A Compositional Algorithm for Computing a Switched System Representation of Neural Network Controllers

Miriam García Soto[1] and Pavithra Prabhakar[2]

***Abstract*— Our broad motivation is to utilize the large body of work on verification techniques for switched affine systems towards verification of neural network-controlled systems. To this end, we explore the problem of computing a switched affine system (SAS) representation of neural network-controlled discrete-time linear dynamical systems by providing a compositional algorithm that computes the piecewise affine (PWA) representation of the neural network. Our algorithm relies on two subroutines - one that computes the PWA representation of a single layer of a neural network, and the other that computes the compositions of PWA representations. We introduce the concept of a composition ordering represented as a binary tree that specifies the order in which the layers of the neural network are composed, and use that to compute the PWA representation of the whole neural network. Our experimental evaluation highlights the critical parameters of the network affecting the runtime complexity. Finally, we illustrate the application of the PWA representation computation toward stability analysis of a neural network-controlled discrete-time linear dynamical system.**

I. Introduction

Neural networks are being used routinely as replacements for traditional controllers owing to their computational and memory efficiency, and their ability to deal with dynamic and uncertain environments [1], [2]. While this has resulted in intelligent systems that are more efficient and robust, their certification remains a barrier to their deployment in real world. Specifically, the large size of the neural networks and the highly non-linear function they represent, combined with the black-box nature of the controllers lacking human intuition, pose challenges toward their scalable automatic verification. Our broad goal through this work is to provide a switched affine system (SAS) representation of neural network-controlled systems enabling the application of the large body of work on SAS verification toward neural network-controlled linear dynamical systems verification. Further, such a representation will provide insights into the function represented by a neural network, thereby enhancing their interpretability.

Our main contribution is a compositional algorithm for computing a piecewise affine system (PWA) representation of the input-output function of a neural network that exploits the compositional nature of this function as a composition of the functions corresponding to the individual layers, and the associative property of the composition operator. Our algorithm takes as input a neural network $\mathcal{N}$ and a composition ordering T provided as a binary tree with leaves corresponding to layers of $\mathcal{N}$. The algorithm builds on two primitives: (1) PWA(W, b) that computes the PWA representation of the function $f(x) = \sigma(Wx + b)$, where σ is the activation function, W the weight matrix and b the bias; and (2) Compose(f, g) that computes the PWA representation of the composition of the functions f and g, that is $g \circ f$. The overarching algorithm essentially computes the PWA representation of the root node of T through a bottom-up evaluation which is captured using a recursive algorithm. The compositional nature of the algorithm makes it amenable to parallelization and efficient updates for evolving neural networks by considering a composition ordering that corresponds to a tree with minimal height.

We performed a thorough experimental evaluation of our algorithm on a large class of randomly generated networks to evaluate the effect of different network parameters on the runtime complexity. First, we noticed that the network size itself doesn't affect the runtime. However, the runtime is proportional to the number of regions in the PWA representation of the neural network. Note that the two primitives are called almost the same number of times (since the number of leaves and internal nodes in a full binary tree are almost the same), however, the algorithm spends more time overall in the computation of the second primitive than the first, implying that composition construction is more expensive than computing the PWA computations for individual layers. This is expected as the size of the PWA representation grows with composition operations.

Finally, to illustrate how the PWA representation computation can be used toward verification, we consider the stability verification of neural network-controlled linear dynamical systems. We apply our algorithm to compute a PWA representation of the neural network and the SAS representation of the closed-loop system. Any algorithm for SAS stability verification can now be used to analyze

This work was partially supported by NSF Grant No. 2008957, an Amazon Research Award and by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 847635.

[1]Miriam García Soto is with the Faculty of Computer Science, Complutense University of Madrid, 28040 Madrid , Spain miriamgs@ucm.es

[2] Pavithra Prabhakar is with the Department of Computer Science, Kansas State University, Manhattan, KS, 66503, USA pprabhakar@ksu.edu

the stability of the neural network-controlled system.

a) Related Work: Safety verification of neural network-controlled systems has been extensively explored in the literature and is widely based on computing the reachable output set of neural networks with techniques ranging from abstraction-based methods [3]–[5] to constraint solving [6]–[8] and symbolic analysis techniques [9]–[14]. Concretely, the approach in [14] computes a PWA representation from a neural network considering the usual layer ordering. Our method generalizes this by enabling arbitrary binary tree orderings.

Stability verification has received relatively less attention in the context of neural network controlled systems. For instance, a Lyapunov function-based approach [15]–[17] and a local integral quadratic constraint-based approach [18] has been explored for systems with feedforward and recurrent neural network controllers, respectively. In addition, a compositional approach has been proposed for learning and verifying neural network controllers for system subtasks [19]. In contrast, stability verification has been extensively studied in the realm of switched linear/affine systems (see [20] for a survey).

There is some recent work on computing PWA representation of a single layer [17] and multi-layer [21] neural network with ReLU activation functions. However, the algorithm presented in [21] is not compositional unlike ours which comes with the possibility of parallelization and efficient updates.

II. Preliminaries

Let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$ and $\mathbb{N}$ denote the set of reals, non-negative reals and natural numbers, respectively. Given $k \in \mathbb{N}$, we denote by $[k]$ the set of natural numbers $\{0, 1, \ldots, k\}$ and by $(k]$, the set $\{1, \ldots, k\}$.

A *partition* $\mathcal{P}$ of $S \subseteq \mathbb{R}^n$ into convex polyhedral sets is a finite set of convex polyhedra $\{P_i\}_{i\in(k]}$ such that $\cup_{i=1}^{k} P_i = S$ and for each $i \neq j$, $\mathcal{P}_i \cap P_j = \emptyset$.

Let f and g be functions such that $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^p$, then the *composition* of g and f is denoted by $g \circ f$ and is defined to be $(g \circ f)(x) = g(f(x))$ for any $x \in \mathbb{R}^n$.

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is said to be *piecewise affine* (PWA) if there exist a partition $\mathcal{P} = \{P_i\}_{i\in(k]}$ of $\mathbb{R}^n$ into polyhedral sets and a set of $m \times n$ matrices $\mathcal{A} = \{A_i\}_{i\in(k]}$ and a set of m-dimensional vectors $\mathcal{B} = \{b_i\}_{i\in(k]}$, such that for all $x \in \mathbb{R}^n$, $f(x) = A_i x + b_i$ when $x \in P_i$. We refer to the tuple $(\mathcal{P}, \mathcal{A}, \mathcal{B})$ or equivalently $\{(P_i, A_i, b_i)\}_{i\in(k]}$ as a PWA *representation* of f.

A Rectified Linear Unit (ReLU) is a function from $\mathbb{R}$ to $\mathbb{R}$, which maps a value to itself if positive and to 0, otherwise. We use σ to represent the ReLU function, that is, for an $x \in \mathbb{R}$, $\sigma(x) = \max(x, 0)$. A multidimensional ReLU function extends this operation to $\mathbb{R}^n$, that is, for $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, $\sigma(x) = (\sigma(x_1), \ldots, \sigma(x_n))$. Also, we will use $\sigma_{[1,i]}$ to denote the function $\sigma_{[1,i]}(x) = (\sigma(x_1), \ldots, \sigma(x_i), x_{i+1}, \ldots, x_n)$, that applies σ to only the first i components of the vector.

III. Discrete-time Control Systems

In this section, we describe the discrete-time closed-loop systems with general state-feedback controllers, PWA controllers and neural network controllers.

A. Closed-loop Systems

We consider closed-loop systems with discrete-time linear dynamics for the plant and a general state-feedback controller. Formally, the closed-loop system is given by $\mathcal{S} = (A, B, G)$, where A and B are matrices in $\mathbb{R}^{n\times n}$ and $\mathbb{R}^{n\times m}$, respectively, and $G : \mathbb{R}^n \to \mathbb{R}^m$ is a state-dependent function. It represents the following system:

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ u(k) = G(x(k)) \end{cases} \tag{1}$$

where state $x(k) \in \mathbb{R}^n$ and control input $u(k) \in \mathbb{R}^m$. An execution of $\mathcal{S}$ starting from x_0 is a sequence $\eta = x(0), x(1), \ldots$ such that $x(0) = x_0$ and $x(k+1) = Ax(k) + BG(x(k))$ for every $k \in \mathbb{N}_{>0}$.

B. Piecewise Affine Controllers

Next, we consider the special case where the controllers are PWA functions. Consider a system $\mathcal{S} = (A, B, G)$, where is G is a PWA function given by the PWA representation $(\mathcal{P}, \mathcal{C}, \mathcal{D})$ with $\mathcal{P} = \{P_i\}_{i\in(r]}$, $\mathcal{C} = \{C_i\}_{i\in(r]}$ and $\mathcal{D} = \{d_i\}_{i\in(r]}$. Then, the closed-loop system can be expressed as the switched (discrete-time) affine system (SAS) $x(k+1) = (A + BC_i)x(k) + Bd_i$ when $x(k) \in P_i$ for every $i \in (r]$.

Switched affine systems are an important class of systems, as they can approximate non-linear systems with arbitrary precision [22], [23]. Verification of several properties including safety [24]–[26] and stability [20], [27] have been extensively investigated for SASs.

C. Neural Network-controllers

Next, we consider closed-loop systems where the controller is a neural network. A neural network (NN) consists of a finite collection of layers, each of which consists of a finite set of nodes, and a finite set of edges connecting nodes of adjacent layers. Each node has an associated bias and an activation function, and each edge has an associated weight. We assume the activation function to be the ReLU function. Hence, we represent the neural network by a set of pairs of matrices and vectors capturing the inter-layer weights and layer biases.

Definition 1. *A neural network is a set $\mathcal{N} = \{(W_i, b_i)\}_{i\in(L]}$, where L is the number of layers, for each $i \in (L]$, W_i is a matrix of dimension $n_i \times n_{i-1}$, and represents the weights on the edges from layer $i-1$ to i, and b_i is a vector of dimension n_i representing the biases associated with the nodes in layer i.*

Let us fix an NN $\mathcal{N} = \{(W_i, b_i)\}_{i\in(L]}$ for the rest of the paper. The control function G represented by the neural network $\mathcal{N}$ is given by $[\![\mathcal{N}]\!]$, which is defined as a composition of the functions represented by individual layers.

Definition 2. *For each $i \in (L]$, $[\![\mathcal{N}]\!]_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ captures the function corresponding to the i-th layer of the NN, and is given by $[\![\mathcal{N}]\!]_i(y) = \sigma(W_i y + b_i)$ for all $y \in \mathbb{R}^{n_{i-1}}$. The input-output function for the NN is given by $[\![\mathcal{N}]\!] : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ where $[\![\mathcal{N}]\!] = [\![\mathcal{N}]\!]_L \circ \ldots \circ [\![\mathcal{N}]\!]_1$.*

IV. PWA representation computation

Our broad objective is to use the large body of work on switched linear system verification toward verification of neural network-controlled dynamical systems. The main contribution of this paper is a novel algorithm for transforming the neural network control function $[\![\mathcal{N}]\!]$ into an equivalent PWA representation. Note that $[\![\mathcal{N}]\!]$ is indeed a piecewise affine function, is an established observation. Although such algorithms for NN with one [17] and more layers [21] have been provided, our algorithm is compositional in nature and provides several benefits, including being amenable to parallelization and efficient updates when neural networks evolve.

A. Algorithm

In this section, we provide an iterative algorithm to compute a PWA representation of a given NN $\mathcal{N}$. The main idea consists of constructing a PWA representation for a specific layer $[\![\mathcal{N}]\!]_i$, and then composing these representations. The crux of our algorithm consists of two subroutines: (1) PWA(W,b) given by Algorithm 1 that provides a PWA representation of function $f_{W,b}(x) = \sigma(Wx+b)$ for any matrix W and vector b corresponding to a single layer of a neural network (correctness provided in Proposition 1) (2) Compose($(\mathcal{P},\mathcal{A},\mathcal{B}),(\mathcal{Q},\mathcal{C},\mathcal{D})$) given by Algorithm 2 that computes the PWA representation of the composition of the functions represented by the PWA representations $(\mathcal{P},\mathcal{A},\mathcal{B})$ and $(\mathcal{Q},\mathcal{C},\mathcal{D})$, respectively (correctness provided in Proposition 2).

Algorithm 1 PWA(W, b)

Require: A matrix $W \in \mathbb{R}^{n\times m}$, a vector $b \in \mathbb{R}^n$
Ensure: PWA representation $(\mathcal{P},\mathcal{A},\mathcal{B})$ of $f_{W,b}$
1: Set $\mathcal{P} = \{\mathbb{R}^m\}$, $\mathcal{A} = \{W\}$ and $\mathcal{B} = \{b\}$
2: **for** $i \in (n]$ **do**
3: Set $\mathcal{P}', \mathcal{A}', \mathcal{B}'$ to be empty sets
4: **for** $(P, A, b') \in (\mathcal{P},\mathcal{A},\mathcal{B})$ **do**
5: $P^0, P^1 = \text{Split}(P, W, b, i)$
6: **for** j = 0, 1 **do**
7: **if** $P^j \neq \emptyset$ **then**
8: Add $(P^j, M_i^j A, M_i^j b')$ to $(\mathcal{P}', \mathcal{A}', \mathcal{B}')$
9: **end if**
10: **end for**
11: **end for**
12: Set $(\mathcal{P},\mathcal{A},\mathcal{B})$ to be $(\mathcal{P}', \mathcal{A}', \mathcal{B}')$
13: **end for**
14: **return** $(\mathcal{P},\mathcal{A},\mathcal{B})$

Algorithm 1 starts with a partition consisting of a single region with the affine function $f(x) = Wx + b$ assigned to it, and then mimics the behavior of applying ReLU iteratively to each dimension. Split(P, W, b, i) splits the region P into $P^0 = P \cap \{x \,|\, [W]_i x + [b]_i \leq 0\}$ and $P^1 = P \cap \{x \,|\, [W]_i x + [b]_i \geq 0\}$, where $[W]_i$ and $[b]_i$ represent the i-th rows. That is, it splits P based on the sign of the i-th neuron on the application of $Wx+b$. In Line 8, for each of the regions P^0 and P^1 that are not empty, the affine dynamics is updated to reflect the application of ReLU. More precisely, M_i^j is an identity matrix if $j = 1$ and is an identity matrix with the i-th diagonal element 0 otherwise. Hence, M_i^1, the identity matrix, is applied to $Ax + b'$ for the case P^1, but M_i^0, the matrix that sets the i-component to 0, is applied in the case of P^0. The following proposition formalizes the correctness of Algorithm 1.

Proposition 1. *The $(\mathcal{P},\mathcal{A},\mathcal{B})$ output by Algorithm 1 is a PWA representation of $f_{W,b}(x) = \sigma(Wx+b)$ for $x \in \mathbb{R}^m$.*

Proof. Initially, in Line 1 $(\mathcal{P},\mathcal{A},\mathcal{B})$ represents the function $f(x) = Wx + b$. The following invariant is maintained: For each i, $(\mathcal{P},\mathcal{A},\mathcal{B})$ at the end of the i-th iteration of the loop (Line 12) represents the function $f_{[1,i]}(x) = \sigma_{[1,i]}(Wx+b)$. This is clear since each iteration mimics the application of σ on the i-th components. Hence, when the loop terminates, the PWA represents the function $f_{W,b}(x) = \sigma(Wx + b)$. □

Remark 1. *Note that in general P could be split into $\{0,1\}^n$ regions if we consider two splits per neuron due to the ReLU application. Our algorithm is iterative and does not explicitly explore all of these possible splittings. Instead, it iteratively splits based on a certain neuron i and only continues forward with those regions that are not empty. Hence, 2^{n-i} combinations that correspond to an empty region are eliminated. These polyhedral regions are efficiently computed by using the Parma Polyhedra Library [28].*

Algorithm 2 Compose($(\mathcal{P},\mathcal{A},\mathcal{B}),(\mathcal{Q},\mathcal{C},\mathcal{D})$)

Require: PWA $(\mathcal{P},\mathcal{A},\mathcal{B})$ for f and $(\mathcal{Q},\mathcal{C},\mathcal{D})$ for g
Ensure: PWA $(\mathcal{R},\mathcal{E},\mathcal{H})$ for the composition $g \circ f$
1: Set $\mathcal{R},\mathcal{E},\mathcal{H}$ to be empty sets
2: **for** $(P, A, b) \in (\mathcal{P},\mathcal{A},\mathcal{B})$ **do**
3: **for** $(Q, C, d) \in (\mathcal{Q},\mathcal{C},\mathcal{D})$ **do**
4: $R := \{x : x \in P, Ax + b \in Q\}$ ▷ Polyhedron
5: $E := CA$ ▷ Matrix
6: $h = Cb + d$ ▷ Vector
7: Add (R, E, h) to $(\mathcal{R},\mathcal{E},\mathcal{H})$
8: **end for**
9: **end for**
10: **return** $(\mathcal{R},\mathcal{E},\mathcal{H})$

Next, we present Algorithm 2 that computes the PWA representation of the composition of two given PWA representation. The correctness of the algorithm is summarized in the following proposition.

Proposition 2. *$(\mathcal{R}, \mathcal{E}, \mathcal{H})$ output by Algorithm 2 is a PWA representation of $h = g \circ f$, where f and g are the functions corresponding to the PWA representations $(\mathcal{P}, \mathcal{A}, \mathcal{B})$ and $(\mathcal{Q}, \mathcal{C}, \mathcal{D})$, respectively.*

Proof. Note that $\mathcal{P}$ is the partitioning of input space of f and $\mathcal{Q}$ is a partitioning of the input space of g (as well as the output space of f). The broad idea of the algorithm is to compute a refinement of $\mathcal{P}$, that is, a further split of the regions of $\mathcal{P}$ such that the new regions upon application of f will be contained within a region of $\mathcal{Q}$. This is accomplished by taking a region P of $\mathcal{P}$ and Q of $\mathcal{Q}$ iteratively and computing the subregion of P for which application of f leads to elements of Q. This is accomplished in Line 4. We need to assign the composed function $g \circ f$ corresponding to this region, which is given by $g \circ f(x) = C(Ax + b) + d = CAx + Cb + d$, where $f(x) = Ax + b$ for $x \in P$ and $g(y) = Cy + d$ for $y \in Q$. Hence, E is assigned CA and h is assigned $Cb + d$. □

Algorithm 3 ComputePWA($\mathcal{N}$, T)

Require: NN $\mathcal{N} = \{(W_i, b_i)\}_{i \in (L]}$, comp. ordering T
Ensure: PWA representation $(\mathcal{R}, \mathcal{E}, \mathcal{H})$ of $[\![\mathcal{N}]\!]$

```
1: for node v in T do
2:     if v is a leaf with label i then
3:         Return PWA(W_i, b_i)
4:     end if
5:     if v is an internal node then
6:         (P, A, B) = ComputePWA(N, T_l)
7:         (Q, C, D) = ComputePWA(N, T_r)
8:         (R, E, H) = Compose((P, A, B), (Q, C, D))
9:     end if
10: end for
11: return (R, E, H)
```

The PWA representation for $[\![\mathcal{N}]\!]$ is broadly computed by computing $[\![\mathcal{N}]\!]_i$ for each i and composing them. Note that composition is an associative operation, that is, $f \circ (g \circ h) = (f \circ g) \circ h$, and hence, there are multiple orderings in which compositions can be applied. First, we define a composition ordering to capture a valid order on the composition of the $[\![\mathcal{N}]\!]_i$s to obtain $[\![\mathcal{N}]\!]$.

Recall that a full binary tree is a binary tree where each node has 0 or 2 children. The nodes with 0 children are referred to as leaves and those with 2 children are referred to as internal nodes. The yield of the binary tree is the sequence of labels of the leaves from left to right.

Definition 3. *A composition ordering T on a neural network with L layers is a full binary tree whose yield is the sequence $1, 2, \ldots, L$.*

Intuitively, each leaf with a label i from $(L]$ represents the function $[\![\mathcal{N}]\!]_i$, and each internal node corresponds to the composition $g \circ f$ of the functions f and g represented by the left and right children, respectively. Figure 1 shows two composition orderings. The left balanced tree T_1 composes Layer 1 and Layer 2, and composes Layer 3 and Layer 4, and then composes the results of those two. The right skewed tree T_2, composes Layer 3 and 4, composes Layer 2 with the result of the previous composition, and finally composes Layer 1 with the result of the last composition.

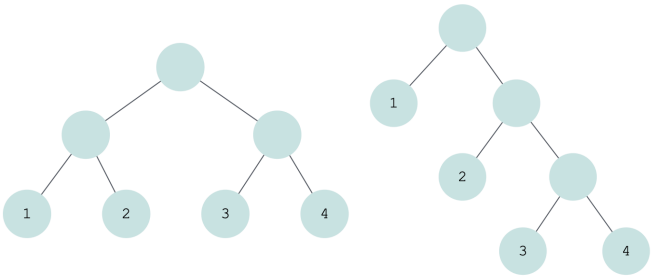

Fig. 1: Two composition orderings

Algorithm 3 summarizes the computation of the PWA representation of $[\![\mathcal{N}]\!]$ by composing PWA representations of the layers in the order specified by a composition ordering T. Note that the algorithm returns the same PWA representation irrespective of the composition ordering, however, some ordering are more beneficial than others for parallelization and efficient updates. For instance, the balanced tree T_1 allows for the computation of the compositions of Layer 1 and Layer 2 and Layer 3 and Layer 4 in parallel, while the structure of T_2 prohibits such parallel applications of Algorithm 2.

Another practically relevant scenario is the need to compute PWA representation in the presence of neural network retraining that happens as new data becomes available. Suppose that only the parameters of a specific layer change. To update the PWA representation of the evolved neural network we need to run Algorithm 1 on the leaf corresponding to this layer, and Algorithm 2 on the compositions corresponding to the internal nodes on the path from that leaf to the root. Specifically, in Figure 1, if Layer 4 is updated, for the T_1 ordering one would need to update compositions corresponding to two internal nodes, whereas in T_2, three compositions would need to be recomputed. This difference could be larger for balanced and skewed tree corresponding to deep networks with balanced trees requiring $O(\log L)$ updates where as skewed trees requiring $O(L)$ updates.

B. Experimental Evaluation

We performed an extensive study of our algorithm on neural networks with varying number of layers L and maximum number of neurons per layer N. Table I reports a selection of the results for illustrative purposes and reports B and R - the theoretical bound on the number of possible regions and the actual number of regions in the PWA representation of the neural network, and $T1$ and $T2$ - the total time spent executing Algorithm 1 and Algorithm 2 in seconds within Algorithm 3. Extended experimental results are available at this link.

L	N	B	R	T1 (s)	T2 (s)
2	6	32	8	0.000547	0.003884
3	6	2048	13	0.002007	0.034174
4	8	1.05e+06	205	0.018842	12.312506
5	7	8.39e+06	10	0.015078	0.526416
5	7	3.36e+07	6519	0.021295	18.146870
5	7	2.68e+08	12787	0.021867	137.985291

TABLE I: Study of Algorithm 3

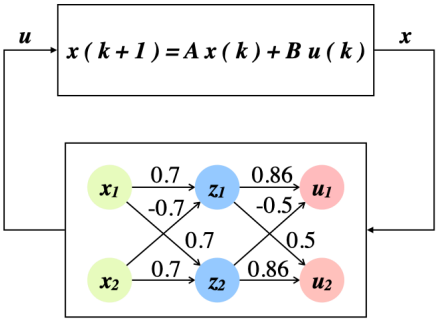

Fig. 2: System with neural network controller

Note that, in general, there is no correlation between the number of regions in the PWA representation and the architecture/size of the neural network. In fact, the actual number of regions is significantly smaller than the theoretical bound $O(2^N)$. This difference occurs because Algorithm 2 may generate empty regions, which are retained and efficiently pruned by Algorithm 3. A large number of empty regions will, in fact, accelerate the construction process. On the other hand, the total time taken by the algorithm (T1 + T2) increases with the number of regions in the PWA representation. We observe that most of the computation time is spent executing Algorithm 2. In contrast, the total computation time for PWA representation computation for individual layers (Algorithm 1) is negligible. Note that the composition time increases with the number of compositions since the PWA representations also grow in size with subsequent compositions.

V. Illustration with Stability Analysis

In this section, we illustrate an end-to-end stability analysis of a neural network-controlled linear dynamical system by first computing a PWA representation of the neural network, then computing a switched affine system representation of the closed-loop system and finally performing the stability analysis on the latter.

A. Closed-loop system description

We consider the closed-loop system in Figure 2, where the matrices for the linear dynamical system are given by:

$$A = \begin{bmatrix} -0.17 & 0.17 \\ 0.41 & 0.71 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0.2 & 0.32 \\ 0.66 & -0.5 \end{bmatrix}.$$

For the system $x(t+1) = Ax(t) + Bu(t)$, with $u(t) = G(x(t))$ to have an equilibrium point at 0, we will assume $G(0) = 0$. Since, G is now represented by a neural network, we will consider a neural network with 0 biases to achieve this condition. We will explore the stability of the closed-loop system with respect to the equilibrium point $x(0) = 0$.

The neural network controller consists of one hidden layer and two nodes at each layer. Concretely, W_1 and W_2 capture the edge weights from the input layer to the hidden layer, and from the hidden layer to the output layer, respectively, and are given by:

$$W_1 = \begin{bmatrix} 0.7 & -0.7 \\ 0.7 & 0.7 \end{bmatrix} \text{ and } \qquad W_2 = \begin{bmatrix} 0.86 & -0.5 \\ 0.5 & 0.86 \end{bmatrix}.$$

Biases b_1 and b_2 are 0 vectors. ReLU function is applied at each node of the hidden and output layers.

B. Illustration of Algorithm 1

Let us consider the function $f_{W_2}(z) = \sigma(W_2 z)$. Algorithm 1 will construct the PWA representation $(\mathcal{Q}, \mathcal{C}, \mathcal{D})$, which in this particular case is just a piecewise linear (PWL) function, since $\mathcal{D}$ will only consist of 0 vectors. Hence, we will drop the third component from our illustration and referred to the system as PWL. Consider $q_1(z) = 0.86z_1 - 0.5z_2$ and $q_2(z) = 0.5z_1 + 0.86z_2$ corresponding to the two rows of W_2. The $\mathcal{Q}$ will correspond to splitting $\mathbb{R}^2$ with respect to the hyperplanes defined by q_1 and q_2, that is, partitioning $\mathbb{R}^2$ with $q_1(z) \geq 0$, $q_1(z) < 0$ and then partitioning each of the non-empty regions with $q_2(z) \geq 0$, $q_2(z) < 0$. In this case, all four regions Q_1, Q_2, Q_3, and Q_4 are non-empty, and are added to the partition $\mathcal{Q}$. The associated matrices are added to $\mathcal{C}$. For instance, Q_2 is determined by the constraints $q_1(z) = 0.86z_1 - 0.5z_2 < 0$, and $q_2(z) = 0.5z_1 + 0.86z_2 \geq 0$, For instance, the matrix corresponding to Q_2 is $C_2 = M_1^0 M_2^1 W_2 = \begin{bmatrix} 0 & 0 \\ 0.5 & 0.86 \end{bmatrix}$.

Similarly, we can compute the PWL representation $(\mathcal{P}, \mathcal{A})$ of f_{W_1}. Specifically, $P_1 = \{x \in \mathbb{R}^2 \,|\, 0.7x_1 - 0.7x_2 \geq 0, 0.7x_1 + 0.7x_2 \geq 0\}$ and $A_1 = \begin{bmatrix} 0.7 & -0.7 \\ 0.7 & 0.7 \end{bmatrix}$.

C. Illustration of Algorithm 2

Next, we will apply Algorithm 2 on $(\mathcal{P}, \mathcal{A})$ and $(\mathcal{Q}, \mathcal{C})$ to compute the PWL representation of $(\mathcal{R}, \mathcal{E})$ of the complete neural network. Specifically, we will illustrate the loop for the regions P_1 and Q_2, which gives rise to the region R_{12} and corresponds to the constraints $p_1(x) = 0.7x_1 - 0.7x_2 \geq 0$, $p_2(x) = 0.7x_1 + 0.7x_2 \geq 0$, $q_1(M_1^0 M_2^1 W_2 x) = 0.252x_1 - 0.952x_2 < 0$, and $q_2(M_1^0 M_2^1 W_2 x) = 0.952x_1 + 0.252x_2 \geq 0$. The matrix associated with R_{12} is $C_2 A_1$ and is given by $\begin{bmatrix} 0 & 0 \\ 0.952 & 0.252 \end{bmatrix}$. The complete PWA representation consists of eight regions, namely, $R_{11}, R_{12}, R_{22}, R_{21}, R_{41}$ each with a unique dynamics and the regions R_{33}, R_{34}, R_{44} with the same dynamics.

D. Stability analysis

Finally, we compose the PWA representation of the neural network with the linear dynamics and obtain the switched linear system with the same partition $R_{11}, R_{12}, R_{22}, R_{21}, R_{41}, \ R_{33}, R_{34}, R_{44}$ and updated dynamics. We note that the matrix norm for each of the resulting matrices is < 1, therefore the system with any arbitrary switching between these matrices is stable, specifically the one computed which has more restricted switching – it will switch under certain constraints.

VI. Conclusions

In this paper, we presented a novel compositional algorithm for the computation of a PWA system representation of a neural network, that provides a switched affine system representation of a neural network-controlled linear dynamical system. This opens up the possibility of the use of switched system verification techniques toward neural network-controlled system verification. Our study of the compositional algorithm for piecewise affine system representation computation of the neural network input-output function highlights the network parameters that affect the runtime. We have illustrated the application of the algorithm and the translation to the stability analysis of a closed-loop system. In the future, we plan to explore rigorous automated verification approaches for different properties of neural network-controlled systems, including safety and stability, using a switched system representation. Also, it would be interesting to explore the benefits of a piecewise affine representation to the interpretability of neural networks.

References

[1] A. Voevoda and D. Romannikov, "Synthesis of neural networks for solving optimal control problems," *IOP Conference Series: Materials Science and Engineering*, vol. 953, no. 1, p. 012065, 2020.

[2] S. Cerf and E. Rutten, "Combining neural networks and control: potentialities, patterns and perspectives," in *IFAC 2023 - 22nd World Congress of the International Federation of Automatic Control*, 2023.

[3] P. Prabhakar and Z. R. Afzal, "Abstraction based output range analysis for neural networks," in *Annual Conference on Neural Information Processing Systems*, 2019.

[4] P. Prabhakar, "Bisimulations for neural network reduction," in *International Conference on Verification, Model Checking, and Abstract Interpretation*, pp. 285–300, Springer, 2022.

[5] Y. Zhong, R. Wang, and S. Khoo, "Expediting neural network verification via network reduction," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023*, pp. 1263–1275, IEEE, 2023.

[6] S. Dutta, X. Chen, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Sherlock - A tool for verification of neural network feedback systems: demo abstract," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC*, pp. 262–263, ACM, 2019.

[7] 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*, pp. 443–452, Springer, 2019.

[8] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification," *arXiv:2103.06624*, 2021.

[9] H. 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 *Formal Methods - The Next 30 Years - Third World Congress, FM 2019*, vol. 11800 of *Lecture Notes in Computer Science*, pp. 670–686, Springer, 2019.

[10] S. Bak, T. Dohmen, K. Subramani, A. Trivedi, A. Velasquez, and P. Wojciechowski, "The octatope abstract domain for verification of neural networks," in *25th International Symposium on Formal Methods*, vol. 14000, pp. 454–472, Springer, 2023.

[11] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. T. Vechev, "PRIMA: general and precise neural network certification via scalable convex hull approximations," *Proc. ACM Program. Lang.*, vol. 6, no. POPL, pp. 1–33, 2022.

[12] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, pp. 41:1–41:30, 2019.

[13] 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 *IEEE symposium on security and privacy (SP)*, 2018.

[14] A. Abate, A. Edwards, and M. Giacobbe, "Neural abstractions," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2022.

[15] A. S. Poznyak, W. Yu, E. N. Sanchez, and J. P. Perez, "Stability analysis of dynamic neural control," *Expert Systems with Applications*, vol. 14, no. 1, pp. 227–236, 1998.

[16] M. Korda, "Stability and performance verification of dynamical systems controlled by neural networks: algorithms and complexity," 2022.

[17] P. Samanipour and H. A. Poonawala, "Stability analysis and controller synthesis using single-hidden-layer relu neural networks," *IEEE Transactions on Automatic Control*, vol. 69, no. 1, pp. 202–213, 2024.

[18] W. Wu, J. Chen, and J. Chen, "Stability analysis of systems with recurrent neural network controllers," *IFAC-PapersOnLine*, vol. 55, no. 12, pp. 170–175, 2022.

[19] R. Ivanov, K. Jothimurugan, S. Hsu, S. Vaidya, R. Alur, and O. Bastani, "Compositional learning and verification of neural network controllers," vol. 20, Sept. 2021.

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

[21] H. Robinson, A. Rasheed, and O. San, "Dissecting deep neural networks," 2020.

[22] T. Dang, O. Maler, and R. Testylier, "Accurate hybridization of nonlinear systems," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, p. 11–20, 2010.

[23] D. Li, S. Bak, and S. Bogomolov, "Reachability analysis of nonlinear systems using hybridization and dynamics scaling," in *Formal Modeling and Analysis of Timed Systems*, pp. 265–282, 2020.

[24] M. Anand, R. Jungers, M. Zamani, and F. Allgöwer, "Path-complete barrier functions for safety of switched linear systems," in *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 7423–7428, 2024.

[25] P. S. Duggirala and A. Tiwari, "Safety verification for linear systems," in *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, pp. 1–10, 2013.

[26] R. Alur, "Formal verification of hybrid systems," in *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, pp. 273–278, 2011.

[27] M. Ogura and C. Martin, "Generalized joint spectral radius and stability of switching systems," *Linear Algebra and its Applications*, vol. 439, no. 8, pp. 2222–2239, 2013.

[28] R. Bagnara, P. M. Hill, and E. Zaffanella, "The parma polyhedra library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems," *Sci. Comput. Program.*, vol. 72, no. 1–2, 2008.