2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

Bounding the Complexity of Formally Verifying Neural Networks: A Geometric Approach

James Ferlez* and Yasser Shoukry*

2021 60th IEEE Conference on Decision and Control (CDC) | 978-1-6654-3659-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CDC45484.2021.9683375

***Abstract*— In this paper, we consider the computational complexity of formally verifying the behavior of Rectified Linear Unit (ReLU) Neural Networks (NNs), where verification entails determining whether the NN satisfies convex polytopic specifications. Specifically, we show that for two different NN architectures – shallow NNs and Two-Level Lattice (TLL) NNs – the verification problem with (convex) polytopic constraints is *polynomial* in the number of neurons in the NN to be verified, when all other aspects of the verification problem held fixed. We achieve these complexity results by exhibiting explicit (but similar) verification algorithms for each type of architecture. Both algorithms efficiently translate the NN parameters into a partitioning of the NN's input space by means of *hyperplanes*; this has the effect of partitioning the original verification problem into polynomially many sub-verification problems derived from the geometry of the neurons. We show that these sub-problems may be chosen so that the NN is purely affine within each, and hence each sub-problem is solvable in polynomial time by means of a Linear Program (LP). Thus, a polynomial-time algorithm for the original verification problem can be obtained using known algorithms for enumerating the regions in a hyperplane arrangement. Finally, we adapt our proposed algorithms to the verification of dynamical systems, specifically when these NN architectures are used as state-feedback controllers for LTI systems. We further evaluate the viability of this approach numerically.**

I. Introduction

Neural Networks (NNs) are increasingly used as feedback controllers in safety-critical cyber-physical systems, so algorithms that can *verify* the safety of such controllers are of crucial importance. Despite this, relatively little attention has been paid to an analysis of their computational complexity. Such considerations are crucial in the verification of controllers, since a verifier may be invoked many times to verify a controller in closed loop ([13], [15] for example).

On the one hand, it is known that the satisfiability of any 3-SAT formula can be encoded as a NN verification problem, but this result requires its variables to be in correspondence with the input dimensions to the network [5]. This means the complexity of verifying a NN depends unfavorably on the dimension of its input space. On the other hand, this result doesn't address the relative difficulty of verifying a NN with a *fixed* input dimension but an increasing number of neurons. The only results in this vein exhibit networks for which the number of affine regions grows exponentially in the number of neurons in the network – see e.g. [7]. However, these merely suggest that the verification problem is still "hard" in the number of neurons in the network (input and output dimensions are fixed). *To our knowledge, there are no polynomial complexity results for this second question.*

In this paper, we prove two such concrete complexity results that explicitly describe the computational complexity of verifying a NN as a function of its size. In particular, we prove that the complexity of verifying either a shallow NN or a Two-Level Lattice NN [3] grows only *polynomially* with the number of neurons in the network to be verified, all other aspects of the verification problem held fixed. These results appear in Section III as Theorem 2 and Theorem 3 for shallow NNs and TLL NNs, respectively. Our proofs for both of these complexity results are existential: that is we propose one concrete verification algorithm for each architecture.

By their mere existence, the complexity results we prove herein demonstrate that the NN verification problem is not per se a "hard" problem as a function of the size of the NN to be verified. However, our results cannot contradict the known results in [5], which specify verification complexity in terms of input dimension: indeed, although we show that the complexity of verifying a shallow or a TLL NN scales polynomially with its size, our complexity claims **necessarily** scale exponentially in the input dimension of the NN. One further observation is in order: while our results do speak directly to the complexity of the verification problem as a function of the number of neurons, they *do not* address the complexity of the verification problem in terms of the *expressivity* of a particular network size; see Section III.

Moreover, the nature of our proposed algorithms means *they have direct applicability to verifying such NNs when they are used as feedback controllers.* In particular, they verify a NN by dividing its input space into regions on which the NN is affine; in this context, verifying an input/output property requires one Linear Program (LP) on each such region, but such an LP can easily be extended to verify certain discrete-time dynamical properties for LTI systems. That is our algorithm can verify whether the *next state* resulting from a state-feedback NN controller lies in a particular polytopic set (e.g. forward invariance of a (polytopic) set of states).

We conclude this paper with a set of experimental results that validate the claims we have made about our proposed TLL verifier. First, we show that our implementation does in fact scale polynomially. And second, we show that it can be adapted to verify the forward invariance of a polytopic set of states (on an LTI system with state feedback TLL controller).

Related work: The work on NN verification has generally focused on practical algorithms rather than theoretical complexity results, although many have noticed empirically that there is a significant complexity associated with the input dimension; [5] is a notable exception, since it also included an NP-completeness result based on the 3-SAT encoding mentioned above. Other examples of pragmatic NN verification approaches include: (i) SMT-based methods; (ii) MILP-based solvers; (iii) Reachability based methods; and (iv) convex relaxations methods. A good survey of these methods an be found in [6]. By contrast, a number of works have focused on the computational complexity of various other verification-related questions for NNs ([5] is

*Department of Electrical Engineering and Computer Science, University of California, Irvine {jferlez,yshoukry}@uci.edu

This work was partially sponsored by the NSF awards #CNS-2002405 and #CNS-2013824.

the exception in that it expressly considers the verification problem itself). Some NN-related complexity results include: computing the minimum adversarial disturbance to a NN is NP hard [16]; computing the Lipschitz constant of a NN is NP hard [14]; reachability analysis is NP hard [9], [12].

II. Preliminaries

A. Notation

We will denote the real numbers by $\mathbb{R}$. For an $(n \times m)$ matrix (or vector), A, we will use the notation $[\![A]\!]_{[i,j]}$ to denote the element in the i^{th} row and j^{th} column of A. Analogously, the notation $[\![A]\!]_{[i,:]}$ will denote the i^{th} row of A, and $[\![A]\!]_{[:,j]}$ will denote the j^{th} column of A; when A is a vector instead of a matrix, both notations will return a scalar corresponding to the corresponding element in the vector. We will use bold parenthesis $\mathbf{(}\cdot\mathbf{)}$ to delineate the arguments to a function that *returns a function*. We will use two special forms of this notation: given an $(m \times n)$ matrix, W, and an $(m \times 1)$ dimensional vector, b, define

$$\mathscr{L}\mathbf{(}W,b\mathbf{)} : x \mapsto Wx + b \tag{1}$$
$$\mathscr{L}_i\mathbf{(}W,b\mathbf{)} : x \mapsto [\![W]\!]_{[i,:]}x + [\![b]\!]_i. \tag{2}$$

We also use the functions `First` and `Last` to return the first and last elements of an ordered list (or by overloading, a vector in $\mathbb{R}^n$). The function `Concat` concatenates two ordered lists, or by overloading, concatenates two vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$ along their (common) nontrivial dimension to get a third vector in $\mathbb{R}^{n+m}$. Finally, an over-bar to indicate (topological) closure of a set: i.e. $\overline{A}$ is the closure of A.

B. Neural Networks

We will exclusively consider Rectified Linear Unit Neural Networks (ReLU NNs). A K-layer ReLU NN is specified by K *layer* functions, $\{L_{\theta^{|i}} : i = 1, \ldots, K\}$, of which we allow two kinds: linear and nonlinear. A *nonlinear* layer, i, with $\mathfrak{i}^{|i}$ inputs and $\mathfrak{o}^{|i}$ outputs is specified by a $(\mathfrak{o}^{|i} \times \mathfrak{i}^{|i})$ matrix of *weights*, $W^{|i}$, and a $(\mathfrak{o}^{|i} \times 1)$ matrix of *biases*, $b^{|i}$:

$$L_{\theta^{|i}} : \mathbb{R}^{\mathfrak{i}} \to \mathbb{R}^{\mathfrak{o}}, \qquad L_{\theta^{|i}} : z \mapsto \max\{Wz + b, 0\} \tag{3}$$

where the max function is taken element-wise, and $\theta^{|i} \triangleq (W^{|i}, b^{|i})$. A *linear* layer is the same as a nonlinear layer, except it omits the nonlinearity $\max\{\cdot, 0\}$ in its layer function; a linear layer will be indicated with a superscript *lin* e.g. $L^{\text{lin}}_{\theta^{|i}}$ Thus, a K-layer ReLU NN function is specified by functionally composing K such layer functions whose input and output dimensions satisfy $\mathfrak{i}_i = \mathfrak{o}_{i-1} : i = 2, \ldots, K$. **We further adopt the convention that the final layer is always a linear layer**, so we may define:

$$\mathscr{NN} = L^{\text{lin}}_{\theta^{|K}} \circ L_{\theta^{|K-1}} \circ \cdots \circ L_{\theta^{|1}} \tag{4}$$

To make the dependence on parameters explicit, we will index a ReLU function $\mathscr{NN}$ by a *list of matrices* $\Theta \triangleq (\theta^{|1}, \ldots, \theta^{|K})$;[1] in this respect, we will often use $\mathscr{NN} = \mathscr{NN}\mathbf{(}\Theta\mathbf{)}$.

The number of layers and the *dimensions* of the associated matrices $\theta^{|i} = (\, W^{|i}, b^{|i} \,)$ specifies the *architecture* of the ReLU NN. Therefore, we will use:

$$\text{Arch}(\Theta) \triangleq ((n, \mathfrak{o}^{|1}), (\mathfrak{i}^{|2}, \mathfrak{o}^{|2}), \ldots, (\mathfrak{i}^{|K}, m)) \tag{5}$$

[1]That is Θ is not the concatenation of the $\theta^{(i)}$ into a single large matrix, so it preserves information about the sizes of the constituent $\theta^{(i)}$.

to denote the architecture of the ReLU NN $\mathscr{NN}\mathbf{(}\Theta\mathbf{)}$.

Definition 1 (Shallow NN)**.** *A **shallow NN** is two-layer NN whose first layer is nonlinear and whose second is linear.*

C. Special NN Operations

The operations in this section will be used to define a Two-Layer Lattice network in Section II-D.

Definition 2 (Sequential (Functional) Composition)**.** *Let $\mathscr{NN}\mathbf{(}\Theta_1\mathbf{)}$ and $\mathscr{NN}\mathbf{(}\Theta_2\mathbf{)}$ be two NNs where $\texttt{Last}(Arch(\Theta_1)) = (\mathfrak{i}, \mathfrak{c})$ and $\texttt{First}(Arch(\Theta_2)) = (\mathfrak{c}, \mathfrak{o})$ for some nonnegative integers $\mathfrak{i}$, $\mathfrak{o}$ and $\mathfrak{c}$. Then the **sequential (or functional) composition** of $\mathscr{NN}\mathbf{(}\Theta_1\mathbf{)}$ and $\mathscr{NN}\mathbf{(}\Theta_2\mathbf{)}$, i.e. $\mathscr{NN}\mathbf{(}\Theta_1\mathbf{)} \circ \mathscr{NN}\mathbf{(}\Theta_2\mathbf{)}$, is a well defined NN, and can be represented by the parameter list $\Theta_1 \circ \Theta_2 \triangleq \texttt{Concat}(\Theta_1, \Theta_2)$.*

Definition 3. *Let $\mathscr{NN}\mathbf{(}\Theta_1\mathbf{)}$ and $\mathscr{NN}\mathbf{(}\Theta_2\mathbf{)}$ be two K-layer NNs with parameter lists:*

$$\Theta_i = ((W_i^{|1}, b_i^{|1}), \ldots, (W_i^{|K}, b_i^{|K})), \quad i = 1, 2. \tag{6}$$

*Then the **parallel composition** of $\mathscr{NN}\mathbf{(}\Theta_1\mathbf{)}$ and $\mathscr{NN}\mathbf{(}\Theta_2\mathbf{)}$ is a NN given by the parameter list*

$$\Theta_1 \parallel \Theta_2 \triangleq \left(\left(\begin{bmatrix} W_1^{|1} \\ W_2^{|1} \end{bmatrix}, \begin{bmatrix} b_1^{|1} \\ b_2^{|1} \end{bmatrix}\right), \ldots, \left(\begin{bmatrix} W_1^{|K} \\ W_2^{|K} \end{bmatrix}, \begin{bmatrix} b_1^{|K} \\ b_2^{|K} \end{bmatrix}\right)\right). \tag{7}$$

That is $\Theta_1 \| \Theta_2$ accepts an input of the same size as (both) Θ_1 and Θ_2, but has as many outputs as Θ_1 and Θ_2 combined.

Definition 4 (n-element min/max NNs)**.** *An **n-element min network** is denoted by the parameter list $\Theta_{\min_n}$. $\mathscr{NN}\mathbf{(}\Theta_{\min_n}\mathbf{)} : \mathbb{R}^n \to \mathbb{R}$ such that $\mathscr{NN}\mathbf{(}\Theta_{\min_n}\mathbf{)}(x)$ is the minimum from among the components of x (i.e. minimum according to the usual order relation $<$ on $\mathbb{R}$). An **n-element max network** is denoted by $\Theta_{\max_n}$, and functions analogously. These networks are described in [3].*

D. Two-Level-Lattice (TLL) Neural Networks

In this paper, we will be especially concerned with ReLU NNs that have the Two-Level Lattice (TLL) architecture, as introduced with the AReN algorithm in [3]. Here we describe both scalar output TLL NNs and multi-output TLL NNs.

1) Scalar TLL NNs: From [3], a scalar-output TLL NN can be described as follows.

Definition 5 (Scalar TLL NN [3, Theorem 2])**.** *A NN that maps $\mathbb{R}^n \to \mathbb{R}$ is said to be **TLL NN of size** (N, M) if the size of its parameter list $\Xi_{N,M}$ can be characterized entirely by integers N and M as follows.*

$$\Xi_{N,M} \triangleq \Theta_{\max_M} \circ \left((\Theta_{\min_N} \circ \Theta_{S_1}) \parallel \ldots \parallel (\Theta_{\min_N} \circ \Theta_{S_M}) \right) \circ \Theta_\ell \tag{8}$$

where

- $\Theta_\ell \triangleq ((W_\ell, b_\ell))$;
- *each Θ_{S_j} has the form $\Theta_{S_j} = \left(S_j, \mathbf{0}_{N,1}\right)$; and*
- $S_j = \left[\, [\![I_N]\!]_{[\iota_1,:]}^T \cdots [\![I_N]\!]_{[\iota_N,:]}^T \right]^T$ *for some sequence $\{\iota_k\} \subseteq \{1, \ldots, N\}$ (I_N is the $(N \times N)$ identity matrix).*

*The linear functions implemented by the mapping $\mathscr{L}_i\mathbf{(}W_\ell, b_\ell\mathbf{)}$ for $i = 1, \ldots, N$ will be referred to as the **local linear functions** of $\Xi_{N,M}$; we assume for simplicity that these linear functions are unique. The matrices $\{S_j | j = 1, \ldots, M\}$ will be referred to as the **selector matrices** of $\Xi_{N,M}$. Each set $s_j \triangleq \{k \in \{1, \ldots, N\} | \exists \iota \in \{1, \ldots, N\}. [\![S_j]\!]_{\iota,k} = 1\}$ is said to be the selector set of S_j.*

2) Multi-output TLL NNs: We will define a multi-output TLL NN with range space $\mathbb{R}^m$ using m equally sized scalar TLL NNs. That is we denote such a network by $\Xi^{(m)}_{N,M}$, with each output-components denoted by $\Xi^k_{N,M}$, $k = 1, \ldots, m$.

E. Hyperplanes and Hyperplane Arrangements

Here we review notation for hyperplanes and hyperplane arrangements; [10] is the main reference for this section.

Definition 6 (Hyperplanes and Half-spaces)**.** *Let $\ell : \mathbb{R}^n \to \mathbb{R}$ be an affine map. Then define:*

$$H^q_\ell \triangleq \begin{cases} \{x | \ell(x) < 0\} & q = -1 \\ \{x | \ell(x) > 0\} & q = +1 \\ \{x | \ell(x) = 0\} & q = 0. \end{cases} \tag{9}$$

*We say that H^0_ℓ is the **hyperplane defined by** ℓ **in dimension** n, and H^{-1}_ℓ and H^{+1}_ℓ are the **negative and positive half-spaces defined by** ℓ, respectively.*

Definition 7 (Hyperplane Arrangement)**.** *Let $\mathcal{L}$ be a set of affine functions where each $\ell \in \mathcal{L} : \mathbb{R}^n \to \mathbb{R}$. Then $\{H^0_\ell | \ell \in \mathcal{L}\}$ is an **arrangement of hyperplanes in dimension** n.*

Definition 8 (Region of a Hyperplane Arrangement)**.** *Let $\mathcal{H}$ be an arrangement of N hyperplanes in dimension n defined by a set of affine functions, $\mathcal{L}$. Then a non-empty open subset $R \subseteq \mathbb{R}^n$ is said to be a (n**-dimensional) region of** $\mathcal{H}$ if there is an indexing function $\mathfrak{s} : \mathcal{L} \to \{-1, +1\}$ such that $R = \bigcap_{\ell \in \mathcal{L}} H^{\mathfrak{s}(\ell)}_\ell$ and $B(x, \delta) \subset R$ for some x and δ. The set of all regions of arrangement $\mathcal{H}$ will be denoted $\mathcal{R}_\mathcal{H}$.*

Theorem 1 ([10])**.** *Let $\mathcal{H}$ be an arrangement of N hyperplanes in dimension n. Then $|\mathcal{R}_\mathcal{H}|$ is at most $\sum_{k=0}^{n} \binom{N}{k}$.*

Remark 1. *Note that for a fixed dimension, n, the bound $\sum_{k=0}^{n} \binom{N}{k}$ grows like $O(N^n/n!)$, i.e. sub-exponentially in N.*

III. Main Results

A. NN Verification Problem

We will consider the following NN verification problem.

Problem 1. *Let $\mathscr{N}\!\!\left[\Theta\right] : \mathbb{R}^n \to \mathbb{R}^m$ be a NN with at least two layers. Furthermore, assume that there are two convex, bounded, full-dimensional polytopes $P_x \subset \mathbb{R}^n$ and $P_y \subset \mathbb{R}^m$ represented as follows:*

- $P_x \triangleq \cap_{i=1}^{\mathsf{N}_x} \overline{H^{-1}_{\ell_{x,i}}} \subset \mathbb{R}^n$ *where $\ell_{x,i} : \mathbb{R}^n \to \mathbb{R}$ is an affine map for each $i = 1, \ldots, \mathsf{N}_x$; and*
- $P_y \triangleq \cap_{i=1}^{\mathsf{N}_y} \overline{H^{-1}_{\ell_{y,i}}} \subset \mathbb{R}^m$ *where $\ell_{y,i} : \mathbb{R}^m \to \mathbb{R}$ is an affine map for each $i = 1, \ldots, \mathsf{N}_y$.*

Then the verification problem is to decide whether the following formula is true:

$$\forall x \in P_x \subset \mathbb{R}^n . \big(\mathscr{N}\!\!\left[\Theta\right](x) \in P_y \subset \mathbb{R}^m\big). \tag{10}$$

*If (10) is true, the problem is **SAT**; otherwise,it is **UNSAT**.*

We proceed with this formulation of Problem 1 for simplicity, and to emphasize the verification complexity in terms of NN parameters. **Even so, our proposed algorithm evaluates NNs on regions where they are affine**, and on such regions, verifying the input/output property in (10) is essentially the same as verifying a control-relevant property such as

$$\mathfrak{L}(x, \mathscr{N}\!\!\left[\Theta\right](x)) \in P_y \tag{11}$$

(for a linear function $\mathfrak{L}$). Examples of (11) appear in forward invariance verification of LTI systems (see Section VI-C) and verifying autonomous robots controlled by NNs [11].

B. Main Theorems

The main results of this paper consist of showing that Problem 1 can be solved in polynomial time complexity in the number of neurons for two classes of networks. In particular, we state the following two theorems.

Theorem 2. *Let Arch$(\Theta) = ((n, \mathfrak{n}), (\mathfrak{n}, m))$ define a shallow network with $\mathfrak{n}$ neurons. Now consider an instance of Problem 1 for $\mathscr{N}\!\!\left[\Theta\right]$: i.e. fixed dimensions n and m, and fixed constraint sets P_x and P_y. Then there is an algorithm that solves this instance of Problem 1 in polynomial time complexity in $\mathfrak{n}$. This algorithm has a worst case runtime of*

$$\mathsf{N}_y \cdot O(m \cdot n^2 \cdot \mathfrak{n}^{n+2}/n!) \cdot \texttt{Cplxty}(\texttt{LP}(\mathfrak{n} + \mathsf{N}_x, n)) \tag{12}$$

where $\texttt{Cplxty}(\texttt{LP}(N, n))$ is the complexity of solving a linear program in dimension n with N constraints.

Theorem 3. *Let $\Xi^{(m)}_{N,M}$ define a multi-output TLL network. Now consider an instance of Problem 1 for $\mathscr{N}\!\!\left[\Xi^{(m)}_{N,M}\right]$: i.e. fixed dimensions n and m, and fixed constraint sets P_x and P_y. Then there is an algorithm that solves this instance of Problem 1 in polynomial time complexity in N and M. This algorithm has a worst case runtime of*

$$\mathsf{N}_y \cdot O(m^{n+2} \cdot n \cdot M \cdot N^{2n+3}/n!) \cdot \texttt{Cplxty}(\texttt{LP}(m \cdot N^2 + \mathsf{N}_x, n))$$

where $\texttt{Cplxty}(\texttt{LP}(N, n))$ is the complexity of solving a linear program in dimension n with N constraints. The algorithm is polynomial in the number of neurons in $\mathscr{N}\!\!\left[\Xi^{(m)}_{N,M}\right]$, since the number of neurons depends polynomially on N and M.

In particular, Theorem 2 and Theorem 3 explicitly indicate that the difficulty in verifying their respective classes of NNs grows only polynomially in the complexity of the network, all other parameters of Problem 1 held fixed. Note also that the polynomial complexity of these algorithms depends on the existence of polynomial-time solvers for linear programs, but such solvers are well known to exist (see e.g. [8]).

Note that Theorem 2 and Theorem 3 do not contradict the 3-SAT embedding of [5], since both algorithms are exponential in the input dimension to the network. Indeed, given that a TLL NN can represent any Continuous, Piecewise Affine (CPWA) function [3] – including the 3-SAT gadgets used in [7] – it follows directly that the satisfiability of any 3-SAT formula can be encoded as an instance of Problem 1 for a TLL NN. Since the input dimensions of both NNs are the same, the conclusion of [5] is preserved.

Finally, it is essential to note that the results in Theorem 2 and Theorem 3 connect the difficulty of verifying a TLL NN (resp. shallow NN) to the *size* of the network not the *expressivity* of the network. The semantics of the TLL NN in particular make this point especially salient, since each distinct affine function represented in the output of a TLL NN can be mapped directly to parameters of the TLL NN itself (see Proposition 3 in Section V). In particular, consider the deep NNs exhibited in [7, Corollary 6]: this parameterized collection of networks expresses a number of unique affine functions that grows exponentially in the number of neurons in the network (i.e. as a function of the number of layers in the network). Consequently, the

size of a TLL required to implement one such network would likewise grow exponentially in the number of neurons deployed in the original network. Thus, although a TLL NN may superficially seem "easy" to verify because of Theorem 3, the efficiency in verifying a TLL NN form could mask the fact that a particular TLL NN implementation is less parameter efficient than some other representation (in terms of neurons required). Ultimately, this trade-off will not necessarily be universal, though, since TLL NNs also have mechanisms for parametric *efficiency*: for example, a particular local linear function need only be implemented once in a TLL NN, no matter how many disjoint regions on which it is activated (as in the case of implementing interpolated zero-order-hold functions, such as in [4]).

C. Proof Sketch of Main Theorems

1) Core Theorem: Polynomial-time Enumeration of Hyperplane Regions: The algorithms that witness the claims in Theorem 2 and 3 have the same broad structure:

Step 1: For the architecture in question, choose (in polynomial time) a hyperplane arrangement with the following two properties:
- *(a)* The number of hyperplanes is polynomial in the number of network neurons;
- *(b)* Problem 1 can be solved in polynomial time on the closure of any region in this arrangement intersected with P_x; i.e. Problem 1 with P_x replaced by $\overline{R} \cap P_x$ can be solved in polynomial time.

Step 2: Iterate over all of the regions in this arrangement, and for each region, solve Problem 1 with P_x replaced by $\overline{R} \cap P_x$.

The details of *Step 1* vary depending on the architecture of the network being verified.However, no matter the details of *Step 1*, this proof structure depends on a polynomial time algorithm to traverse the regions in a hyperplane arrangement. But it is known that there exist polynomial algorithms to perform such enumerations. The following result from [1] is one example; an "optimal", if not practical, option is [2].

Theorem 4 ([1] Theorem 3.3)**.** *Let $\mathcal{L} = \{\ell_1, \ldots, \ell_N\}$ be a set of affine functions, $\ell_i : \mathbb{R}^n \to \mathbb{R}$, that can be accessed in $O(1)$ time, and let $\mathcal{HL} = \{H^0_\ell | \ell \in \mathcal{L}\}$ be the associated hyperplane arrangement.*

Then there is an algorithm to traverse all of the regions in $\mathcal{HL}$ that has runtime

$$O(n \cdot N^{n+1}/n!) \cdot \texttt{Cplxty}(\texttt{LP}(N, n)) \tag{13}$$

where $\texttt{Cplxty}(\texttt{LP}(N, n))$ is the complexity of solving a linear program in dimension n with N constraints.

Note that there is more to Theorem 4 than just the sub-exponential bound on the number of regions in a hyperplane arrangement (see Theorem 1 in Section II). Indeed, although there are only $O(N^n/n!)$ regions in an arrangement of N hyperplanes in dimension n, it must be inferred which of the 2^N possible activations correspond to *valid* regions. That this is possible in polynomial time is the main contribution of Theorem 4, and thus facilitates the results in this paper.

2) Theorem 2 and Theorem 3: Given Theorem 4, the proofs of Theorem 2 and Theorem 3 depend on finding a suitable hyperplane arrangement, as described in *Step 1*.

In both cases, we note that the easiest closed convex polytope on which to solve Problem 1 is one on which the underlying NN is affine. Indeed, suppose for the moment that $\mathscr{NN}(\Theta)$ is affine on the entire constraint set P_x with $\mathscr{NN}(\Theta) = \ell_0$ on this domain. Under this assumption, solving the verification problem for a single output constraint, $\ell_{y,i}$, entails solving the following linear program:

$$\begin{aligned} y_i = \max &(\ell_{y,i} \circ \ell_0)(x) \\ \text{s.t. } &\ell_{x,i'}(x) \leq 0 \text{ for } i' = 1, \ldots, \mathsf{N}_x. \end{aligned} \tag{14}$$

Of course if $y_i > 0$, then Problem 1 is UNSAT under these assumptions; otherwise it is SAT for the constraint $\ell_{y,i}$ and the next output constraint needs to be considered. Given the known (polynomial) efficiency of solving linear programs, it thus makes sense to select a hyperplane arrangement for *Step 1* with the property that the NN is affine on each region of the arrangement. Although this is a difficult problem for a general NN, the particular structure of shallow NNs and TLL NNs allow such a selection to be accomplished efficiently.

To this end, we make the following definition.

Definition 9 (Switching Affine Function/Hyperplane)**.** *Let $\mathscr{NN}(\Theta) : \mathbb{R}^n \to \mathbb{R}^m$ be a NN. A set of affine functions $\mathcal{S} = \{\ell_1, \ldots, \ell_N\}$ with $\ell_\iota : \mathbb{R}^n \to \mathbb{R}$ is said to be a set of **switching affine functions for** $\mathscr{NN}(\Theta)$ if $\mathscr{NN}(\Theta)$ is affine on every region of the hyperplane arrangement $\mathcal{HS} = \{H^0_\ell | \ell \in \mathcal{S}\}$. $\mathcal{HS}$ is then said to the be an **arrangement of switching hyperplanes** of $\mathscr{NN}(\Theta)$.*

For both shallow NNs and TLL NNs, we will show that a set of switching hyperplanes is immediately evident (i.e. in polynomial complexity) from the parameters of those architectures directly; this satisfies *Step 1(b)*. However it also further implies that this choice of switching hyperplanes has a *number* of hyperplanes that is polynomial in the number of neurons in either network; this satisfies *Step 1(a)*.

IV. Polynomial-time Algorithm to Verify Shallow NNs

This section consists of several propositions that address the various aspects of *Step 1*, as described in Subsection III-C.1. Theorem 2 is a direct consequence of these propositions.

Proposition 1. *Let $\Theta = ((W^{|1}, b^{|1}), (W^{|2}, b^{|2}))$ define a shallow NN with Arch$(\Theta) = ((n, \mathfrak{n}), (\mathfrak{n}, m))$. Then the set of affine functions*

$$\mathcal{S}(\Theta) \triangleq \{\mathscr{L}_i(W^{|1}, b^{|1}), i = 1, \ldots, \mathfrak{n}\} \tag{15}$$

is a set of switching affine functions for $\mathscr{NN}(\Theta)$, and $\mathcal{HS}(\Theta) = \{H^0_\ell | \ell \in \mathcal{S}(\Theta)\}$ is a set of switching hyperplanes.

Proof. A region in the arrangement $\mathcal{HS}(\Theta)$ exactly assigns to each neuron a status of strictly active or strictly inactive. But forcing a particular activation on *each* of the neurons forces the shallow NN to operate on an affine region. □

Proposition 2. *Let $\Theta = ((W^{|1}, b^{|1}), (W^{|2}, b^{|2}))$ define a shallow NN with Arch$(\Theta) = ((n, \mathfrak{n}), (\mathfrak{n}, m))$, and let $\mathcal{HS}(\Theta)$ be as in Proposition 1. Then the complexity of determining the active linear function of $\mathscr{NN}(\Theta)$ on a region of $\mathcal{HS}(\Theta)$ is at most*

$$O(m \cdot \mathfrak{n} \cdot n). \tag{16}$$

Proof. This runtime is clearly dominated by the cost of doing the matrix multiplication $W^{|2} \cdot W^{|1}$. Given that Arch$(\Theta) = ((n, \mathfrak{n}), (\mathfrak{n}, m))$, this operation has the claimed runtime. □

Theorem 2 now follows.

Proof. (Theorem 2.) First note that $|\mathcal{HS}(\Theta)| = \mathfrak{n}$. Thus, by Proposition 1, the closure of regions from $\mathcal{HS}(\Theta)$ covers $\mathbb{R}^n$, and $\mathscr{N}(\Theta)$ is an affine function on each such closure.

Thus, an algorithm to solve Problem 1 for $\mathscr{N}(\Theta)$ can be obtained by enumerating the regions in $\mathcal{HS}(\Theta)$ using the algorithm from Theorem 4, and for each such region, $\mathfrak{s}$ (see Definition 8), solving one linear program:

$$\begin{aligned} y_i = \max &(\ell_{y,i} \circ \ell_0)(x) \\ \text{s.t. } &\ell_{x,i'}(x) \leq 0 \text{ for } i' = 1, \ldots, \mathsf{N}_x \\ \text{and } &\mathfrak{s}(\ell) \cdot \ell(x) \leq 0 \text{ for } \ell \in \mathcal{S}(\Theta). \end{aligned} \tag{17}$$

for each output polytope constraint $i \in 1, \ldots, \mathsf{N}_y$. The claimed runtime then follows directly by incorporating the cost of computing the active affine function on each such region (Proposition 2) and bounding the size of each enumeration LP (Theorem 4) by the size of the LP above, which has at most $\mathfrak{n}+\mathsf{N}_x$ constraints in dimension n. □

V. Polynomial-time Algorithm to Verify TLL NN

This section consists of several propositions that address the various aspects of *Step 1*, as described in Subsection III-C.1. Theorem 3 is a direct consequence of these propositions.

Proposition 3. *Let $\Xi^{(m)}_{N,M}$ define a TLL NN. Then define*

$$\mathcal{S}(\Xi^{\kappa}_{N,M}) \triangleq \{\mathscr{L}_i(W^{\kappa}_{\ell}, b^{\kappa}_{\ell}) - \mathscr{L}_j(W^{\kappa}_{\ell}, b^{\kappa}_{\ell}) \,|\, i < j \in \{1, .., N\}\} \tag{18}$$

and $\mathcal{HS}(\Xi^{\kappa}_{N,M}) \triangleq \{H^0_{\ell} \,|\, \ell \in \mathcal{S}(\Xi^{\kappa}_{N,M})\}$. Furthermore, define $\mathcal{S}(\Xi^{(m)}_{N,M}) \triangleq \cup_{\kappa=1}^{m} \mathcal{S}(\Xi^{\kappa}_{N,M})$ and $\mathcal{HS}(\Xi^{(m)}_{N,M}) \triangleq \cup_{\kappa=1}^{m} \mathcal{HS}(\Xi^{\kappa}_{N,M})$.

Then $\mathcal{S}(\Xi^{(m)}_{N,M})$ is a set of switching affine functions for $\Xi^{(m)}_{N,M}$, and the κ^{th} component of $\Xi^{(m)}_{N,M}$ is an affine function on each region of $\mathcal{HS}(\Xi^{(m)}_{N,M})$ and is exactly equal to $\mathscr{L}_i(W^{\kappa}_{\ell}, b^{\kappa}_{\ell})$ for some i.

Proof. Let R be a region in $\mathcal{HS}(\Xi^{(m)}_{N,M})$. Each such region is contained in exactly one region from each of the arrangements $\mathcal{HS}(\Xi^{\kappa}_{N,M})$, so it suffices to show that each component TLL is linear on the regions of its own arrangement.

Thus, let R_κ be a region in $\mathcal{HS}(\Xi^{\kappa}_{N,M})$. We claim that $\mathscr{N}(\Xi^{\kappa}_{N,M})$ is linear on R_κ. To see this, note by definition of a region, there is an indexing function $\mathfrak{s} : \mathcal{S}(\Xi^{\kappa}_{N,M}) \to \{-1, +1\}$ such that $R_k = \bigcap_{\ell \in \mathcal{S}(\Xi^{\kappa}_{N,M})} H^{\mathfrak{s}(\ell)}_{\ell}$. Thus, R_k is a unique order region by construction: each such half-space identically orders the outputs of two linear functions, and since R_k is n-dimensional it is contained in just such a half space for each and every possible pair. Applying the unique ordering property of R_κ to the definition of the TLL NN implies that there exists an index $\iota \in \{1, \ldots, N\}$ such that $\mathscr{N}(\Xi^{\kappa}_{N,M})(x) = [\![W^{\kappa}_{\ell} x + b^{\kappa}_{\ell}]\!]_{\iota}$ for all $x \in R_\kappa$. □

Proposition 4. *Let $\Xi^{(m)}_{N,M}$ define a multi-output TLL NN, and let $\mathcal{HS}(\Xi^{(m)}_{N,M})$ be as in Proposition 3. Then for any region R of $\mathcal{HS}(\Xi^{(m)}_{N,M})$ the affine function of $\mathscr{N}(\Xi^{(m)}_{N,M})$ that is active on R can be found by a polynomial algorithm of runtime*

$$O(m \cdot M \cdot (N+1)). \tag{19}$$

Proof. From the proof of Proposition 3 we know that a region in $\mathcal{HS}(\Xi^{(m)}_{N,M})$ is a unique order region for the linear functions $(W^{\kappa}_{\ell}, b^{\kappa}_{\ell})$ of each component. In other words, the indexing function of the region R specifies a strict ordering of each pair of linear functions from each component of $\mathscr{N}(\Xi^{(m)}_{N,M})$, and hence the component-wise local linear functions of $\mathscr{N}(\Xi^{(m)}_{N,M})$ are *pairwise-ordered* on R.

These pairwise comparisons can be used to identify the active affine function on each $\min$ group, $\Theta_{\min_N}$ (of each output component) by means of successive comparison in a bubble-sort-type way. Thus, resolving the active function on each $\min$ group requires N comparisons, each of which is a direct look up in the region indexing function, and hence $O(1)$. Moreover, the same argument applies to the $\max$ operation for each component, only resolving the active affine function there requires M comparisons instead. Since there are $m \cdot M$ $\min$ groups in total, and there are m $\max$ groups in total, resolving the active affine function runs in $O(m \cdot M \cdot N + m \cdot M) = O(m \cdot M \cdot (N+1))$ as claimed. □

Theorem 3 now follows from these propositions.

Proof. (Theorem 3.) This proof follows exactly the same structure as the proof of Theorem 2. The salient differences are that $|\mathcal{S}(\Xi^{(m)}_{N,M})| = m \cdot N \cdot (N-1)/2 = O(m \cdot N^2)$ (Proposition 3), and the cost of obtaining the active affine function on each region thereof is now specified as $O(m \cdot M \cdot (N+1))$ (Proposition 4). The claimed runtime for TLL networks follows mutatis mutandis from Theorem 4. □

VI. Numerical Results

To validate the claims we have made about the polynomial efficiency of the TLL verification problem, we implemented a version of the algorithm described in Section V. Then we conducted three separate experiments on a selection of randomly generated TLL networks of various sizes.

- *A.* We used our tool to merely enumerate the regions in $\mathcal{HS}(\Xi^{(1)}_{N,M})$ (see Proposition 3); this verifies that our implemented hyperplane-region enumeration algorithm is in fact polynomial, and confirms Theorem 1.
- *B.* For each TLL NN, we randomly generated a polytope in its domain to serve as an input constraint for an instance of Problem 1. We verified each network/input constraint with a single, randomly generated output constraint.
- *C.* Finally, we randomly generated an LTI system, and used our tool to check whether the same polytope associated with each TLL network was forward invariant when said network was used as a state feedback controller.

These experiments were conducted on randomly generated TLL networks with $n = 2$ and $m = 1$ for sizes $N = 8, 16, 24, 32, 40, 48, 56,$ and 64, with $M = N$ for each network. We generated 20 instances of each size. A 3D plot of one such network is depicted in Figure 1 *(d)*.

We implemented a polynomial-time enumerator for the regions in a hyperplane arrangement that was further able to evaluate the verification LP on each such region. We used Python and a parallelism abstraction library, charm4py; all experiments were conducted on a system with a total of 24 Intel E5-2650 v4 2.20GHz cores (48 virtual cores) of which our tool was allocated 24. The system had 256 GB of RAM.

A. Region Enumeration

Figure 1 *(a)* shows the number of regions our tool found in $\mathcal{HS}(\Xi^{(1)}_{N,M})$ for each TLL network, as well as the execution time required. For reference, the maximum number

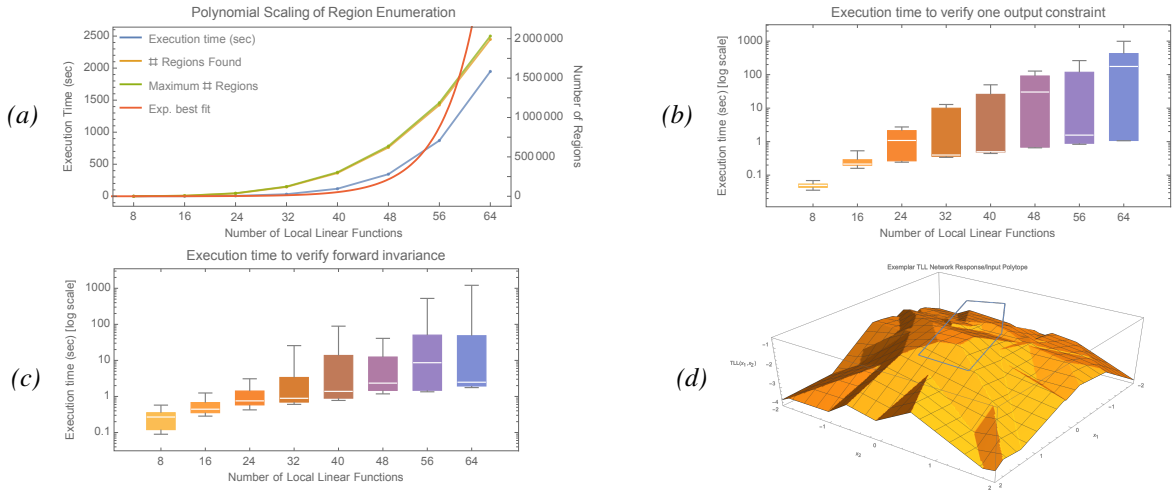

Fig. 1: *(a)* Polynomial growth in number of regions and time needed to enumerate them (red curve is an exponential best-fit to execution time for reference); *(b)* Box plot of execution times to verify a single random output constraint for each of several TLL sizes; *(c)* Box plot of execution time to verify forward invariance of a polytopic set for each of several size TLLs; *(d)* a TLL network and input constraint polytope used in output-constraint/forward-invariance experiments ($N = 64$).

of regions possible is shown for each size, as determined by Theorem 1; note that $\mathcal{HS}(\Xi^{(1)}_{N,M})$ has degeneracy in it, because its hyperplanes are differences between a common set of affine functions. Hence, $\mathcal{HS}(\Xi^{(1)}_{N,M})$ has slightly fewer regions than the theoretical maximum that would be expected for a random arrangement. Also shown is an exponential best-fit to the execution time of our tool (red curve): it attests that our tool has a polynomial region enumeration implementation (compare to the number of regions enumerated).

B. Output Constraint Verification

For each example TLL network, we randomly generated an input constraint polytope and an output constraint to verify. Since $m = 1$, an output constraint amounts to a random threshold, combined with a random choice of $\leq$ or $\geq$ to specify the constraint. A box-and-whisker plot of our tool's execution time on these verification problems is shown in Figure 1 *(b)*. The variability in execution time is due to the fact that our tool terminates early when a region of $\mathcal{HS}(\Xi^{(1)}_{N,M})$ is found to generate a violation of the constraint.

C. LTI System Forward Invariance Verification

For each of the TLL networks, we randomly generated LTI system matrices of the appropriate dimension for the TLL network to serve as a state-feedback controller. Then we used our tool to verify that each input polytope P_x satisfies

$$\forall x \in P_x \;.\; \left(Ax + B\mathcal{M}(\Xi^{(1)}_{N,M})(x)\right) \in P_x. \qquad (20)$$

That is P_x is forward invariant for the system (A, B) with closed-loop controller $\mathcal{M}(\Xi^{(1)}_{N,M})$. This is easily accomplished since we consider only affine regions of $\Xi^{(1)}_{N,M}$: the verification LP from before can be amended with a new objective function as follows, one for each of the $i = 1, \ldots, \mathsf{N}_x$ constraints comprising P_x: $\max(\ell_{x,i} \circ (A + B\ell_0))(x)$.

The results of this experiment are shown in Figure 1 *(c)*. The execution times are generally better than in the previous experiment, despite the fact that twice as many "output" constraints are checked. This is because *all* input polytopes were not invariant, so the algorithm was able to find a counter example early.

References

[1] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996.

[2] H Edelsbrunner, J O'Rourke, and R Seidel. Constructing Arrangements of Lines and Hyperplanes with Applications. *SIAM Journal on Computing*, 15(2):23, 1986.

[3] James Ferlez and Yasser Shoukry. AReN: Assured ReLU NN Architecture for Model Predictive Control of LTI Systems. In *Hybrid Systems: Computation and Control 2020 (HSCC'20)*. ACM, 2020.

[4] James Ferlez, Xiaowu Sun, and Yasser Shoukry. Two-Level Lattice Neural Network Architectures for Control of Nonlinear Systems, 2020.

[5] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 97–117. Springer International, 2017.

[6] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer. Algorithms for Verifying Deep Neural Networks, 2019.

[7] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the Number of Linear Regions of Deep Neural Networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2924–2932. Curran Associates, Inc., 2014.

[8] Arkadi S. Nemirovski and Michael J. Todd. Interior-point methods for optimization. *Acta Numerica*, 17:191–234, 2008.

[9] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability Analysis of Deep Neural Networks with Provable Guarantees, 2018.

[10] Richard P Stanley. An Introduction to Hyperplane Arrangements.

[11] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pages 147–156. Association for Computing Machinery, 2019.

[12] H.-D. Tran, D. Manzanas Lopez, P. Musau, X. Yang, L.-V. Nguyen, W. Xiang, and T. Johnson. Star-Based Reachability Analysis of Deep Neural Networks. In *Formal Methods – The Next 30 Years*, Lecture Notes in Computer Science. Springer International, 2019.

[13] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 3–17. Springer International Publishing, 2020.

[14] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: Analysis and efficient estimation. In *Advances in Neural Information Processing Systems 31*. Curran Associates, 2018.

[15] Yuh-Shyang Wang, Lily Weng, and Luca Daniel. Neural Network Control Policy Verification With Persistent Adversarial Perturbation. In *International Conference on Machine Learning*, pages 10050–10059. PMLR, 2020.

[16] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S. Dhillon, and Luca Daniel. Towards Fast Computation of Certified Robustness for ReLU Networks, 2018.