Counter-example guided synthesis of neural network Lyapunov functions for piecewise linear systems

Hongkai Dai[1], Benoit Landry[2], Marco Pavone[2] and Russ Tedrake[1,3]

***Abstract*— We introduce an algorithm for synthesizing and verifying piecewise linear Lyapunov functions to prove global exponential stability of piecewise linear dynamical systems. The Lyapunov functions we synthesize are parameterized by feedforward neural networks with leaky ReLU activation units. To train these neural networks, we design a loss function that measures the maximal violation of the Lyapunov conditions in the state space. We show that this maximal violation can be computed by solving a mixed-integer linear program (MILP). Compared to previous learning-based approaches, our learning approach is able to certify with high precision that the learned neural network satisfies the Lyapunov conditions not only for sampled states, but over the entire state space. Moreover, compared to previous optimization-based approaches that require a pre-specified partition of the state space when synthesizing piecewise Lyapunov functions, our method can automatically search for both the partition and the Lyapunov function simultaneously. We demonstrate our algorithm on both continuous and discrete-time systems, including some for which known strategies for partitioning of the Lyapunov function would require introducing higher order Lyapunov functions.**

I. INTRODUCTION

Proving stability of dynamical systems has been a central theme in the control community. One particular criterion, Lyapunov stability, has attracted tremendous interests. This criterion guarantees the convergence of dynamical system states through the existence of a Lyapunov function, which can be pictured as a bowl-shaped function with positive values everywhere except at the equilibrium, and function values decreasing along trajectories following the system dynamics. Various approaches have been developed to synthesize Lyapunov functions for different types of systems. Lyapunov functions for linear systems can be obtained through solving Algebraic Lyapunov equation or Linear Matrix Inequalities (LMI) [5]. For some nonlinear systems, it is possible to compute Lyapunov functions through sum-of-squares (SOS) optimization [19].

In this paper, we are interested in a particular type of hybrid dynamical systems, piecewise linear (PWL) systems, and synthesizing their Lyapunov functions. A PWL system has hybrid dynamics, where each of the mode is defined by a conic polyhedron region, and the dynamics remain linear within each mode [10]. PWL systems have attracted attention in the control community, as these systems retain much of the simplicity of linear systems, while being able to approximate

Stanford Robotics Center Fellowship sponsored by FANUC

[1] Toyota Research Institute, USA [2] Stanford University, USA [3] Computer Science and Artificial Intelligence Lab, MIT, USA, hongkai.dai@tri.global, blandry@stanford.edu, pavone@stanford.edu, russt@mit.edu

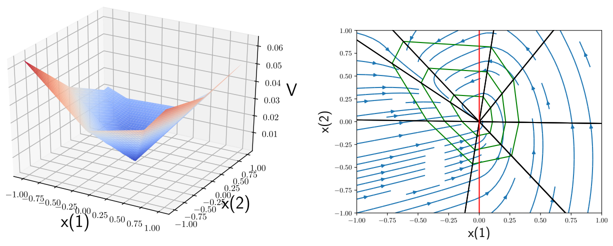


Fig. 1: (left) Learned Lyapunov function for a continuous piecewise linear dynamical system with 2 states. (right) The phase portrait of the system. We also draw contours of Lyapunov function $V(x) = 0.005, 0.01, 0.015$ in green lines. The boundaries between each piece in the piecewise linear Lyapunov function is drawn in black, and the red line is the boundary between the hybrid modes.

complicated (hybrid) nonlinear systems by partitioning the state space of nonlinear systems into smaller regions, and linearizing their nonlinear dynamics within each of those regions. One approach to synthesize Lyapunov functions for PWL systems is to look for *common Lyapunov functions*, where a single smooth Lyapunov function is shared across all modes [24]. But many stable PWL systems do not admit a common Lyapunov function [10]. An alternative is to search for piecewise linear or piecewise quadratic Lyapunov functions [4], [11]. With piecewise Lyapunov functions, the state space is cut into different partitions; a linear/quadratic Lyapunov function is synthesized within each partition, and then stitched together along their borders. One challenge in synthesizing piecewise linear/quadratic Lyapunov functions is determining how to partition the state space. A common choice is to align the boundary of each partition in the Lyapunov function with the boundary of each hybrid mode of the system, i.e., each mode has a linear/quadratic Lyapunov function. However, many stable systems do not admit piecewise linear or quadratic Lyapunov functions in which the partition aligns with the hybrid modes [28]. In this paper, we propose a novel method to overcome this challenge. Our method can synthesize piecewise linear Lyapunov functions without requiring explicit partitioning of the state space. Instead, it is able to search for the partition automatically. As a result, we can find piecewise linear Lyapunov functions for stable systems which requires more partitions in the piecewise linear Lyapunov function than the number of hybrid modes.

With recent advances in deep learning, many researchers have proposed learning Lyapunov-like functions for dynamical systems using feedforward neural networks [17], [22],

[13], [2]. In [22], the authors attempt to learn a Lyapunov function using gradient descent and a loss function which encourages the Lyapunov function to decrease over a set of randomly sampled states. In [2], the authors use a similar approach as [22], but the training set contains counter- example states (states where the Lyapunov condition is violated). These counter-example states are generated using an SMT solver [6] at each iteration. When using the SMT solver, [2] allows a small violation (≈ 0.01) of the Lyapunov conditions, and doesn't certify the satisfiability within a neighbourhood around the equilibrium state. In our approach, we also generate a training set containing the counter-examples, namely the **worst** adversarial states (the states with the maximal violation of Lyapunov conditions). In the previous work, the counter-example states are generated from simulation [12], SMT solvers [2] or solving LMI relaxations [20]. Thanks to the special structure of PWL dynamical systems and of our neural networks, we show that it is possible to find these states through mixed-integer linear programming (MILP), which can globally certify the Lyapunov condition with high accuracy ($\approx 10^{-5}$ with modern MILP solvers), much higher than SMT solvers.

In this work, we represent Lyapunov functions using fully connected neural networks with leaky ReLU activation units [18]. Based on the *universal approximation theorem*, such neural networks can approximate any continuous functions if the network is big enough [16], and hence can approximate complicated Lyapunov functions, including the piecewise linear/quadratic Lyapunov functions synthesized with previous approaches. [26], [27] have shown that for many supervised learning tasks (like image classification), it is possible to find counter examples for a classifier with (leaky) ReLU activation functions by solving an MILP. Similarly in our approach, for each neural network representing a candidate Lyapunov function, we solve an MILP to find the maximal violation of the Lyapunov conditions together with the worst adversarial states. Our approach improves the candidate Lyapunov function using gradient descent, and a loss function made up of two major components. The first is an empirical Lyapunov condition violation on all adversarial states detected in the previous iterations (similar to [2]); the second component, taking insight from *bilevel optimization* [1], [14], is the maximal violation of the Lyapunov conditions in the entire state space as the MILP optimal costs. We show that using both components simultaneously, our training converges faster than using either one separately.

Interestingly, in the event that the system turns out to be unstable, the adversarial states produced by our method can be efficiently used as initial states from which to simulate it and potentially prove its instability. This is similar to the other counter-example guided approaches, but contrasts with many optimization-based approaches that would simply fail to find Lyapunov functions without providing any information with respect to the stability of the underlying system.

It is worth mentioning that some previous approaches can also find the partition of the state space automatically and synthesize a piecewise Lyapunov function [25], [8]. These approaches adopt a multi-step process, that they first attempt certain partition of the state space, and then try to find a piecewise Lyapunov function for this given partition. If the Lyapunov function is not found, then the partition is refined for another trial. Unlike these multi-step approaches, our approach find the partition and the Lyapunov function simultaneously, by optimizing the neural network which encodes both the state partition and the piecewise Lyapunov function.

II. Problem Statement

We are interested in finding the Lyapunov function for the following continuous-time or discrete-time piecewise linear (PWL) systems

$$\text{Continuous-time} \quad \dot{x} = A_i x \text{ if } P_i x \leq 0 \tag{1a}$$

$$\text{Discrete-time } x_{n+1} = A_i x_n \text{ if } P_i x_n \leq 0 \tag{1b}$$

Notice that the domain of the i'th mode is a conic polyhedron $\mathcal{P}_i = \{x | P_i x \leq 0\}$, that all mode boundaries pass through the origin. We denote the total number of modes as N.

We aim to show the global exponential stability of PWL systems by finding piecewise linear Lyapunov functions $V(x)$ satisfying

$$V(x) \geq \epsilon_1 |x|_1 \; \forall x \neq 0 \tag{2a}$$

$$dV(x) \leq -\epsilon_2 V(x) \; \forall x \tag{2b}$$

$$V(0) = 0 \tag{2c}$$

where ϵ_1 and ϵ_2 are given positive constants, $|x|_1$ is the l_1 norm of vector x. In (2b) we use $dV(x_n)$ to denote $V(x_{n+1}) - V(x_n)$ for discrete-time system, and $dV(x) = \dot{V}(x)$ for continuous-time system. It is straightforward to verify that condition (2a)-(2c) imply LaSalle's theorem [15], and hence global exponential stability.

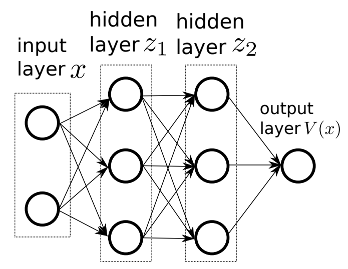
Note that special care must be taken when enforcing the Lyapunov conditions on functions that are not smooth [23], like the ones produced by our approach. For example, at their non-differentiable points, our method enforces the Lyapunov condition for all valid subgradients.

III. Approach

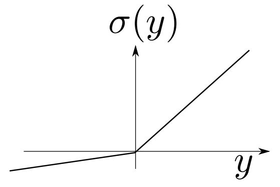
In this section, we describe how to represent the Lyapunov function using neural networks with leaky ReLU activation functions. We then show that we can compute the adversarial states (counter-example states with the worst violation of the Lyapunov conditions) through mixed-integer linear programming (MILP). Finally, we show how to use adversarial states and MILP results to learn a Lyapunov function.

A. Neural networks as Lyapunov functions

We can prove that if a piecewise affine Lyapunov function exists for the PWL system in (1a)(1b), then there always exists a fully connected neural network with leaky ReLU activation functions and no bias terms, whose output is also a Lyapunov function (The proof can be found in the appendix section VI) Such neural network is illustrated in Fig.2a. Suppose that our network has K hidden layers, if we denote



(a) A fully connected network with 2 hiddlen layers.



(b) A leaky ReLU unit. We use a binary variable β to indicate whether the leaky ReLU unit is active ($\beta = 1, y \geq 0$) or inactive ($\beta = 0, y \leq 0$).

the output of the i'th hidden layer as z_i (with z_0 denoting the input x, and z_{K+1} denoting the network output $V(x)$), then the neural network can be formulated as

$$z_{i+1} = \sigma(W_i z_i),\ i = 0, \ldots, K-1 \tag{3}$$
$$V(x) = z_{K+1} = w_K^T z_K \tag{4}$$

where W_i is the linear weight matrix of the i'th layer. $\sigma(\bullet)$ denotes the leaky ReLU activation function

$$\sigma(y) = \max(cy, y) \tag{5}$$

where $c < 1$ is a given constant. The leaky ReLU function σ is drawn in Fig.2b. Since leaky ReLU is piecewise linear, the network output $V(x)$ also becomes a piecewise linear function of the input state x. Note that our network doesn't have bias terms in each linear layer, hence the network output satisfies $V(kx) = kV(x)$ for any positive scalar k, and the condition (2c) is trivially satisfied.

In order to enforce that the network output $V(x)$ satisfies the Lyapunov condition (2a)-(2b) for the entire unbounded state space, we only need to enforce the conditions within a bounded neighbourhood around the origin. This holds because both the system dynamics and Lyapunov function are homogeneous function of x, so if $V(x)$ satisfies condition (2a)(2b) when x is restricted to a bounded region $\mathcal{S}$ around the origin, these conditions are also satisfied for the region $\{kx | x \in \mathcal{S}, k > 0\}$, which is the entire state space. In this paper, we use a bounded polytope as this set $\mathcal{S}$, and we denote the intersection of the i'th mode domain $\mathcal{P}_i$ with this bounded polytope $\mathcal{S}$ as $\bar{\mathcal{P}}_i$

$$\bar{\mathcal{P}}_i = \mathcal{P}_i \cap \mathcal{S} = \{x | \bar{P}_i x \leq q_i\} \tag{6}$$

$\bar{\mathcal{P}}_i$ is a bounded polytope with the origin on the boundary of each polytope. We are interested in the bounded region instead of the unbounded entire state space, because when restricted to a bounded domain, a piecewise linear function can be converted to mixed-integer linear constraints (for more details, refer to the big-M trick in [9]). As the network output is a piecewise linear function of the input state, we can thus obtain mixed-integer linear constraints on x, when the state is restricted to the bounded domain $\cup_{i=1}^N \bar{\mathcal{P}}_i$.

It is worth mentioning that the leaky ReLU function $\sigma(y)$ is not differentiable at $y = 0$. Here we consider all possible *subgradients* of the leaky ReLU function $d\sigma \in [c, 1]$ at $y = 0$ when we compute the gradient $\dot{V} = \frac{\partial V}{\partial x}\dot{x}$ on a continuous-time PWL system. Namely we require the Lyapunov function to decrease with any possible subgradients.

B. Find adversarial states through solving MILP

We aim to find the worst adversarial states for a given neural network, by solving the following two optimization problems whose costs are the violation of Lyapunov conditions (2a)

$$\max_{x \in \cup_{i=1}^N \bar{\mathcal{P}}_i} \epsilon_1 |x|_1 - V(x) \tag{7a}$$

and (2b)

$$\max_{x \in \cup_{i=1}^N \bar{\mathcal{P}}_i} dV(x) + \epsilon_2 V(x) \tag{7b}$$

If the maximal costs of these two optimization problems are 0 (obtained at $x = 0$), then the Lyapunov conditions are satisfied globally. Note that as we mentioned in the previous subsection III-A, we only need to consider the state x within the union of bounded region $\bar{\mathcal{P}}_i$ defined in (6).

In the subsequent paragraphs, we will show that both optimization problems (7a) and (7b) can be cast as mixed-integer linear programs, whose global optimal solution can be readily computed with off-the-shelf solvers [7].

1) Cast (7a) *as an MILP:* We first show that the network output $V(x)$ and the input x satisfy mixed-integer linear constraints. We adopt the idea in [27], and introduce binary variable vectors $\beta_i, i = 1, \ldots, K$ to indicate whether the leaky ReLU units in the i'th hidden layer are active or not (as mentioned in Fig.2b). Namely

$$\beta_i(j) = 1 \Rightarrow W_i(j,:)z_i \geq 0,\ z_{i+1}(j) = W_i(j,:)z_i \tag{8a}$$
$$\beta_i(j) = 0 \Rightarrow W_i(j,:)z_i \leq 0,\ z_{i+1}(j) = cW_i(j,:)z_i \tag{8b}$$

Here we use the notation $W_i(j,:)$ to denote the j'th row of matrix W_i. The symbol $\Rightarrow$ means "implies". There are standard procedures (like big-M[1] or convex hull approaches) in mixed-integer formulation literature to convert the implication relationship in (8) to mixed-integer linear constraints [21], [9]. In the subsequent presentation, we will present only the implication relationship with "$\Rightarrow$" symbol, since the corresponding mixed-integer linear constraints can be readily obtained.

By replacing the nonlinear (piecewise linear) relationship in the network hidden layers (Eq.(3)) with the mixed-integer linear constraints (8), we obtain the mixed-integer linear constraints on the decision variables z_{K+1} (in Eq.(4), z_{K+1} is the network output), the network input x, the slack variables z_i, and the binary variables β_i. The readers could refer to [27] on the mixed-integer linear constraint formulation of ReLU network for more details.

The objective in (7a) also contains the l_1 norm $|x|_1$. We can also write this l_1 norm as mixed-integer linear constraints

[1]For example, to convert the relationship $\beta = 0 \Rightarrow a^T x = b$ to mixed-integer linear constraints, we could use the big-M trick as $-M\beta \leq b - a^T x \leq M\beta$ where M is a big number

with the binary variable α

$$\alpha(i) = 1 \Rightarrow x(i) \geq 0, |x(i)| = x(i) \tag{9a}$$

$$\alpha(i) = 0 \Rightarrow x(i) \leq 0, |x(i)| = -x(i) \tag{9b}$$

Since both $|x|_1$ and $V(x)$ satisfy mixed-integer linear constraints, the nonlinear objective in (7a) can be converted to linear objective subject to mixed-integer linear constraints. Finally, the constraint $x \in \cup_i \bar{\mathcal{P}}_i$ in (7a) can also be formulated as mixed-integer linear constraints below, with binary variable ζ indicating the active hybrid mode

$$x = \sum_{i=1}^{N} s_i, 1 = \sum_{i=1}^{N} \zeta(i) \tag{10a}$$

$$\bar{P}_i s_i \leq q_i \zeta(i), i = 1, \ldots, N \tag{10b}$$

Note that s_i is the slack variable. The mixed-integer linear constraint (10) requires that when mode i is inactive, $\zeta(i) = 0, s_i = 0$; when mode i is active, $\zeta(i) = 1, s_i = x$. This formulation is also used to partition the state space in [3].

The optimization problem (7a) is cast as a mixed-integer linear program with constraints (8)(9)(10).

To show that the optimization problem (7b) can be cast as an MILP, we will discuss the discrete-time and continuous-time PWL systems separately, as dV has different forms.

2) Cast (7b) *as an MILP for discrete-time systems:* For discrete-time systems, $dV(x_n) = V(x_{n+1}) - V(x_n)$. We can first write the constraints on x_{n+1} as

$$x_{n+1} = \sum_{i=1}^{N} A_i s_i \tag{11}$$

where s_i is the slack variable introduced in Eq.(10), $s_i = x_n$ when the i'th mode is active, and $s_i = 0$ for inactive hybrid mode. Constraint (11) is linear.

In the same way we can derive the mixed-integer linear constraint for the network output $V(x)$ and input x in (8), we can obtain the mixed-integer linear constraints on $V(x_{n+1})$ and x_{n+1}. As a result, we cast (7b) as an MILP.

3) Cast (7b) *as an MILP for continuous-time systems:* For continuous-time systems, $dV(x) = \dot{V}(x) = \frac{\partial V}{\partial x}\dot{x}$, which is a piecewise linear function of x. To see this, note that $\frac{\partial V}{\partial x}$ is a piecewise constant function. $\dot{x}$ is a piecewise linear function of x, so the product becomes a piecewise linear function. Given this intuition, in the next few paragraphs we present the detailed mixed-integer linear constraint formulation.

Similar to the discrete-time systems constraint (11), we impose the constraint on $\dot{x}$ as

$$\dot{x} = \sum_{i=1}^{N} A_i s_i \tag{12}$$

where s_i is introduced as slack variables in formulating the hybrid mode constraint (10).

We notice that $\dot{V} = \frac{\partial V}{\partial x}\dot{x}$ can be computed using the chain rule between subsequent layers of the neural network as

$$\dot{V} = \frac{\partial V}{\partial x}\dot{x} = \left(\prod_{i=0}^{K} \frac{\partial z_{i+1}}{\partial z_i}\right)\dot{x} \tag{13}$$

where we use the notation $z_{K+1} = V(x), z_0 = x$. We also introduce the slack variable y_i defined in the following recursive manner

$$y_0 = \dot{x} \tag{14a}$$

$$y_{i+1} = \frac{\partial z_{i+1}}{\partial z_i} y_i, i = 0, \ldots, K \tag{14b}$$

Combining (13) and (14), it is easy to see that $\dot{V} = y_{K+1}$. Next we show that y_i and y_{i+1} satisfy mixed-integer linear constraints. As $z_{i+1} = \sigma(W_i z_i)$ (Eq.(3)), we can compute the gradient as

$$\frac{\partial z_{i+1}}{\partial z_i} = \left.\frac{\partial \sigma(t)}{\partial t}\right|_{t=W_i z_i} W_i \tag{15}$$

Substituting $\frac{\partial z_{i+1}}{z_i}$ in (14b) with the right hand-side of (15) we obtain

$$y_{i+1} = \left.\frac{\partial \sigma(t)}{\partial t}\right|_{t=W_i z_i} W_i y_i \tag{16}$$

For the leaky ReLU function $\sigma(t)$, its subgradient has the following form

$$\left.\frac{\partial \sigma(t)}{\partial t}\right|_{t=W_i(j,:)z_i} = \begin{cases} 1 \text{ if } W_i(j,:)z_i > 0 \\ c \text{ if } W_i(j,:)z_i < 0 \\ [c, 1] \text{ if } W_i(j,:)z_i = 0 \end{cases} \tag{17}$$

As we introduced binary variables β_i to indicate the activation of the ReLU units in the i'th layer, combining the subgradient in (17) and (16), we obtain the following mixed-integer linear constraints

$$\beta_i(j) = 1 \Rightarrow y_{i+1}(j) = W_i(j,:)y_i,\ W_i(j,:)z_i \geq 0 \tag{18a}$$

$$\beta_i(j) = 0 \Rightarrow y_{i+1}(j) = cW_i(j,:)y_i,\ W_i(j,:)z_i \leq 0 \tag{18b}$$

With the mixed-integer linear constraints (18) on y_i, β_i and the linear constraint (12), we cast the optimization problem (7b) as an MILP.

For both discrete-time and continuous-time PWL systems, we can solve the MILPs in (7a) and (7b) to global optimality. The optimal costs are the worst violation of the Lyapunov conditions (2a) and (2b), and the optimal solution x are the worst adversarial states.

C. Computing gradients of MILP costs w.r.t network parameters

Since our goal is to find the network parameters so as to decrease the Lyapunov condition violation to 0, we need to understand how the optimal costs of MILPs in (7a)(7b), representing the worst violation of the Lyapunov conditions, would change when the network parameters vary. To this end, we aim to compute the gradient of the MILP optimal cost w.r.t the network parameters. This gradient will be used when training the network. By descending the network parameters along this gradient direction, we can decrease the violation of the Lyapunov conditions. The network parameters show up as coefficients in the cost and constraints in the MILPs (for example, in constraint (8) and (18)), hence we need to understand how the MILP optimal cost changes when the cost/constraint coefficients vary.

For a generic MILP

$$\eta_\theta = \max_{x,\gamma} a_\theta^T x + b_\theta^T \gamma \tag{19a}$$
$$\text{s.t } A_\theta x + B_\theta \gamma \le c_\theta \tag{19b}$$
$$\gamma \text{ are binary} \tag{19c}$$

where its cost/constraint coefficients a, b, A, B, c are all differentiable functions of θ (θ are the neural network weights). Its optimal cost η_θ also becomes a differentiable function of θ. To see this, consider when we solve this MILP to optimality, with optimal solution x^*, γ^*, and the active linear constraints at the solution are $A_\theta^{\text{act}} x^* + B_\theta^{\text{act}} \gamma^* = c^{\text{act}}$, where we select the rows in Eq.(19b) that the left hand-side equal to the right hand-side. The optimal continuous variables x^* can be computed as $x^* = (A_\theta^{\text{act}})^\dagger (c_\theta^{\text{act}} - B_\theta^{\text{act}}\gamma^*)$, where $(A_\theta^{\text{act}})^\dagger$ is the pseudo-inverse of A_θ^{act}. We substitute this x^* to the cost function in (19), and obtain the optimal cost as a function of θ

$$\eta_\theta = a_\theta^T \left(A_\theta^{\text{act}}\right)^\dagger \left(c_\theta^{\text{act}} - B_\theta^{\text{act}}\gamma^*\right) + b_\theta^T \gamma^* \tag{20}$$

Since a, b, c, A, B are all differentiable function of θ, we can compute the gradient $\frac{\partial \eta_\theta}{\partial \theta}$, namely the gradient of the MILP optimal cost w.r.t the weights of the neural network through back propagation. It is worth mentioning that when computing this gradient $\frac{\partial \eta_\theta}{\partial \theta}$, we assume that an infinitesimal change of θ does not change the binary variable solution γ^* nor the indices of active constraints.

Since the maximal violation of Lyapunov condition can be computed as the optimal costs of MILPs (7a)(7b), we can also compute the gradient of the maximal violation w.r.t the network parameters. This gradient will be used in the training procedure in the next sub-section.

D. Training

Our goal is to learn a neural network such that the output satisfies the Lyapunov conditions. We define the following loss function, such that by decreasing this loss function to zero, the violation of Lyapunov condition will diminish

$$\begin{aligned}\text{loss}(\theta, \mathcal{X}_1, \mathcal{X}_2) = & w_1 \sum_{x\in\mathcal{X}_1} \max(0, \epsilon_1|x|_1 - V_\theta(x)) + \\ & w_2 \sum_{x\in\mathcal{X}_2} \max(0, dV_\theta(x) + \epsilon_2 V_\theta(x)) + \\ & w_3 \underbrace{\max_{x\in\cup_i\bar{\mathcal{P}}_i} \epsilon_1|x|_1 - V_\theta(x)}_{\text{MILP in (7a)}} + \\ & w_4 \underbrace{\max_{x\in\cup_i\bar{\mathcal{P}}_i} dV_\theta(x) + \epsilon_2 V_\theta(x)}_{\text{MILP in (7b)}}\end{aligned} \tag{21}$$

where $w_1, \ldots, w_4$ are all given non-negative weights. $\mathcal{X}_1, \mathcal{X}_2$ are training data sets. The first two terms in Eq.(21) penalize the violation of the Lyapunov conditions (2a)(2b) on the training sets $\mathcal{X}_1, \mathcal{X}_2$ respectively. The last two terms are the MILPs introduced in III-B, which compute the maximal violation of Lyapunov conditions and the adversarial states.

Algorithm 1 explains our training process. When computing the gradient of the loss function in line 10 of the algorithm, the gradient of the first two terms in (21) are computed through back propagation on the training sets $\mathcal{X}_1, \mathcal{X}_2$. The gradients of the last two terms in (21) can be computed through the procedure explained in sub-section III-C. Note that in each iteration, the newly discovered adversarial states $x_{\text{adv}}^1, x_{\text{adv}}^2$ are added to the training sets. Our algorithm is similar to those in *bilevel optimization* [1], in which in the outer level a loss function is minimized using gradient descent; in the inner level, the loss function is computed by solving a maximization problem (MILP in our case), and the gradient of the maximal cost is used in the outer level.

Algorithm 1 Learning Lyapunov function

```
1:  pre-train a neural network $V_\theta(x)$
2:  success = FALSE
3:  while not success do
4:      Solve MILP $\max_{x\in\cup_i\bar{\mathcal{P}}_i} \epsilon_1|x|_1 - V_\theta(x)$ (Eq.(7a)) with
        optimal solution $x_{\text{adv}}^1$ and optimal cost $\eta_1(\theta)$
5:      Solve MILP $\max_{x\in\cup_i\bar{\mathcal{P}}_i} dV_\theta(x) + \epsilon_2 V_\theta(x)$ (Eq.(7b))
        with optimal solution $x_{\text{adv}}^2$ and optimal cost $\eta_2(\theta)$
6:      if $\eta_1(\theta) == 0$ and $\eta_2(\theta) == 0$ then
7:          success = TRUE. Return
8:      else
9:          Compute loss$(\theta, \mathcal{X}_1, \mathcal{X}_2)$ defined in (21).
10:         Compute the gradient of the loss $\frac{\partial \text{loss}}{\partial \theta}$.
11:         Descend the network parameter $\theta$ along the gradient
            direction $\theta = \theta - \text{step\_size} * \frac{\partial \text{loss}}{\partial \theta}$.
12:         Add $x_{\text{adv}}^1$ to $\mathcal{X}_1$, $x_{\text{adv}}^2$ to $\mathcal{X}_2$.
13:     end if
14: end while
```

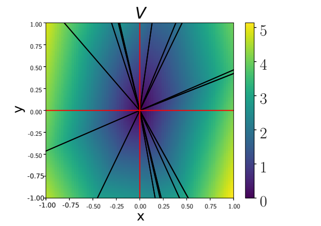
In order to start the training process with a good initial network, in line 1 of Algorithm 1, we pre-train the network with Algorithm 2. Note that the loss function (22) in the pre-training algorithm 2 is just the Lyapunov condition violation on the randomly generated training state $\mathcal{X}_4$. It doesn't require solving MILP as in (21) hence the pretraining is significantly faster than the actual training in Algorithm 1.

IV. Results

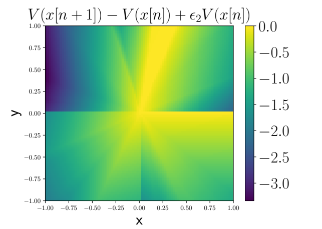
In this section we show that our approach successfully learns Lyapunov functions with non-trivial partitions on a set of discrete-time and continuous-time systems. We include systems which do not admit piecewise linear or quadratic Lyapunov functions by partitioning it according to the hybrid modes. All leaky ReLU units have slope $c = 0.1$ in the negative region. In all examples (except one case in Example 2), our approach finds the Lyapunov function in the first trial, without any tuning on the network structure. We regard the Lyapunov condition being satisfied when the MILP losses are less than 10^{-5}.

Algorithm 2 Pre-train the network on sampled states

```
1: Generate a set of random states $\mathcal{X}_4$.
2: iter = 0
3: while $iter <$ iter_max do
4:     Compute the total violation of Lyapunov condition on the training set $\mathcal{X}_4$ as
       $$\text{loss}(\theta, \mathcal{X}_4) = w_1 \sum_{x\in\mathcal{X}_4} \max(0, \epsilon_1|x|_1 - V_\theta(x)) + w_2 \sum_{x\in\mathcal{X}_4} \max(0, dV_\theta(x) + \epsilon_2 V_\theta(x)) \quad (22)$$
5:     Compute the gradient of the loss in (22) w.r.t $\theta$ through back propagation.
6:     $\theta = \theta -$ step_size $* \frac{\partial \text{loss}(\theta,\mathcal{X}_4)}{\partial\theta}$.
7:     $iter = iter + 1$.
8: end while
```



(a) Lyapunov function value. We also draw the boundaries of each piece in the piecewise Lyapunov function (black lines), and the boundaries of hybrid modes (red lines).



(b) Satisfaction of the condition $V(x_{n+1}) - V(x_n) \leq -\epsilon_2 V(x_n) \forall x_n$ by the learned Lyapunov function.

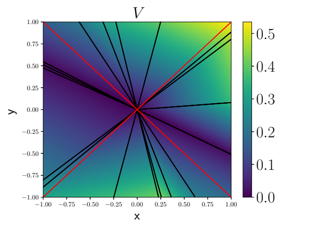
Fig. 3: Lyapunov function for discrete-time system in Eq.(23)

A. Example 1 (discrete-time)

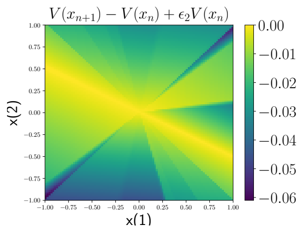
We consider the discrete-time system introduced in [4]

$$x_{n+1} = \begin{cases} \begin{bmatrix} -0.999 & 0 \\ -0.139 & 0.341 \end{bmatrix} x_n, & x_n \in (0,\infty)\times(-\infty,0) \\ \begin{bmatrix} 0.436 & 0.323 \\ 0.388 & -0.049 \end{bmatrix} x_n, & x_n \in [0,\infty)\times[0,\infty) \\ \begin{bmatrix} -0.457 & 0.215 \\ 0.491 & 0.49 \end{bmatrix} x_n, & x_n \in (-\infty,0]\times(-\infty,0] \\ \begin{bmatrix} -0.022 & 0.344 \\ 0.458 & 0.271 \end{bmatrix} x_n, & x_n \in (-\infty,0)\times(0,\infty) \end{cases} \quad (23)$$

We learn a neural network with 2 hidden layers, each layer is of width 4. We show the Lyapunov function in Fig.3. As shown in the plot, the boundaries of the Lyapunov function (black lines) produced by our method are not trivial and do not align with the boundaries (red lines) of each hybrid mode.



(a) Lyapunov function value. We also draw the boundaries of each piece in the piecewise Lyapunov function (black lines), and the boundaries of hybrid modes (red lines).



(b) Satisfaction of the condition $V(x_{n+1}) - V(x_n) \leq -\epsilon_2 V(x_n) \forall x_n$ by the learned Lyapunov function.

Fig. 4: Lyapunov function for discrete-time system in Eq.(24)

B. Example 2 (discrete-time)

We consider the discrete-time system introduced in [28],

$$x_{n+1} = \begin{cases} \begin{bmatrix} 1 & 0.01 \\ -0.05 & 0.897 \end{bmatrix} x_n, |x_n(1)| \leq |x_n(2)| \\ \begin{bmatrix} 1 & 0.05 \\ -0.01 & 0.897 \end{bmatrix} x_n, |x_n(1)| > |x_n(2)| \end{cases} \quad (24)$$

The learned Lyapunov function for this system is shown in Fig. 4. The network has two hidden layers; the first hidden layer has 8 units, and the second hidden layer has 4 units.

In [28], the authors further showed that by modifying the dynamics to

$$x_{n+1} = \begin{cases} \begin{bmatrix} 1 & 0.01 \\ -0.05 & 0.997 \end{bmatrix} x_n, |x_n(1)| \leq |x_n(2)| \\ \begin{bmatrix} 1 & 0.05 \\ -0.01 & 0.998 \end{bmatrix} x_n, |x_n(1)| > |x_n(2)| \end{cases} \quad (25)$$

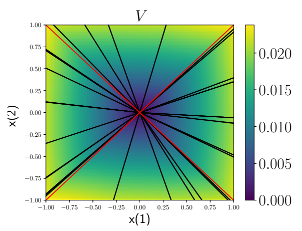
this new system doesn't have a piecewise linear or quadratic Lyapunov function when each piece is determined by the domain of each hybrid mode. (However they could synthesize a piecewise 4-th order Lyapunov function). On the other hand, using our approach, we could find a piecewise linear Lyapunov function for the system in (25). We visualize our Lyapunov function in Fig.5. Our network has 3 hidden layers with 8 units on each of the first two hidden layers, and 2 units on the last hidden layer.

C. Example 3 (continuous-time)

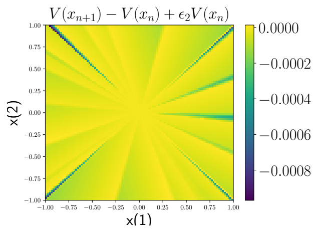
We consider a continuous time system introduced in [10]

$$\dot{x} = \begin{cases} \begin{bmatrix} -0.1 & 1 \\ -10 & -0.1 \end{bmatrix} x, & x(1)x(2) \geq 0 \\ \begin{bmatrix} -0.1 & 10 \\ -1 & -0.1 \end{bmatrix} x, & x(1)x(2) < 0 \end{cases} \quad (26)$$

This system does not have a piecewise linear Lyapunov function when the partition aligns with the mode (certified by the approach in [10]). On the other hand, because our method is able to search over the partitions, it successfully

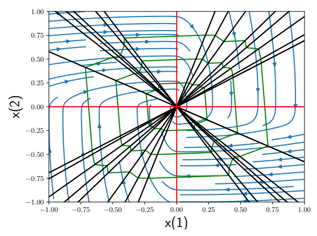


(a) Lyapunov function value. We also draw the boundaries of each piece in the piecewise Lyapunov function (black lines), and the boundaries of hybrid modes (red lines).

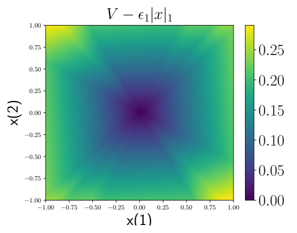


(b) Satisfaction of the condition $V(x_{n+1}) - V(x_n) \leq -\epsilon_2 V(x_n) \forall x_n$ by the learned Lyapunov function.

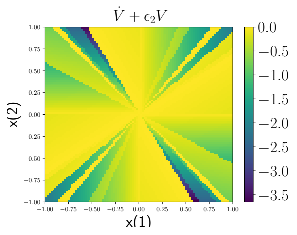
Fig. 5: Lyapunov function for discrete-time system in Eq.(25)



(a) The phase portrait of system (26). The green lines are the contours of $V = 0.05, 0.1, 0.15$. All trajectories point inward w.r.t the contours. The boundaries of the pieces in the piecewise linear Lyapunov function (black lines) do not align the with boundaries of the hybrid modes (red lines).



(b) Satisfaction of condition $V(x) \geq \epsilon_1|x|_1 \quad \forall x$ by the learned Lyapunov function.



(c) Satisfaction of condition $\dot{V} \leq -\epsilon_2 V \; \forall x$ by the learned Lyapunov function.

Fig. 6: Lyapunov function for system in Eq.(26).

.

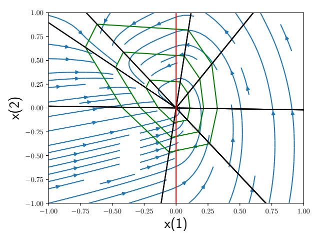
identifies a piecewise linear Lyapunov function. We use a network containing 2 hidden layers, with width 8 in the first hidden layer, and width 4 in the second hidden layer. The results are shown in Fig.6.

D. *Example 4 (continuous-time)*

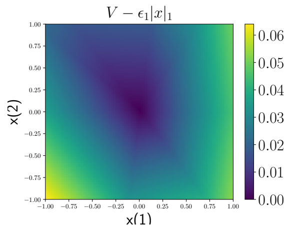
We consider the continuous-time PWL system in [10]

$$\dot{x} = \begin{cases} \begin{bmatrix} -5 & -4 \\ -1 & -2 \end{bmatrix} x, \; x(1) \leq 0 \\ \begin{bmatrix} -2 & -4 \\ 20 & -2 \end{bmatrix} x, \; x(1) > 0 \end{cases} \tag{27}$$

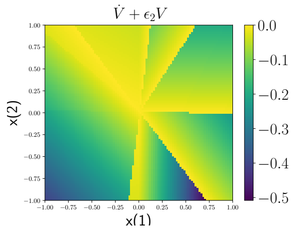
Again, this system does not have a piecewise linear Lyapunov function, if each piece is aligned with each hybrid mode (since the origin is not a vertex of the hybrid modes, a Lyapunov function with pieces aligned with each mode



(a) The phase portrait of system (27). The green lines are the contours of $V = 0.005, 0.01, 0.015$. All trajectories point inward w.r.t the contours. The boundaries of the pieces in piecewise linear Lyapunov function (black lines) do not align the with boundaries of the hybrid modes (red lines).



(b) Satisfaction of condition $V(x) \geq \epsilon_1|x|_1 \quad \forall x$ by the learned Lyapunov function.



(c) Satisfaction of condition $\dot{V} \leq -\epsilon_2 V \; \forall x$ by the learned Lyapunov function.

Fig. 7: Learned Lyapunov function for continuous-time system in Eq.(27).

.

	Adversarial states only	MILP costs only	Adversarial states + MILP costs
Example 1	**110**	119	117
Example 2	4156	4027	**3074**
Example 3	3256	2751	**1311**
Example 4	77	60	**54**

TABLE I: Number of iterations to converge with different loss functions.

would never have the origin as a unique global minimum.) Once again, our approach can easily learn a piecewise linear Lyapunov function. We use a network with 2 hidden layers, with width 4 on the first hidden layer, and width 2 on the second hidden layer. The results are shown in Fig.7.

Finally, we perform an ablation study to compare the convergence rate of our method using three different loss functions 1) Only training on adversarial states (w_3, w_4 are zero in Eq (21)). 2) Only training with MILP costs (w_1, w_2 are zero in Eq (21)). 3) Training with both adversarial states and MILP costs (all $w_1, \ldots, w_4$ are non-zero). The result is summarized in Table I. For the majority of the experiments the learning process converges fastest with both adversarial states and MILP costs. Even though our algorithm involves solving several MILPs, its runtimes remain well within acceptable ranges for each of the examples provided. Specifically, their computation times range from 9s (example 1) to 780s (example 2) on a laptop with Intel Xeon processor.

V. Discussion and Conclusion

In this paper, we showed that we can synthesize and certify Lyapunov functions to prove global exponential convergence of piecewise linear dynamical systems. Our Lyapunov functions are the outputs of neural networks with leaky ReLU activation functions and no bias terms. To learn this Lyapunov function, we apply gradient descent algorithm on the network weights. In each iteration of the gradient descent , we solve MILPs to compute the maximal violation of the Lyapunov conditions, and the worst adversarial states. We append these worst adversarial states to our training set, and compute the gradient of the loss function by differentiating the MILP optimal cost w.r.t the network weights. We demonstrate that our approach can be applied to both continuous-time and discrete-time systems. Unlike previous approaches which require a pre-specified partition of the state space, our approach can freely search the boundary of each piece in the piecewise linear Lyapunov function.

One major limitation of our approach is that we need to solve nonlinear nonconvex problems through gradient descent. This is contrary to previous approaches in which the piece boundaries are fixed, and the Lyapunov function is synthesized through convex optimization. Hence we do not guarantee convergence to a Lyapunov function even if one exists. Moreover, since we need to solve MILPs at each iteration of the learning process, and the number of binary variables scales proportionally with the number of leaky ReLU units in the network, it is computationally expensive to use large neural networks.

We note that our approach can be readily extended to piecewise affine dynamical systems whose hybrid mode boundaries do not need to pass through the origin, and we can also verify the region of attraction for these systems. We will include these results in future work.

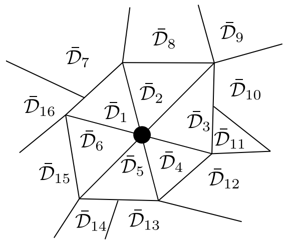
VI. Appendix

In the appendix, we aim to prove that if a stable piecewise linear dynamical system has continuous piecewise affine Lyapunov function, then it has a Lyapunov function that can be represented by a neural network with (leaky) ReLU activation units and no bias terms. To prove this, we first show that we can construct a continuous piecewise linear Lyapunov function from the continuous piecewise affine Lyapunov function, then we prove that this piecewise linear Lyapunov function can be represented by the neural network with zero bias terms.

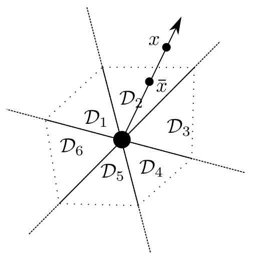
As a first step, we show the existence of a piecewise linear Lyapunov function. We make the distinction between affine function and the linear function, that an affine function $ax+b$ has a constant term b, while the linear function ax does not. Namely a piecewise linear function $V(x)$ is homogeneous $V(kx) = kV(x)\ \forall k \ge 0$.

Theorem 1: If a piecewise linear system in (1) has a piecewise affine Lyapunov function $\bar{V}(x)$, then there exists a piecewise linear Lyapunov function $V(x)$.

Proof: For this piecewise affine Lyapunov function $\bar{V}(x)$, consider an affine piece containing the origin, namely



(a)



(b)

Fig. 8: (left) The domain $\bar{\mathcal{D}}_i$ is a polyhedron piece in the piecewise affine Lyapunov function $\bar{V}(x)$. The origin (black circle) has to be a shared vertex of the neighbouring domains $\bar{\mathcal{D}}_1, \ldots, \bar{\mathcal{D}}_6$. (right) We construct a new piecewise linear Lyapunov function $V(x)$, by removing all the domain boundaries that do not pass through the origin. For the domains $\bar{\mathcal{D}}_1, \ldots, \bar{\mathcal{D}}_6$ neighbouring the origin, we keep the boundaries that pass through the origin, and extend these remaining boundaries to infinity.

$0 \in \bar{\mathcal{D}}_i$, where the polyhedral region $\bar{\mathcal{D}}_i$ is the domain of this piece (do not confuse $\bar{\mathcal{D}}_i$ with the domain of a hybrid mode). Since $\bar{V}(x)$ is an affine function on this domain $\bar{\mathcal{D}}_i$, and $\bar{V}(0)$ is the unique minimum of $\bar{V}(x)$, the origin must be a vertex of the domain $\bar{\mathcal{D}}_i$, because the unique minimal of an affine function $\bar{V}(x)$ on a polyhedron domain $\bar{\mathcal{D}}_i$ has to be a vertex of the polyhedron. Hence the origin is the common vertex of the neighbouring domains, as shown in Fig.8a. And within each domains neighbouring the vertex, the Lyapunov function $\bar{V}(x)$ has to be a linear function of x, rather than an affine function, since $\bar{V}(0) = 0$.

To construct a new piecewise linear Lyapunov function $V(x)$ from $\bar{V}(x)$, we remove all the domain boundaries in $V(x)$ that do not pass through the origin. For the domains neighbouring the origin, we keep the boundaries that pass through the origin, and extend these boundaries to infinity. Namely for each of the domain $\bar{\mathcal{D}}_i$ neighbouring the origin, we compute the *cone* of $\bar{\mathcal{D}}_i$ as $\mathcal{D}_i = \{kx | x \in \bar{\mathcal{D}}_i, k \ge 0\}$. This process is shown in Fig.8b. The newly constructed function $V(x)$ has the same form in $\mathcal{D}_i$ as $\bar{V}(x)$ in $\bar{\mathcal{D}}_i$. Namely if $\bar{V}(x) = a_i^T x$ when $x \in \bar{\mathcal{D}}_i \cap \mathcal{D}_i$, then

$$V(x) = a_i^T x \text{ if } x \in \mathcal{D}_i \tag{28}$$

We need to prove that this newly constructed function $V(x)$ is a valid Lyapunov function. Here we use the fact that both the dynamical equation and the Lyapunov function are homogeneous functions of state x. Specifically, when $x \in \bar{\mathcal{D}}_i \cap \mathcal{D}_i$, $V(x) = \bar{V}(x)$, hence $V(x)$ satisfies the Lyapunov conditions when $x \in \bar{\mathcal{D}}_i \cap \mathcal{D}_i$. When $x \notin \bar{\mathcal{D}}_i \cap \mathcal{D}_i$ (for example, x in Fig.8b), we shoot a ray connecting the origin and x. There exists a point $\bar{x} = kx, k > 0$, s.t $\bar{x} \in \bar{\mathcal{D}}_i \cap \mathcal{D}_i$ (shown in Fig.8b). Hence we can prove that $V(x)$ is strictly positive as $V(x) = V(kx)/k = V(\bar{x})/k = \bar{V}(\bar{x})/k \ge \epsilon_1|\bar{x}|_1/k = \epsilon_1|x|_1$. The first equality is because $V(x)$ is a linear function of x in $\mathcal{D}_i$; the second equality is because $\bar{x} = kx$ by definition; the third equality is because $V(\bar{x}) = \bar{V}(\bar{x})$ as $V(\bar{x})$ coincides with $\bar{V}(\bar{x})$ within $\bar{x} \in \bar{\mathcal{D}}_i \cap \mathcal{D}_i$; the last inequality is because $\bar{V}$ is a Lyapunov function thus satisfying condition $\bar{V}(\bar{x}) \geq \epsilon_1 |\bar{x}|_1$ (Eq.(2a)). Hence we prove that $V(x)$ also satisfies the strict positivity condition $V(x) \geq \epsilon_1 |x|_1$. Likewise, we can show that $dV(x) = d\bar{V}(\bar{x})/k \leq -\epsilon_2 \bar{V}(\bar{x})/k = -\epsilon_2 V(\bar{x})/k = -\epsilon_2 V(x)$. The first equality is because the dynamics satisfies $\dot{\bar{x}} = k\dot{x}$; the inequality is because $\bar{V}$ is a Lyapunov function thus satisfying condition $d\bar{V}(\bar{x}) \leq -\epsilon_2 \bar{V}(\bar{x})$ (Eq.(2b)); the last two equalities hold as we explained before. We therefore prove that the newly constructed piecewise linear function $V(x)$ satisfies the Lyapunov conditions (2a)-(2c). ∎

Based on universal approximation theorem [16], any continuous function can be approximated with arbitrarily high accuracy by a neural network with one hidden layer and (leaky) ReLU activation functions. Since a neural network with (leaky) ReLU activation functions represents a piecewise affine relationship between the input and the output, the theorem implies that a piecewise linear Lyapunov function can always be represented by a neural network with one hidden layer and (leaky) ReLU activation functions. In this paper we restrict to neural networks for which the bias terms are all zero. Next we prove that there is no loss of generality with this restriction, that a continuous piecewise linear function can always be represented by a (leaky) ReLU neural network with zero bias terms. For clarity, we prove the claim for neural networks with scalar outputs (which is the network used in this paper), but the theorem is also trivially extended to networks with multiple outputs.

Theorem 2: If a neural network $\phi(\mathbf{x})$ with one hidden layer, a scalar output layer and (leaky) ReLU activation function is piecewise linear, namely it satisfied $\phi(k\mathbf{x}) = k\phi(\mathbf{x})\ \forall k \geq 0, \forall \mathbf{x}$, then this network has all its biases equal to 0.

Proof: This network can be formulated as

$$\phi(\mathbf{x}) = \mathbf{w}_2^T \sigma(W_1 \mathbf{x} + \mathbf{b}_1) + b_2 \tag{29}$$

where $W_1, \mathbf{b}_1$ are the weights/bias in the hidden layer, and $\mathbf{w}_2, b_2$ are the weights/bias in the output layer. If any entry in $\mathbf{w}_2$ is zero, then we could just remove that entry and the corresponding rows in W_1 and $\mathbf{b}_1$ and obtain a smaller network with the same output. Hence we can safely suppose $\mathbf{w}_2(i) \neq 0\ \forall i$. Our goal is to prove that both $\mathbf{b}_1 = 0$ and $b_2 = 0$.

We can prove this claim by by contradiction. First, suppose $\mathbf{b}_1 \neq \mathbf{0}$. Without loss of generality, we suppose that the first m entries in $\mathbf{b}_1$ are non-zero, the other entries are 0. We use $W_1(i,:)$ to denote the i'th row of matrix W_1, and $\bar{W}_1$ to denote the sub-matrix containing first m rows of W_1; $\bar{\mathbf{w}}_2, \bar{\mathbf{b}}_1$ to denote the sub-vector containing the first m entries of $\mathbf{w}_2, \mathbf{b}_1$ respectively. We further make the following assumption

Assumption 1: there are no two rows in the matrix $\bar{W}_1, \bar{\mathbf{b}}_1$ satisfying $\bar{W}_1(i,:)/\bar{\mathbf{b}}_1(i) = \bar{W}_1(j,:)/\bar{\mathbf{b}}_1(j)$ for $1 \leq i,j \leq m$. Namely in the matrix $[\bar{W}_1\ \bar{\mathbf{b}}_1]$, there is not a row being a muliplier of another row.

If such two rows exist, we can add the j'th row of $\bar{W}_1, \bar{\mathbf{b}}_1, \bar{\mathbf{w}}_2$ to their i'th rows, and remove the j'th row. This new network has the same output as before. Hence the assumption 1 is always valid.

The condition $\phi(k\mathbf{x}) = k\phi(\mathbf{x}) \forall k \geq 0$ means that for any fixed $\mathbf{x}$, the function $\phi(k\mathbf{x})$ as a function of k has a fixed slope, namely

$$\frac{\partial \phi(k\mathbf{x})}{\partial k} = \text{constant}\ \forall k \geq 0 \tag{30}$$

, in the remaining of the proof we will show that we can construct certain $\mathbf{x}$ such that the slope in (30) is not a constant if $\mathbf{b}_1 \neq \mathbf{0}$.

Since $\bar{\mathbf{b}}_1$ is supposed to be non-zero, there exists a vector $\mathbf{x}^*$ such that some entries in $\bar{W}_1\mathbf{x}^*$ take the opposite sign as the corresponding entries in $\bar{\mathbf{b}}_1$, namely there exists $1 \leq i \leq m$ such that $sign(W_1(i,:)\mathbf{x}^*) = -sign(\mathbf{b}_1(i))$. Without loss of generality we assume such entries are the first n entries of $\bar{W}_1\mathbf{x}^*$ and $\bar{\mathbf{b}}_1$. We can further require that $\mathbf{x}^*$ is so small such that $|\bar{W}_1(i,:)\mathbf{x}| \leq |\bar{\mathbf{b}}_1(i)|\ \forall i$, hence $\mathbf{x}^*$ also satisfies $sign(\bar{W}_1\mathbf{x}^* + \bar{\mathbf{b}}_1) = sign(\bar{\mathbf{b}}_1)$, namely adding small $\bar{W}_1\mathbf{x}^*$ to $\bar{\mathbf{b}}_1$ doesn't change its sign. We will show that there exists some $k > 0$ such that along the ray $k\mathbf{x}^*$, the condition $\phi(k\mathbf{x}^*) = k\phi(\mathbf{x}^*)$ does not hold.

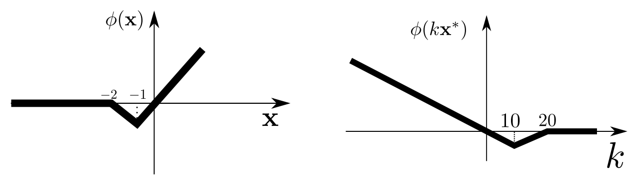
To show this, among all the entries i such that $sign(\bar{W}_1(i,:)\mathbf{x}^*) = -sign(\bar{\mathbf{b}}_1(i))$, we compute the ratio $k^i = -\bar{\mathbf{b}}_1(i)/(\bar{W}_1(i,:)\mathbf{x}^*)$. Apparently $k^i > 0$ since the numerator and denominator have different signs. We further assume the array $[k^i], i = 1, 2, \ldots, n$ has a unique minimal (if the minimal is not unique, namely there are two indices i, j such that $k^i = k^j$ and both k^i, k^j are the smallest entries of the array, then according to assumption 1, we only need to perturb x^* a little bit to break the tie, while the perturbed x^* still satisfies the sign requirement). Without loss of generality, we can assume the smallest k^i is k^1, and the second smallest k^i is k^2, namely $0 < k^1 < k^2 \leq k^i\ \forall i > 2$. Notice that $\phi(k\mathbf{x}^*) = k\phi(\mathbf{x}^*)$ indicates that $\frac{\partial \phi(k\mathbf{x}^*)}{\partial k}$ is a constant that is independent of k, but we will show that the slope $\frac{\partial \phi(k\mathbf{x}^*)}{\partial k}$ is different for $0 < k < k^1$ versus $k^1 < k < k^2$. Notice that when k increases from 0 to k^2, the signs of all entries $\bar{W}(i,:)k\mathbf{x}^* + \mathbf{b}_1(i)$ don't change except for $\bar{W}(1,:)k\mathbf{x}^* + \mathbf{b}_1(1)$, which flips the sign at $k = k^1$. Since the (leaky) ReLU unit $\sigma(y)$ takes a different slope as y changes the sign, and none of the entries in $\mathbf{w}_1$ is zero, the gradient $\frac{\partial \phi(k\mathbf{x}^*)}{\partial k}$ changes when k increases from below k^1 to above k^1. Algebraically we see it by writing the slope using the chain rule

$$\frac{\partial \phi(k\mathbf{x}^*)}{\partial k} = \sum_i \mathbf{w}_2(i)\bar{W}_1(i,:)\mathbf{x}^*\ \frac{\partial \sigma(y)}{\partial y}\bigg|_{y=\bar{W}_1(i,:)k\mathbf{x}^* + \mathbf{b}_1(i)} \tag{31}$$

we can see that all the terms in equation (31) don't change with k, except for $i = 1$, which changes when y flips sign at $k = k^1$. Therefore we obtain the contradiction that $\frac{\partial \phi(k\mathbf{x}^*)}{\partial k}$ is not a constant, and $\phi(k\mathbf{x}^*) \neq k\phi(\mathbf{x}^*)$. So we conclude that the bias $\mathbf{b}_1$ in the hidden layer has to be 0.

If $b_2 \neq 0$, then since $\phi(\mathbf{0}) = 0$, we obtain

$$\mathbf{w}_2^T \sigma(\mathbf{b}_1) + b_2 = 0 \tag{32}$$



(a) The network output as a function of input $\mathbf{x}$.

(b) The network output $\phi(k\mathbf{x}^*)$ as a function of scalar k.

Fig. 9: To illustrate the proof for theorem 2, we draw a simple neural network $\phi(x) = 2\sigma(x+1) - \sigma(x+2)$ in Fig. 9a. The output of the network $\phi(\mathbf{x})$ passes through the origin. Here $W_1 = [1,1]^T, \mathbf{b}_1 = [1,2]^T, \mathbf{w}_2 = [2,-1]^T, b_2 = 0$. We can find a small $\mathbf{x}^* = -0.1$, such that $sign(W_1\mathbf{x}^*) = -sign(\mathbf{b}_1)$, but $sign(W_1\mathbf{x}^* + \mathbf{b}_1) = sign(\mathbf{b}_1)$. We also draw the function $\phi(k\mathbf{x}^*)$ as a function of k in Fig 9b. Here we compute $k^1 = -\mathbf{b}_1(1)/(W_1(1,:)\mathbf{x}^*) = 10$ and $k^2 = -\mathbf{b}_1(2)/(W_1(2,:)\mathbf{x}^*) = 20$. When $0 < k < k^1$, we have $\frac{\partial\phi(k\mathbf{x}^*)}{\partial k} = -0.1$, but when $k^1 < k < k^2$ we have a different slope $\frac{\partial\phi(k\mathbf{x}^*)}{\partial k} = 0.1$, this demonstrates that $\phi(k\mathbf{x}) = k\phi(\mathbf{x})$ cannot hold for this network with non-zero bias.

this implies that $\mathbf{b}_1 \neq \mathbf{0}$, but from the discussion above $\mathbf{b}_1$ has to be a 0-vector, hence $b_2 = 0$.

The idea of the proof is visually illustrated in the Fig 9. ■

References

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

[2] Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control. In *Advances in Neural Information Processing Systems*, pages 3240–3249, 2019.

[3] Hongkai Dai, Gregory Izatt, and Russ Tedrake. Global inverse kinematics via mixed-integer convex optimization. *The International Journal of Robotics Research*, 38(12-13):1420–1441, 2019.

[4] Giancarlo Ferrari-Trecate, Francesco Alessandro Cuzzola, Domenico Mignone, and Manfred Morari. Analysis of discrete-time piecewise affine and hybrid systems. *Automatica*, 38(12):2139–2146, 2002.

[5] Pascal Gahinet, Arkadii Nemirovskii, Alan J Laub, and Mahmoud Chilali. The lmi control toolbox. In *Proceedings of 1994 33rd IEEE Conference on Decision and Control*, volume 3, pages 2038–2041. IEEE, 1994.

[6] Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International conference on automated deduction*, pages 208–214. Springer, 2013.

[7] Incorporate Gurobi Optimization. Gurobi optimizer reference manual. *URL http://www. gurobi. com*, 2018.

[8] Sigurdur F Hafstein, Christopher M Kellett, and Huijuan Li. Computing continuous and piecewise affine lyapunov functions for nonlinear systems. *Journal of Computational Dynamics*, 2(2):227, 2015.

[9] https://yalmip.github.io/tutorial/bigmandconvexhulls/. *Big-M and convex hulls*, Sep 2016.

[10] Mikael Johansson. *Piecewise linear control systems*. PhD thesis, Ph. D. Thesis, Lund Institute of Technology, Sweden, 1999.

[11] Mikael Johansson and Anders Rantzer. Computation of piecewise quadratic lyapunov functions for hybrid systems. In *1997 European Control Conference (ECC)*, pages 2005–2010. IEEE, 1997.

[12] 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.

[13] J Zico Kolter and Gaurav Manek. Learning stable deep dynamics models. In *Advances in Neural Information Processing Systems*, pages 11126–11134, 2019.

[14] Benoit Landry, Zachary Manchester, and Marco Pavone. A differentiable augmented lagrangian method for bilevel nonlinear optimization. *arXiv preprint arXiv:1902.03319*, 2019.

[15] Joseph LaSalle. Some extensions of liapunov's second method. *IRE Transactions on circuit theory*, 7(4):520–527, 1960.

[16] 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.

[17] Michael Lutter, Boris Belousov, Kim Listmann, Debora Clever, and Jan Peters. Hjb optimal feedback control with deep differential value functions and action constraints. *arXiv preprint arXiv:1909.06153*, 2019.

[18] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[19] Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000.

[20] Hadi Ravanbakhsh and Sriram Sankaranarayanan. Counter-example guided synthesis of control lyapunov functions for switched systems. In *2015 54th IEEE conference on decision and control (CDC)*, pages 4232–4239. IEEE, 2015.

[21] Arthur Richards and Jonathan How. Mixed-integer programming for control. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 2676–2683. IEEE, 2005.

[22] Spencer M Richards, Felix Berkenkamp, and Andreas Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. *arXiv preprint arXiv:1808.00924*, 2018.

[23] Daniel Shevitz and Brad Paden. Lyapunov stability theory of nonsmooth systems. *IEEE Transactions on automatic control*, 39(9):1910–1914, 1994.

[24] RN Shorten and KS Narendra. On the stability and existence of common lyapunov functions for stable linear switching systems. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171)*, volume 4, pages 3723–3724. IEEE, 1998.

[25] Miriam García Soto and Pavithra Prabhakar. Averist: Algorithmic verifier for stability of linear hybrid systems. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 259–264, 2018.

[26] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[27] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

[28] Jun Xu and Lihua Xie. Homogeneous polynomial lyapunov functions for piecewise affine systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 581–586. IEEE, 2005.