Reachable Polyhedral Marching (RPM): An Exact Analysis Tool for Deep-Learned Control Systems

Joseph A. Vincent[1] and Mac Schwager[1]

***Abstract*— We present a tool for computing exact forward and backward reachable sets of deep neural networks with rectified linear unit (ReLU) activation. We then develop algorithms using this tool to compute invariant sets and regions of attraction (ROAs) for control systems with neural networks in the feedback loop. Our algorithm is unique in that it builds the reachable sets by incrementally enumerating polyhedral regions in the input space, rather than iterating layer-by-layer through the network as in other methods. When performing safety verification, if an unsafe region is found, our algorithm can return this result without completing the full reachability computation, thus giving an anytime property that accelerates safety verification. Furthermore, we introduce a method to accelerate the computation of ROAs in the case that deep learned components are homeomorphisms, which we find is surprisingly common in practice. We demonstrate our tool in several test cases. We compute a ROA for a learned van der Pol oscillator model. We find a control invariant set for a learned torque-controlled pendulum model. We also verify specific safety properties for multiple deep networks related to the ACAS Xu aircraft collision advisory system. Finally, we apply our algorithm to find ROAs for an image-based aircraft runway taxi problem. Algorithm source code: `https://github.com/StanfordMSL/Neural-Network-Reach`.**

I. Introduction

In this paper we present the Reachable Polyhedral Marching (RPM) algorithm for computing forward and backward reachable sets of deep neural networks with rectified linear unit (ReLU) activation. Our algorithm provides a building block for proving safety properties for autonomous systems with learned perception, dynamics, or control components in the loop. Specifically, given a set in the input space, RPM computes the set of all corresponding outputs (the forward reachable set, or image, of the input set). Similarly, given a set of outputs, RPM computes the set of all corresponding inputs under the ReLU network (the backward reachable set, or preimage, of the output set). We use these capabilities to compute finite-time reachable sets, as well as infinite-time invariant sets and regions of attraction (ROAs). It is well known that for any ReLU network there exists an equivalent continuous piecewise-affine (PWA) function, that is, the input space for a ReLU network can be tessellated into polyhedra, and over each polyhedron the neural network evaluates to an affine function. RPM explicitly finds this equivalent PWA representation for a given ReLU network. Figure 1 illustrates how RPM works. The algorithm iteratively enumerates the polyhedra and affine functions defining the PWA function by solving a series of Linear Programs (LPs), and following analytical edge flipping rules to determine neighboring polyhedra. The algorithm starts with a random polyhedron, then solves for its neighboring polyhedra, then neighbors of neighbors, etc., until the whole input space is tessellated. In this way, our method is geometrically similar to fast marching methods in optimal control [1], path planning [2], [3], and graphics [4], [5]. We then use computational methods for PWA reachability to compute forward and backward reachable sets for each polyhedron and associated affine function. Finally, we also propose an accelerated

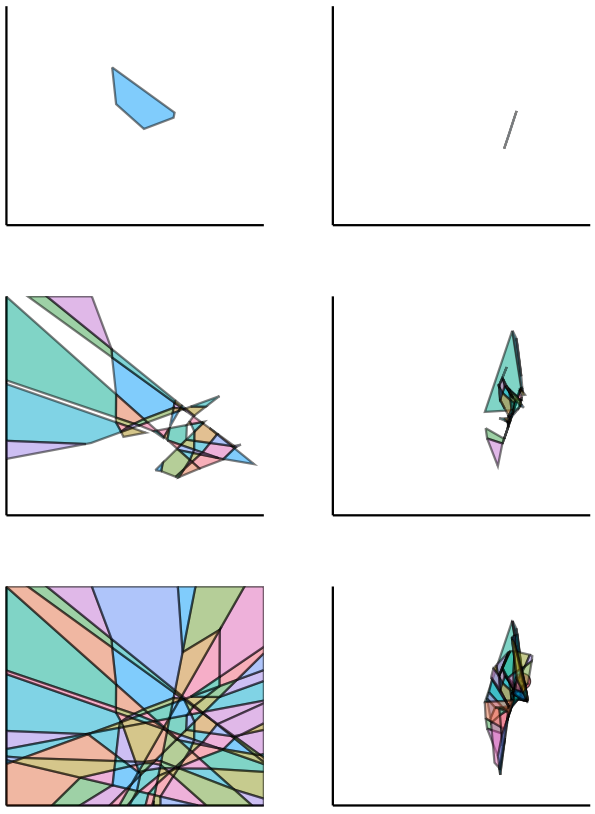


Fig. 1: This figure shows how RPM incrementally explores polyhedra corresponding to affine regions of a ReLU neural network (left column) and the corresponding forward reachable set of these regions under the neural network map (right column). This example uses a randomly initialized network with 2 inputs, 2 outputs, and 2 hidden layers of 10 neurons each; resulting in 166 affine regions.

[1]Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, USA, {josephav, schwager}@stanford.edu

The first author was supported in part by a Dwight D. Eisenhower Transportation Fellowship. The NASA University Leadership Initiative (grant #80NSSC20M0163) provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not any NASA entity. We are grateful for this support.

backward reachability procedure in the case that the deep network is a homeomorphism (a continuous bijection with continuous inverse). We find, surprisingly, that a variety of organically trained ReLU networks are homeomorphisms, even through they are not constructed or trained with this property in mind (e.g. the closed-loop TaxiNet network trained for autonomous aircraft runway taxiing), therefore backward reachability and ROA computations with RPM can be significantly accelerated for these systems.

All existing algorithms that compute exact reachable sets of neural networks iterate through the network layer-by-layer [6], [7], [8]. The layer-by-layer approaches obtain the entire reachable set at once at the end of the computation, rather than revealing the reachable set piece by piece throughout the computation. Consequently, if the computation must end early due to memory constraints or a computer fault, no usable result is obtained. In contrast, RPM builds the reachable set one polyhedron at a time, leading to a partial, but still potentially useful result if computation is halted before the algorithm runs to completion. Incremental computation is better suited to finding counterexamples to safety properties and terminating early, as well as providing a more memory efficient approach where a region of the reachable set already computed can be saved externally to free up memory for remaining computation. Furthermore, for learned dynamical systems, RPM can be used to certify local asymptotic stability, compute nonconvex ROAs, compute intricate control invariant sets, and verify input-output safety properties, as we demonstrate in this paper. These factors are the core motivation for RPM.

In numerical examples, we compute ROAs for a learned van der Pol oscillator, and show this computation enjoys a 16x speedup because the learned dynamics are homeomorphic. We then pair RPM with the MPT3 [9] Matlab toolbox to find an a control invariant set for a learned torque-controlled pendulum. We also show faster identification of unsafe inputs compared to a state of the art reachability method [8] in a safety verification problem involving ACAS Xu, a neural network policy for aircraft collision avoidance. Finally, we apply our algorithm to find ROAs for an image-based airplane runway taxiing system, TaxiNet [10]. The closed-loop system is PWA with $\sim 100,000$ regions, two orders of magnitude larger than PWA systems for which other approaches have been demonstrated.

The contributions of this paper are

- An incremental method for computing the explicit PWA representation of a ReLU network.
- A theorem to analytically determine an input space polyhedron from its neighbor by flipping neuron activations in the ReLU network.
- An accelerated PWA backward reachability algorithm for homeomorphic PWA functions.
- Application of PWA analysis tools to compute regions of attraction for equilibria, and infinite time forward-invariant and control-invariant sets for ReLU networks.
- Examples demonstrating the computation of exact ROAs and control invariant sets, safety property verification, and ROA computation for a ReLU network with two orders of magnitude more affine regions than other examples in the literature.

This paper builds upon an earlier conference version [11], which first introduced the RPM algorithm to convert a ReLU network into a PWA representation. This paper improves upon [11] by (i) adding a formal proof of correctness for the main RPM algorithm, (ii) extending the RPM framework to compute invariant sets and ROAs, (iii) introducing an accelerated method for computing ROAs for homeomorphic neural networks, and (iv) demonstrating the computation of invariant sets and ROAs in three new examples.

The paper is organized as follows. We give related work in Section II and give background and state the problem in Section III. Section IV presents our main algorithm, RPM, and explains its derivation. In Section V we use RPM to perform forward and backward reachability computations for ReLU networks over multiple time steps. Next, in Section VI we summarize existing results on computing robust control invariant sets for PWA dynamical systems. In Section VII we use RPM to compute ROAs using these PWA invariant set tools. Finally, Section VIII presents numerical results using RPM in the aforementioned examples, and we offer conclusions in Section IX. A proof for the main theorem on the correctness of RPM is presented in the Appendix.

II. Related Work

Though the analysis of neural networks is a young field, a broad literature has emerged to address varied questions related to interpretability, trustworthiness, and safety verification. Much work has been dedicated to characterizing the expressive potential of ReLU networks by studying how the number of affine regions scales with network depth and width [12], [13], [14]. Other research includes encoding piecewise-affine (PWA) functions as ReLU networks [15], [16], learning deep signed distance functions and extracting level sets [5], and learning neural network dynamics or controllers that satisfy stability conditions [17], [18], which may be more broadly grouped with correct-by-construction training approaches [19], [20].

Spurred on by pioneering methods such as Reluplex [21], the field of neural network verification has emerged to address the problem of analyzing properties of neural networks over continuous input sets. A survey of the neural network verification literature is given by [22]. Reachability approaches are a subset of this literature and are especially useful for analysis of learned dynamical systems. Reachability methods can be categorized into overapproximate and exact methods. Overapproximate methods often compute neuron-wise bounds either from interval arithmetic or symbolically [23], [24], [25], [26], [27]. Optimization based approaches such as mixed-integer linear programming are also used to solve for bounds on a reachable output set in [28], [29], [30], [31]. Other approaches include modeling the network as a hybrid system [32], abstracting the domain [33], and performing layer-by-layer operations on zonotopes [34]. Further, [35], [36] demonstrate how overapproximate

approaches can be used to perform closed-loop reachability of a dynamical system given a neural network controller.

Exact reachability methods have also been proposed, although to a lesser degree [6], [7], [8]. These methods generally perform the set operations of affine transformation, intersection/division, and projection layer-by-layer through a ReLU network. Similar layer-by-layer approaches have also been proposed to solve for the explicit PWA representation of a ReLU network [37], [38], [39].

Our RPM algorithm inherits all the advantages of exact methods, but is unique in that it does not iterate layer-by-layer, but rather the reachable set incrementally. This makes our high-level procedure more similar to explicit model predictive control methods [40] than to other exact reachability methods. In contrast to layer-by-layer methods, if our algorithm encounters a region with an unsafe input, it can return that result immediately without computing the entire reachable set. Finally, all intermediate polyhedra and affine map matrices of layer-by-layer methods must be stored in memory until the algorithm terminates, whereas with RPM once a polyhedron-affine map pair is computed it can be sent to external memory and only a binary vector (the neuron activations) needs to be stored to continue the algorithm. This is especially useful because, even for networks with only a few hundred neurons, the number of affine regions in a specified input set can be on the order of 10^5–10^6 and no method has been shown to accurately estimate the number of regions without explicit enumeration.

In this paper we demonstrate how RPM can be used not only for finite-time verification and reachability, but also for computing infinite-time invariant sets and ROAs of learned dynamical systems. Computing these sets tends to be much harder than computing finite-time reachable sets, but the associated guarantees hold for all time, unlike with finite-time reachable sets. Few methods exist for computing invariant sets or ROAs for dynamical systems represented as neural networks, and those that do rely on an optimization-based search of a Lyapunov-like function. In [41], the authors learn a quadratic Lyapunov function using semidefinite programming which results in an ellipsoidal ROA, a drawback of which is that ellipsoids are not very expressive sets. In contrast, nonconvex ROAs can be computed by [42], [43], [44]. These methods learn Lyapunov functions for a given autonomous dynamical system and verify the Lyapunov conditions either using many samples and Lipschitz continuity or mixed integer programming. In [45], an optimization and sampling-based approach for synthesizing control invariant sets is given, but as with searching for Lyapunov functions, it may fail in finding a control invariant set when one exists. All of these methods inherit the drawbacks of Lyapunov-style approaches wherein a valid Lyapunov-like function may not be found when one exists, and certifying the Lyapunov conditions is challenging.

In the case of explicitly defined PWA dynamics, there are specially designed algorithms for computing ROAs based on convex optimization approaches for computing Lyapunov functions [46], [47], [48], or reachability approaches [49]. A drawback to the Lyapunov approaches for PWA dynamics is they can be too conservative and many require the user to provide an invariant domain, which itself can be very challenging to find. Futhermore, in Section 7.2 of [46], motivating examples are given where a variety of Lyapunov approaches fail to find valid Lyapunov functions for very small PWA systems which are known to be stable.

In contrast to the Lyapunov approaches, the reachability approach requires the user to supply an initial 'seed' set of states that is a ROA and uses backward reachability to grow the size of the ROA. Finding a seed ROA can be difficult for general nonlinear systems, but for PWA systems can be easily found by finding a ROA in one of the affine regions with a stable equilibrium. The benefit of a reachability-based approach for computing ROAs is that it is a straightforward and reliable procedure for PWA systems. The drawback to reachability-based approaches is the poor scaling of the backward reachability computation at each time-step. Our proposed method for finding ROAs of neural networks follows the backward reachability approach from the literature. In addition, we show that our backward reachability algorithm enjoys considerable speedup when the ReLU network is homeomorphic, which can be checked via a simple procedure.

III. Background and Problem Statement

We begin by formalizing a model of a ReLU network, and defining various concepts related to polyhedra and PWA functions. We then state the main problem of computing forward and backward reachable sets for ReLU networks.

A. ReLU Networks

An L-layer feedforward neural network implements a function $\mathbf{y} = F_{\mathbf{\Theta}}(\mathbf{x})$, with the map $F_{\mathbf{\Theta}} : \mathbb{R}^n \mapsto \mathbb{R}^m$ defined recursively by the iteration

$$\mathbf{z}_i = \sigma_i(\mathbf{W}_i \mathbf{z}_{i-1} + \mathbf{b}_i), \quad i = 1, \dots, L \tag{1}$$

where $\mathbf{z}_0 = \mathbf{x}$ is the input, $\mathbf{y} = \mathbf{z}_L$ is the output, and $\mathbf{z}_i$ is the hidden layer output of the network at layer i. The function $\sigma_i(\cdot)$ is the activation function at layer i, and $\mathbf{W}_i \in \mathbb{R}^{l_i \times l_{i-1}}$ and $\mathbf{b}_i \in \mathbb{R}^{l_i}$ are the weights and biases, respectively, where l_i denotes the number of neurons in layer i. We assume $\sigma_L(\cdot)$ is an identity map and all hidden layer activations are the rectified linear unit (ReLU),

$$\sigma_i(\mathbf{z}_i^{\text{pre}}) = [\max(0, z_{i1}^{\text{pre}}) \cdots \max(0, z_{il_i}^{\text{pre}})]^\top, \tag{2}$$

where $\mathbf{z}_i^{\text{pre}} = [z_{i1}^{\text{pre}} \cdots z_{il_i}^{\text{pre}}]^\top = \mathbf{W}_i \mathbf{z}_{i-1} + \mathbf{b}_i$ is the pre-activation hidden state at layer i. We can rewrite the iteration in (1) in a homogeneous form

$$\begin{bmatrix} \mathbf{z}_i \\ 1 \end{bmatrix} = \sigma_i\Big(\mathbf{\Theta}_i \begin{bmatrix} \mathbf{z}_{i-1} \\ 1 \end{bmatrix}\Big), \tag{3}$$

where

$$\mathbf{\Theta}_i = \begin{bmatrix} \mathbf{W}_i & \mathbf{b}_i \\ \mathbf{0}^\top & 1 \end{bmatrix} \quad \text{for} \quad i = 1, \dots, L-1, \tag{4a}$$

$$\mathbf{\Theta}_L = \begin{bmatrix} \mathbf{W}_L & \mathbf{b}_L \end{bmatrix}. \tag{4b}$$

We define $\mathbf{\Theta} = (\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_L)$ as the tuple containing all the parameters that define the function $F_{\mathbf{\Theta}}(\cdot)$.

Given an input $\mathbf{x}$ to a ReLU network, every hidden neuron has an associated binary *neuron activation* of zero or one, corresponding to whether the preactivation value is nonpositive or positive, respectively.[1] Specifically, the activation for neuron j in layer i is given by

$$\lambda_{(ij)}(\mathbf{x}) = \begin{cases} 1 & \text{for} \quad z_{ij}^{\text{pre}}(\mathbf{x}) > 0 \\ 0 & \text{for} \quad z_{ij}^{\text{pre}}(\mathbf{x}) \leq 0. \end{cases} \tag{5}$$

We define the activation pattern at layer i as the binary vector $\lambda_{(i)}(\mathbf{x}) = [\lambda_{(i1)}(\mathbf{x}) \cdots \lambda_{(il_i)}(\mathbf{x})]^\top$ and the activation pattern of the whole network as the tuple $\lambda(\mathbf{x}) = (\lambda_{(1)}(\mathbf{x}), \ldots, \lambda_{(L-1)}(\mathbf{x}))$. For a network with $N = \sum_{i=1}^{L-1} l_i$ neurons there are 2^N possible combinations of neuron activations, however not all are realizable by the network due to inter-dependencies between neurons from layer to layer, as we will see later in the paper. Empirically, the number of activation patterns achievable by the network is better approximated by N^n for input space with dimension n [50].

We can write equivalent expressions for the neural network function in terms of the activation pattern. Define the diagonal activation matrix for layer i as $\mathbf{\Lambda}_{(i)}(\mathbf{x}) = \text{diag}([\lambda_{(i)}(\mathbf{x})^\top 1]^\top)$, where a 1 is appended at the end[2] to match the dimensions of the homogeneous form in (3). Using the activation matrix, we can write the iteration defining the ReLU network from (3) as

$$\begin{bmatrix} \mathbf{z}_i \\ 1 \end{bmatrix} = \mathbf{\Lambda}_{(i)}(\mathbf{x}) \mathbf{\Theta}_i \begin{bmatrix} \mathbf{z}_{i-1} \\ 1 \end{bmatrix}. \tag{6}$$

Finally, the map from input $\mathbf{x}$ to the hidden layer preactivation z_{ij}^{pre} can be written explicitly from the iteration (6) as

$$z_{ij}^{\text{pre}}(\mathbf{x}) = \boldsymbol{\theta}_{ij} \Big(\prod_{l=1}^{i-1} \mathbf{\Lambda}_{(l)}(\mathbf{x}) \mathbf{\Theta}_l \Big) \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \tag{7}$$

where $\boldsymbol{\theta}_{ij}$ is the j^{th} row of the matrix $\mathbf{\Theta}_i$. Similarly, the whole neural network function $\mathbf{y} = F_{\mathbf{\Theta}}(\mathbf{x})$ can be written explicitly as a map from $\mathbf{x}$ to $\mathbf{y}$ as

$$\mathbf{y} = \mathbf{\Theta}_L \Big(\prod_{i=1}^{L-1} \mathbf{\Lambda}_{(i)}(\mathbf{x}) \mathbf{\Theta}_i \Big) \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}. \tag{8}$$

It is well known that in (8), the output y is a continuous and PWA function of the input x [12], [13], [14]. Next, we mathematically characterize PWA functions and give useful definitions.

B. Polyhedra and PWA Functions

Definition 1 ((Convex) Polyhedron): A convex polyhedron is a closed convex set $P \subset \mathbb{R}^n$ defined by a finite collection of M halfspace constraints,

$$P = \{\mathbf{x} \mid \mathbf{a}_i^\top \mathbf{x} \leq b_i, \quad i = 1, \ldots, M\},$$

where $\mathbf{a}_i, \mathbf{x} \in \mathbb{R}^n$, $b_i \in \mathbb{R}$, and each $(\mathbf{a}_i, b_i)$ pair defines a halfspace constraint. Defining matrix $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_N]^\top$ and vector $\mathbf{b} = [b_1 \cdots b_N]^\top$, we can equivalently write

$$P = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}. \tag{9}$$

We henceforth refer to convex polyhedra as just polyhedra.

The above representation of a polyhedron is known as the halfspace representation, or H-representation. One can equivalently represent a polyhedron as the convex hull of a set of points and rays, known as the vertex representation, or V-representation, which we do not use in this paper. Note:

- A polyhedron can be bounded or unbounded.
- A polyhedron can occupy a dimension less than the ambient dimension (if $(\mathbf{a}_i, b_i) = \alpha_{ij}(-\mathbf{a}_j, -b_j)$ for some pairs (i, j) and some positive scalars α_{ij}).
- A polyhedron can be empty ($\nexists \mathbf{x}$ such that $\mathbf{A}\mathbf{x} \leq \mathbf{b}$).
- Without loss of generality, the halfspace constraints for a polyhedron can be normalized so that $\|\mathbf{a}_i\| \in \{0, 1\}$ (by dividing the unnormalized parameters $\mathbf{a}_i$ and b_i by $\|\mathbf{a}_i\|$), where $\|\cdot\|$ is the ℓ_2 norm. $\|\mathbf{a}\| = 0$ represents a degenerate constraint that is trivially satisfied if $b_i \geq 0$.
- An H-representation of a polyhedron may have an arbitrary number of redundant constraints, which are constraints that are implied by other constraints. Removing redundant constraints does not change the polyhedron.

Definition 2 (Polyhedral Tessellation): A polyhedral tessellation $\mathcal{P} = \{P_1, \ldots, P_N\}$ is a finite set of polyhedra that tessellate a set $\mathcal{X} \subset \mathbb{R}^n$. That is, $\cup_{i=1}^N P_i = \mathcal{X}$ and $|P_i \cap P_j|_n = 0$, for all $i \neq j$, where $|\cdot|$ denotes the n-dimensional Euclidean volume of a set (the integral over the set with respect to the Lebesgue measure of dimension n).

Intuitively, the polyhedra in a tessellation together cover the set $\mathcal{X}$, and can only intersect with each other at shared faces, edges, vertices, etc.

Definition 3 (Neighboring Polyhedra): Two polyhedra, P_i and P_j, are neighbors if their intersection has nonzero Euclidean volume in $n-1$ dimensions, that is, $|P_i \cap P_j|_{n-1} > 0$.

Intuitively, neighboring polyhedra are adjacent to one another in the tessellation, and share a common face. A polyhedral tessellation naturally induces a graph in which each node is a polyhedron and edges exist between neighboring polyhedra. An example of a polyhedral tessellation in $\mathbb{R}^2$ (tessellating the input space of a ReLU network) is shown in Figure 1.

Definition 4 (Piecewise Affine (PWA) Function): A PWA function is a function $F_{\text{PWA}}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$ that is affine over each polyhdron in a polyhedral tessellation $\mathcal{P}$ of $\mathbb{R}^n$. Specifically, $F_{\text{PWA}}(\mathbf{x})$ is defined as

$$F_{\text{PWA}}(\mathbf{x}) = \mathbf{C}_k \mathbf{x} + \mathbf{d}_k \quad \forall \mathbf{x} \in P_k \quad \forall P_k \in \mathcal{P}, \tag{10}$$

where $\mathbf{C}_k \in \mathbb{R}^{m \times n}$, $\mathbf{d}_k \in \mathbb{R}^m$.

Note that this specification of a PWA function requires a collection of tuples $\{(\mathbf{A}_1, \mathbf{b}_1, \mathbf{C}_1, \mathbf{d}_1), \ldots, (\mathbf{A}_N, \mathbf{b}_N, \mathbf{C}_N, \mathbf{d}_N)\}$, where $\mathbf{A}_k, \mathbf{b}_k$ define the polyhedron, and $\mathbf{C}_k, \mathrm{d}_k$ define the affine

[1] This choice is arbitrary. Some works use the opposite convention.

[2] Note that the homogeneous form adds a dummy neuron at each layer that is always active, since the last element of the hidden state is $\mathbf{z}_{i(l_i+1)} = 1$ in homogeneous form, so $\mathbf{z}_{i(l_i+1)} > 0$ is always true.

function over that polyhedron. Here we consider only *continuous* PWA functions, in which case the parameters are not independent, but must be such that each affine function continuously joins with the affine functions in each neighboring polyhedron. We refer to the PWA representation in (10) as the *explicit* PWA representation, as each affine map and polyhedron is written explicitly without specifying the relationship between them. There also exist other representations, such as the lattice representation [51], [52] and the, so called, canonical representation [53], [54], in which the polyhedra are expressed implicitly as functions of the affine maps, or vice versa. Indeed, a ReLU network is also an example of a PWA representation in which the polyhedra and affine maps are implicitly contained in the neural network weights.

Our goal in this paper is to transcribe the ReLU network function $\mathbf{y} = F_{\mathbf{\Theta}}(\mathbf{x})$ into its equivalent explicit PWA representation $\mathbf{y} = F_{\text{PWA}}(\mathbf{x})$, and then apply tools for PWA reachability analysis. We next define basic reachability concepts and state the overall problems that we seek to solve.

Definition 5 (Forward Reachable Set): The forward reachable set of a set of inputs $\mathcal{X}$ under the map $F(\mathbf{x})$ is the image of $\mathcal{X}$ under F, $\mathcal{Y} = \{F(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}$.

Definition 6 (Backward Reachable Set): The backward reachable set of a set of outputs $\mathcal{Y}$ under the map $F(\mathbf{x})$ is the preimage of $\mathcal{Y}$ under F, $\mathcal{X} = \{\mathbf{x} \mid F(\mathbf{x}) \in \mathcal{Y}\}$.

In this paper we seek to solve the following problems.

Problem 1: Given a ReLU network $\mathbf{y} = F_{\mathbf{\Theta}}(\mathbf{x})$ and a polyhedral input set $\mathcal{X} \subset \mathbb{R}^n$, compute the forward reachable set $\mathcal{Y}$. Similarly, given a polyhedral output set $\mathcal{Y} \subset \mathbb{R}^m$ compute the backward reachable set $\mathcal{X}$.

Problem 2: Given a ReLU network which implements a discrete-time dynamical system $\mathbf{x}^+ = F_{\mathbf{\Theta}}(\mathbf{x})$ and domain $\mathcal{X}$, identify the existence of stable fixed-point equilibria and compute associated ROAs for each.

Problem 3: Given a ReLU network which implements a discrete-time dynamical system $\mathbf{x}^+ = F_{\mathbf{\Theta}}(\mathbf{x})$, and given a domain of states $\mathcal{X}$, compute a robust control invariant set.

Problem 4: Given a ReLU network which implements a function $\mathbf{y} = F_{\mathbf{\Theta}}(\mathbf{x})$, determine whether the network is a homeomorphism (bijective with continuous inverse).

In the next section we introduce our main algorithm for transcribing the ReLU network function $\mathbf{y} = F_{\mathbf{\Theta}}(\mathbf{x})$ into its equivalent explicit PWA representation $\mathbf{y} = F_{\text{PWA}}(\mathbf{x})$. We then address the solution of the above problems in the subsequent sections.

IV. From ReLU Network to PWA Function

First, we seek to construct the explicit PWA representation of a ReLU network. Our method enumerates each polyhedral region and its associated affine map directly from the ReLU network. In Sec. IV-A we show how polyhedra and affine maps are computed from the activation pattern of a ReLU network. In Sec. IV-B we show how polyhedral representations are reduced to a minimal form, which is used in Sec. IV-C to determine neighboring polyhedra given a current polyhedron. This leads to a recursive procedure, which we call Reachable Polyhedral Marching (RPM), in which we explore an expanding front of polyhedra, ultimately giving the explicit PWA representation, as explained in Sec. IV-D.

A. Determining Polyhedral Regions from Activation Patterns

We show here that the network activation pattern $\lambda(\mathbf{x})$ from (5) has a one-to-one correspondence with the regions of the polyhedral tessellation underlying the ReLU network. We show how to explicitly extract the half-space constraints defining the polyhedron from the activation pattern.

Consider the expression for the preactivation value z_{ij}^{pre} in (7). We can re-write this equation as

$$z_{ij}^{\text{pre}} = [\bar{\mathbf{a}}_{ij}^\top \quad -\bar{b}_{ij}] \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \tag{11}$$

where

$$[\bar{\mathbf{a}}_{ij}^\top \quad -\bar{b}_{ij}] = \boldsymbol{\theta}_{ij} \Big(\prod_{l=1}^{i-1} \mathbf{\Lambda}_{(l)}(\mathbf{x}) \mathbf{\Theta}_l \Big). \tag{12}$$

The activation $\lambda_{(ij)}$ is decided by the test $z_{ij}^{\text{pre}} > 0$, which we can write as $\bar{\mathbf{a}}_{ij}^\top \mathbf{x} > \bar{b}_{ij}$, defining a halfspace constraint in the input space. Specifically, we have the following cases,

$$\begin{cases} \bar{\mathbf{a}}_{ij}^\top \mathbf{x} > \bar{b}_{ij} & \text{if} \quad \lambda_{(ij)} = 1 \\ \bar{\mathbf{a}}_{ij}^\top \mathbf{x} \leq \bar{b}_{ij} & \text{if} \quad \lambda_{(ij)} = 0. \end{cases} \tag{13}$$

This defines a halfspace constraint, which may or may not be open (due to the strict '$>$' inequality). However, on the boundary $\bar{\mathbf{a}}_{ij}^\top \mathbf{x} = \bar{b}_{ij}$, we have $z_{ij}^{\text{pre}} = 0$ from (11), leading to the post activation hidden state $z_{ij} = \lambda_{(ij)} z_{ij}^{\text{pre}} = 0$. We see that the value of the activation $\lambda_{(ij)}$ is irrelevant on the boundary, as the post activation state evaluates to zero regardless. We can therefore replace the '$>$' with '$\geq$' in (13) without loss of generality.

Finally, to obtain the standard normalized form for a halfspace constraint ($\mathbf{a}_{ij}^\top \mathbf{x} \leq b_{ij}$), we define

$$\mathbf{a}_{ij} = (1 - 2\lambda_{(ij)}) \frac{\bar{\mathbf{a}}_{ij}}{\|\bar{\mathbf{a}}_{ij}\|} \tag{14a}$$

$$b_{ij} = (1 - 2\lambda_{(ij)}) \frac{\bar{b}_{ij}}{\|\bar{\mathbf{a}}_{ij}\|}, \tag{14b}$$

where, in the degenerate case when $\bar{\mathbf{a}}_{ij} = \mathbf{0}$, we define $\mathbf{a}_{ij} = \mathbf{0}$ and $b_{ij} = \bar{b}_{ij}$. Hence, we obtain one halfspace constraint $\mathbf{a}_{ij}^\top \mathbf{x} \leq b_{ij}$ for each neuron activation state $\lambda_{(ij)}$.

Given a specific input vector $\mathbf{x}$, we can then take the resulting activation pattern of the network $\lambda(\mathbf{x})$, and directly extract the halfspace constraints that apply at that activation state from (13). In fact, (12) shows that $(\mathbf{a}_{ij}, b_{ij})$ are actually functions of the activation patterns at earlier layers in the network, so $\mathbf{a}_{ij}(\lambda_{(l<i)})$ and $b_{ij}(\lambda_{(l<i)})$. Indeed, consider perturbing $\mathbf{x}$ as $\mathbf{x}' = \mathbf{x} + \delta$. The halfspace constraints $(\mathbf{a}_{ij}, b_{ij})$ will remain fixed under this perturbation until δ is large enough to change the activation pattern of an earlier layer $\lambda_{(l)}$, $l < i$. Consider the space of all such perturbed input values $\mathbf{x}$ that do not result in a change in any neuron activation. We have

$$P_\lambda = \{\mathbf{x} \mid \mathbf{a}_{ij}^\top \mathbf{x} \leq b_{ij} \quad i = 1, \ldots, L-1,\ j = 1, \ldots, l_i\}, \tag{15}$$

which is a polyhedron. We see that for each activation pattern λ, there exists an associated polyhedron P in the input space over which that activation pattern remains constant. The procedure for determining the polyhedron associated with an activation pattern is formalized in Algorithm 1. A unique activation pattern λ_k can then be defined for each affine region k. To be clear, we refer to the activation pattern for region k as λ_k, and use subscripts in parentheses to refer to the specific layer and neuron activation values. Lastly, for a fixed activation pattern, from (8) the ReLU network simplifies to the affine map $\mathbf{y} = \mathbf{C}_k\mathbf{x} + \mathbf{d}_k$ where

$$\begin{bmatrix}\mathbf{C}_k & \mathbf{d}_k\end{bmatrix} = \mathbf{\Theta}_L\Big(\prod_{i=1}^{L-1}\mathbf{\Lambda}_{k,(i)}(\mathbf{x})\mathbf{\Theta}_i\Big). \tag{16}$$

Algorithm 1: Polyhedron from Activation Pattern

Input: λ
Output: $\mathbf{A}, \mathbf{b}$
$k \leftarrow 1$
for $i = 1, \dots, L-1,\ j = 1, \dots, l_i$ **do**
  $\bar{\mathbf{a}}_{ij},\ \bar{b}_{ij} \leftarrow$ (12)
  $\mathbf{a}_{ij},\ b_{ij} \leftarrow$ (14)
  $\mathbf{A}[k,:],\ \mathbf{b}[k] \leftarrow \mathbf{a}_{ij},\ b_{ij}$
  $k \leftarrow k+1$
end
return $\mathbf{A}, \mathbf{b}$

B. Finding Essential Constraints

As mentioned previously, a polyhedron may have redundant constraints. We find that many of the constraints for the polyhedron P_k generated by λ_k are either duplicates or redundant, and can thus be removed. We define more formally the concepts of duplicate and redundant constraints.

Definition 7 (Duplicate Constraints): A constraint $\mathbf{a}_j^\top\mathbf{x} \le b_j$ is duplicate if there exists a scalar $\alpha > 0$ and a constraint with a lower index $\mathbf{a}_i^\top\mathbf{x} \le b_i$, $i < j$, such that $\alpha[\mathbf{a}_j^\top\ b_j] = [\mathbf{a}_i^\top\ b_i]$.

Definition 8 (Redundant & Essential Constraints): A constraint is redundant if the feasible set does not change upon its removal. An essential constraint is one which is not redundant.

We next describe how to remove the redundant constraints in a H-representation, leaving only the essential halfspace constraints. We first normalize all constraints, remove any numerically duplicate constraints, and consider the resulting H-representation $\mathbf{A}\mathbf{x} \le \mathbf{b}$. To determine if the remaining i^{th} constraint is essential or redundant, we define a new set of constraints with the i^{th} constraint removed,

$$\tilde{\mathbf{A}} = [\mathbf{a}_1 \dots \mathbf{a}_{i-1}\ \mathbf{a}_{i+1} \dots \mathbf{a}_M]^\top \tag{17a}$$

$$\tilde{\mathbf{b}} = [b_1 \dots b_{i-1}\ b_{i+1} \dots b_M]^\top, \tag{17b}$$

and solve the linear program

$$\max_{\mathbf{x}} \quad \mathbf{a}_i^\top\mathbf{x} \tag{18a}$$

$$\text{subject to} \quad \tilde{\mathbf{A}}\mathbf{x} \le \tilde{\mathbf{b}}. \tag{18b}$$

If the optimal objective value is less than or equal to b_i, constraint i is redundant. Note, it is critical that any numerically duplicate constraints are removed before this procedure. We formalize this procedure in Algorithm 2.

Algorithm 2: Essential H-representation

Input: λ
Output: $\mathbf{A}, \mathbf{b}$
$\mathbf{A}, \mathbf{b} \leftarrow$ Algorithm 1
$\mathbf{A}, \mathbf{b} \leftarrow$ remove duplicate constraints of $\mathbf{A}, \mathbf{b}$
$\tilde{\mathbf{A}}, \tilde{\mathbf{b}} \leftarrow \mathbf{A}, \mathbf{b}$ ▷ make copy of $\mathbf{A}, \mathbf{b}$
for $\mathbf{a}_i, b_i \in \mathbf{A}, \mathbf{b}$ **do**
  $\tilde{\mathbf{A}}, \tilde{\mathbf{b}} \leftarrow$ remove $\mathbf{a}_i, b_i$ from $\tilde{\mathbf{A}}, \tilde{\mathbf{b}}$
  $p^* \leftarrow$ optimal objective value of (18)
  if $p^* > b_i$ **then**
    add $\mathbf{a}_i, b_i$ back to $\tilde{\mathbf{A}}, \tilde{\mathbf{b}}$
  end
end
$\mathbf{A}, \mathbf{b} \leftarrow \tilde{\mathbf{A}}, \tilde{\mathbf{b}}$
return $\mathbf{A}, \mathbf{b}$

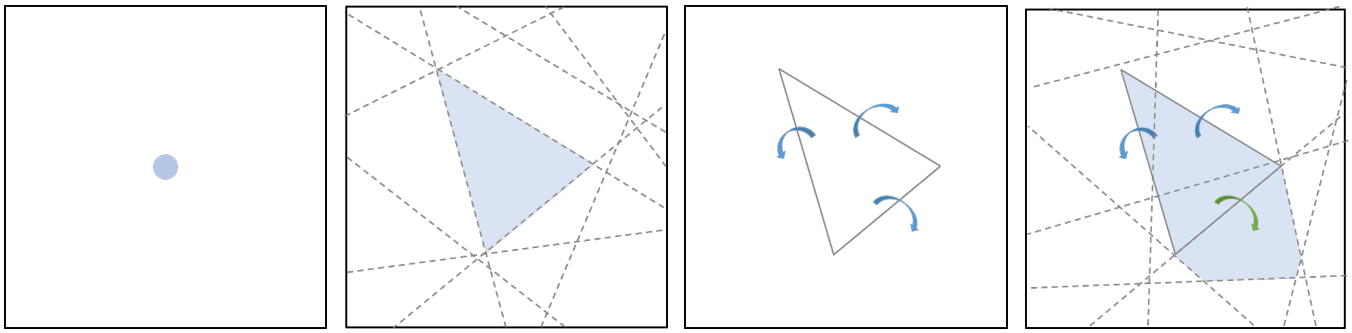
In the worst case, a single LP must be solved for each constraint to determine whether it is essential or redundant. However, heuristics exist to practically avoid this worst case complexity. For computational efficiency, Algorithm 2 can be modified to first identify and remove a subset of redundant constraints using the bounding box heuristic [55]. We observe that this can result in identifying as many as 90% of the redundant constraints. We find that other heuristics such as ray-shooting do not improve performance in our tests.

C. Determining Neighboring Activation Patterns

Consider two nieghboring polyhedra P_k and $P_{k'}$ such that their shared face is identified by $\mathbf{a}_\eta^\top\mathbf{x} = b_\eta$ and $\mathbf{a}_\eta^\top\mathbf{x} \le b_\eta$ is an essential constraint for P_k, while $-\mathbf{a}_\eta^\top\mathbf{x} \le -b_\eta$ is an essential constraint for $P_{k'}$. We call $\mathbf{a}_\eta^\top\mathbf{x} \le b_\eta$ the neighbor constraint. Given the neuron activation pattern λ for P_k, we next describe a procedure to find the activation pattern λ' for $P_{k'}$, from which we can find the essential halfspace constraints from the procedure described in the previous section. This allows us to systematically generate all the activation patterns that are realizable by the network, and to obtain the polyhedra and affine maps associated with each of those activation patterns.

Intuitively, to generate $\lambda_{k'}$ one ought to simply flip the activation of the neurons defining $\mathbf{a}_\eta^\top\mathbf{x} \le b_\eta$, and compute all the resulting new halfspace constraints from later layers in the network from (12). However, this intuitive procedure is incomplete because it does not correctly account for the influence that one neuron may have on other neurons in later layers. Flipping one neuron may lead to other neurons in later layers being flipped as well. To correctly determine the downstream effects of flipping a neuron, we provide a theorem for when neuron activations may need to be flipped.

Theorem 1 (Neighboring Activation Patterns): For a given region P_k, each neuron defines a hyperplane



(a) (b) (c) (d)

Fig. 2: This figure illustrates the main steps of the RPM algorithm (Algorithm 4). (a) The algorithm is initialized with a point in the input space which is evaluated by the ReLU network to determine the activation pattern λ. (b) Then the polyhedron corresponding to λ is computed using Algorithm 1. (c) Next, the essential constraints for the polyhedron, which in turn connect it to unexplored neighboring regions (indicated by blue arrows), are determined using Algorithm 2. (d) Then the activation pattern for a neighboring region (indicated by the green arrow) is found using Algorithm 3. The process then repeats by recursively exploring neighboring regions in this manner until the entire input set is explored.

$\mathbf{a}_{k,ij}^\top \mathbf{x} = b_{k,ij}$, as given by (14). Suppose regions P_k and $P_{k'}$ neighbor each other and are separated by $\mathbf{a}_\eta^\top \mathbf{x} = b_\eta$ where $P_k \subset \{\mathbf{x} \mid \mathbf{a}_\eta^\top \mathbf{x} \leq b_\eta\}$. Then,

1) if $[\mathbf{a}_{k,ij}^\top \quad -b_{k,ij}] \neq \alpha[\mathbf{a}_\eta^\top \quad -b_\eta]$ for some $\alpha \in \{0, 1\}$, the activation of neuron ij does not change, $\lambda_{k,ij} = \lambda_{k',ij}$.
2) If $[\mathbf{a}_{k,ij}^\top \quad -b_{k,ij}] = \alpha[\mathbf{a}_\eta^\top \quad -b_\eta]$ for some $\alpha \in \{0, 1\}$, then $[\mathbf{a}_{k',ij}^\top \quad -b_{k',ij}] = \alpha[\mathbf{a}_\eta^\top \quad -b_\eta]$ for some $\alpha \in \{-1, 0\}$, and the activation $\lambda_{k',ij}$ should be set to ensure this holds.

The proof for Theorem 1 is in the Appendix. Then, the procedure to generate a neighboring activation pattern is defined in Algorithm 3. We present Algorithm 3 in as simple a manner as possible, however, for a more efficient approach, one need not loop over all neurons in the network, but only those which meet condition 2) of Theorem 1.

Algorithm 3: Neighboring Activation Pattern

Input: λ_k, $\mathbf{a}_\eta$, b_η, $\mathbf{\Theta}$
Output: $\lambda_{k'}$

$\lambda_{k'} \leftarrow \lambda_k$ ▷ Initialize neighbor region λ
for $i = 1, \ldots, L-1,\ j = 1, \ldots, l_i$ **do**
  $\mathbf{a}_{k',ij},\ b_{k',ij} \leftarrow$ (14)
  if $\mathbf{a}_{k',ij} = \mathbf{a}_\eta$ **and** $b_{k',ij} = b_\eta$ **then**
    $\lambda_{k',ij} \leftarrow \neg\lambda_{k,ij}$
  else if $\mathbf{a}_{k',ij} = \mathbf{0}$ **and** $b_{k',ij} = 0$ **then**
    $\lambda_{k',ij} \leftarrow 0$
end
return $\lambda_{k'}$

D. Reachable Polyhedral Marching

Building on Algorithms 2 & 3, we now define our main algorithm, Reachable Polyhedral Marching (RPM) in Algorithm 4 for explicitly enumerating all polyhedra in the input space of a ReLU network, resulting in the explicit PWA representation. First, we start with an initial point in the input space and evaluate the network to find the activation pattern. Note, this initial point should lie in the interior of a polyhedral region, not on the boundary. From this, the essential H-representation is found using Algorithm 2. For each essential constraint we generate a neighboring activation pattern using Algorithm 3. The process then repeats with each new neighboring activation pattern being added to a working set. For a given starting polyhedron each neighbor polyhedron is enumerated, and since all polyhedra are connected via neighbors, Algorithm 4 is guaranteed to enumerate every polyhedron in the input space. Figure 2 illustrates the steps of RPM and Figure 1 shows the result of applying Algorithm 4 to a randomly initialized ReLU network.

Algorithm 4: Reachable Polyhedral Marching

Input: λ_0, $\mathbf{\Theta}$
Output: Explicit PWA representation

PWA = $\emptyset$; visited = $\emptyset$
working set = $\{\lambda_0\}$
while working set $\neq \emptyset$ **do**
  $\lambda_k \leftarrow$ pop next λ off working set
  $\mathbf{A}_k, \mathbf{b}_k \leftarrow$ Algorithm 2 ▷ Retrieve essential H-rep
  $\mathbf{C}_k,\ \mathbf{d}_k \leftarrow$ (16) ▷ Retrieve affine map
  push $(\mathbf{A}_k, \mathbf{b}_k, \mathbf{C}_k, \mathbf{d}_k)$ onto PWA
  for $\mathbf{a}_{k,i}, b_{k,i} \in \mathbf{A}_k, \mathbf{b}_k$ **do** ▷ For each neighbor
    $\lambda_{k'} \leftarrow$ Algorithm 3 ▷ $\mathbf{a}_\eta, b_\eta = \mathbf{a}_{k,i}, b_{k,i}$
    if $\lambda_{k'} \notin$ visited $\cup$ working set **then**
      push $\lambda_{k'}$ onto working set
    end
  end
  push λ_k onto visited
end
return PWA

V. Reachability

A. Forward Reachability

The forward reachable set of a PWA function over some input set is simply the union over the forward reachable sets

of each polyhedron comprising the input set. The image of a polyhedron under an affine map is

$$P_{image} = \{\mathbf{y} \mid \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{d}, \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}\}. \tag{19}$$

For invertible $\mathbf{C}$, the H-representation of the image is

$$P_{image} = \{\mathbf{y} \mid \mathbf{A}\mathbf{C}^{-1}\mathbf{y} \leq \mathbf{b} + \mathbf{A}\mathbf{C}^{-1}\mathbf{d}\}. \tag{20}$$

In the case the affine map is not invertible, more general polyhedral projection methods such as block elimination, Fourier-Motzkin elimination, or parametric linear programming can compute the H-representation of the image [56], [57]. Our implementation uses the block elimination projection in the case of a non-invertible affine map [58].

Our RPM algorithm is used to perform forward reachability as follows. We first specify a polyhedral input set whose image through the ReLU network we want to compute. This is a set over which to perform the RPM algorithm. For each activation pattern λ_k we also compute the image of $(\mathbf{A}_k, \mathbf{b}_k)$ under the map $\mathbf{y} = \mathbf{C}_k\mathbf{x} + \mathbf{d}_k$. To do this, an additional line is introduced between lines 9 and 10 of Algorithm 4

$$(\mathbf{A}_k, \mathbf{d}_k)_{forward} \leftarrow \text{project}(\mathbf{A}_k, \mathbf{d}_k, \mathbf{C}_k, \mathbf{d}_k) \tag{21}$$

where $\text{project}(\mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d})$ applies (20) if $\mathbf{C}$ invertible and block elimination on (19) otherwise.

B. Backward Reachability

The preimage of a polyhedron under an affine map is

$$P_{pre} = \{\mathbf{x} \mid \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{d}, \quad \mathbf{A}\mathbf{y} \leq \mathbf{b}\} \tag{22a}$$

$$P_{pre} = \{\mathbf{x} \mid \mathbf{A}\mathbf{C}\mathbf{x} \leq \mathbf{b} - \mathbf{A}\mathbf{d}\}. \tag{22b}$$

Like forward reachability, performing backward reachability only requires a small modification to Algorithm 4. We first specify a polyhedral output set whose preimage we would like to compute. For each activation pattern λ_k, we then compute the intersection of the preimage of the given output set under the map $\mathbf{C}_k\mathbf{x} + \mathbf{d}_k$ with P_k (the polyhedron for which the affine map is valid). Two additional lines are thus introduced between lines 9 and 10 of Algorithm 4

$$P_{pre} \leftarrow \text{Equation } 22b \tag{23a}$$

$$(\mathbf{A}_k, \mathbf{b}_k)_{backward} \leftarrow P_k \cap P_{pre}. \tag{23b}$$

Multiple backward reachable sets can be solved for simultaneously at the added cost of repeating (23a) and (23b) for each output set argument to Algorithm 4.

Finally, we address the issue of finding forward and backward reachable sets iterated over multiple time steps. For this, we note that a ReLU network that is applied iteratively over T timesteps, $\mathbf{x}_{t+1} = F(\mathbf{x}_t)$ for $t = 0, \ldots, T-1$, is mathematically equivalent to a single ReLU network consisting of T copies of the original network concatenated end to end, $\mathbf{x}_T = F_T(\mathbf{x}_0) := F \circ \cdots \circ F(\mathbf{x}_0)$. If the original network $F(\mathbf{x})$ has r hidden layer neurons and N layers, the equivalent multi-time step network $F_T(\mathbf{x})$ has Tr neurons and $T(N-1)+1$ layers. We simply perform RPM for forward or backward reachability on the equivalent multi-time step network $F_T(\mathbf{x})$.

VI. Computing Invariant Sets

In this section we describe a general framework for computing robust control invariant sets of discrete-time PWA dynamical systems which was first put forth in [49]. Given a dynamical system represented as a ReLU network we can use Algorithm 4 to retrieve the explicitly defined PWA system and use the following techniques to compute robust control invariant sets. We first define a few useful terms.

Definition 9 (Fixed Point): A vector $\mathbf{x} \in \mathbb{R}^n$ is a fixed point of the function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^n$ if $f(\mathbf{x}) = \mathbf{x}$.

Definition 10 (Invariant Set): A (forward) invariant set of an autonomous dynamical system $\mathbf{x}^+ = f(\mathbf{x})$ is a set of states for which any trajectory starting in the invariant set remains in the set for all time.

Building on these definitions, a set of states is robust control invariant if there always exists a control input such that the subsequent state under the dynamics will remain in the set under all possible disturbance values. Computing robust invariant sets for autonomous systems or control invariant sets for deterministic controlled systems are special instances of this general problem. We now summarize the main results from [49].

Suppose we have the system model

$$\mathbf{x}^+ = f(\mathbf{x}, \mathbf{u}, \mathbf{w}) \tag{24a}$$

$$(\mathbf{x}, \mathbf{u}) \in Y \subset \mathcal{X} \times \mathcal{U} \tag{24b}$$

$$\mathbf{w} \in W(\mathbf{x}, \mathbf{u}) \subset \mathcal{W} \tag{24c}$$

where f is PWA, $\mathcal{X} = \mathbb{R}^n$, $\mathcal{U} = \mathbb{R}^m$, and $\mathcal{W} = \mathbb{R}^p$. We define the set of admissible state, control, and disturbance triples as

$$\gamma = \{(\mathbf{x}, \mathbf{u}, \mathbf{w}) \mid (\mathbf{x}, \mathbf{u}) \in Y, \ \mathbf{w} \in W(\mathbf{x}, \mathbf{u})\}. \tag{25}$$

It follows that the set of admissible inputs and states are

$$U(\mathbf{x}) = \{\mathbf{u} \mid (\mathbf{x}, \mathbf{u}) \in Y\} \tag{26a}$$

$$X = \{\mathbf{x} \mid \exists \mathbf{u} \ s.t. \ (\mathbf{x}, \mathbf{u}) \in Y\} = Proj_{\mathcal{X}}(Y). \tag{26b}$$

Additionally, let X_f represent a target robust control invariant set. The general strategy is to start with X_f and step backwards in time to compute set of states that lead to X_f. We can then iterate upon this until convergence or as long our computational resources allow.

Definition 11 (Predecessor Set): Given a set $\Omega \subset \mathcal{X}$, the predecessor set $Pre(\Omega)$ is the set of states for which there exists an admissible input such that, for all allowable disturbances, the successor state is in Ω, i.e.,

$$Pre(\Omega) := \{\mathbf{x} \mid \exists \mathbf{u} \in U(\mathbf{x}) \ s.t. \ f(\mathbf{x}, \mathbf{u}, \mathbf{w}) \in \Omega \ \forall \mathbf{w} \in W(\mathbf{x}, \mathbf{u})\}. \tag{27}$$

Let X_t denote the t-step predecessor set to X_f. That is,

$$X_0 = X_f \tag{28a}$$

$$X_{t+1} = Pre(X_t). \tag{28b}$$

A set $S \subset X$ is robust control invariant if for any $\mathbf{x} \in S$ there exists a $\mathbf{u} \in U(\mathbf{x})$ such that $f(\mathbf{x}, \mathbf{u}, \mathbf{w}) \in S$ for all $\mathbf{w} \in W(\mathbf{x}, \mathbf{u})$. It is known that X_t is robust control invariant

for all t if and only if X_f is robust control invariant. Thus, the problem of computing robust control invariant sets involves finding an initial robust control invariant set X_f and then computing predecessor sets X_t. For PWA systems it is often advantageous to compute X_f for an affine region of f since computing robust control invariant sets is much easier for affine systems than for nonlinear systems.

Next we present the main result of [49] which leads to Algorithm 5 for computing robust control invariant sets. Define the set of robust state and control tuples and the preimage operation as

$$\Sigma := \{(\mathbf{x}, \mathbf{u}) \in Y \mid f(\mathbf{x}, \mathbf{u}, \mathbf{w}) \in \Omega \ \forall \mathbf{w} \in W(\mathbf{x}, \mathbf{u})\} \quad (29a)$$

$$\Phi := f^{-1}(\Omega) = \{(\mathbf{x}, \mathbf{u}, \mathbf{w}) \mid f(\mathbf{x}, \mathbf{u}, \mathbf{w}) \in \Omega\}. \quad (29b)$$

Then the main result from [49] is

$$Pre(\Omega) = Proj_{\mathcal{X}}(\Sigma) \quad (30a)$$

$$\Sigma = Y \setminus Proj_{\mathcal{X}\times\mathcal{U}}(\gamma \setminus \Phi) \quad (30b)$$

where $A \backslash B$ is the set difference between sets A and B. Algorithm 5 outlines the process for computing the i-step predecessor set of a seed robust control invariant set X_0. If γ and X_f are each a union of polyhedra then each X_t will be a union of polyhedra and Φ can be computed by adapting the backward reachability algorithm from Section V and standard computational geometry methods can be used to perform the necessary set projections and differences. Algorithm 5 is very general and faster algorithms exist for special cases, such as if the disturbance is additive [59].

Algorithm 5: Robust Control Invariant Set [49]

Input: γ, X_0
Output: X_T

Compute the projection $Y = Proj_{\mathcal{X}\times\mathcal{U}}(\gamma)$
for $t = 1, \ldots, T$ **do**
  Compute the preimage $\Phi_t = f^{-1}(X_{t-1})$
  Compute the set difference $\Delta_t = \gamma \setminus \Phi_t$
  Compute the projection $\Psi_t = Proj_{\mathcal{X}\times\mathcal{U}}(\Delta_t)$
  Compute the set difference $\Sigma_t = Y \setminus \Psi_t$
  Compute the projection $X_t = Proj_{\mathcal{X}}(\Sigma_t)$
end
return X_T

VII. Computing Regions of Attraction

In Section VI we gave a general overview of existing results for computing robust control invariant sets of PWA dynamical systems which we can apply to ReLU networks after computing the explictly defined PWA representation using Algorithm 4. In this section we specifically consider the problem of computing invariant ROAs given a ReLU network that represents a deterministic autonomous system. A common goal in the analysis of nonlinear dynamical systems is to identify the maximal set of states that is both a ROA and an invariant set. We define a ROA as follows,

Definition 12 (Region of Attraction): A region of attraction (ROA) of an autonomous dynamical system $\mathbf{x}^+ = f(\mathbf{x})$ is a set of states which asymptotically converge to a stable fixed point $\mathbf{x}_{fp}$. The set of *all* states which asymptotically converge to a stable fixed point is the maximal ROA, i.e. $\text{ROA}_{max} = \{\mathbf{x} \mid \lim_{t\to\infty} f(\mathbf{x}) = \mathbf{x}_{fp}\}$.

Note that under our definition, a ROA need not be an invariant set (although the maximal ROA is invariant), but since our approach uses seed ROAs that are also invariant sets, as we grow the ROA backwards in time it remains an invariant set. Here we consider computing, via backward reachability, ROAs of ReLU networks that model the discrete-time dynamics of an autonomous system. We first identify a stable fixed point, then compute a small seed ROA around this equilibrium. Then we use our backward reachability method to find the t-step backward reachable set of the seed ROA, which approximates the maximal ROA of the considered equilibrium. In the following subsections we detail the procedure for finding fixed points, checking whether they are stable equilibria, and computing seed ROAs. We also describe how the computation of the t-step ROA can be accelerated if the ReLU network is homeomorphic and how to check this condition.

A. Finding Fixed Points

To find fixed points of a ReLU network we use RPM to solve for the explicit PWA representation and for each region k solve the following system of equations.

$$\mathbf{C}_k\mathbf{x} + \mathbf{d}_k = \mathbf{x} \quad (31a)$$

$$\implies (\mathbf{I} - \mathbf{C}_k)\mathbf{x} = \mathbf{d}_k \quad (31b)$$

We consider $\mathbf{x}$ a fixed point if it is the unique solution of (31b) and lies on the interior of P_k. We do not consider the case of non-unique solutions. A fixed point in P_k is a stable equilibrium if $\mathbf{C}_k$ is a stable dynamics matrix. That is, if the eigenvalues of $\mathbf{C}_k$ have magnitude strictly less than one.

B. Finding Seed ROAs

For our backward reachability algorithm we require a polyhedral seed ROA to start from. Once a stable fixed point is located we translate the corresponding affine system to a linear one via a coordinate transformation. We then use existing techniques for computing invariant polyhedral ROAs of stable linear systems [60], [61], [62], for which we not [60] is the most reliable and straightforward. Once a polyhedral ROA is found it can be scaled it up or down to ensure it is a subset of the polyhedron for which the local affine system is valid.

C. The Homeomorphic Case

The backward reachability algorithm as previously presented can be made more efficient if we know the preimage to be computed is a connected set. In this case, we can alter the backward reachability algorithm to only add neighbor polyhedra to the working set if they are neighbors of a polyhedron known to be in the preimage. When initialized within the connected preimage, this procedure restricts the polyhedra RPM has to enumerate, leading to a substantial reduction in the compute time.

It is known that the preimage of a connected set will also be connected if the function mapping between the input and output spaces is a homeomorphism.

Definition 13 (Homeomorphism): A function $f : \mathcal{X} \to \mathcal{Y}$ between two metric spaces is a homeomorphism if:

- f is bijective
- f is continuous
- f^{-1} is continuous.

Furthermore, the composition of homeomorphisms is also a homeomorphism.

Specific conditions for a PWA function to be homeomorphic were originally given in [63], and restated in some useful ways in [64] and [65].

Theorem 2 (Homeomorphic PWA Functions [64]): Assume affine maps $\mathbf{C}_k\mathbf{x} + \mathbf{d}_k$ for regions $P_k \in \mathcal{P}$. Let $\mathbf{C}[r]$ denote the matrix consisting of the first r rows and columns of $\mathbf{C}$. If for each $r = 1, \ldots, n$,

$$\text{sgn det}(\mathbf{C}_k[r]) = \sigma_r \quad \forall k \tag{32}$$

where $\sigma_r \neq 0$ and sgn det() is the sign of the determinant, then the PWA function is a homeomorphism. That is, all minor determinants of a given order must have the same sign across all affine map matrices.

Note that all $\mathbf{C}_k$ nonsingular is a necessary condition ($\sigma_r \neq 0$). This theorem gives us a simple way to check whether a ReLU network is a homeomorphism once we have obtained the explicitly defined PWA function.

Our method of decomposing ReLU networks into their explicit PWA representations using RPM and checking the homeomorphism condition is, to the best of our knowledge, the first approach for determining the invertibility of an arbitrary ReLU network. We emphasize that it is not immediately obvious that ReLU networks can be homeomorphisms because, as the authors of [66] note, the ReLU activation itself is not homeomorphic and layer weight matrices often map between spaces of different dimension. But even though the equation of a ReLU network may not look homeomorphic, the underlying PWA function may still be homeomorphic.

Somewhat surpisingly, we find the homeomorphic property holds for many of the learned dynamical systems used in Section VIII without special modifications to the training procedure. Because of this, we think investigation into learning homeomorphic neural networks is a promising direction for future research. Finally, we are not aware of any other work that checks and uses the homeomorphism conditions to accelerate PWA backward reachability, but we have found that utilizing this information can result in significant speedups, as discussed in Section VIII-A.

VIII. Examples

Examples A&C are run on a 2013 Dell Latitude E6430s laptop with an Intel Core i7-3540M processor and 16GB of RAM. Examples B&D are run on a desktop with two Intel Xeon E5-2650 v3 processors and 64GB of RAM. Our implementation uses the GLPK open-source LP solver and the JuMP optimization package [67]. The polyhedron coloring in plots is random.

A. van der Pol Oscillator Example

In this example we illustrate ROA computation for a learned reverse-time van der Pol oscillator dynamical system. We use the reverse-time dynamics because they give rise to a bounded, nonconvex ROA (the boundary is the limit cycle). We first train a ReLU network to learn a 10 Hz discrete-time model from the continuous-time model

$$\dot{x}_1 = -x_2 \tag{33a}$$

$$\dot{x}_2 = x_1 + x_2(x_1^2 - 1). \tag{33b}$$

The learned dynamics network has two hidden layers with 20 neurons each and the resulting PWA function found with RPM has 248 affine regions with a stable fixed point near the origin. We then find a seed ROA for the stable affine system as described in Section VII-B. Next, we verify the network is a homeomorphism by evaluating the conditions in Section VII-C; from this we know that a connected set in the output space will have a connected preimage. To find a T-step ROA, we simply perform backward reachability on the seed ROA for the desired number of steps.

We show the resulting 5, 10, 15, 20, 25, and 30-step ROAs in Figure 3. Note that in this example we compute a T-step ROA without needing intermediate ROAs by simply concatenating T dynamics networks. However, we show multiple ROAs to illustrate how the learned ROA approaches the maximal ROA for the true dynamics (bounded in blue) as the steps increase. By verifying the learned dynamics function is homeomorphic, we can run RPM on a much more restricted domain, providing a significant computational speedup. Specifically, the 10-step dynamics network has 33,946 regions, but by knowing the ROA is connected RPM only explores 2,735 regions, resulting in the computation taking 1 minute instead of 16 minutes, a 16x speedup.

B. Pendulum Example

In this example we pair our RPM tool with the popular multi-parametric toolbox (MPT3) [9] to find a control invariant set for a learned pendulum model. We first learn a ReLU network that models the continuous-time dynamics

$$\ddot{\theta} = \frac{u + mgl\sin\theta}{ml^2} \tag{34}$$

as a discrete-time model where $m = 1$kg, $g = 9.81$m/s^2, $l = 1$m, and θ is measured clockwise from the upright position. Using RPM over the domain $\theta \in [-180°, 180°]$, $\dot{\theta} \in [-180°/\text{s}, 180°/\text{s}]$, $u \in [-5, 5]$ N-m, we find the underlying PWA function has 217 regions. We then use MPT3 to compute the control invariant set shown in Figure 4. Compared to Algorithm 5, MPT3 uses a slightly different, but still recursive approach. Rather than starting with a seed control invariant set, MPT3 computes the following recursion

$$X_0 = \mathcal{X} \tag{35a}$$

$$X_{i+1} = \{\mathbf{x} \mid f(\mathbf{x}, \mathbf{u}) \in X_i, \mathbf{u} \in \mathcal{U}\} \tag{35b}$$

for a compact polyhedral domain $\mathcal{X} \cup \mathcal{U}$. If $X_{t+1} = X_t$, then X_t is a control invariant set. The control invariant set in

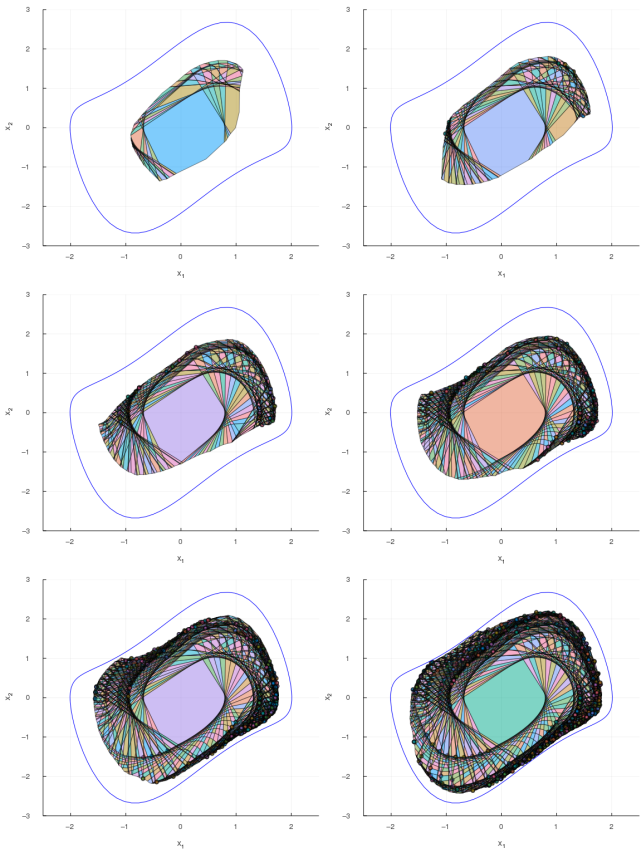


Fig. 3: 5, 10, 15, 20, 25, and 30-step regions of attraction (ROAs) for the learned van der Pol system. The learned dynamics network is continuously invertible, implying that preimages of connected sets are connected. This property restricts the space explored with RPM, resulting in a 16x speedup in ROA computation. The blue line represents the boundary of the true continuous-time maximal ROA. As we perform more computation (i.e. more steps), the ROA of the learned system approaches the ROA of the true system.

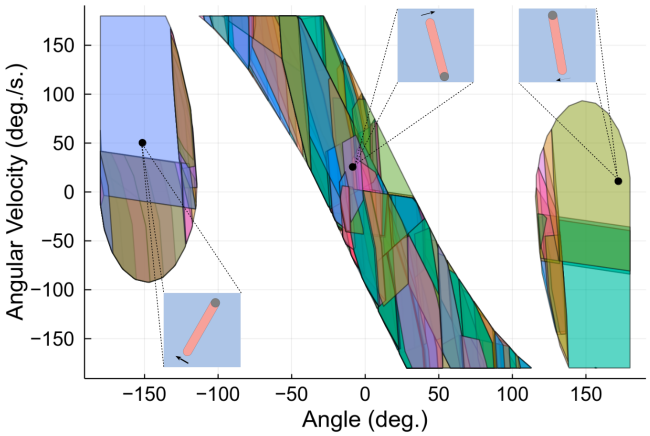


Fig. 4: A control invariant set for a learned, torque controlled pendulum system. The domain of the dynamics is $\theta \in (-180°, 180°)$, $\dot{\theta} \in [-180°/\text{s}, 180°/\text{s}]$, $u \in [-5, 5]$ N-m, where $0°$ is upright. Note that we restrict the pendulum so it is not allowed to pass through the down position. When RPM is paired with MPT3 we are able to compute complicated control invariant sets of learned systems, like this unconnected set, as opposed to convex approximations. Within each connected component a sample state is illustrated to help visualize the possible pendulum configurations.

Figure 4 converged in 61 steps after 10 hours of computation, and is represented by 958 polyhedra.

As shown in Figure 4, the control invariant set is the union of 3 disjoint sets. The leftmost set represents the states for which there is enough control authority to keep the pendulum clockwise of the swing-down position, but not enough to swing back up to upright. The middle set represents the states for which there is enough control authority to keep the pendulum from swinging all the way down. Finally, rightmost set has a similar interpretation to the leftmost set, being the states for which there is enough control authority to keep the pendulum counterclockwise of the swing-down position, but not enough to swing back upright. With the explicit PWA representation given by RPM, the wide array of PWA analysis tools offered by the MPT3 toolbox is able to be leveraged for learned dynamics and controls models.

C. Aircraft Collision Avoidance Example

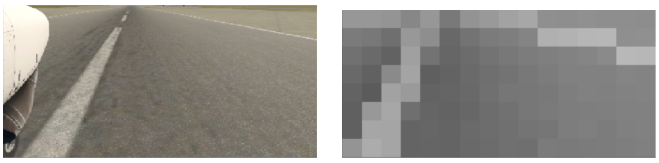
The ACAS Xu networks are 45 distinct policy networks designed to issue advisory warnings to avoid mid-air collisions for unmanned aircarft. Each network has five inputs, five outputs (advisories), and 300 neurons. Inputs are relative distance, angles, and speeds (see [68] for more details).

The appendix of [21] lists ten safety properties the networks should satisfy. Here we consider Property 3. This property is satisfied if the network never outputs a "Clear of Conflict" advisory when the intruder is directly ahead and moving towards the ownship. We use the backward reachability algorithm to verify whether Property 3 is satisfied. A nonempty backward reachable set implies that some allowable inputs will map to unsafe outputs and the safety property is not satisfied. Table I shows the results of verifying Property 3 for four of the networks. A comparison is given against the fastest existing exact reachability method based on the face lattice representation [8] as well as the Reluplex and Marabou verification algorithms [21], [69]. Each algorithm was run on a single core. We note that a parallelized algorithm is also provided for the face lattice approach [8], and our RPM algorithm is also amenable to a parallelized implementation.

The results in Table I show the advantage of our proposed method when verifying network properties that are found to not hold. Our RPM algorithm is anytime, in the sense that once an unsafe input polyhedron is found, the algorithm can terminate without completing the full reachability computation. This is in contrast to all existing exact reachability methods that proceed layer-by-layer through the network. They must solve the entire backward reachability problem to conclude any verification result, whether safe or unsafe.

TABLE I: ACAS Xu Property 3 Verification Results

Network	Result	Time (s) RPM	Time (s) Face Lattice	Time (s) Reluplex	Time (s) Marabou
$N_{1,7}$	Unsafe	**0.01**	6.66	2.15	0.70
$N_{1,8}$	Unsafe	**0.01**	5.45	4.32	1.49
$N_{3,8}$	Safe	19.72	**9.35**	231.28	40.13
$N_{5,6}$	Safe	31.92	**17.28**	366.39	54.07



(a) Full observation. (b) Downsampled observation.

Fig. 5: The X-Plane 11 flight simulator is used to generate realistic image observations (left) that are then downsampled (right). The downsampled images are used to learn an image-based controller, as well as to learn an observation model from state to image observation. Putting these two components together along with a learned dynamics network results in closed-loop system dynamics represented as a single ReLU network. These images and learned controller/observation models are from [10].

D. Verifying Image-Based Control

In this example we apply the tools of Section VI to a larger dynamical system which models the closed-loop behavior of a taxiing airplane controlled from camera images. We consider the system introduced in [10] where a taxiing airplane is regulated to the centerline of a runway via rudder-angle control input based on runway images taken from a camera on the wing. Because of the image observation, we cannot model the observation from physical principles. Instead, we modify the learned models in [10] to obtain a network that approximates the observation given a state

$$\mathbf{y}_t = f_{obs}(\mathbf{x}_t), \tag{36}$$

and a control network

$$u = f_{ctrl}(\mathbf{y}_t). \tag{37}$$

The continuous time dynamics are

$$\dot{p} = v \sin\theta \tag{38a}$$

$$\dot{\theta} = \frac{v}{L} \tan u \tag{38b}$$

for crosstrack position p, heading error θ, rudder angle u, and speed $v = 5$ m/s. We learn a 5Hz discrete-time model,

$$\mathbf{x}_{t+1} = f_{dyn}(\mathbf{x}_t, u), \tag{39}$$

and then have the closed-loop system

$$\mathbf{x}_{t+1} = f_{cl}(\mathbf{x}_t) = f_{dyn}(\mathbf{x}_t, f_{ctrl} \circ f_{obs}(\mathbf{x}_t)) \tag{40}$$

which can be represented as one ReLU network.

Next, we use Algorithm 4 to retrieve the explicit PWA function for f_{cl} over the domain $p \in [-5, 5]$ m., $\theta \in [-15°, 15°]$. The resulting PWA function has 93,298 regions, is homeomorphic, and took 44 hours to compute. This PWA function is significantly larger than other PWA dynamics that ROA methods have been tested on in the literature ($O(100,000)$ vs regions vs $O(1,000)$ regions). For example, to use the LP framework of [47] for synthesizing a Lyapunov function would require an LP with $\sim 400,000$ variables, likely many more constraints, and a preprocessing step which is more expensive than solving the LP.

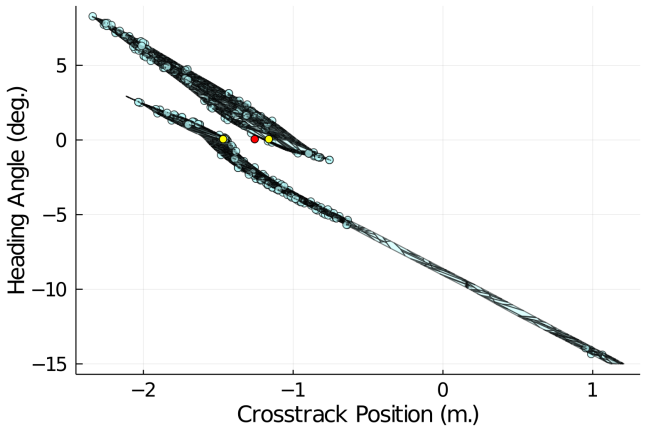


Fig. 6: 51-step regions of attraction (ROAs) for the two stable fixed points of the image-based airplane runway taxiing system from Fig. 5. The closed-loop dynamics has 93,298 affine regions, precluding the use of Lyapunov-based approaches. This plot shows the useful 'anytime' property of our proposed approach where, at each step, we compute valid ROAs resulting in 51-step ROAs of useful size. The two stable fixed points are shown in yellow, and the one unstable fixed point is shown in red. The teal points represent polyhedra which are too small to plot as sets. These ROAs are represented by 13,805 polyhedra.

After computing the explicit PWA function, we find three fixed points, two stable and one unstable. Next, we find a seed ROA for each stable fixed point and perform backward reachability to grow the ROAs. During the backward reachability computation, we use a few strategies to make the problem more tractable. First, we utilize the homeomorphic property of the dynamics to only explore connected backward reachable sets. Second, we compute backward reachable sets iteratively with the 1-step PWA dynamics instead of in one shot using concatenated ReLU networks. Third, at each step we merge the polyhedra in the ROA into a union of polyhedra with as few polyhedra as possible.

In Figure 6 we show two 51-step ROAs, one associated with each stable fixed point (yellow). The unstable fixed point is shown in red. Any state within these ROAs will asymptotically converge to its respective stable fixed point.

IX. Conclusions

We proposed the Reachable Polyhedral Marching (RPM) algorithm to incrementally construct an exact PWA representation of a ReLU network. RPM computes an explicit PWA function representation for a given ReLU network which can then be used to quickly find forward and backward reachable sets of ReLU networks. In addition, we demonstrated how,

for nonlinear dynamical systems modeled as ReLU networks, the RPM tool can be leveraged to (i) compute intricate robust control invariant sets and (ii) certify stability and compute ROAs. RPM also provides a novel way to determine invertibility of ReLU networks, which we show leads to accelerated computation of ROAs. Finally, the incremental property of RPM is shown to be especially useful for identifying the existence of unsafe inputs during verification.

In future work we are interested in investigating the homeomorphic properties of ReLU networks. There are many studies on the invertibility of neural networks that propose to restrict the network architecture and training such that invertibility is guaranteed with high probability. However, these restrictions might be loosened by further understanding the connection between PWA homeomorphisms and ReLU networks. Another area for future work is to use the insights gained from Theorem 1 to develop an inverse RPM method for encoding explicit PWA functions as ReLU networks. Existing work in this direction yields ReLU networks with many more parameters than necessary, not taking full advantage of the dependency of PWA parameters. Also of interest is the potential to use the PWA decomposition given by RPM to assess where in the domain the ReLU network is likely to generalize well and where it is not.

ACKNOWLEDGEMENTS

The authors would like to thank Yue Meng for helpful insights and feedback on the RPM algorithm.

REFERENCES

[1] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[2] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin, "Path planning for mobile robot navigation using voronoi diagram and fast marching," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 2376–2381.

[3] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

[4] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[5] J. Lei and K. Jia, "Analytic marching: An analytic meshing solution from deep implicit surface networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5789–5798.

[6] W. Xiang, H.-D. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with relu activations," *arXiv preprint arXiv:1712.08163*, 2017.

[7] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *International Symposium on Formal Methods*. Springer, 2019, pp. 670–686.

[8] X. Yang, H.-D. Tran, W. Xiang, and T. Johnson, "Reachability analysis for feed-forward neural networks using face lattices," *arXiv preprint arXiv:2003.01226*, 2020.

[9] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *2013 European control conference (ECC)*. IEEE, 2013, pp. 502–510.

[10] S. M. Katz, A. L. Corso, C. A. Strong, and M. J. Kochenderfer, "Verification of image-based neural network controllers using generative models," *Journal of Aerospace Information Systems*, vol. 19, no. 9, pp. 574–584, 2022.

[11] J. A. Vincent and M. Schwager, "Reachable Polyhedral Marching (RPM): A Safety Verification Algorithm for Robotic Systems with Deep Neural Network Components," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9029–9035.

[12] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in neural information processing systems*, 2014, pp. 2924–2932.

[13] B. Hanin, "Universal function approximation by deep neural nets with bounded width and relu activations," *Mathematics*, vol. 7, no. 10, p. 992, 2019.

[14] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," in *International Conference on Learning Representations*, 2018.

[15] J. He, L. Li, J. Xu, and C. Zheng, "Relu deep neural networks and linear finite elements," *Journal of Computational Mathematics*, 2019.

[16] D. Rolnick and K. Kording, "Reverse-engineering deep relu networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8178–8187.

[17] J. Z. Kolter and G. Manek, "Learning stable deep dynamics models," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 128–11 136.

[18] W. Jin, Z. Wang, Z. Yang, and S. Mou, "Neural certificates for safe control policies," *arXiv preprint arXiv:2006.08465*, 2020.

[19] X. Lin, H. Zhu, R. Samanta, and S. Jagannathan, "Art: Abstraction refinement-guided training for provably correct neural networks," in *Formal Methods in Computer-Aided Design*, 2020.

[20] S. Mell, O. Brown, J. Goodwin, and S.-H. Son, "Safe predictors for enforcing input-output specifications," *arXiv preprint arXiv:2001.11062*, 2020.

[21] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

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

[23] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[24] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, and I. Dhillon, "Towards fast computation of certified robustness for relu networks," in *International Conference on Machine Learning (ICML)*, 2018.

[25] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1599–1614.

[26] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6367–6377.

[27] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in neural information processing systems*, 2018, pp. 4939–4948.

[28] S. Dutta, S. Jha, S. Sanakaranarayanan, and A. Tiwari, "Output range analysis for deep neural networks," *arXiv preprint arXiv:1709.09130*, 2017.

[29] P. Kouvaros and A. Lomuscio, "Formal verification of cnn-based perception systems," *arXiv preprint arXiv:1811.11373*, 2018.

[30] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," *arXiv preprint arXiv:1706.07351*, 2017.

[31] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," *arXiv preprint arXiv:1805.02242*, 2018.

[32] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 169–178.

[33] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, pp. 1–30, 01 2019.

[34] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 3–18.

[35] K. D. Julian and M. J. Kochenderfer, "A reachability method for verifying dynamical systems with deep neural network controllers," *arXiv preprint arXiv:1903.00520*, 2019.

[36] N. Rober, M. Everett, and J. P. How, "Backward reachability analysis for neural feedback loops," *arXiv preprint arXiv:2204.08319*, 2022.

[37] H. Robinson, A. Rasheed, and O. San, "Dissecting deep neural networks," *arXiv preprint arXiv:1910.03879*, 2019.

[38] B. Hanin and D. Rolnick, "Complexity of linear regions in deep networks," in *International Conference on Machine Learning*, 2019, pp. 2596–2604.

[39] S. Xu, J. Vaughan, J. Chen, A. Zhang, and A. Sudjianto, "Traversing the local polytopes of relu neural networks: A unified approach for network verification," *arXiv preprint arXiv:2111.08922*, 2021.

[40] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[41] H. Yin, P. Seiler, and M. Arcak, "Stability analysis using quadratic constraints for systems with neural network controllers," *IEEE Transactions on Automatic Control*, pp. 1–1, 2021.

[42] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Conference on Robot Learning*. PMLR, 2018, pp. 466–476.

[43] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado, "Learning region of attraction for nonlinear systems," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 6477–6484.

[44] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, "Lyapunov-stable neural-network control," *arXiv preprint arXiv:2109.14152*, 2021.

[45] T. Wei and C. Liu, "Safe control with neural network dynamic models," in *Learning for Dynamics and Control Conference*. PMLR, 2022, pp. 739–750.

[46] P. Biswas, P. Grieder, J. Löfberg, and M. Morari, "A survey on stability analysis of discrete-time piecewise affine systems," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 283–294, 2005.

[47] M. Rubagotti, L. Zaccarian, and A. Bemporad, "A lyapunov method for stability analysis of piecewise-affine systems over non-invariant domains," *International Journal of Control*, vol. 89, pp. 1–25, 10 2015.

[48] A. Hassibi and S. Boyd, "Quadratic stabilization and control of piecewise-linear systems," in *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, vol. 6, 1998, pp. 3659–3664 vol.6.

[49] S. Rakovic, E. Kerrigan, D. Mayne, and J. Lygeros, "Reachability analysis of discrete-time systems with disturbances," *IEEE Transactions on Automatic Control*, vol. 51, no. 4, pp. 546–561, 2006.

[50] B. Hanin and D. Rolnick, "Deep relu networks have surprisingly few activation patterns," in *Advances in Neural Information Processing Systems*, 2019, pp. 361–370.

[51] R. H. Wilkinson, "A method of generating functions of several variables using analog diode logic," *IEEE Transactions on Electronic Computers*, vol. EC-12, no. 2, pp. 112–129, 1963.

[52] J. Xu, T. J. van den Boom, and B. De Schutter, "Irredundant lattice piecewise affine representations and their applications in explicit model predictive control," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 4416–4421.

[53] L. Chua and A.-C. Deng, "Canonical piecewise-linear representation," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 1, pp. 101–111, 1988.

[54] T. Kevenaar and D. Leenaerts, "A comparison of piecewise-linear model descriptions," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 39, no. 12, pp. 996–1004, 1992.

[55] R. Suard, J. Lofberg, P. Grieder, M. Kvasnica, and M. Morari, "Efficient computation of controller partitions in multi-parametric programming," in *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, vol. 4. IEEE, 2004, pp. 3643–3648.

[56] K. Fukuda, "Cddlib reference manual," *Report version 093a, McGill University, Montréal, Quebec, Canada*, 2003.

[57] A. Maréchal, D. Monniaux, and M. Périn, "Scalable minimizing-operators on polyhedra via parametric linear programming," in *International Static Analysis Symposium*. Springer, 2017, pp. 212–231.

[58] B. Legat, R. Deits, M. Forets, D. Oyama, S. Timme, F. Pacaud, S. Guadalupe, M. Besançon, J. TagBot, and E. Saba, "Juliapolyhedra/cddlib.jl: v0.6.1," Mar. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3733590

[59] M. Vašak, M. Baotić, and N. Perić, "Efficient computation of the one-step robust sets for piecewise affine systems with polytopic additive uncertainties," in *2007 European Control Conference (ECC)*, 2007, pp. 1430–1435.

[60] J.-C. Hennet, "Discrete time constrained linear systems," *Control and dynamic systems*, vol. 71, pp. 157–214, 1995.

[61] A. Alessio, M. Lazar, A. Bemporad, and W. Heemels, "Squaring the circle: An algorithm for generating polyhedral invariant sets from ellipsoidal ones," *Automatica*, vol. 43, no. 12, pp. 2096–2103, 2007.

[62] D. Y. Rubin, H.-N. Nguyen, and P.-O. Gutman, "Yet another algorithm for the computation of polyhedral positive invariant sets," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*, 2018, pp. 698–703.

[63] T. Fujisawa and E. Kuh, "Piecewise-linear theory of nonlinear networks," *Siam Journal on Applied Mathematics*, vol. 22, pp. 307–328, 1972.

[64] W. C. Rheinboldt and J. S. Vandergraft, "On piecewise affine mappings in r^n," *SIAM Journal on Applied Mathematics*, vol. 29, no. 4, pp. 680–689, 1975.

[65] M. Radons, "A note on surjectivity of piecewise affine mappings," *Optimization Letters*, vol. 13, no. 2, pp. 439–443, 2019.

[66] G. Naitzat, A. Zhitnikov, and L.-H. Lim, "Topology of deep neural networks." *J. Mach. Learn. Res.*, vol. 21, no. 184, pp. 1–40, 2020.

[67] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[68] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 2016, pp. 1–10.

[69] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 443–452.

APPENDIX

In this section we give a proof for Theorem 1. Similar arguments are used to characterize the change in Jacobians of PWA components in [54] and the references therein.

Lemma 3 (Stationary Neurons): If $[\mathbf{a}_{k,ij}^\top \quad -b_{k,ij}] \neq \alpha[\mathbf{a}_\eta^\top \quad -b_\eta]$ for some $\alpha \in \{0,1\}$, then the activation of neuron ij does not change, $\lambda_{k,ij} = \lambda_{k',ij}$.

Proof: First, it cannot be the case that $[\mathbf{a}_{k,ij}^\top \quad -b_{k,ij}] = -[\mathbf{a}_\eta^\top \quad -b_\eta]$ (i.e. $\alpha = -1$) because this would imply that the interior of P_k is of dimension lower than the ambient dimension, which cannot be according to Definition 2. Next, if $[\mathbf{a}_{k,ij}^\top \quad -b_{k,ij}] \neq \alpha[\mathbf{a}_\eta^\top \quad -b_\eta]$ for some $\alpha \in \{0,1\}$, then we know that the constraint $\mathbf{a}_{k,ij}^\top \mathbf{x} \leq b_{k,ij}$ is not active on the boundary between P_k and $P_{k'}$. Formally, $\forall \mathbf{x} \in \operatorname{int}(P_k \cap P_{k'})$

$$\mathbf{a}_{k,ij}^\top(\mathbf{x})\mathbf{x} < b_{k,ij}(\mathbf{x}). \tag{41}$$

The inequality is strict because if the relationship held with equality $\forall \mathbf{x} \in \operatorname{int}(P_k \cap P_{k'})$ then we would have $[\mathbf{a}_{k,ij}^\top \quad -b_{k,ij}] = \alpha[\mathbf{a}_\eta^\top \quad -b_\eta]$ for some $\alpha \in \{0,1\}$, by definition of the neighbor constraint, a contradiction. Furthermore if equality held on a strict subset of $\operatorname{int}(P_k \cap P_{k'})$ then the shared face of P_k and $P_{k'}$ would not be identified by $\mathbf{a}_\eta^\top \mathbf{x} = b_\eta$, but instead by both $\mathbf{a}_\eta^\top \mathbf{x} = b_\eta$ and $\mathbf{a}_{k,ij}^\top \mathbf{x} = b_{k,ij}$, another contradiction. Given the above strict inequality and noting

that $\mathbf{a}_{k,ij}^\top(\mathbf{x})\mathbf{x} - b_{k,ij}(\mathbf{x})$ is a continuous function of $\mathbf{x}$, $\exists \boldsymbol{\delta}$ such that $\mathbf{x}+\boldsymbol{\delta} \in \text{int}(P_{k'})$ and the inequality remains strict,

$$\mathbf{a}_{k,ij}^\top(\mathbf{x}+\boldsymbol{\delta})(\mathbf{x}+\boldsymbol{\delta}) < b_{k,ij}(\mathbf{x}+\boldsymbol{\delta}). \tag{42}$$

Thus, since the constraints associated with such a neuron ij are valid on the interior of both P_k and $P_{k'}$, the neuron activation does not change. ∎

Lemma 4 (Effects of Activation Flips): If $[\mathbf{a}_{k,ij}^\top \quad -b_{k,ij}] = \alpha[\mathbf{a}_\eta^\top \quad -b_\eta]$ for some $\alpha \in \{0,1\}$, then $[\mathbf{a}_{k',ij}^\top \quad -b_{k',ij}] = \alpha'[\mathbf{a}_\eta^\top \quad -b_\eta]$ for some $\alpha' \in \{-1,0\}$.

Proof:

Consider the affine functions defined by neuron ij in regions P_k and $P_{k'}$,

$$\bar{\mathbf{a}}_{k,ij}^\top\mathbf{x} - \bar{b}_{k,ij} \quad \forall \mathbf{x} \in P_k \tag{43a}$$
$$\bar{\mathbf{a}}_{k',ij}^\top\mathbf{x} - \bar{b}_{k',ij} \quad \forall \mathbf{x} \in P_{k'} \tag{43b}$$

(note that the overbar accent indicates the parameters are unnormalized as in (12). Since ReLU networks are continuous functions, $\forall \mathbf{x} \in P_k \cap P_{k'}$ we know

$$\bar{\mathbf{a}}_{k,ij}^\top\mathbf{x} - \bar{b}_{k,ij} = \bar{\mathbf{a}}_{k',ij}^\top\mathbf{x} - \bar{b}_{k',ij} \tag{44a}$$
$$\implies \begin{bmatrix}\bar{\mathbf{a}}_{k',ij}^\top & -\bar{b}_{k',ij}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} = \begin{bmatrix}\bar{\mathbf{a}}_{k,ij}^\top & -\bar{b}_{k,ij}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} \tag{44b}$$
$$\implies \begin{bmatrix}\bar{\mathbf{a}}_{k',ij}^\top & -\bar{b}_{k',ij}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} = \beta\begin{bmatrix}\mathbf{a}_\eta^\top & -b_\eta\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} \tag{44c}$$

for some $\beta \geq 0$. Thus, we know the normalized constraint $\mathbf{a}_{k',ij}^\top\mathbf{x} \leq b_{k',ij}$ must be such that,

$$\begin{bmatrix}\mathbf{a}_{k',ij}^\top & -b_{k',ij}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} = \alpha'\begin{bmatrix}\mathbf{a}_\eta^\top & -b_\eta\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} \tag{45}$$

for some $\alpha' \in \{-1,0,1\}$ and $\forall \mathbf{x} \in P_k \cap P_{k'}$. We next seek to remove this relationship's dependence on $\mathbf{x}$.

First, suppose $b_\eta = 0$, then $P_k \cap P_{k'}$ is a subset of the $n-1$-dimensional linear subspace $\{\mathbf{x} \mid \mathbf{a}_\eta^\top\mathbf{x} = 0\}$. Since $|P_k \cap P_{k'}|_{n-1} \neq 0$, there exist $\mathbf{x}_1, \ldots, \mathbf{x}_{n-1} \in P_k \cap P_{k'}$ which are linearly independent. Then the vectors

$$\begin{bmatrix}\mathbf{x}_1\\1\end{bmatrix}, \ldots, \begin{bmatrix}\mathbf{x}_{n-1}\\1\end{bmatrix} \in \mathbb{R}^{n+1} \tag{46}$$

are also linearly independent. Now assume $\mathbf{x}_{n-1} \neq \mathbf{0}$ and

$$\mathbf{x}_n = \beta\mathbf{x}_{n-1} \tag{47}$$

for some $\beta \neq 1$ such that $\mathbf{x}_n \in P_k \cap P_{k'}$. Then the vectors

$$\begin{bmatrix}\mathbf{x}_1\\1\end{bmatrix}, \ldots, \begin{bmatrix}\mathbf{x}_n\\1\end{bmatrix} \in \mathbb{R}^{n+1} \tag{48}$$

are linearly independent since $\mathbf{x}_n$ is only in the span of $\mathbf{x}_{n-1}$, but with $\beta \neq 1$, $[\mathbf{x}_n^\top \quad 1]^\top$ is not in the span of $[\mathbf{x}_{n-1}^\top \quad 1]^\top$. Now consider the matrix formed by these vectors,

$$\mathbf{X} = \begin{bmatrix}\mathbf{x}_1^\top & 1\\ \ldots & \ldots\\ \mathbf{x}_n^\top & 1\end{bmatrix} \in \mathbb{R}^{n\times n+1}. \tag{49}$$

Since we know $\mathbf{a}_\eta^\top\mathbf{x}_i = 0 \quad \forall i \in 1, \ldots, n$,

$$\mathbf{X}\begin{bmatrix}\mathbf{a}_{k',ij}\\-b_{k',ij}\end{bmatrix} = \alpha'\mathbf{X}\begin{bmatrix}\mathbf{a}_\eta\\-b_\eta\end{bmatrix} \tag{50a}$$
$$\mathbf{X}\begin{bmatrix}\mathbf{a}_{k',ij}\\-b_{k',ij}\end{bmatrix} = \mathbf{0}. \tag{50b}$$

Thus, $[\mathbf{a}_{k',ij}^\top \quad -b_{k',ij}]^\top \in \text{null}(\mathbf{X})$. Furthermore, by the rank-nullity theorem, $\text{nullity}(\mathbf{X}) = 1$ (rank is n by construction and $\mathbf{X} \in \mathbb{R}^{n\times n+1}$). Finally, for some $\alpha' \in \{-1,0,1\}$,

$$[\mathbf{a}_{k',ij}^\top \quad -b_{k',ij}]^\top, [\mathbf{a}_\eta^\top \quad -b_\eta]^\top \in \text{null}(\mathbf{X}) \tag{51a}$$
$$\implies [\mathbf{a}_{k',ij}^\top \quad -b_{k',ij}] = \alpha'[\mathbf{a}_\eta^\top \quad -b_\eta]. \tag{51b}$$

Next, suppose $b_\eta \neq 0$, then $P_k \cap P_{k'}$ is a subset of the $n-1$-dimensional affine span $\{\mathbf{x} \mid \mathbf{a}_\eta^\top\mathbf{x} = b_\eta\}$. Since $|P_k \cap P_{k'}|_{n-1} \neq 0$ there exist $\mathbf{x}_1, \ldots, \mathbf{x}_n \in P_k \cap P_{k'}$ which define the vertex points of an $n-1$-dimensional simplex on $P_k \cap P_{k'}$. The vertices of a simplex are known to be affinely independent. Therefore, by Lemma 5, the vectors

$$\begin{bmatrix}\mathbf{x}_1\\1\end{bmatrix}, \ldots, \begin{bmatrix}\mathbf{x}_n\\1\end{bmatrix} \in \mathbb{R}^{n+1} \tag{52}$$

are linearly independent. Now, in a similar manner to the previous case, one can construct the matrix

$$\mathbf{X} = \begin{bmatrix}\mathbf{x}_1^\top & 1\\ \ldots & \ldots\\ \mathbf{x}_n^\top & 1\end{bmatrix} \in \mathbb{R}^{n\times n+1}. \tag{53}$$

Since we know $\mathbf{a}_\eta^\top\mathbf{x}_i - b_\eta = 0 \quad \forall i \in 1, \ldots, n$,

$$\mathbf{X}\begin{bmatrix}\mathbf{a}_{k',ij}\\-b_{k',ij}\end{bmatrix} = \alpha'\mathbf{X}\begin{bmatrix}\mathbf{a}_\eta\\-b_\eta\end{bmatrix} \tag{54a}$$
$$\mathbf{X}\begin{bmatrix}\mathbf{a}_{k',ij}\\-b_{k',ij}\end{bmatrix} = \mathbf{0}. \tag{54b}$$

As in the previous case, we know that $\text{nullity}(\mathbf{X}) = 1$. Finally, for some $\alpha' \in \{-1,0,1\}$,

$$[\mathbf{a}_{k',ij}^\top \quad -b_{k',ij}]^\top, [\mathbf{a}_\eta^\top \quad -b_\eta]^\top \in \text{null}(\mathbf{X}) \tag{55a}$$
$$\implies [\mathbf{a}_{k',ij}^\top \quad -b_{k',ij}] = \alpha'[\mathbf{a}_\eta^\top \quad -b_\eta]. \tag{55b}$$

Lastly, $\begin{bmatrix}\mathbf{a}_{k',ij}^\top & -b_{k',ij}\end{bmatrix} = \begin{bmatrix}\mathbf{a}_\eta^\top & -b_\eta\end{bmatrix}$ cannot hold as it leaves $P_{k'}$ lower than the ambient dimension, so $\alpha' \neq 1$. Thus, we arrive at the desired result. ∎

Lemma 5 (Affine Independence to Linear Independence): Given affinely independent vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^n$, $[\mathbf{x}_1^\top \quad 1]^\top, \ldots, [\mathbf{x}_k^\top \quad 1]^\top \in \mathbb{R}^{n+1}$ are linearly independent.

Proof: The vectors $[\mathbf{x}_1^\top \quad 1]^\top, \ldots, [\mathbf{x}_k^\top \quad 1]^\top$ are linearly independent if and only if

$$\lambda_1\begin{bmatrix}\mathbf{x}_1\\1\end{bmatrix} + \lambda_2\begin{bmatrix}\mathbf{x}_2\\1\end{bmatrix} + \ldots + \lambda_k\begin{bmatrix}\mathbf{x}_k\\1\end{bmatrix} = \mathbf{0} \tag{56a}$$
$$\implies \lambda_1 = \lambda_2 = \ldots = \lambda_k = 0. \tag{56b}$$

This condition is equivalent to

$$\left.\begin{aligned}\lambda_1\mathbf{x}_1 + \lambda_2\mathbf{x}_2 + \ldots + \lambda_k\mathbf{x}_k = \mathbf{0}\\ \lambda_1 + \lambda_2 + \ldots + \lambda_k = 0\end{aligned}\right\} \tag{57a}$$
$$\implies \lambda_1 = \lambda_2 = \ldots = \lambda_k = 0, \tag{57b}$$

i.e. $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are affinely independent. ∎