Polynomial-Time Reachability for LTI Systems With Two-Level Lattice Neural Network Controllers

James Ferlez and Yasser Shoukry, *Senior Member, IEEE*

***Abstract*—In this letter, we consider the computational complexity of bounding the reachable set of a Linear Time-Invariant (LTI) system controlled by a Rectified Linear Unit (ReLU) Two-Level Lattice (TLL) Neural Network (NN) controller. In particular, we show that for such a system and controller, it is possible to compute the exact one-step reachable set in *polynomial* time in the size of the TLL NN controller (number of neurons). Additionally, we show that a tight bounding box of the reachable set is computable via two polynomial-time methods: one with polynomial complexity in the size of the TLL and the other with polynomial complexity in the Lipschitz constant of the controller and other problem parameters. Finally, we propose a pragmatic algorithm that adaptively combines the benefits of (semi-)exact reachability and approximate reachability, which we call L-TLLBox. We evaluate L-TLLBox with an empirical comparison to a state-of-the-art NN controller reachability tool. In our experiments, L-TLLBox completed reachability analysis as much as 5000x faster than this tool on the same network/system, while producing reach boxes that were from 0.08 to 1.42 times the area.**

***Index Terms*—Computer-aided control design, neural networks, linear systems.**

I. Introduction

NEURAL Networks (NNs) are increasingly used to control dynamical systems in safety critical contexts. As a result, the problem of *formally* verifying the safety properties of NN controllers in closed loop is a crucial one. Despite this, comparatively little attention has been paid to the *time-complexity* of such reachability analysis. Understanding – and improving – the complexity of NN verification algorithms is thus crucial to designing provably safe NN controllers: it bears directly on the size of NNs that can be pragmatically verified.

Formal verification of NNs is usually formulated in terms of static input-output behavior, but there are few results analyzing the time complexity of such input-output verification [6], [9], [12]. We know of no paper that directly analyzes the time-complexity of exact reachability analysis for LTI systems with NN controllers, although [11] comes closest. Note: exact reachability is distinct from (polynomial-time) set-based reachability methods, which consider an over-approximated set of possible controller outputs in each state [2]. Formally, [11] only provides a complexity result for verifying the input-output behavior of a NN, but the underlying methodology, star sets, suggests a complexity analysis for exact reachability of LTI systems. Unfortunately, that algorithm produces exponentially many star sets – in the number of neurons – just to verify the input-output behavior of a NN once [11, Th. 1]; this exponential complexity compounds with each additional time step in reachability analysis. No such analysis is provided for the accompanying approximate star-set reachability analysis.

In this letter, we show that for a certain class of ReLU NN controllers – viz. Two-Level Lattice (TLL) NNs [5] – exact (or quantifiably approximate) reachability analysis for a controlled discrete-time LTI system is worst-case polynomial time complexity in the size (number of neurons) of the TLL NN controller. Thus, we show that LTI reachability analysis for the TLL NN architecture is dramatically more efficient (per neuron) than the same problem with general NNs (i.e., exponential complexity [11]; see above). In this sense, our results motivate for *directly designing TLL NN controllers in the first place*, since reachability for a TLL NN controller is more efficient to compute (TLL NNs are similarly beneficial in other problems: e.g., verification [4]). Moreover, TLL NNs can realize *the same functions* that general ReLU NNs can,[1] so no generality in realizable controllers is lost by this choice.

In particular, we prove several polynomial-complexity results related to the *one-step reachable set* of a discrete-time LTI system: i.e., the set $X_{t+1} = \{Ax + B\mathcal{N}(x) | x \in X_t\}$ for a given polytopic set of states[2] X_t and a TLL controller $\mathcal{N}$. Moreover, we consider the computation of both the *exact* set X_{t+1} and an ϵ-tight *bounding box* of X_{t+1}. All claimed complexities are worst case and with respect to *a fixed state-space dimension*,[3] n. These results are summarized as:

(i) The exact one-step reachable set, X_{t+1}, can be computed in polynomial time-complexity in the size of the TLL NN (Theorem 1).

Manuscript received 16 September 2022; revised 19 November 2022; accepted 6 December 2022. Date of publication 22 December 2022; date of current version 6 January 2023. This work was supported in part by NSF under Award CNS-2002405, Award CNS-2013824, and Award ECCS-2139781, and in part by C3.ai Digital Transformation Institute. Recommended by Senior Editor M. Arcak. *(Corresponding author: James Ferlez.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: jferlez@uci.edu; yshoukry@uci.edu).

Digital Object Identifier 10.1109/LCSYS.2022.3231556

[1] See the TLL form of Continuous Piecewise-Affine functions [10].

[2] Polytopic input constraints are a natural – and ubiquitous – choice, since ReLU NNs are affine on convex polytopic regions; hence, our complexity results are also expressed in terms of the complexity of a Linear Program.

[3] The reachability (verification) problem for a NN alone is known to be able to encode satisfiability of any 3-SAT formula; in particular, this result matches 3-SAT variables to input dimensions to the network [9].

(ii) An ϵ-tight bounding box for X_{t+1} can be computed via three algorithms with time-complexities:
   a) polynomial in size of the TLL (Theorem 2); or
   b) polynomial in the Lipschitz constant of the controller, the accuracy, ϵ, the norm of the B matrix and the volume of X_t (Proposition 1); or
   c) the minimum complexity of *(ii-a)* and *(ii-b)* (Theorem 3); this uses polynomial-time Lipschitz constant computation for TLLs (Lemma 1).

Here an ϵ-tight bounding box of X_{t+1} is one that is within $\epsilon > 0$ of the exact, coordinate-aligned bounding box of X_{t+1}.

In addition, we propose an algorithm that *adaptively* combines notions of exact and approximate *bounding box* reachability for TLL NNs in order to obtain an extremely effective approximate reachability algorithm, which we call L-TLLBox. We validate this method by empirically comparing an implementation of L-TLLBox[4] with the state-of-art NN reachability tool, NNV [12]. On a test suite of TLL NNs derived from the TLL Verification Benchmark in the 2022 VNN Competition [1], L-TLLBox performed LTI reachability analysis as much as 5000x faster than NNV on the same reachability problem; L-TLLBox produced reach boxes of 0.08 to 1.42 times the area produced by NNV.

Related work: There is a large literature on the complexity of set-based reachability for LTI systems; [2] provides a good summary. For the complexity of LTI-NN reachability, [11] is the closest to providing an explicit, exact result. The complexity of approaches based on NN over-approximation have been considered in [7], [8]. The literature on the complexity of input-output verification of NNs is larger but still small: [11] falls in this category as well; [9] is important for its NP-completeness result based on the 3-SAT encoding; and [4], [6] consider the complexity of verifying TLL NNs. Other NN-related complexity results include: computing the minimum adversarial disturbance is NP hard [14], and computing the Lipschitz constant is NP hard [13].

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 angle brackets $\langle\cdot\rangle$ to delineate the arguments to a function that *returns a function*. We use one special form of this notation: for a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and $i \in \{1, \ldots, m\}$ define $\pi_i(f) : x \mapsto [\![f(x)]\!]_{[i,:]}$. Finally, $\|\cdot\|$ will refer to the max-norm on $\mathbb{R}^n$, unless otherwise specified.

B. Neural Networks

We consider only Rectified Linear Unit Neural Networks (ReLU NNs). A K-layer ReLU NN is specified by K *layer* functions; a layer may be either linear or nonlinear. Both types of layer are specified by a parameter list $\theta \triangleq (W, b)$ where W is a $(\overline{d} \times \underline{d})$ matrix and b is a $(\overline{d} \times 1)$ vector. Specifically, the *linear* and *nonlinear* layers specified by θ are denoted by L_θ and $L^\sharp_\theta$, respectively, and are defined as:

$$L_\theta : \mathbb{R}^{\underline{d}} \to \mathbb{R}^{\overline{d}}, \qquad L_\theta : z \mapsto Wz + b \tag{1}$$

$$L^\sharp_\theta : \mathbb{R}^{\underline{d}} \to \mathbb{R}^{\overline{d}}, \qquad L^\sharp_\theta : z \mapsto \max\{L_\theta(z), 0\}. \tag{2}$$

where the max function is taken element-wise. Thus, a K-layer ReLU NN function is specified by functionally composing K such layer functions whose parameters $\theta^{|i}, i = 1, \ldots, K$ have dimensions that satisfy $\underline{d}^{|i} = \overline{d}^{|i-1} : i = 2, \ldots, K$; we will consistently use the superscript notation $^{|k}$ to identify a parameter with layer k. Whether a layer function is linear or not will be further specified by a set of linear layers, $\texttt{lin} \subseteq \{1, \ldots, K\}$. For example, a typical K-layer NN has $\texttt{lin} = \{K\}$, which together with a list of K layer parameters defines the NN: $\mathcal{N} = L_{\theta^{|K}} \circ L^\sharp_{\theta^{|K-1}} \circ \cdots \circ L^\sharp_{\theta^{|1}}$.

To indicate the dependence on parameters, we will index a ReLU $\mathcal{N}$ by a *list of NN parameters* $\Theta \triangleq (\texttt{lin}, \theta^{|1}, \ldots, \theta^{|K})$; i.e., we will often write $\mathcal{N}(\Theta) : \mathbb{R}^{\underline{d}^{|1}} \to \mathbb{R}^{\overline{d}^{|K}}$.

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

In this letter, we consider only Two-Level Lattice (TLL) ReLU NNs. Thus, we formally define NNs with the TLL architecture using the succinct method exhibited in [6]; the material in this subsection is derived from [5], [6].

A TLL NN is most easily defined by way of three generic NN composition operators. Hence, the following three definitions lead to the TLL NN in Definition 4.

Definition 1 (Sequential (Functional) Composition): Let $\mathcal{N}(\Theta_i) : \mathbb{R}^{\underline{d}_i^{|1}} \to \mathbb{R}^{\overline{d}_i^{|K_i}}$, $i = 1, 2$ be two NNs with parameter lists $\Theta_i \triangleq (\texttt{lin}_i, \theta_i^{|1}, \ldots, \theta_i^{|K_i})$, $i = 1, 2$ such that $\overline{d}_1^{|K_1} = \underline{d}_2^{|1}$. Then the **sequential (or functional) composition** of $\mathcal{N}(\Theta_1)$ and $\mathcal{N}(\Theta_2)$, i.e., $\mathcal{N}(\Theta_1) \circ \mathcal{N}(\Theta_2)$, is a NN that is represented by the parameter list $\Theta_1 \circ \Theta_2 \triangleq (\texttt{lin}_1 \cup (\texttt{lin}_2 + K_1), \theta_1^{|1}, \ldots, \theta_1^{|K_1}, \theta_2^{|1}, \ldots, \theta_2^{|K_2})$, where $\texttt{lin}_2 + K_1$ is an element-wise sum.

Definition 2: Let $\mathcal{N}(\Theta_i) : \mathbb{R}^{\underline{d}_i^{|1}} \to \mathbb{R}^{\overline{d}_i^{|K}}$, $i = 1, 2$ be two K-layer NNs with parameter lists $\Theta_i = (\texttt{lin}, (W_i^{|1}, b_i^{|1}), \ldots, (W_i^{|K}, b_i^{|K}))$, $i = 1, 2$ such that $\underline{d}_1^{|1} = \underline{d}_2^{|1}$; also note the common set of linear layers, $\texttt{lin}$. Then the **parallel composition** of $\mathcal{N}(\Theta_1)$ and $\mathcal{N}(\Theta_2)$ is a NN given by:

$$\Theta_1 \| \Theta_2 \triangleq \left(\texttt{lin}, \left(\begin{bmatrix} W_1^{|1} \\ W_2^{|1} \end{bmatrix}, \begin{bmatrix} b_1^{|1} \\ b_2^{|1} \end{bmatrix}\right), \ldots, \left(\begin{bmatrix} W_1^{|K} & \mathbf{0} \\ \mathbf{0} & W_2^{|K} \end{bmatrix}, \begin{bmatrix} b_1^{|K} \\ b_2^{|K} \end{bmatrix}\right)\right) \tag{3}$$

where $\mathbf{0}$ is a sub-matrix of zeros of the appropriate size. 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 3 (n-element min/max NNs): An ***n*-element min network** is denoted by the parameter list $\Theta_{\min_n}$. $\mathcal{N}(\Theta_{\min_n}) : \mathbb{R}^n \to \mathbb{R}$ such that $\mathcal{N}(\Theta_{\min_n})(x)$ 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 [5].

The ReLU NNs defined in Definition 1-3 can be arranged to define a TLL NN as shown in [4, Figure 1]. We formalize this construction by first defining a *scalar* TLL NN, and then extend this notion to a *multi-output* TLL NN [6].

Definition 4 (Scalar TLL NN [6]): A NN from $\mathbb{R}^n \to \mathbb{R}$ is a **TLL NN of size** (N, M) if its parameter list $\Xi_{N,M}$ can be characterized entirely by integers N and M as follows.

$$\Xi_{N,M} \triangleq \Theta_{\max_M} \circ \left((\Theta_{\min_N} \circ \Theta_{S_1}) \| \ldots \| (\Theta_{\min_N} \circ \Theta_{S_M})\right) \circ \Theta_\ell \tag{4}$$

[4]https://github.com/jferlez/FastBATLLNN

where

- $\Theta_\ell \triangleq (\{1\}, \theta_\ell)$ for $\theta_\ell \triangleq (W_\ell, b_\ell)$;
- each Θ_{S_j} has the form $\Theta_{S_j} = (\{1\}, (S_j, \mathbf{0}))$ where $\mathbf{0}$ is the column vector of N zeros, and where
  - $S_j = [\,[\![I_N]\!]_{\iota_1,:}^T \ \dots \ [\![I_N]\!]_{\iota_N,:}^T\,]^T$ for a length-N sequence $\{\iota_k\}$ where $\iota_k \in \{1, \dots, N\}$ and I_N is the $(N \times N)$ identity matrix.

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

Definition 5 (Multi-Output TLL NN [6]): A NN that maps $\mathbb{R}^n \to \mathbb{R}^m$ is said to be a **multi-output TLL NN of size** (N, M) if its parameter list $\Xi_{N,M}^{(m)}$ can be written as

$$\Xi_{N,M}^{(m)} = \Xi_{N,M}^1 \| \dots \| \Xi_{N,M}^m \tag{5}$$

for m equally-sized scalar TLL NNs, $\Xi_{N,M}^1, \dots, \Xi_{N,M}^m$, which will be referred to as the **(output) components of** $\Xi_{N,M}^{(m)}$.

Finally, we have the following definition.

Definition 6 (Non-Degenerate TLL): A scalar TLL NN $\Xi_{N,M}$ is **non-degenerate** if each function $\ell_i \triangleq \pi_i(L_{\Theta_\ell})$ (see Definition 4) is realized on some open set. That is, for each $i = 1, \dots, N$ there exists an open set $V_i \subset \mathbb{R}^n$ such that

$$\forall x \in V_i . \mathcal{N}(\Xi_{N,M})(x) = \ell_i(x). \tag{6}$$

III. Problem Formulation

The main object of our attention is the reachable set of a discrete-time LTI system in closed-loop with a state-feedback TLL NN controller. To this end, we define the following.

Definition 7 (One-Step Closed-Loop Reachable Set): Let $x_{t+1} = Ax_t + Bu_t$ be a discrete-time LTI system with states $x_t \in \mathbb{R}^n$ and controls $u_t \in \mathbb{R}^m$. Furthermore, let $X \subset \mathbb{R}^n$ be a compact, convex polytope, and let $\mu : \mathbb{R}^n \to \mathbb{R}^m$ be a state-feedback controller. Then the **one-step reachable set from** X_t **under feedback control** μ is defined as:

$$X_{t+1} \triangleq \mathcal{R}(X_t, \mu) \triangleq \{Ax + B\mu(x) | x \in X_t\}. \tag{7}$$

For a compact, convex polytope, $X_0 \subset \mathbb{R}^n$, the **T-step reachable set from X_0 under control** μ is the set X_T that is defined according to the recursion:

$$X_t \triangleq \mathcal{R}(X_{t-1}, \mu), t = 1, \dots, T. \tag{8}$$

In one instance, we will be interested in computing X_t *exactly* from X_{t-1} (or by recursive application, X_0). However, we will also be interested in two different approximations for the reachable set X_t: a one-step bounding box for X_t from X_{t-1}, and a bounding box for X_t obtained by propagating bounding boxes from X_0. Thus, we have the following.

Definition 8 (One-Step ϵ-Bounding Box): Let A, B, X_t and μ be as in Definition 7. Then a **one-step ϵ bounding box reachable from** X_t is a box $\mathrm{B}_{t+1} = \mathsf{X}_{i=1}^n [l_i, r_i] \subset \mathbb{R}^n$ s.t.:

1) $X_{t+1} = \mathcal{R}(X_t, \mu) \subseteq \mathrm{B}_{t+1}$; and
2) for each $i = 1, \dots, n$, there exist points $x_{l_i}, x_{r_i} \in \mathcal{R}(X_t, \mu)$ such that:

$$\left| [\![x_{l_i}]\!]_{[i,:]} - l_i \right| < \epsilon \text{ and } \left| [\![x_{r_i}]\!]_{[i,:]} - r_i \right| < \epsilon. \tag{9}$$

The idea of a one-step ϵ bounding box can be extended to *approximate* reachability by propagating one-step bounding boxes recursively instead of the previous reachable set itself.

Definition 9 (ϵ-Bounding Box Propagation): Let A, B, X_0 and μ be as in Definition 7.

Let $\mathrm{B}_0 \triangleq X_0$ by convention. Then an **ϵ-bounding box propagation** of X_0 is a sequence of bounding boxes, $\mathrm{B}_t^{X_0}$, $t = 0, \dots, T$ such that:

- for all $t = 1, \dots, T$, $\mathrm{B}_t^{X_0}$ is an ϵ-bounding box for the system with initial set of states $\mathrm{B}_{t-1}^{X_0}$.

Note: although an ϵ-bounding box propagation only approximates the reachable set X_t, the amount of over-approximation depends only on ϵ and the dynamics – *not the controller*. Thus, any desired approximation error to X_t can be obtained by computing ϵ-box propagations of suitably small *subsets* of X_0 (to compensate for the propagation of each bounding box approximation through the dynamics).

Finally, as a consequence of considering ReLU NNs and polytopic state sets, our complexity results can be written in terms of the complexity of solving a linear program (LP).

Definition 10 (LP Complexity): Let $\mathrm{LP}(\eta, \nu)$ be the complexity of an LP in dimension ν with η inequality constraints.

This complexity is polynomial in both parameters, subject to the usual caveats associated with digital arithmetic.

IV. Exact Reachability for TLL NN Controllers

Our first complexity result shows that exact one-step reachability for an LTI system controlled by a TLL NN is computable in polynomial time in the size of the TLL.

Theorem 1: Let A, B, and X_t be as defined in Definition 7, where X_t is the intersection of N_{X_t} linear constraints. Moreover, suppose this system is controlled by a state-feedback TLL NN controller $\mathcal{N}(\Xi_{N,M}^{(m)}) : \mathbb{R}^n \to \mathbb{R}^m$ (Section II-C).

For a fixed state dimension, n, the reachable set X_{t+1} can be represented as the union of at most $O(m^n \cdot N^{2n}/n!)$ compact, convex polytopes, and these polytopes can be computed in time complexity at most (also for fixed n):

$$O((m \cdot N)^{2^{n+1}} \cdot m^{n+2} \cdot n \cdot M \cdot N^{2n+3} \cdot \mathrm{LP}(mN^2 + N_{X_t}, n)/n!).$$

Proof: This follows almost directly from the result in [6], where it is shown that a multi-output TLL with parameters $\Xi_{N,M}^{(m)}$ has at most as many linear (affine) regions as there are regions in a hyperplane arrangement with $O(m \cdot N^2)$ hyperplanes. Clearly, each of these potential regions can contribute one polytope to the reachable set X_{t+1}. According to [6], these regions can be enumerated in time complexity:

$$O(m^{n+2} \cdot n \cdot M \cdot N^{2n+3} \cdot \mathrm{LP}(mN^2 + N_{X_t}, n)/n!) \tag{10}$$

which includes the complexity of identifying the active linear function on each of those regions [6, Proposition 4]. The LP complexity in (10) depends on N_{X_t} because it is necessary to obtain the intersection of the $O(m \cdot N^2)$ regions with X_t.

Thus, it remains to determine the reachable set with respect to the TLL's realized affine function on each such region. The complexity of this operation is bounded by the complexity of transforming each such polytope through the A matrix and B times the affine function realized by the TLL on that region. This can be accomplished by Fourier-Motzkin elimination to determine the resulting polytopes that add together to form the associated constituent polytope of X_{t+1}. This operation has complexity $O((m \cdot N)^{2^{n+1}})$. ■

V. Bounding-Box Reachability for TLL NN Controllers

We begin with the following useful definition.

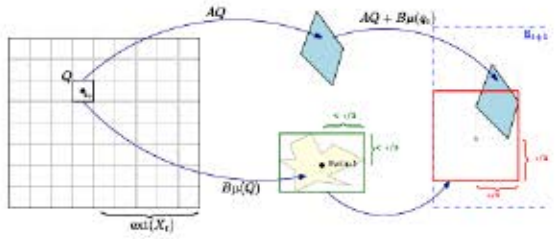

Fig. 1. Illustration of the proof of Proposition 1.

Definition 11 (Center/Extent of $X \subset \mathbb{R}^n$): Let $X \subset \mathbb{R}^n$ be a compact set. Then the **center of** X is the point $x_c \in \mathbb{R}^n$ such that for each $i = 1, \ldots, n$:

$$[\![x_c]\!]_{[i,:]} \triangleq \tfrac{1}{2} \cdot \left(\min_{x \in X} [\![x]\!]_{[i,:]} + \max_{x \in X} [\![x]\!]_{[i,:]} \right) \tag{11}$$

Also, define the **extent of** X **along coordinate** i as:

$$ext_i(X) \triangleq \max_{x \in X} [\![x - x_c]\!]_{[i,:]}, \tag{12}$$

and the **extent of** X as $\mathrm{ext}(X) \triangleq \max_{i=1,\ldots,n} \mathrm{ext}_i(X)$.

A. One-Step ϵ Bounding Box Reachability

Now we can state our second main result: that a one-step bounding box can be computed in polynomial time in the size of a TLL NN controller. We provide two such results, each of which is polynomial in different aspects of the problem.

Theorem 2: Let A, B, X_t and $\mathcal{N}(\Xi^{(m)}_{N,M})$ be as in the statement of Theorem 1.

Then a one-step $\epsilon = 0$ bounding box from X_t is computable in time complexity at most (for fixed dimension, n):

$$O(m^{n+2} \cdot n^2 \cdot M \cdot N^{2n+3} \cdot \mathrm{LP}(mN^2 + N_{X_t}, n)/n!). \tag{13}$$

Proof: This follows almost directly from the proof of Theorem 1. For each of the convex polytopes describing the reachable set, the Fourier-Motzkin elimination can be replaced by $2 \cdot n$ LPs to compute its bounding box; these can be combined to determine an $\epsilon = 0$ bounding box for X_{t+1} without increasing the complexity noted above. ■

The result in Theorem 2 certainly meets the criteria of a polynomial-time computation of a one-step bounding box. Unfortunately, the dependence on N and M in Theorem 2 is significant despite being polynomial. However, since we are considering only bounding box reachability, it makes sense to regard the TLL controller as a generic Lipschitz-continuous controller instead: this allows the dependence on its size to be replaced with a dependence on its Lipschitz constant, at the expense of additional (polynomial) dependence on the size of the set X_t and the norm of the matrix B.

Proposition 1: Let A, B, X_t and μ be as in the statement of Definition 7. Furthermore, suppose that μ is Lipschitz continuous on X_t, with Lipschitz constant at most $\|\mu\|$.

Then an ϵ-bounding box from X_t is computable in complexity at most (for fixed dimension, n):

$$O\left(\left(2 \cdot ext(X_t) \cdot \tfrac{2\|B\| \cdot \|\mu\|}{\epsilon}\right)^n \cdot LP(2 \cdot n, n)\right). \tag{14}$$

Proof: X_t can be covered by a grid of $(2 \cdot \mathrm{ext}(X_t) \cdot 2\|B\| \cdot \|\mu\|/\epsilon)^n$ hypercubes whose edges are of width $\epsilon/(2\|B\| \cdot \|\mu\|)$. Denote by Q an arbitrary such hypercube, and let q_c denote its center. Now observe that for all $q \in Q$:

$$\|B\mu(q) - B\mu(q_c)\| \leq \|B\| \cdot \|\mu\| \cdot \|q - q_c\| \leq \tfrac{\epsilon}{2} < \epsilon. \tag{15}$$

Consequently, the set $Q' \triangleq \{x : \|AQ + B\mu(q_c) - x\| < \epsilon\}$ is guaranteed to be in any ϵ-bounding box of X_{t+1}, as is the exact bounding thereof, which we denote by $\mathrm{box}(Q')$; see Fig. 1. Clearly $\cup_Q \mathrm{box}(Q')$ is likewise so contained.

Thus, an ϵ-bounding box from X_t can be obtained by examining each Q and computing $\mathrm{box}(Q')$. The latter operation entails computing a bounding box for AQ, which has the complexity of $LP(2 \cdot n, n)$; see Fig. 1. ■

In particular, for some problems, the quantities in (14) may be much smaller than terms like N^{2n} in (13). In fact, this explains why this type of result is typically not used for NN reachability: it is computationally expensive to compute the exact Lipschitz constant of a generic NN – indeed, it is of exponential complexity in the number of neurons for a deep NN. For a non-degenerate TLL NN, however, it is trivial to compute its exact Lipschitz constant (over all of $\mathbb{R}^n$).[5]

Lemma 1: A bound on the Lipschitz constant of a TLL NN $\Xi^{(m)}_{N,M}$ over $\mathbb{R}^n$ is computable in complexity $O(m \cdot N \cdot n)$. For a non-degenerate TLL (Definition 6), this bound is tight.

Proof: This is a straightforward application of [6, Proposition 3] or the related result [3, Proposition 4]. The only affine functions realizable by a TLL are those described by its linear layer (see Definition 4). If the TLL is not degenerate, then each of those is realized in its output, hence also lower bounding its Lipschitz constant. The claim follows, since there are N such local linear functions per output, each of whose Lipschitz constants can be computed in $O(n)$. ■

Theorem 3: Let A, B, X_t and $\mathcal{N}(\Xi^{(m)}_{N,M})$ be as in the statement of Theorem 1.

Then for any $\epsilon > 0$, an ϵ-bounding box from X_t can be computed with complexity no more than the maximum of (13) and (14) for $\|\mathcal{N}(\Xi^{(m)}_{N,M})\|$ bounded according to Lemma 1.

VI. The L-TLLBox Algorithm

Theorem 3 establishes a trade-off in computational complexity between two methods for computing an ϵ-bounding box from states X_t. However, it requires a commitment to the full computational complexity of one algorithm or the other. Moreover, the difference in computational complexity between Theorem 2 and Proposition 1 depends on the characteristics of the TLL controller on the set X_t (assuming $\mathrm{ext}(X_t)$ and $\|B\|$ are fixed). If the TLL controller has relatively *few linear regions* intersecting X_t but a relatively *large Lipschitz constant* on X_t, then Theorem 2 will have lower complexity; recall that Theorem 2 enumerates the linear regions of the TLL controller that intersect X_t. If, on the other hand, the TLL controller has relatively *numerous linear regions* intersecting X_t but a relatively *small Lipschitz constant* on X_t, then Proposition 1 will instead have lower complexity. Thus, the trade off in complexity between Theorem 2 and Proposition 1 amounts to roughly the following: reachability via Proposition 1 is more efficient when the Lipschitz constant of the TLL controller yields a partition of X_t into hypercubes (see proof of Proposition 1) each of which "typically" contains many linear regions of the TLL.

This is a salient observation in light of the way that Proposition 1 employs the Lipschitz constant of the TLL controller in question. In Proposition 1, the Lipschitz constant of the TLL controller is really used to create a subdivision of X_t into sets such that the output of the TLL controller lies within box of sufficiently small width: see (15). However, note that the Lipschitz constant of a TLL controller over the *entire set* X_t will generally be larger than the Lipschitz constant of the

[5]Even when this bound is approximate, it depends only on the parameters of one layer; for general deep ReLU NNs, a bound of similar computational complexity involves multiplying weight matrices of successive layers.

TLL controller on any subset of X_t. Also, there may be subsets of X_t where the TLL controller rapidly switches between large-Lipschitz-constant affine functions such that its output is nevertheless confined to a small box (e.g., a high-frequency saw-tooth function with small amplitude). This suggests the following improvement on Theorem 2 and Proposition 1: identify large subsets of X_t where the output of the TLL controller is bounded within a small box – thereby replacing the enumeration of many linear regions of the TLL controller, as in Theorem 2, *or* the enumeration of many "Lipschitz-width" hypercubes, as in Proposition 1.

Thus, we introduce L-TLLBox as a practical, "adaptive" algorithm, which implements this strategy via the recent tool FastBATLLNN [4]. In particular, FastBATLLNN provides a fast algorithm for obtaining an ϵ-tight bounding box on the output of a TLL controller subject to a convex, polytopic input constraint. This means that FastBATLLNN can be used to directly and efficiently identify subsets of X_t where the output of the TLL controller is confined to a small box, without enumerating all of the linear regions of the TLL. That is, for a convex, polytopic set $P \subset X_t$ and a box $\mathsf{B} = \mathsf{X}_{i=1}^{m}[l_i, r_i]$ FastBATLLNN can efficiently decide the query:

$$\forall x \in P . \mathscr{N}(\Xi_{N,M}^{(m)})(x) \in \mathsf{B} \tag{16}$$

by an algorithm that has *half* the crucial exponent of the algorithms of Theorem 1 and Theorem 2 [6] – i.e., **without enumerating the affine regions of the TLL**. Thus, for P as above, a tight bounding box on $\mathscr{N}(\Xi_{N,M}^{(m)})(P)$ can be obtained by roughly log invocations of FastBATLLNN in a binary search on the endpoints of a bounding box.

In this context, the structure of L-TLLBox can be summarized as follows. Start with a hypercube Q_1^0 of edge length $2 \cdot \text{ext}(X_t)$, so that Q_1^0 is large enough to capture the whole set X_t. Then use FastBATLLNN to determine a sufficiently tight bounding box on the set $\mathscr{N}(\Xi_{N,M}^{(m)})(Q_1^0 \cap X_t)$. If this bounding box is small enough that its endpoints differ by less than $\epsilon/\|B\|$ from its center (see (15)), then the reachable bounding box, B_{t+1} can be updated directly as in Proposition 1. Otherwise, Q_1^0 should be refined into 2^n hypercubes, each with half its edge lengths, denoted by Q_p^1, $p = 1, \ldots, 2^n$, and the process is repeated recursively on each. As above, the recursion stops for a hypercube Q_p^d at depth d only if FastBATLLNN returns an bounding box for $\mathscr{N}(\Xi_{N,M}^{(m)})(Q_p^d \cap X_t)$ whose endpoints are within $\epsilon/\|B\|$.

This basic recursion is described in Algorithm 1. Three functions in Algorithm 1 require explanation:

- `BBox`(P) computes an exact bounding box for the convex polytopic set P using LPs;
- `Subdivide`(Q, p) returns the p^{th} hypercube obtained by splitting each edge of the hypercube Q in half;
- `FastBATLLNN`$(\Xi_{N,M}^{(m)}, P, \epsilon)$ returns an ϵ-tight bounding box on the set $\mathscr{N}(\Xi_{N,M}^{(m)})(P)$.

Formally, we have the following Theorem, which describes the *worst-case* runtime of L-TLLBox.

Theorem 4: Let A, B, X_t and $\mathscr{N}(\Xi_{N,M}^{(m)})$ be as in the statement of Theorem 3.

Then for any $\epsilon > 0$, L-TLLBox can compute an ϵ bounding box from X_t with a worst-case time complexity of

$$O\Big(\log_2 \big[\epsilon \cdot \|\mathscr{N}(\Xi_{N,M}^{(m)})\| \cdot \text{ext}(X_t) \cdot \log_2 \big[\tfrac{\text{ext}(X_t)\cdot\|B\|\cdot\|\mu\|}{\epsilon}\big]\big] \\ m \cdot K \cdot 2^{K\cdot n} \cdot n \cdot M \cdot \tfrac{N^{n+3}}{n!} \cdot \text{LP}(N + N_{X_t}, n)\Big) \tag{17}$$

where $K = \lceil \log_2(2 \cdot \text{ext}(X_t) \cdot \frac{2\|B\|\cdot\|\mu\|}{\epsilon}) \rceil$.

Algorithm 1: L-TLL Box Core Recursion

input : $\epsilon > 0$
A an $(n \times n)$ matrix
B an $(n \times m)$ matrix
X_t a compact, convex polytope of states
$\Xi_{N,M}^{(m)}$, parameters of a TLL NN to verify
output: $\mathsf{B}_{t+1} = \mathsf{X}_{i=1}^{n}[l_i, r_i]$, an ϵ-bounding box from X_t

```
global d ← 0
global B_{t+1} ≜ X_{i=1}^n [l_i, r_i] ← X_{i=1}^n [∞, −∞]
function LTLLBox(ε, A, B, X_t, Ξ_{N,M}^{(m)})
    d ← d + 1 // Increment depth counter
    Q^d_local ← BBox(X_t)
    for p in 1, ..., 2^n do
        // Subdivide Q^d_local into 2^n hypercubes
        Q_p^{d+1} ← Subdivide(Q^d_local, p)
        /* Get a bounding box on TLL output to
           ε/2 error for inputs in Q_p^{d+1} ∩ X_t      */
        TLLBx ← FastBATLLNN(Ξ_{N,M}^{(m)}, Q_p^{d+1} ∩ X_t, ε/2)
        if Width(TLLBx) < ε/(2 · ‖B‖) then
            /* Output of TLL is small enough on
               Q_p^{d+1} that we can update B_{t+1}     */
            for i in 1, ..., n do
                if Min(⟦BBox(AQ_p^{d+1})⟧_{[i,:]}) − Min(⟦TLLBx
                  ⟧_{[i,:]}) < l_i then
                    l_i ← Min(⟦TLLBx⟧_{[i,:]})
                end
                if Max(⟦BBox(AQ_p^{d+1})⟧_{[i,:]}) + Max(⟦TLLBx
                  ⟧_{[i,:]}) > r_i then
                    r_i ← Max(⟦TLLBx⟧_{[i,:]})
                end
            end
            return
        else // Need to refine on Q_p^{d+1}
            LTLLBox(ε, A, B, Q_p^{d+1} ∩ X_t, Ξ_{N,M}^{(m)})
        end
    end
end
```

Proof: L-TLLBox recursively subdivides a hypercube of edge length at most $2 \cdot \text{ext}(X_t)$ into 2^n hypercubes with each recursion. Let d denote the number of recursions, so that at depth d, L-TLLBox has created at most $2^{d\cdot n}$ hypercubes. Let Q_p^d, $p = 1, \ldots, 2^{d\cdot n}$ denote the hypercubes at depth d.

In the worst case, L-TLLBox must recurse on every single hypercube at each depth until all of the resultant subdivided hypercubes have edge length at most $2\cdot\text{ext}(X_t)\cdot(2\|B\|\cdot\|\mu\|)/\epsilon$. This explains the factor $K\cdot 2^{K\cdot n}$: the complexity at each depth is *added* to the runtime, so the cumulative runtime is dominated by the runtime for the largest recursion depth.

On any given subdivided hypercube, Q_p^d, the complexity of L-TLLBox is dominated by using FastBATLLNN in a binary search on each of the m real-valued TLLs comprising $\mathscr{N}(\Xi_{N,M}^{(m)})$. This is necessary to determine an $\epsilon/2$ bounding box on the output of $\mathscr{N}(\Xi_{N,M}^{(m)})$ when its input is constrained to the set Q_p^d. Since the Lipschitz constant of $\mathscr{N}(\Xi_{N,M}^{(m)})$ is known, the invocations of FastBATLLNN on each output is associated with a binary search over an interval of width $2 \cdot \|\mathscr{N}(\Xi_{N,M}^{(m)})\| \cdot 2 \cdot \text{ext}(X_t)/2^d$ until iterations of the search are lie in an interval of width $\epsilon/2$. Thus, each output requires

$$\Big\lceil \log_2\Big(\epsilon \cdot \|\mathscr{N}(\Xi_{N,M}^{(m)})\| \cdot \text{ext}(X_t)/2^{3-d}\Big)\Big\rceil \tag{18}$$

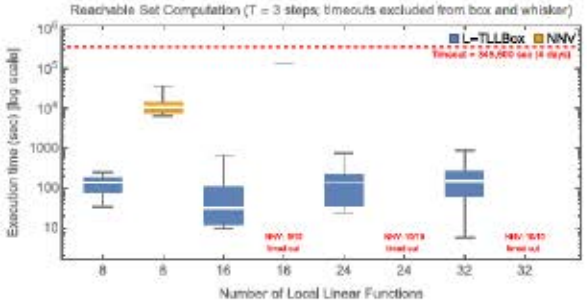

Fig. 2. Execution time of L-TLLBox compared to NNV.

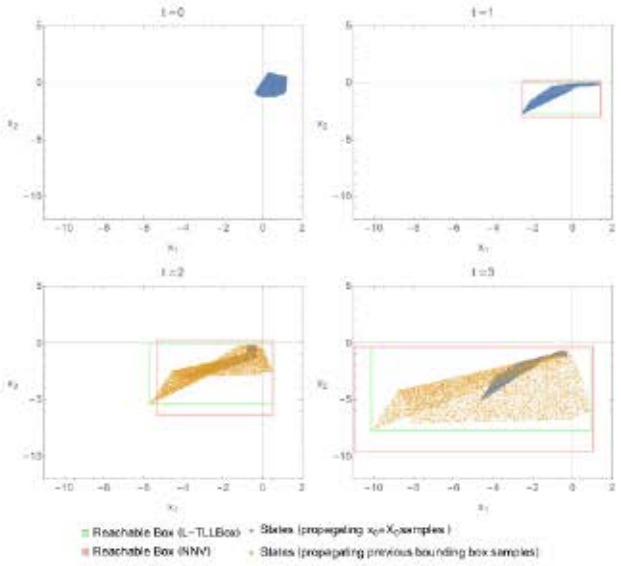

Fig. 3 Example reachability box progressions computed by L-TLLBox (25 sec.) and NNV (139,000 sec.); this was the sole $N = 16$ reachability sequence completed by NNV.

invocations of FastBATLLNN, so the total number of invocations of FastBATLLNN is less than:

$$O\left(\log_2\left[\epsilon \cdot \|\mathcal{N}(\Xi_{N,M}^{(m)})\| \cdot \text{ext}(X_t) \cdot \log_2\left[\frac{\text{ext}(X_t)\cdot\|B\|\cdot\|\mu\|}{\epsilon}\right]\right]\right)$$

From [4], each invocation of FastBATLLNN has complexity:

$$n \cdot M \cdot N^{n+3} \cdot \text{LP}(N + N_{X_t}, n)/n!. \tag{19}$$

This explains the formula (17). ■

VII. Experiments

To evaluate L-TLLBox as an LTI reachability tool, we used it to perform multi-step bounding box propagation on a number of TLL NN controllers; see Definition 9. We compared the results to NNV's [12] approximate reachability analysis setting. For this evaluation we selected 40 networks from the TLL Verification Benchmark in the 2022 VNN competition [1]: the first 10 examples from each of the sizes $N = M = 8, 16, 24$ and 32 were used, and these TLLs were converted to a fully-connected Tensorflow format that NNV could import. Each TLL had $n = 2$ inputs and $m = 1$ output, so we took these as our state and control dimensions, respectively and generated one random A and B matrix for each TLL NN. We likewise generated one polytopic set of states to serve as X_0 for each TLL/system combination. Reachability analysis was performed on both tools for $T = 3$ discrete time steps. Both tools were given at most 4 days of compute time per TLL/system combination on a standard Microsoft Azure E2ds v5 instance; this instance has one CPU core running at 2.8GHz and 16Gb of RAM and 32Gb swap.

The execution time results of this experiment are summarized by the box-and-whisker plot in Fig. 2. L-TLLBox was able to complete all reachability problems well within the timeout. However, NNV only completed the 10 reachability problems for size $N = 8$ and one reachability problem for size $N = 16$; it timed out at 4 days for all other problems. On problems where both tools completed the entire reachability analysis, L-TLLBox ranged from 32x faster (the first instance of size $N = 8$) to 5,392x faster (the commonly completed instance of $N = 16$). For problems that both algorithms finished, the final reachability boxes produced by L-TLLBox were anywhere from 0.08 to 1.42 times the area of those produced by NNV. Fig. 3 shows one sequence of reach boxes output by L-TLLBox and NNV.

VIII. Conclusion

In this letter, we presented several polynomial complexity results for reachability of an LTI system with a TLL NN controller, including L-TLLBox; these results improve on the exponential complexity for the same reachability problem with a general NN controller, and thus provide a motivation for designing NN controllers using the TLL architecture. As a result, there are a numerous opportunities for future work such as: considering reachability for more general convex sets (e.g., ellipsoidal sets), and generalizing FastBATLLNN so that L-TLLBox can be extended to exact reachability.

References

[1] "VNN Competition," 2022. [Online]. Available: https://sites.google.com/view/vnn2022
[2] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annu. Rev. Control Robot. Auton. Syst.*, vol. 4, no. 1, pp. 369–395, 2021.
[3] U. S. Cruz, J. Ferlez, and Y. Shoukry, "Safe-by-repair: A convex optimization approach for repairing unsafe two-level lattice neural network controllers," in *Proc. 61st IEEE Conf. Decis. Control (CDC)*, 2022, pp. 1–19.
[4] J. Ferlez, H. Khedr, and Y. Shoukry, "Fast BATLLNN: Fast box analysis of two-level lattice neural networks," in *Proc. ACM Hybrid Syst. Comput. Control (HSCC)*, 2022, pp. 1–7.
[5] J. Ferlez and Y. Shoukry, "AReN: Assured ReLU NN architecture for model predictive control of LTI systems," in *Proc. Hybrid Syst. Comput. Control (HSCC)*, 2020, pp. 1–11.
[6] J. Ferlez and Y. Shoukry, "Bounding the complexity of formally verifying neural networks: A geometric approach," in *Proc. 60th IEEE Conf. Decis. Control (CDC)*, 2021, pp. 5104–5109.
[7] C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu. "POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems." 2022. [Online]. Available: http://arxiv.org/abs/2106.13867
[8] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verifying the safety of autonomous systems with neural network controllers," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 1, pp. 1–7, 2020.
[9] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Computer Aided Verification* (Lecture Notes in Computer Science). Cham, Switzerland: Springer Int., 2017, pp. 97–117. [Online]. Available: https://doi.org/10.1007/978-3-319-63387-9_5
[10] J. M. Tarela and M. V. Martínez, "Region configurations for realizability of lattice Piecewise-Linear models," *Math. Comput. Model.*, vol. 30, no. 11, pp. 17–27, 1999.
[11] H.-D. Tran et al., "Star-based reachability analysis of deep neural networks," in *Formal Methods—The Next 30 Years* (Lecture Notes in Computer Science). Cham, Switzerland: Springer Int., 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-53288-8_1
[12] H.-D. Tran et al., "NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *Proc. Comput.-Aided Verification*, 2020, pp. 3–17.
[13] A. Virmaux and K. Scaman, "Lipschitz regularity of deep neural networks: Analysis and efficient estimation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 3839–3848.
[14] T.-W. Weng et al. "Towards Fast Computation of Certified Robustness for ReLU Networks." 2018. [Online]. Available: http://arxiv.org/abs/1804.09699