Stability analysis and control using single-hidden-layer ReLU neural networks

Hasan A. Poonawala and Pouya Samanipour

***Abstract*—This paper presents algorithms to solve analysis and controller synthesis problems for dynamical systems modeled as a recurrent single-hidden-layer rectified linear unit neural network (ReLU NN), or equivalently, a piecewise affine dynamical system. Such models are interesting since they may arise through the use of modern machine learning methods for system identification, or as closed-loop solutions in certain classes of model predictive control (MPC) problems. A key idea in the proposed approach is to use piecewise affine Lyapunov functions parametrized as ReLU neural networks, and similar controllers. This compatible representation between the Lyapunov function and the models simplifies the automation of analysis of and controller synthesis for learned models. The controllers and Lyapunov functions are 'learned' using both weight-updates and network architecture search, without gradients. We demonstrate the proposed algorithm on examples involving learned models, explicit MPC controllers, and constrained controller synthesis.**

I. Introduction

Modern machine learning has made fitting complex functions to large amounts of data a realistic task to accomplish [1]. This ability is increasingly finding applications in control and dynamical systems. Two obvious applications are in system identification, where the right hand side of the dynamics models are learned from data [2]–[4], and analysis, where Lyapunov functions are 'learned' [5]–[10].

If we are to increasingly model dynamical systems using over-parametrized models like neural networks, we require methods to analyze such models, and to possibly synthesize controllers from them. The potential complexity of such models suggests that the methods we develop must be amenable to automatic, and preferably efficient, computations.

Automatic methods for Lyapunov-based analysis and synthesis must deal with the fact that Lyapunov-based methods require solving a constraint satisfaction problem at infinite states [11]. One approach to doing so is to reformulate these infinite constraints involving states into finite constraints involving the parameters of the Lyapunov function and dynamical system. This method is preferable since it often leads to convex optimization problems that yield valid Lyapunov functions [12]–[15] or controllers [14]. These methods are parametrization-specific, and to the best of our knowledge, such reformulations have not been developed for models and Lyapunov functions in the form of neural networks. Section II reviews the literature in detail.

Hasan A. Poonawala and Pouya Samanipour are with the Department of Mechanical Engineering, University of Kentucky, Lexington, KY 40506, USA. hasan.poonawala,samanipour.pouya@uky.edu

This work was supported by the Department of Mechanical Engineering at the University of Kentucky.

An alternative approach to dealing with infinite constraints is inspired by supervised machine learning [16], where a model that works on an infinite input set may be learned from a finite dataset. This idea leads to sampling-based methods for 'learning' Lyapunov functions [5]–[10]. Unlike standard machine learning, in Lyapunov analysis we require that the stability conditions hold at all states, even when learned using finite states. This requirement forces use of a *verification* step. Again, Section II reviews the literature in detail.

The main **idea** in this paper is to use Lyapunov functions parametrized as single-hidden-layer rectified linear unit neural networks (ReLU NNs) for analysis and controller synthesis. This choice turns out to have advantages over other parametrizations, at least when the dynamical systems are piecewise affine. Briefly, the resulting method is fully automated, formulates a quadratic program (QP) instead of a semidefinite program (SDP), and avoids the verification step.

To use ReLU NN Lyapunov functions for analysis, we must tackle four **problems**. The first is that the piecewise nature of these functions suggests that any algorithm may get bogged down in combinatorial enumerations. The second is that it is not obvious how to update the weights of the ReLU NN Lyapunov function if we wish to avoid gradient descent using finite samples. The third is that we do not know how wide the ReLU NN needs to be. The fourth is that we consider continuous time systems, unlike most work on learning Lyapunov functions, and so we must account for the non-differentiable nature of the Lyapunov functions we choose when formulating Lyapunov conditions.

Contributions.: We propose a sound algorithm for verifying stability properties of dynamical systems models involving single-hidden-layer ReLU NNs using single-hidden-layer ReLU NN Lyapunov functions in a fully automated way. To do so, we first formulate the stability conditions at all states into exactly equivalent constraints on the parameters of the dynamical system and ReLU NN Lyapunov function. Then, we convert these constraints into a sequential quadratic optimization problem for finding a Lyapunov function. Specifically, each iteration of the quadratic optimization problem 'trains' the outer layer weights, keeping the hidden layer fixed. Between iterates, we intelligently add neurons to the Lyapunov function's neural network, by viewing their role as defining a partition of the state space. We extend this analysis method to create controller synthesis algorithms based on bilinear optimization. For controlled dynamical systems where the input enters linearly, we propose a method to synthesize a controller parametrized by a ReLU NN partially derived from the system dynamics.

II. Related Work

In this section, we review various approaches to automatically searching for Lyapunov functions, which can be grouped into the two categories below. We also briefly explain how our work relates to these approaches.

A. Mathematical Optimization

Theorems of Alternatives [14], [17]–[19] are used to convert Lyapunov stability conditions involving quantifiers on the state into quantifier-free constraints. Which particular theorem is applied by a method depends on the parametrizations of the dynamical systems and Lyapunov function.

Sums-of-squares-based analysis is widely used for polynomial systems [20], [21]. A good amount of work focuses on homogenous Lyapunov functions for homogenous dynamical systems. [22]. Most remaining work has focused on stability analysis for switched and hybrid dynamical systems [23], [24]. These works include [15], [15], [25]–[28].

Common [23], multiple [29], or piecewise [25] quadratic Lyapunov functions are used to design switching rules [30]–[35] between affine or linear dynamics modes. Recent work in [36] reports that synthesis using PWQ Lyapunov functions may not be practical beyond a small number of cells.

Piecewise linear (PWL) Lyapunov functions were studied as an alternative to piecewise quadratic (PWQ) functions in [37] and [26]. A few methods attempt to find piecewise linear Lyapunov functions for piecewise affine systems [26], [38]–[40] and piecewise linear systems [41]. Despite some theoretical [37] and computational advantages [26] of PWL functions over PWQ ones, the latter still dominate stability analysis.

The ability to convert Lyapunov-based analysis of piecewise affine (PWA) dynamical systems into convex optimization problems has motivated attempts to automate the search for PWQ and PWL Lyapunov functions [26]–[28], [40], [42]. These works observe that an automated partitioning scheme may enable analysis of systems for which using the same partition as the dynamics for the PWQ or PWL Lyapunov function fails to produce a valid Lyapunov function. They use the optimal variables to identify which cells to refine, and then divide these cells into two or more cells, typically along a longest edge.

Despite this broad work in analysis of PWA systems, only recently has their representation using ReLU neural networks been explored (see below). Our approach uses inhomogenous forms of Farkas' Lemma [17] to reformulate the Lyapunov stability conditions for Lyapunov functions and dynamical systems parametrized by ReLU NNs. Therefore, our approach checks Lyapunov stability conditions at all states, unlike sampling-based methods. We also refine partitions when these conditions fail, however we do so by adding neurons to the single hidden layer in the ReLU NN.

B. Learning From Finite Samples

Several methods are inspired by the intuitive idea that if we make a candidate Lyapunov function satisfy stability conditions at a sufficient number of finite state samples, the stability conditions may be satisfied at all points in a region of interpolation between these samples. To ensure that this 'generalization' happens, these approaches search for a counterexample, a state where the conditions fail, with the hope that none will be found. The differences between various learning-from-samples methods lies in what tools are available to implement a search for a counter example, and how this counterexample is used.

One approach for using the counterexample is to add it to the training samples [5], [6], [8], [9], [41] and repeat the learning process. A second approach uses the counterexample to prune the set of parameters in which a valid Lyapunov function may be found [10], [43]. Either a Lyapunov function is found in the remaining set, or the set is pruned to emptiness and no Lyapunov function exists for the system in the considered class.

Instead of manually derived optimization problems that drive the automatic search for a Lyapunov function or a counterexample, some research directions use satisfiability modulo theory (SMT) solvers [44], [45]. These methods have been applied to dynamics that are piecewise constant [46], set-valued dynamics [47], nonlinear systems [6], [8], [48].

In our work, we obtain a set of stability conditions for each cell in the partition created by the Lyapunov function and dynamical system. We may view cells for which these stability conditions fail as counterexamples to the existence of a Lyapunov function with that partition, and we use them to add neurons to our Lyapunov function. These neurons effectively divide the cell that generates them, in addition to other cells.

III. Problem Description

We consider the **stability analysis** problem for dynamical systems of the form

$$\dot{x} = A_c x + \mathrm{PWA}(x) + \mathrm{ReLU}(x), \tag{1}$$

where terms $\mathrm{PWA}(x)$ and $\mathrm{ReLU}(x)$ denote piecewise affine functions and rectified neural networks respectively. Section IV defines these functions more precisely. For now, we mention that the piecewise affine functions are continuous with cells that are polytopes, and the rectified neural networks have a single hidden layer.

We also consider the **controller synthesis** problem for dynamical systems of the form

$$\dot{x} = A_c x + b_c + \mathrm{PWA}(x) + \mathrm{ReLU}(x) + Bu, \tag{2}$$

using controllers of the form

$$u = K_c x + k_c + \mathrm{ReLU}(x). \tag{3}$$

The natural motivation for this choice is that the closed-loop under (3) of system (2) reduces to the form (1) that we can analyze, enabling synthesis.

For both problems, we will use Lyapunov functions $V(x)$ that are parametrized as $\mathrm{ReLU}(x)$. The methods we develop are able to deal with closed-loop properties such as stability, asymptotic stability, and exponential stability. However, we focus on asymptotic stability to simplify the exposition, with the remaining properties described in Section VIII-D.

IV. Representations of Piecewise Affine Functions

This section introduces notation we use in this paper, for both standard piecewise affine functions and rectified linear unit neural networks. We refer to the standard representation of piecewise affine functions (see Section IV-B) as an **explicit parametrization**, while single-hidden-layer ReLU NNs (see Section IV-C) are an **implicit parametrization** of piecewise affine functions.

Notation: The indices of the elements of a set S form the set $I(S)$. We use the bold symbol $\mathbf{x}_S$ to denote a set of variables $\{x_i\}_{i\in I(S)}$, and $\mathbf{x}_I$ to denote a set of variables $\{x_i\}_{i\in I}$.

We denote the convex hull of S by $\mathrm{conv}\,(S)$, the interior of S by $\mathrm{Int}\,(S)$, the boundary of S by ∂S, and closure of S by $\overline{S}$. Let $B_\epsilon(x) = \{y \in R^n : \|y - x\| < \epsilon$, an ϵ-ball at x.

Let I be a set of indices. Given a matrix E, E^I denotes a matrix formed by stacking the i^{th} rows of E, for $i \in I$. The transpose of matrix A is A^T.

For $v, u \in \mathbb{R}^n$, $v \succeq u \iff v_i \geq u_i$ for $1 \leq i \leq n$. The symbols $\preceq$, $\succ$, and $\prec$ imply the same element-wise rule corresponding to $\leq$, $>$, and $<$ respectively.

Let $\mathcal{B}^m \subset \mathbb{R}^m = \{0,1\}^m$, the set of m-dimensional vectors whose elements are 0 or 1. The vector $\mathbf{1}_m \in \mathcal{B}^m$ has all elements equal to unity.

A. Partitions and Refinements

A partition $\mathcal{P}$ is a collection of subsets $\{X_i\}_{i\in I(\mathcal{P})}$, where $X_i \subseteq \mathbb{R}^n$, $n \in \mathbb{N}$, and $\overline{\mathrm{Int}\,(X_i)} = X_i$ for each $i \in I(\mathcal{P})$. Furthermore, $\mathrm{Int}\,(X_i) \cap \mathrm{Int}\,(X_j) = \emptyset$ for each pair $i, j \in I(\mathcal{P})$ such that $i \neq j$. We refer to $\cup_{i\in I(\mathcal{P})} X_i$ as the domain of $\mathcal{P}$, which we also denote by $\mathrm{Dom}(\mathcal{P})$. We also refer to the subsets X_i in $\mathcal{P}$ as the cells of the partition. We assume that there exists a neighborhood of x that intersects with only a finite number of cells in $\mathcal{P}$, for each $x \in \mathrm{Dom}(\mathcal{P})$.

Let $\mathcal{P} = \{Y_i\}_{i\in I}$ and $\mathcal{R} = \{Z_j\}_{j\in J}$ be two partitions of a set $S = \mathrm{Dom}\,(\mathcal{P}) = \mathrm{Dom}\,(\mathcal{R})$. A partition $\mathcal{R}$ is a *refinement* of $\mathcal{P}$ if $Z_j \cap Y_i \neq \emptyset$ implies that $Z_j \subseteq Y_i$. We denote the set of refinements of a partition $\mathcal{P}$ as $\mathrm{Ref}(\mathcal{P})$.

In this paper, we only consider piecewise functions where the **cells** in the underlying partitions **are only polytopes**.

B. Piecewise Affine Functions

We explicitly parameterize a piecewise affine function $\mathrm{PWA}(x)$ by a partition $\mathcal{P} = \{X_i\}_{i\in I(\mathcal{P})}$ and a collection of matrices $\mathbf{A}_\mathcal{P} = \{A_i\}_{i\in I(\mathcal{P})}$ and vectors $\mathbf{a}_\mathcal{P} = \{a_i\}_{i\in I(\mathcal{P})}$ such that

$$\mathrm{PWA}(x) = A_i x + a_i, \text{ if } x \in X_i, \text{ where} \tag{4}$$

$$X_i = \{x \in \mathbb{R}^n : E_i x + e_i \succeq 0\}. \tag{5}$$

Note that a generic piecewise affine function may not be continuous unless we appropriately constrain the parameters A_i, a_i, E_i, and e_i [26], [38]. We assume that any piecewise affine function available in this explicit form meets such constraints, and is therefore always continuous.

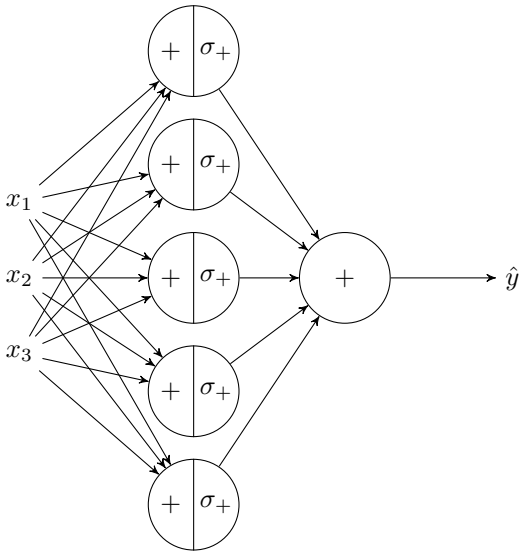

Fig. 1: Depiction of a single-hidden-layer rectified linear unit (ReLU) neural network with five hidden neurons, where the input $x \in \mathbb{R}^3$ and the output is scalar.

C. Single-Hidden-Layer ReLU NNs

A single-hidden-layer ReLU neural network defines a map from $x \in \mathbb{R}^n$ to output $\mathrm{ReLU}(x) \in \mathbb{R}^o$:

$$\mathrm{ReLU}(x) = W\sigma_+.(Hx + b) + c, \tag{6}$$

where $W \in \mathbb{R}^{o\times m}$, $H \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^{o\times 1}$, and the function $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function. We choose the activation as the rectified linear activation σ_+ given by

$$\sigma_+(x) = \max(x, 0). \tag{7}$$

The period in the notation $\sigma_+.(v)$ in (6) implies that we apply this function to a vector $v \in \mathbb{R}^m$ in an element-wise manner to produce another vector in $\mathbb{R}^m$.

The pair (H^i, b^i) – the i^{th} row of H and the i^{th} element of b – together represent a single neuron in the hidden layer of the ReLU neural network, as seen in Figure 1. We refer to the pair (H, b) as the neurons of the network. The matrix W and vector c denote the outer layer weights, since they define the linear combination of the hidden layer outputs that produces the output of the network.

The function $\mathrm{ReLU}(x)$ is piecewise affine in x. We therefore refer to it as an implicit parametrization of a piecewise affine function. The next section shows how we transition between the two representations.

V. Converting Representations

Traditional methods in analysis of piecewise dynamical systems focus on explicit representations of the dynamics and partition [25], [26]. To work in an exact manner with dynamics like (1) that include ReLU neural network functions we need to be able to simultaneously work with both explicit and implicit representations. This section shows how we do so.

A. Implicit Encoding Of A Partition Induced by $\mathrm{ReLU}(x)$

A function $\mathrm{ReLU}(x)$, where $x \in \mathbb{R}^n$, induces a partition of $\mathbb{R}^n$ that depends on the parameter H, b as follows. Given parameters $H \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, we define the $\{0,1\}$-vector-valued function $d^{H,b}\colon X \to \mathcal{B}^m$ by the element-wise function

$$\left(d^{H,b}\right)_i(x) = \begin{cases} 1, & \text{if } H^i x + b^i > 0, \\ 0, & \text{if } H^i x + b^i < 0, \end{cases} \tag{8}$$

where H^i and b^i are the i^{th} rows of H and b respectively. We may then define a $\{0,1\}$-matrix-valued function $D^{H,b}(x)$ as the diagonal matrix derived from $d^{H,b}(x)$. These functions are piecewise constant in x, and defined on the interiors of the cells in the partition of the piecewise affine function that the ReLU NN encodes. Using the matrix function $D^{H,b}(x)$, the ReLU neural network model becomes

$$\begin{aligned} \hat{y}(x) &= W\sigma_+.(Hx+b)+c && (9)\\ &= WD^{H,b}(x)\,(Hx+b)+c && (10)\\ &= WD^{H,b}(x)H\ x + WD^{H,b}(x)b + c && (11) \end{aligned}$$

where we assume that ${D^{H,b}}_{ii}(0)\cdot 0 = 0$ to handle the lack of definition of ${D^{H,b}}_{ii}(0)$. Clearly, two points x_1 and x_2 belong to the same cell in the partition when $d^{H,b}(x_1) = d^{H,b}(x_2)$. The vector $d^{H,b}(x)$ therefore serves as an **implicit** encoding of which cell a point belongs to. We present an example by way of illustration.

Example 1. Consider $x \in \mathbb{R}^2$ and

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \text{ and } b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

While there are 16 possible values for a $\{0,1\}$-valued vector in four dimensions, the function $d^{H,b}(x)$ will have only four values v_i for $i \in \{1,2,3,4\}$, corresponding to the four quadrants in the plane:

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \ v_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \ v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \text{ and } v_4 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Note that $d^{H,b}(x)$ is treated as undefined on the axes, otherwise it would have another five values corresponding to the origin and the four rays originating from it along the axes. ■

For any m, the set $\mathcal{B}^m$ has 2^m possible elements. The possible encodings of points may be a subset of $\mathcal{B}^m$, as Example 1 shows. We may generate all valid encodings of the cells by computing $d^{H,b}(X)$, the image of $X \subseteq \mathbb{R}^n$ under $d^{H,b}$, which we denote as $\mathcal{BS}(X,H,b)$. Since $d^{H,b}(x)$ is constant over the interior of a cell, we may calculate $\mathcal{BS}(X,H,b)$ from a sufficiently dense set of **samples** of X.

Let $I_{H,b}$ denote the index set $I\left(\mathcal{BS}(X,H,b)\right)$. We denote the the i^{th} element of $\mathcal{BS}(X,H,b)$ as $d_i^{H,b}$, where $i \in I_{H,b}$, and define $D_i^{H,b} = \operatorname{diag}\left(d_i^{H,b}\right)$.

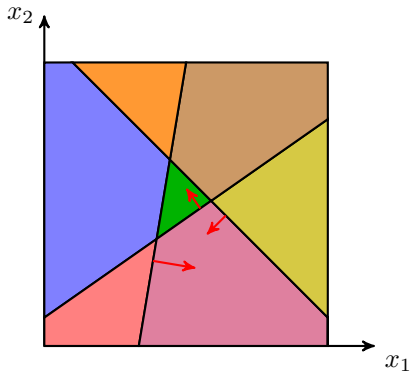

Fig. 2: Three neurons, with orientation marked by red arrows, implicitly define seven regions in the plane. However, only the green region is exactly defined by conditions of the form (13).

B. Explicitly Enumerating and Describing All Cells In A Partition Induced by $\mathrm{ReLU}(x)$

To implement an exact check of Lyapunov stability conditions at all states in X, we will need an explicit characterization of the (polytopic) cells in a partition of X as a set of linear inequalities. Since we may generate the encodings $\mathcal{BS}(X,H,b)$ by sampling X, one potential approach to deriving this explicit characterization is to consider the following collection of cells $\mathcal{P}_{H,b}$:

$$\mathcal{P}_{H,b} = \{X_i \colon i \in I_{H,b}\}, \tag{12}$$

where

$$X_i = \left\{x \in \mathbb{R}^n \colon D_i^{H,b}Hx + D_i^{H,b}b \geq 0\right\}. \tag{13}$$

We need the collection $\mathcal{P}_{H,b}$ above to be a partition of X. As the next example shows, this need is not automatically met.

Example 2. Consider $X = \mathbb{R}^2$ and

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and } b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

The set $\mathcal{BS}(X,H,b)$ will have only four values, that is, $\mathcal{BS}(X,H,b) = \{v_1, v_2, v_3, v_4\}$, where

$$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \ v_3 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \text{ and } v_4 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

However, the non-trivial cells defined by (13) include the first quadrant, the halfspace on the positive side of the x-axis, and the halfspace on the positive side of the y-axis. These sets do not form a partition, since they do not have disjoint interiors, and their union does not cover $\mathbb{R}^2$. ■

Figure 2 depicts another example. The heart of the matter is that a neuron (H^i, b^i) implicitly defines two halfspaces, but explicitly defines only one when using conditions like (13). We may need the neuron $(-H^i, -b^i)$ to also be contained in (H,b), so that both halfspaces may be explicitly defined. Sufficient conditions for when $\mathcal{P}_{H,b}$ defines a partition for X is given in the next result.

Proposition 1. *Let $X \subseteq \mathbb{R}^n$ have non-empty interior, and $H \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ be a collection of m neurons. If*

1) *the Hamming distance between any two elements in* $\mathcal{BS}(X, H, b)$ *is at least* 2,
2) *every boundary of the set* X *is a subset of* $H^i x + b^i = 0$ *for some* i*, where* $1 \leq i \leq m$*, and*
3) *for every* $i \in \{1, \ldots, m\}$ *there exists* $x \in \mathrm{Int}\,(X)$ *such that* $H^i x + b^i > 0$,

then the set $\mathcal{P}_{H,b}$ *derived from* $\mathcal{BS}(X, H, b)$ *forms a partition of* X.

Proof: Due to the definition of the map $d^{H,b}(x)$ in (8) and the cells of $\mathcal{P}_{H,b}$ in (13), $\overline{\mathrm{Int}\,(X_i)} = X_i$ for all $i \in I_{H,b}$. If the hyperplane corresponding to a neuron (H^k, b^k) in (H, b), for some $k \in \{1, \ldots, m\}$, passes through $x^* \in X$, then we can find two points $x_1, x_2 \in \mathrm{Int}\,(X)$ close to x^* where $d^{H,b}(x_1)$ and $d^{H,b}(x_2)$ can only differ only due to the sign of $H^k x + b^k$. Again, by definition of (8), condition 1 implies that the neuron $(-H^k, -b^k)$ must also be a neuron in (H, b), otherwise the Hamming distance between $d^{H,b}(x_1)$ and $d^{H,b}(x_2)$ can only be one, which is a contradiction. By (13), two points on either side of the hyperplane given by $H^k x + b^k = 0$ must belong to disjoint sets, so that the interiors of X_i, for all $i \in I_{H,b}$, are disjoint. Therefore, $\mathcal{P}_{H,b}$ is a partition.

Condition 2 ensures that if $x \in X_i$ for some $i \in I_{H,b}$, then $x \in X$. Condition 3 ensures that $\mathcal{P}_{H,b}$ is non-empty if X has non-empty interior. Therefore, conditions 2 and 3 together ensure that $\mathcal{P}_{H,b}$ is a partition of X. ∎

If X, H, and b meet the conditions in Proposition 1, then $\mathcal{BS}(X, H, b)$ correctly enumerates all cells in a partition of X, and (13) defines each of these cells as a set of linear inequalities.

C. Converting $\mathrm{ReLU}(x)$ to $\mathrm{PWA}(x)$

If a ReLU neural network function $\mathrm{ReLU}(x)$ has neurons that satisfy Proposition 1, then $\mathcal{P}_{H,b}$ forms a partition of $\mathbb{R}^n$. We may then describe $\mathrm{ReLU}(x)$ as an explicit piecewise affine function over a *partition* $\mathcal{P}_{H,b}$:

$$\begin{aligned} \hat{y}(x) = W D_i^{H,b} H\ x + W D_i^{H,b} b + c, \\ \text{if } D_i^{H,b} H\ x + D_i^{H,b} b \geq 0. \end{aligned} \tag{14}$$

D. Partially converting $\mathrm{PWA}(x)$ to $\mathrm{ReLU}(x)$

The machinery of the previous sections allow us to partially convert a function in form $\mathrm{PWA}(x)$ to the form $\mathrm{ReLU}(x)$. By partial, we mean that we may describe the affine function using a partition encoded by hidden neurons.

Let the partition associated with $\mathrm{PWA}(x)$ be $\mathcal{Q} = \{Z_i\}_{i \in I(\mathcal{Q})}$ where $Z_i = \{x \in \mathbb{R}^n : E_i x + e_i \succeq 0\}$. Let $X = \mathrm{Dom}\,(\mathcal{Q})$, and the affine function over Z_i be $A_i x + a_i$.

We construct hidden neurons H and b by concatenating the matrices E_i and vectors e_i respectively for all $i \in I(\mathcal{Q})$. This construction ensures that conditions 2 and 3 in Proposition 1 are met. Next, we compute $\mathcal{BS}(X, H, b)$. If the Hamming distance between two elements $d_i^{H,b}, d_j^{H,b} \in \mathcal{BS}(X, H, b)$ is 1, and the neuron (H^k, b^k) in (H, b) defines a boundary between $X_i, X_j \in \mathcal{P}_{H,b}$, then we must include $(-H^k, -b^k)$ in (H, b). Addition of these neurons in (H, b) ensures that the first condition in Proposition 1 is satisfied for the resulting neurons (H, b). Therefore, at the end of this construction, the collection of sets $\mathcal{P}_{H,b}$ will be form a partition of $X = \mathrm{Dom}\,(\mathcal{Q})$, by Proposition 1.

The final step is to identify which affine term from $\mathrm{PWA}(x)$ must be used for each cell in $\mathcal{P}_{H,b}$. That is, we identify an abstraction function $\pi : I_{H,b} \to I(\mathcal{Q})$. Let (H, b) consist of m neurons. To an element $Z_j \in \mathcal{Q}$ we may assign a vector $v_j \in \mathcal{B}^m$ which has non-zero elements corresponding to those neurons in (H, b) that define the boundaries of Z_j. These neurons must exist by construction of H, b. Then, $X_i \subseteq Z_j$, where $X_i \in \mathcal{P}_{H,b}$, if the k^{th} element of $d_i^{H,b} \in \mathcal{BS}(X, H, b)$ is non-zero whenever the k^{th} element of v_j is non-zero.

With these constructions, we may redefine $\mathrm{PWA}(x)$ as

$$\begin{aligned} \mathrm{PWA}(x) = A_{\pi(i)} x + a_{\pi(i)}, \\ \text{if } D_i^{H,b} H x + D_i^{H,b} b \geq 0 \end{aligned} \tag{15}$$

We remark that exactly fitting a function $\mathrm{ReLU}(x)$ with parameters (W, H, b, c) to $\mathrm{PWA}(x)$ is not straightforward, since training to zero loss using supervised learning techniques is impractical. Fortunately, the partial conversion above suffices for the algorithms we propose. We may now represent the right hand side of (1) as an explicit piecewise affine function, where the partition depends on neurons (H, b) derived from both $\mathrm{ReLU}(x)$ and from converting $\mathrm{PWA}(x)$ using the method described above.

VI. Lyapunov ReLU Neural Networks

We will use candidate Lyapunov functions of the form

$$V(x) = w_V \sigma_+ \left(H_V x + b_V\right). \tag{16}$$

A candidate Lyapunov function must be positive definite, i.e., $V(0) = 0$ and $V(x) > 0$ when $x \neq 0$. This section formulates equivalent constraints on the parameters w_V, H_V, and b_V such that these conditions are met.

To facilitate these equivalent constraints we make two assumptions:

A1 $\mathcal{P}_{H_V, b_V} \in \mathrm{Ref}\,(\mathcal{P}_{H,b})$, where H, b are defined by (1).
A2 $0 \notin \mathrm{Int}\,(Z_i)$, for all $i \in I_{H_V, b_V}$.

Assumption **A1** is easily satisfied by constructing H_V, b_V using the neurons H, b associated with the dynamics (obtained for $\mathrm{PWA}(x)$ using methods in Section V-D). Specifically, we assume that H_V and b_V are of the form

$$H_V = \begin{bmatrix} H \\ H_V' \end{bmatrix}, \quad b_V = \begin{bmatrix} b \\ b_V' \end{bmatrix}, \tag{17}$$

for some H_V' and b_V'. We achieve assumption **A2** by ensuring that H_V, b_V contains n independent hyperplanes passing through the origin, and their reflections.

To ensure positive definiteness of $V(x)$ in (16), we use its corresponding explicit representation as a piecewise affine function, similar to (14):

$$\begin{aligned} V = w_V D_i^{H_V, b_V} \left(H_V x + b_V\right) \text{ if} \\ D_i^{H_V, b_V} H_V x + D_i^{H_V, b_V} b_V \succeq 0. \end{aligned} \tag{18}$$

We distinguish between cells that contain $x = 0$ on the boundary and those that do not, similar to the approach by

Johansson in [26]. Note that by assumption **A2**, the origin is always on the boundary of cells. Let

$$I^{lin}_{H_V,b_V} = \{i \in I_{H_V,b_V} : 0 \in \partial X_i\}, \tag{19}$$
$$I^{aff}_{H_V,b_V} = \{i \in I_{H_V,b_V} : 0 \notin \partial X_i\}. \tag{20}$$

To ensure that $V(0) = 0$, we constrain the constant term $w_V D_i^{H_V,b_V} b_V$ to be zero for regions X_i where $i \in I^{lin}_{H_V,b_V}$. Depending on whether a cell contains the origin on the boundary or not, we use the following results to ensure sign definiteness of such explicit piecewise affine functions.

Lemma 2 (Lemma 4.7 [26])**.** *The following are equivalent*

1) $Ex \succeq 0, Ex \neq 0 \implies p^T x > 0$.
2) $\exists v \succ 0$ *such that* $E^T v = p$.

To convert the conditional statement $Gx + g \succeq 0 \implies p^T x + q \leq 0$ into a constraint without conditions, we use the following result shown in [38]:

Lemma 3 ([38])**.** *Let the set* $\{x \in \mathbb{R}^n : Gx + g \succeq 0\}$ *be non-empty, where* $G \in \mathbb{R}^{l \times n}$, $g \in \mathbb{R}^l$ *for some* $l \in \mathbb{N}$. *Let* $p \in \mathbb{R}^n$ *and* $q \in \mathbb{R}$. *Then, the following are equivalent*

1) $Gx + g \succeq 0 \implies p^T x + q \leq 0$.
2) $\exists v \in \mathbb{R}^l$, $v \succeq 0$ *such that* $G^T v + p = 0$ *and* $g^T v + q \leq 0$.

Lemma 4. *Consider a function* V *as defined in equations* (16). *If there exist* $\epsilon_1 > 0$, $\epsilon_2 > 0$ *and* μ_i *for* $i \in I_{H_V,b_V}$ *such that*

$$\mu_i^T D_i H_V - w_V D_i H_V = 0, \quad \forall i \in I^{lin}_{H_V,b_V}, \tag{21}$$
$$w_V D_i b_V = 0, \quad \forall i \in I^{lin}_{H_V,b_V} \tag{22}$$
$$, \mu_i \succeq \epsilon_1 \mathbf{1}, \quad \forall i \in I^{lin}_{H_V,b_V}, \tag{23}$$
$$\mu_i^T D_i H_V - w_V D_i H_V = 0, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{24}$$
$$\mu_i^T D_i b_V - w_V D_i b_V + \epsilon_2 \leq 0, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{25}$$
$$\mu_i \succeq 0, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{26}$$

then $V(x)$ *is strictly positive for* $x \neq 0$.

Proof: This result is a straightforward application of Lemmas 2 and 3 to a strict positivity constraint applied piecewise for all the regions in $\mathcal{P}_{H_V,b_V}$ defined by the ReLU Lyapunov NN with parameters w_V, H_V, b_V. ∎

VII. Lyapunov-based Stability Conditions

Conditions (21)-(26) determine when a function of the form in (16) is a candidate Lyapunov function. In this section, we present constraints that determine when a candidate Lyapunov function is a valid Lyapunov function certifying asymptotic stability. Section VIII-D describes modifications that verify other closed-loop properties.

If the dynamics contains a term of the form $\mathrm{PWA}(x)$ in (1), we assume that the function $\mathrm{ReLU}(x)$ has neurons (H, b) which include neurons that result from applying the procedure in Section V-D to $\mathrm{PWA}(x)$. In turn, we have an abstraction function π available, enabling mapping of cells of $\mathcal{P}_{H,b}$ to the affine functions of $\mathrm{PWA}(x)$.

A. Stability Conditions At A Single State

When $V(x)$ is differentiable at x, and the dynamics $\dot{x} = f(x)$ are continuous, stability may be assessed through the condition that the Lie derivative $\mathcal{L}_f V$ of V along $f(x)$ be negative definite or negative semi-definite for all x in some region $S \ni 0$:

$$\mathcal{L}_f V = \langle \nabla V, f(x) \rangle < 0, \tag{27}$$

where ∇V is the gradient of V and $\langle \cdot, \cdot \rangle$ is the usual inner product.

At a state x, let the affine Lyapunov function be $V = p^T x + q$, and the dynamics be $\dot{x} = Ax + a$. The Lyapunov stability condition (27) at this state is

$$p^T (Ax + a) < 0, . \tag{28}$$

B. Stability Conditions For A Single Cell

Since the Lyapunov functions and dynamics we consider are piecewise affine, the parameters involved in condition (28) are identical for all states in connected cells. If such a cell $X_i \in \mathcal{P}_{H_V,b_V}$ is defined by the constraints $E_i x + e_i \geq 0$, then we can compactly represent the condition at all states in the cell through the quantified condition

$$E_i x + e_i \geq 0, x \neq 0 \implies p_i^T (A_i x + a_i) < 0. \tag{29}$$

Depending on whether $i \in I^{lin}_{H_V,b_V}$ or not, we may use either Lemma 2 or Lemma 3 to eliminate the quantifier.

C. Non-smoothness

When x belongs to the boundary of cells in $V(x)$ (where it is non-differentiable) the quantity $\nabla V(x)$ must be replaced by its multi-valued extension [49], [50]. Therefore, we may need multiple conditions of the form (28) to hold. Since the Lyapunov function is derived from the partition of the dynamics, the points of discontinuity of the dynamics are a subset of those of the Lyapunov function. The main **idea** here is that checking conditions for each cell, which includes its boundaries, automatically takes care of the multi-valued conditions at the points of discontinuity of V. Note that our closed-loop dynamics are continuous, even if non-differentiable, and therefore no sliding behavior occurs [51].

D. Stability Conditions For Multiple Cells

We assume that we are checking stability conditions over a connected region defined by a subset of $\mathcal{P}_{H_V,b_V}$. Given parameters w_V, H_V, and b_V, we may obtain a set of conditions that verify asymptotic stability of a dynamical system by combining conditions that determine candidacy with the conditions that determine decrease of the Lyapunov function along solutions, for each cell.

First, we simplify the notation. The construction of H_V, b_V in (17) using (H, b) allows us to rewrite the dynamics term of the form $\mathrm{ReLU}(x)$ as

$$W\sigma_+ (Hx + b) \to W_x \sigma_+ (H_V x + b_V), \tag{30}$$

where W_x combines W with a matrix of zeroes:

$$W_x = \begin{bmatrix} W & 0 \end{bmatrix}. \tag{31}$$

Now that we only deal with neurons (H_V, b_V), we drop the superscripts in $D_i^{H_V,b_V}$ for simplicity. Finally, we define the following symbols for each $i \in i_{H_V,b_V}$:

$$p_i = w_V D_i H_V, \tag{32}$$
$$q_i = w_V D_i b_V, \tag{33}$$
$$A_i = A_c + A_{\pi(i)} + W_x D_i H_V, \text{ and} \tag{34}$$
$$a_i = a_{\pi(i)} + W_x D_i b_V. \tag{35}$$

Theorem 5. *Consider a dynamical system of the form 1, and a candidate Lyapunov ReLU neural network function $V(x)$ of the form (16). If the parameters of $V(x)$ satisfy the constraints (21)-(26), in addition to the constraints*

$$0 = (D_i H_V)^T \nu_i + A_i^T (p_i)^T, \quad \forall i \in I^{lin}_{H_V,b_V}, \tag{36}$$
$$a_i = 0, \quad \forall i \in I^{lin}_{H_V,b_V} \tag{37}$$
$$\nu_i \succ 0, \quad \forall i \in I^{lin}_{H_V,b_V}, \tag{38}$$
$$0 = (D_i H_V)^T \nu_i + A_i^T (p_i)^T, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{39}$$
$$0 > (D_i b_V)^T \nu_i + a_i^T (p_i)^T, \quad \forall i \in I^{aff}_{H_V,b_V}, \text{ and} \tag{40}$$
$$\nu_i \succeq 0, \quad t_i \geq 0, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{41}$$

then the origin $x = 0$ is asymptotically stable.

Proof: Constraints (21)-(26) ensure that $V(x)$ is a valid candidate Lyapunov function, by Lemma 4. We must show that the Lyapunov function decreases appropriately along solutions of the dynamical system, where we consider solutions to be Caratheodory solutions [40], [52]. We do so using the generalized Clarke gradient $\partial V(x)$ of $V(x)$ [52]–[54].

Since $V(x)$ is piecewise affine, at every x, $\partial V(x)$ is simply the convex hull of the set $\{p_i\}_{i \in I(x)}$, where $I(x) = \{i \in I_{H_V,b_V} : x \in X_i\}$ [54], and p_i is given in (32). This generalized gradient is single-valued on the interior of cells in $\mathcal{P}_{H_V,b_V}$, and set-valued on boundaries between cells. If the constraints above are feasible, then by Lemmas 2 and 3, we may conclude that $p_i^T(A_i^T x + a_i) < 0$ for all $x \neq 0$, $x \in X_i \in \mathcal{P}_{H_V,b_V}$, for each $i \in I_{H_V,b_V}$. This enumeration over all $i \in I_{H_V,b_V}$ ensures that this condition is checked at x for each $i \in I(x)$. Therefore, we may conclude that $p^T f < 0$ for all $p \in \partial V(x)$, at each $x \neq 0$, where f is the dynamics at x. From results in [54], we may therefore conclude that at $\dot{V}(t) < 0$ almost everywhere along solutions of the dynamical system, so that the origin of (1) is (locally) asymptotically stable. ∎

To summarize, we have derived conditions on the parameters w_V, H_V, and b_V of a candidate Lyapunov function $V(x)$ that when satisfied allow us to conclude that a dynamical system of the form (1) is asymptotically stable. These conditions are exact, and cover all $x \in X$, $x \neq 0$, allowing us to avoid the verification step used by sampling-based approaches to Lyapunov function search. In the next sections, we use these conditions to search for a Lyapunov function using sequential optimization, and to synthesize controllers for controlled systems of the form (2).

VIII. Algorithms For Stability Analysis

This section proposes an algorithm for finding Lyapunov functions of the form (16) that verify stability properties of the origin of dynamical systems of the form (1). The key insight is that the role of the hidden neurons parametrized by (H_V, b_V) should be to define a partition, while the output layer weights w_V should define a function over that partition. This insight leads to an alternation between choosing the weights of the output layer of the ReLU NN Lyapunov function for a given set of neurons, and inserting new neurons. We use optimization problem $\text{Opt}(H_V, b_V)$ in Section VIII-A to choose w_V, wherein H_V, b_V are fixed. If the optimal value is non-zero, we use the solution to insert new neurons into the Lyapunov ReLU neural network. In effect, we reformulate the non-convex optimization problem typically solved using gradient descent as an alternation between convex optimization and neural architecture search.

A. 'Learning' Outer-Layer Weights

The conditions in Theorem 5 allow us to search for a Lyapunov function $V(x)$ given (H_V, b_V) using linear programming. If this linear program is feasible, we verify asymptotic stability of the origin. What should we do when it is infeasible? We solve this problem by introducing slack variables for some of the constraints, and changing the objective function to the sum of the norms of these slack variables.

For each $i \in I_{H_V,b_V}$, we introduce slack variables s_i for constraints (36) and (39), and slack variables t_i for (40). We then define the objective function $J_{H_V,b_V}(w_V)$ given by

$$J_{H_V,b_V}(w_V) = \sum_{i \in I_{H_V,b_V}} \|s_i\|_2^2 + \|t_i\|_2^2, \tag{42}$$

The optimization problem $\text{Opt}(H_V, b_V)$ that implements a search for w_V given $\mathcal{Q}$ is

$$\min_{w_V, \mu_i, \nu_i, s_i, t_i} \quad J_{H_V,b_V}(w_V) \tag{43}$$
$$\text{s.t.} \quad \mu_i^T D_i H_V - p_i = 0, \quad \forall i \in I^{lin}_{H_V,b_V}, \tag{44}$$
$$q_i = 0, \quad \forall i \in I^{lin}_{H_V,b_V} \tag{45}$$
$$, \mu_i \succeq \epsilon_1 \mathbf{1}, \quad \forall i \in I^{lin}_{H_V,b_V}, \tag{46}$$
$$\mu_i^T D_i H_V - p_i = 0, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{47}$$
$$\mu_i^T D_i b_V - q_i + \epsilon_2 \leq 0, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{48}$$
$$\mu_i \succeq 0, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{49}$$
$$s_i = (D_i H_V)^T \nu_i + A_i^T (p_i)^T, \quad \forall i \in I^{lin}_{H_V,b_V}, \tag{50}$$
$$a_i = 0, \quad \forall i \in I^{lin}_{H_V,b_V} \tag{51}$$
$$\nu_i \succeq \epsilon_2 \mathbf{1}, \quad \forall i \in I^{lin}_{H_V,b_V}, \tag{52}$$
$$s_i = (D_i H_V)^T \nu_i + A_i^T (p_i)^T, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{53}$$
$$t_i \geq \epsilon_2 + (D_i b_V)^T \nu_i + a_i^T (p_i)^T, \quad \forall i \in I^{aff}_{H_V,b_V} \tag{54}$$
$$\nu_i \succeq 0, \quad t_i \geq 0, \quad \forall i \in I^{aff}_{H_V,b_V}, \tag{55}$$

where $\epsilon_1 > 0$ and $\epsilon_2 > 0$. The optimization problem $\text{Opt}(H_V, b_V)$ in (43)-(55) contains several variables, of which w_V is most important, and the rest are related to establishing properties of the function $V(x)$. We state the following result:

Lemma 6. *The optimization problem* $\text{Opt}(H_V, b_V)$ *in* (43)-(55) *is always feasible.*

Proof: This result is by construction. By including n independent neurons with no bias terms in H_V, b_V, we may ensure that the Lyapunov candidate function is positive definite. Therefore constraints (21)-(26) will always be feasible by themselves. The remaining constraints are always feasible for any value of w_V, H_V, b_V due to use of slack-like variables s_i and t_i. Therefore $\text{Opt}(H_V, b_V)$ is always feasible. ■

While $\text{Opt}(H_V, b_V)$ is always feasible, only the case where the optimal value is zero is useful, due to Theorem 5. The next section describes how we continue searching for a valid ReLU NN Lyapunov function then the optimal value of $\text{Opt}(H_V, b_V)$ is non-zero.

B. Adding Neurons

In this work, we add neurons (H^{new}, b^{new}) such that they define hyperplanes that split cells $X_i \in \mathcal{P}_{H_V,b_V}$ for which slack variables s_i and/or t_i in the solution to $\text{Opt}(H_V, b_V)$ are non-zero. Consider two index sets I_s and I_t, where

$$I_s = \{i \in I_{H_V,b_V} : s_i \neq 0\}, \text{ and}$$
$$I_t = \{i \in I_{H_V,b_V} : t_i \neq 0\}.$$

At least one of these sets are non-empty when $\text{Opt}(H_V, b_V)$ has non-zero optimal value. These index sets lead to the following rules for refining cells:

1) If I_s is not empty, split all cells in I_s.
2) Otherwise, split all cells in I_t.

If the slack variable s_i is zero in the solution to $\text{Opt}(H_V, b_V)$, then by constraint (50), the vector $-A_i^T p_i$ is a strictly positive combination of the hyperplanes that define $X_i \in \mathcal{P}_{H_V,b_V}$, where $i \in I^{lin}_{H_V,b_V}$. If $X_i = \{x \in X : E_i \succeq 0\}$, then we may find $\mu \succ 0$ such that

$$-A_i^T p_i = E_i^T \mu^*$$

. If the slack is non-zero, then this combination μ^* may have negative elements. The idea in [40], [42] is to use μ^* to define a splitting hyperplane, such that at $-A_i^T p_i$ is a positive combination of the hyperplanes defining one of the resulting cell. The work in [40], [42] focus on conewise linear systems where the cones have exactly n sides, so that E_i is square and invertible.

Here, we generalize this approach to deal with cells that are polytopes defined by an arbitrary number of hyperplanes, not just cones with exactly n-sides and apex at the origin. We solve for μ^* as the optimal point of the following quadratic optimization problem.

$$\min_{\mu} \quad \|\mu\|_2^2, \tag{56}$$
$$\text{s.t.} \quad E_i^T \mu + A_i^T p_i = 0, \text{ and} \tag{57}$$
$$e_i^T \mu + a_i^T p_i \leq 0. \tag{58}$$

We then define

$$\mu^*_{bin} = d^{I,0}(\mu^*) - d^{I,0}(-\mu^*), \tag{59}$$

Algorithm 1 Verifying Stability using Lyapunov ReLU NNs

Require: PWA(x), H, b, $\epsilon_1 > 0$, $\epsilon_2 \geq 0$
Ensure: ReLU NN Lyapunov function $V(x)$ that verifies the origin is (asymptotically) stable.
  $H_V \leftarrow H$, $b_V \leftarrow b$ {Initialize network}
  $J_{H_V,b_V}(w_V) \leftarrow \infty$
  while $J_{H_V,b_V}(w_V) \neq 0$ **do**
    Solve $\text{Opt}(H_V, b_V)$.
    $I_s \leftarrow \{i \in I_{H_V,b_V} : s_i \neq 0\}$
    $I_t \leftarrow \{i \in I_{H_V,b_V} : t_i \neq 0\}$
    for $i \in I_s$ **do**
      Compute (H^{new}, b^{new}), add to (H_V, b_V) (see Section VIII-B)
    end for
    if $I_s = \emptyset$ **then**
      for $i \in I_t$ **do**
        Compute (H^{new}, b^{new}), add to (H_V, b_V) (see Section VIII-B)
      end for
    end if
  end while
  return $V(x) = w_V \sigma_+ . (H_V x + b_V)$.

which has non-zero entries for strictly positive and negative entries of μ^*. We may compute the neuron (H^{new}, b^{new}) that serves to split X_i as

$$H^{new} = \begin{cases} (\mu^*_{bin})^T H, & \text{if } s_i \neq 0 \\ (\mu^*)^T H, & \text{if } s_i = 0, \end{cases} \tag{60}$$

and

$$b^{new} = \begin{cases} (\mu^*_{bin})^T b, & \text{if } s_i \neq 0 \\ (\mu^*)^T b, & \text{if } s_i = 0. \end{cases} \tag{61}$$

We use μ^*_{bin} based on the observation that in a few cases, μ may have values where some values are small and others large, so that (H^{new}, b^{new}) is very similar to an existing neuron. The vector μ^*_{bin} makes (H^{new}, b^{new}) divide a cell into more uniformly sized subsets. Empirically, we find that using μ^* itself is more effective when splitting cells for which $t_i \neq 0$.

Finally, we update (H_V, b_V) by concatenating them with neurons (H^{new}, b^{new}) and $(-H^{new}, -b^{new})$. This concatenation of matrices and vectors is equivalent to inserting a neuron (and its reflection) in the ReLU neural network Lyapunov function.

C. Main Algorithm

Algorithm 1 describes the algorithm resulting from these choices. We can show the following properties.

Proposition 7. *Algorithm 1 is sound.*

Proof: The algorithm terminates when $\text{Opt}(H_V, b_V)$ has optimal value zero. By Theorem 5, the candidate Lyapunov function $V(x)$ is therefore a valid Lyapunov function certifying asymptotic stability of the origin. ■

D. Extensions To Other Closed-Loop Properties

The discussion so far has focused on asymptotic stability. The extension to properties such as stability, exponential stability, and ultimate boundedness are straight forward.

1) Stability: We simply choose $\epsilon_2 = 0$.

2) Exponential Stability: We may show exponential stability by requiring that $\dot{V} \leq -kV$ for some $\alpha > 0$. This constraint implies that for each cell $X_i \in I_{H_V,b_V}$, we need

$$p^T(Ax+a) < -\alpha p^T x, \text{ or } \left(p^T A + \alpha p^T\right) + p^T a < 0.$$

The constraints in our optimization-based algorithms barely change, with the linear dynamics matrix being modified from A to $A + \alpha I$. We must either choose α through some side information, or use some procedure to search for it. We may use methods inspired by bisection algorithms, however we do not pursue this further here.

3) Ultimate Boundedness: In some cases, we may need to show ultimate boundedness to some set X_0 containing the origin, instead of asymptotic stability. For example, when models are learned from data, the vector field at the origin may be non-zero, even if small. Instead of adding a correction term, which would be added to the vector field at all states, we may show ultimate boundedness of the learned dynamics.

If this set is characterized by the constraint $Ex + e \geq 0$, then we can add (E, e), $(-E, -e)$ to the neurons (H_V, b_V), and remove the Lyapunov decrease conditions (50)-(52), corresponding to cells containing the origin in their interior or boundary.

4) Convex Lyapunov Functions: If we constrain the outer weights w_V of the Lyapunov ReLU neural network to satisfy $w_V \succeq 0$, then the Lyapunov function will be convex.

IX. Algorithms For Controller Synthesis

Consider a controlled dynamical system of the form

$$\dot{x} = A_c x + b_c + W\sigma_+.(Hx+b) + \text{PWA}(x) + Bu, \quad (62)$$

where we assume that (H, b) are compatible with $\text{PWA}(x)$ using methods in Section V-D.

We will use a neural network controller of the form (3), with parameters W_u, H_u, and b_u, combined with an affine term $K_c x + k_c$. Like the Lyapunov function in Section VIII, we construct H_u, and b_u to include H and b. We choose the Lyapunov functions to have the same hidden-layer neurons as the controller: $H_V = H_u$, $b_V = b_u$, but with possibly different outer layer parameters $W_u \neq w_V$. In other words, the controller and Lyapunov function share their hidden layer neurons. With these choices, we may represent the closed-loop system as

$$\dot{x} = A_c x + b_c + (W_x + BW_u)\,\sigma_+.(H_V x + b_V) + \text{PWA}(x).$$

We obtain an explicit piecewise representation of the closed-loop functions as

$$\dot{x} = A_i x + a_i, \text{ if } D_i H_V x + D_i b_V \geq 0, \quad (63)$$

where

$$A_i = A_c + BK_c + A_{\pi(i)} + (W_x + BW_u)\,D_i H_V, \text{ and} \quad (64)$$
$$a_i = b_c + Bk_c + a_{\pi(i)} + (W_x + BW_u)\,D_i b_V. \quad (65)$$

For synthesis, we minimize the same objective as in (42). We may then derive stability conditions for verifying asymptotic stability of the controlled closed-loop that are identical to those in Theorem 5, except the affine dynamics are now given by (64) and (65) instead of (34) and (35). As a result, we will obtain a nearly identical optimization problem $\text{Opt}_{\text{synth}}(H_V, b_V)$ to $\text{Opt}(H_V, b_V)$, except that the the constraints are now bilinear in the optimization variables, with additional variables K_c, k_c, and W_u. This bilinearity arises because the terms A_i and a_i in (64) and (65) depend on the optimization variables.

We use alternate convex search (ACS) [55] to solve this optimization problem with bilinear constraints. Each convex problem arises by keeping either w_V or W_u fixed, at the value of the solution from the previously solved optimization problem. Therefore, we obtain a nested iterative algorithm, where the inner optimization finds local minima for the bilinear problem $\text{Opt}_{\text{synth}}(H_V, b_V)$, and the outer optimization enables insertion of nodes so that the local optima may have value zero. This algorithm is nearly identical to Algorithm 1, except that we solve $\text{Opt}_{\text{synth}}(H_V, b_V)$ using ACS, instead of $\text{Opt}(H_V, b_V)$ using quadratic programming.

A. Constrained Controller Synthesis

The proposed framework makes it easy to include polytopic constraints on the controller $u = \text{ReLU}(x)$. Since these constraints are affine in u, they are in turn piecewise affine in x. If we use the neurons in the candidate Lyapunov neural network to define the controller, then we simply require that over each cell $X_i \in \mathcal{P}_{H_V,b_V}$, an additional set of affine inequalities in x hold, which is a routine step in our approach.

The advantage of this ability to constrain the control is that we may replace techniques such as Model Predictive Control for a few cases in which they are applied, namely to asymptotically stabilize linear systems while respecting actuator limits.

X. Examples

We present two examples that demonstrate the performance of the search for a ReLU NN Lyapunov function proposed in Section VIII. In addition, we present two examples for synthesis of controllers. We use the `Mosek` optimization package and `Julia v1.5` to implement all computations, using a computer with a 2.6 GHz processor and 16 GB RAM. The values of ϵ_1 and ϵ_2 are set to 1 and 0.001 respectively. We use a tolerance of 10^{-8} when checking if a number is non-zero. Table I summarizes the performance of Algorithm 1 and its extension to controller synthesis on the examples below. Please see supplementary materials for full details.

Example 3 (Recurrent Neural Network)**.** We approximate the right hand side of the dynamics of a simple pendulum using a single-hidden-layer ReLU neural network with 20 neurons. The state-space dynamics of the pendulum are

$$\begin{bmatrix}\dot{x}_1 \\ \dot{x}_2\end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{c}{m}x_2 - gl^2\sin(x_1)\end{bmatrix},$$

Example	Problem	Computation time	Neurons	Regions	Verified property
3	Analysis	1.09 sec	52	75	LS
4	Analysis	82.34 sec	342	278	LAS
5	Synthesis	0.27 sec	12	16	GAS
6	Synthesis	3.33 sec	12	6	LAS

TABLE I: Summary of examples of applying the proposed methods. LS: Local stability; LAS: Local Asymptotic Stability; GAS: Global Asymptotic Stability.

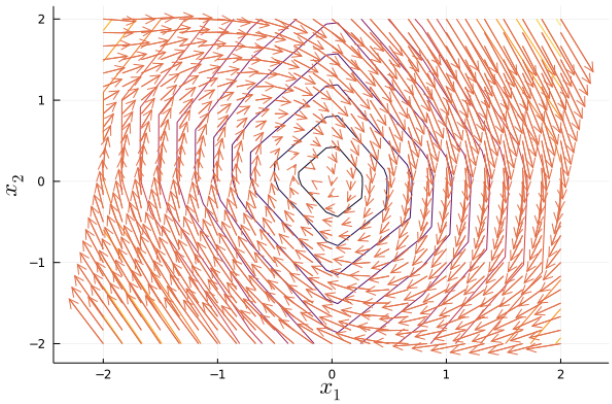

Fig. 3: Depiction of the dynamics (arrows) and the level sets (colored curves) of the Lyapunov function verifying stability of the origin for Example 3.

where $m = 0.1$ kg, $l = 0.5$ m, c = $0.1Ns/rad$, and $g = 9.81$ m/s^2 [9]. We use the ADAM optimization algorithm provided by the `Optim` package in `Julia`, with a learning rate of $1e^{-4}$, on 10000 samples with batch size 3000. We check stability conditions on the region corresponding to $\|x\|_\infty \leq 2$, excluding a neighborhood of the origin corresponding to the constraint $\|x\|_\infty \leq 0.1$. Figure 3 depicts the level sets of the Lyapunov function found by Algorithm 1.

Example 4 (MPC)**.** We consider the control policy derived for a discrete-time linear dynamical system using Model Predictive Control, similar to [10], with dynamics

$$x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} u_t. \tag{66}$$

The MPC problem features the stage-wise quadratic cost $x_t^T x_t + u_t^2$, actuator constraints $|u| < 4$, and state constraint $\|x\|_\infty < 5$. We use the `MPT3` toolbox in `Matlab` to obtain an explicit controller. We verify the stability of this explicit MPC controller when used for the continuous time dynamical system

$$\dot{x} = \begin{bmatrix} 0 & 100 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 75 \\ 50 \end{bmatrix} u, \tag{67}$$

which produces the discrete-time dynamical system in (66) under a discretization time of 0.01 seconds. Figure 4 depicts the level sets of the Lyapunov function found by Algorithm 1.

While we find a Lyapunov function in Example 4, we encountered examples related to explicit MPC where the number of regions and neurons involved become too large to manage in our implementation.

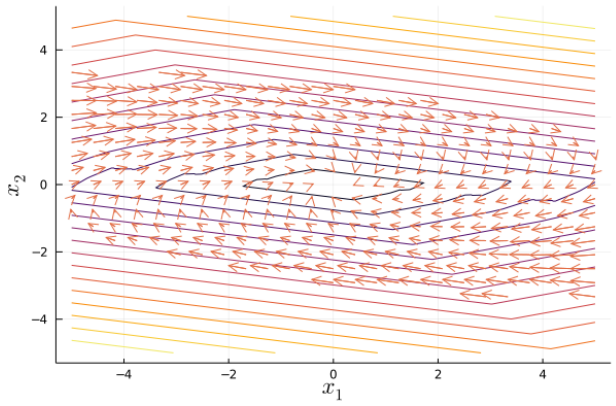

Fig. 4: Depiction of the closed-loop dynamics (arrows) and the level sets (colored curves) of the Lyapunov function verifying asymptotic stability of the origin for Example 4.

Example 5 (Asymptotically Stabilization)**.** We consider a system of the form (1) that is open-loop unstable, where

$$A_c = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad b_c = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

$$W = \begin{bmatrix} -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \\ -1.0 & -0.0 \\ -0.0 & -1.0 \\ -1.0 & -0.0 \\ -0.0 & -1.0 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \\ -1.0 & -0.0 \\ -0.0 & -1.0 \end{bmatrix}, \quad b = \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ -0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.0 \\ 0.0 \\ -0.0 \\ -0.0 \end{bmatrix},$$

and there are no terms of the form $\mathrm{PWA}(x)$. Our algorithm finds a controller with the same neurons as the dynamics, and is verified by a Lyapunov function, shown in Figure 5, with the same neurons as the dynamics.

Example 6 (Constrained Controller Synthesis)**.** We use the method described in Section IX-A to synthesize a state feedback controller $u = \mathrm{ReLU}(x)$ for the double integrator that satisfies actuator limits on a domain. Specifically, $|u(x)| < 4$ for all states x such that $\|x\|_\infty \leq 5$.

The dynamics involves four regions corresponding to the intersection between $\{x \in \mathbb{R}^2 : \|x\|_\infty \leq 5\}$ and the four quadrants. The procedure in Section V-D generates this partition using 8 neurons. The bilinear optimization $\mathrm{Opt}_{\mathrm{synth}}(H_V, b_V)$ is solved twice, in between which four neurons are added. Figure 6 shows the closed-lop dynamics and the level sets of the Lyapunov ReLU neural network function found by the method in Section IX.

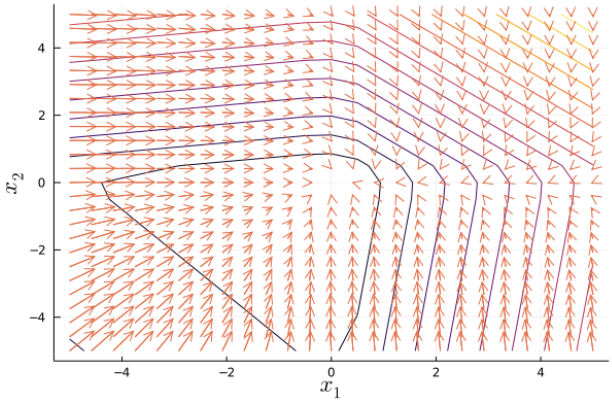

Fig. 5: Depiction of the closed-loop dynamics (arrows) and the level sets (colored curves) of the Lyapunov function verifying asymptotic stability of the origin for Example 5.

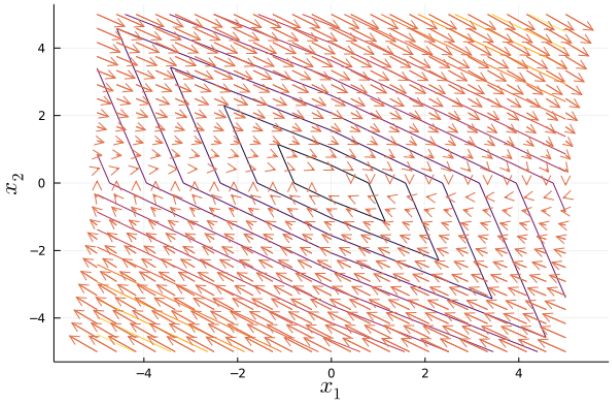

Fig. 6: Depiction of the closed-loop dynamics (arrows) and the level sets (colored curves) of the Lyapunov function verifying asymptotic stability of the origin for Example 6.

XI. Limitations And Future Work

We have presented examples to demonstrate successful applications of our algorithm. Here, we discuss the challenges that exist to making this algorithm widely useful.

Limitations: The proposed solution relies heavily on the piecewise affine nature of the dynamics. The approach of enumerating all cells may be impractical when the numbers of cells or neurons are large. While we have examples where the optimization problem contains more than 200 regions and 300 neurons, these numbers grow dramatically beyond these values in some examples, and the search fails.

A related limitation is the use of sampling to identify all regions in the partition defined by the neural networks. A large number of neurons may create small regions that have a low probability of being sampled, invalidating the conclusions. The use of mixed-integer linear programming to handle ReLU functions [41] may then be a better approach.

The procedure we propose for adding neurons is heuristic, and its effect on future iterations is unknown. The justification to add neurons is based on the intuition that the complexity of the network is related to the partition it is defined on, but this claim is not formal.

The bilinear nature of the optimizations used for controller synthesis implies that this procedure may be far less reliable than the analysis algorithm. However, this bilinearity is largely unavoidable when searching for both controllers and Lyapunov-based certificates at the same time.

Future Work: This work presents a few clear avenues for future work. First, a characterization of dynamical systems and closed-loop properties for which a piecewise affine Lyapunov function must exist will guide application of the proposed methods, similar to work in [37]. Second, the algorithm focuses on continuous dynamical systems, which polynomial methods may also handle. The piecewise nature suggests that these methods will provide greater value for discontinuous dynamics or controllers, for which analysis and synthesis are difficult [36]. Automating the more involved technical issues is likely feasible [40], but possibly challenging [52]–[54]. Third, we will develop a characterization of the how the computational complexity depends on the number of neurons and the dimension of the state.

XII. Conclusion

This paper introduces an computational framework to search for single-layer ReLU NN Lyapunov functions that verify properties of piecewise affine dynamical systems potentially represented as single-layer ReLU NN dynamical systems. Such models are also known as recurrent neural networks, which are often learned from data. The design of our algorithm makes it sound by construction, which is an important consideration in Lyapunov-based guarantees. Furthermore, we avoid the need to train networks using gradient descent. Inspired by refinement-based approaches, we instead increase the number of neurons in the network, thereby increasing the complexity of the neural network in response to the failure to find a Lyapunov function.

While the algorithm is sound, several features are potentially computationally expensive, and the performance of the algorithm is not well characterized. Despite these challenges, this work points to promising automated algorithms for analyzing dynamics models learned from data, and even synthesizing controllers for them. We anticipate that potential for optimized implementations, and the relentless advance of computational power, to be factors in favor of such methods.

References

[1] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/

[2] L. Breiman, "Hinging hyperplanes for regression, classification, and function approximation," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 999–1013, 1993.

[3] J.-N. Lin and R. Unbehauen, "Canonical piecewise-linear approximations," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 39, no. 8, pp. 697–699, 1992.

[4] A. Toriello and J. P. Vielma, "Fitting piecewise linear continuous functions," *European Journal of Operational Research*, vol. 219, no. 1, pp. 86–95, 2012.

[5] H. Ravanbakhsh and S. Sankaranarayanan, "Learning lyapunov (potential) functions from counterexamples and demonstrations," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[6] Y.-C. Chang, N. Roohi, and S. Gao, "Neural lyapunov control," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dé Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[7] M. Korda, "Stability and performance verification of dynamical systems controlled by neural networks: algorithms and complexity," *arXiv preprint arXiv:2102.02273*, 2021.

[8] A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo, "Formal synthesis of lyapunov neural networks," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 773–778, 2021.

[9] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 466–476.

[10] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado, "Learning lyapunov functions for piecewise affine systems with neural network controllers," *arXiv preprint arXiv:2008.06546*, 2020.

[11] H. Khalil, *Nonlinear Systems*, ser. Pearson Education. Prentice Hall, 2002.

[12] S. Boyd, V. Balakrishnan, E. Feron, and L. ElGhaoui, "Control system analysis and synthesis via linear matrix inequalities," in *1993 American Control Conference*, June 1993, pp. 2147–2154.

[13] S. Prajna and A. Papachristodoulou, "Analysis of switched and hybrid systems - beyond piecewise quadratic methods," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 4, June 2003, pp. 2779–2784 vol.4.

[14] S. Prajna, P. A. Parrilo, and A. Rantzer, "Nonlinear control synthesis by convex optimization," *IEEE Transactions on Automatic Control*, vol. 49, no. 2, pp. 310–314, Feb 2004.

[15] M. Johansson and A. Rantzer, "Computation of piecewise quadratic Lyapunov functions for hybrid systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 555–559, Apr 1998.

[16] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[17] O. L. Mangasarian, *Nonlinear Programming*, ser. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994.

[18] I. Polik and T. Terlaky, "A survey of the s-lemma," *SIAM Review*, vol. 49, no. 3, pp. 371–418, 2007.

[19] A. Papachristodoulou and S. Prajna, "On the construction of lyapunov functions using the sum of squares decomposition," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 3, 2002, pp. 3482–3487 vol.3.

[20] S. Prajna, A. Papachristodoulou, and P. A. Parrilo, "Introducing sostools: a general purpose sum of squares programming solver," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 1, Dec 2002, pp. 741–746 vol.1.

[21] J. Anderson and A. Papachristodoulou, "Advances in computational lyapunov analysis using sum-of-squares programming," *Discrete & Continuous Dynamical Systems - B*, vol. 20, p. 2361, 2015.

[22] R. Sepulchre and D. Aeyels, "Homogeneous lyapunov functions and necessary conditions for stabilization," *Mathematics of Control, Signals and Systems*, vol. 9, no. 1, pp. 34–58, 1996.

[23] D. Liberzon, J. P. Hespanha, and A. Morse, "Stability of switched systems: a lie-algebraic condition," *Systems & Control Letters*, vol. 37, no. 3, pp. 117 – 122, 1999.

[24] R. Goebel and R. Sanfelice, *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, 2012.

[25] H. Lin and P. J. Antsaklis, "Stability and stabilizability of switched linear systems: A survey of recent results," *IEEE Transactions on Automatic Control*, vol. 54, no. 2, pp. 308–322, Feb 2009.

[26] M. Johansson, "Piecewise linear control systems," Ph.D. dissertation, Lund University, 1999.

[27] J. Oehlerking, H. Burchardt, and O. Theel, "Fully automated stability verification for piecewise affine systems," in *Hybrid Systems: Computation and Control*, A. Bemporad, A. Bicchi, and G. Buttazzo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 741–745.

[28] R. Iervolino, F. Vasca, and L. Iannelli, "Cone-copositive piecewise quadratic Lyapunov functions for conewise linear systems," *IEEE Tran. on Automatic Control*, vol. 60, no. 11, pp. 3077–3082, 2015.

[29] M. S. Branicky, "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 475–482, April 1998.

[30] M. A. Wicks, P. Peleties, and R. A. DeCarlo, "Construction of piecewise Lyapunov functions for stabilizing switched systems," in *Proceedings of 1994 33rd IEEE Conference on Decision and Control*, vol. 4, Dec 1994, pp. 3492–3497 vol.4.

[31] S. Pettersson and B. Lennartson, "Stabilization of hybrid systems using a min-projection strategy," in *Proceedings of the 2001 American Control Conference*, vol. 1, 2001, pp. 223–228 vol.1.

[32] T. Hu, L. Ma, and Z. Lin, "Stabilization of switched systems via composite quadratic functions," *IEEE Transactions on Automatic Control*, vol. 53, no. 11, pp. 2571–2585, Dec 2008.

[33] P. Bolzern and W. Spinelli, "Quadratic stabilization of a switched affine system about a nonequilibrium point," in *Proceedings of the 2004 American Control Conference*, vol. 5, June 2004, pp. 3890–3895.

[34] T. Soga and N. Otsuka, "Quadratic stabilizability for polytopic uncertain continuous-time switched linear systems by output feedback," in *Proceedings of the 2010 American Control Conference*, June 2010, pp. 3920–3925.

[35] L. Hetel and E. Bernuau, "Local stabilization of switched affine systems," *IEEE Transactions on Automatic Control*, vol. 60, no. 4, pp. 1158–1163, April 2015.

[36] E. Treadway and R. B. Gillespie, "Vector field control methods for discretely variable passive robotic devices," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 375–389, 2021.

[37] F. Blanchini, "Nonquadratic Lyapunov functions for robust control," *Automatica*, vol. 31, no. 3, pp. 451 – 461, 1995.

[38] H. A. Poonawala, N. Lauffer, and U. Topcu, "Training classifiers for feedback control with safety in mind," *Automatica*, vol. 128, p. 109509, 2021.

[39] H. A. Poonawala, N. Lauffer, and U. Topcu, "Training classifiers for feedback control," in *2019 American Control Conference (ACC)*, July 2019, pp. 4961–4967.

[40] H. A. Poonawala, "Stability analysis of conewise affine dynamical systems using conewise linear lyapunov functions," *IEEE Control Systems Letters*, pp. 1–1, 2020.

[41] H. Dai, B. Landry, M. Pavone, and R. Tedrake, "Counter-example guided synthesis of neural network lyapunov functions for piecewise linear systems," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 1274–1281.

[42] H. A. Poonawala, "Stability Analysis Via Refinement Of Piece-wise Linear Lyapunov Functions," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 1442–1447.

[43] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado, "Learning lyapunov functions for hybrid systems," *arXiv preprint arXiv:2012.12015*, 2020.

[44] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[45] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[46] R. Alur, S. Kannan, and S. La Torre, "Polyhedral flows in hybrid automata," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 1999, pp. 5–18.

[47] P. Prabhakar and M. G. Soto, "Counterexample guided abstraction refinement for stability analysis," in *International Conference on Computer Aided Verification*, 2016, pp. 495–512.

[48] D. Ahmed, A. Peruffo, and A. Abate, "Automated and sound synthesis of lyapunov functions with smt solvers," in *Tools and Algorithms for the Construction and Analysis of Systems*, A. Biere and D. Parker, Eds. Cham: Springer International Publishing, 2020, pp. 97–114.

[49] F. H. Clarke, Y. S. Ledyaev, R. J. Stern, and P. R. Wolenski, *Nonsmooth analysis and control theory*. Springer Science & Business Media, 2008, vol. 178.

[50] A. F. Filippov and F. M. Arscott, *Differential equations with discontinuous righthand sides*, ser. Mathematics and its Applications, 1988.

[51] L. Q. Thuan and M. K. Camlibel, "Continuous piecewise affine dynamical systems do not exhibit zeno behavior," *IEEE Transactions on Automatic Control*, vol. 56, no. 8, pp. 1932–1936, 2011.

[52] J. Cortes, "Discontinuous dynamical systems," *IEEE Control Systems Magazine*, vol. 28, no. 3, pp. 36–73, June 2008.

[53] M. Della Rossa, A. Tanwani, and L. Zaccarian, "Smooth approximation of patchy Lyapunov functions for switched systems," *IFAC-PapersOnLine*, vol. 52, no. 16, pp. 364–369, 2019.

[54] R. Baier, L. Grüne, and S. F. Hafstein, "Linear programming based Lyapunov function computation for differential inclusions," *Discrete & Continuous Dynamical Systems - B*, vol. 17, p. 33, 2012.

[55] J. Gorski, F. Pfeuffer, and K. Klamroth, "Biconvex sets and optimization with biconvex functions: a survey and extensions," *Math Meth Oper Res*, pp. 373–407, 2007.

Hasan A. Poonawala Hasan A. Poonawala is an Assistant Professor in the Department of Mechanical Engineering at the University of Kentucky. He holds a Master's degree in Mechanical Engineering from the University of Michigan (2009), and a Ph.D. in Electrical Engineering from the University of Texas at Dallas (2014). Dr. Poonawala worked as a postdoctoral researcher at the University of Texas at Austin, on combining AI and control theory. His research expertise spans mechatronics, control of multi-agent systems, vision-based motion control, and classifier-in-the-loop systems. His current research focuses on controlling robotic systems using high-dimensional sensor data, machine learning, and control theory.

PLACE PHOTO HERE

Pouya Samanipour Pouya Samanipour is a PhD student in the Department of Mechanical Engineering at the University of Kentucky. He received a Bachelor's degree in Electrical Engineering from Imam Khomeini International University, Qazvin, Iran in 2011, and a Master's degree in Electrical Engineering from Iran University of Science and Technology, Tehran, Iran in 2015. Currently, his research interests include control theory and robotics.