compact set, and the minimal of a positive function $\zeta(d)$ on a compact set is still positive, hence $\kappa > 0$. As a result, setting $\bar{V}(\cdot) = V(\cdot)/\kappa$ will satisfy (21), because $\bar{V}(x) = \bar{V}(x^* + td) = V(x^* + td)/\kappa = \phi_d(t)/\kappa \geq \phi_d(t)/\zeta(d) \geq \phi_d(t)/(\phi_d(t)/\psi_d(t)) = \psi_d(t) = \epsilon |R(x - x^*)|_1$ where we choose $d = (x - x^*)/|x - x^*|$ and $t = |x - x^*|$. ■

D. LQR simulation on 3D quadrotor

We simulate the 3D quadrotor with an LQR controller, and plot its Lyapunov function $x^T S x$ (where S is the solution to the Riccati equation) along the simulated trajectories in Fig. 14. If the dynamics were linear, then this quadratic Lyapunov function would always decrease; on the other hand, with the actual nonlinear dynamics the Lyapunov function (for the linear dynamical system) can increase. For our neural network dynamics, the discrepancy between the approximated dynamics and the actual dynamics will likewise cause the Lyapunov function to increase on the actual dynamical system.

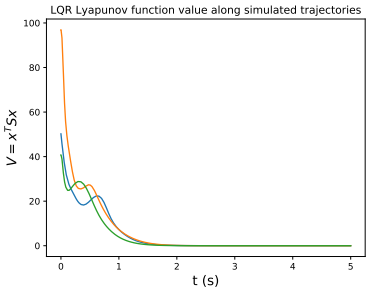


Fig. 14: Value of the Lyapunov function $V = x^T S x$ for an LQR controller on a 3D quadrotor.

E. Termination tolerance

In practice, due to solver's numerical tolerance, we declare convergence of Algorithm 1 and 2 when the MIPs (10a) (10b) has optimal cost in the order of 10^{-6}.