Abstraction-based Safety Analysis of Linear Dynamical Systems with Neural Network Controllers

Ratan Lal[1] Pavithra Prabhakar[2]

Abstract— **We consider the safety verification problem of a closed-loop discrete-time linear dynamical system with a neural network controller. The crux of safety verification relies on computing output reachable sets of the dynamical system and the neural network. Reachable set computation time of the neural network grows with the network size. To address the scalability issue, our main approach consists of abstracting the neural network controller into a smaller *annotated interval neural network* (AINN), and using this to compute an over-approximation of the reachable set of the closed-loop system. We present a novel approach for output reachable set computation of an AINN by decomposing it into two reachable set computation problems on neural networks, which we then compute using star sets. Our experimental analysis on two benchmarks demonstrate the trade-off in the precision and time for reachable set computation.**

I. Introduction

Neural networks are increasingly being deployed as controllers in safety-critical domains, which has necessitated rigorous analysis methods for guaranteeing the correct functioning of these systems. Of particular interest is safety analysis of closed loop dynamical systems with neural networks as controllers, which consists of checking if the states reached by the closed loop system after a given number of loop iterations do not intersect with an unsafe set. The crux of safety verification is the computation of the set of output valuations of the dynamical system and the neural network, which are then applied repeatedly to compute the reachable set of the closed loop system, and checked for intersection with an unsafe set.

Output reachable set computation of neural networks has received extensive attention in recent years [15]. Computation of the output set is computationally challenging (NP-complete); hence, efficient methods for computing precise over-approximations of the output set have been investigated (see related work for details). The large size of the network remains a challenge. We propose an orthogonal approach for over-approximating the input-output relation via a novel network reduction technique that is applicable to a wide range of activation functions.

A neural network is a deterministic system, that is, the output is unique given an input. To capture an over-approximation of the input-output, we require a non-deterministic system. We propose a novel data structure, *annotated interval neural network*, that allows interval weights and biases, and consist of a natural number annotation on the nodes for bookkeeping during the abstraction-refinement process. The choice of weights and biases from the corresponding intervals at each step of the computation allows multiple possible outputs for a given input. Our abstraction procedure consists of merging the nodes of a neural network based on a partitioning of the nodes. Biases and weights of the abstract system are interval hulls of the corresponding biases and weights in the concrete system. The abstraction procedure is sound for any continuous activation function. Our data structure is similar to the interval neural networks [22] with the addition of annotations on the nodes, however, the abstraction construction is simplified as we do not need scaling the interval hull of the weight by a factor equivalent to the number of nodes merged in the abstract source node. This is circumvented through the additional annotations. More interestingly, we can show that the abstraction is unique given a partition, irrespective of the sequence of abstractions performed to reach a certain partition, which is violated by the abstraction in [22]. Further, our abstraction technique is applicable to any continuous activation function; however, we need verification methodologies for analyzing the abstract INNs.

We present a novel algorithms for computing the output reachable set of the annotated interval neural network (AINN) $\mathcal{N}$, which consists of decomposing the reachable set computation problem on $\mathcal{N}$ into two reachable set computation problems on standard neural networks. More precisely, reachable set for AINN $\mathcal{N}$ is obtained by a convex combination of the reachable set of two neural networks $\mathcal{N}_l$ and $\mathcal{N}_u$, where $\mathcal{N}_l$ and $\mathcal{N}_u$ represent the neural networks with the same architecture as $\mathcal{N}$ but with weights and biases that are the lower and upper bounds of the corresponding intervals. We use the star set based method [28] to compute the reachset set of $\mathcal{N}_l$ and $\mathcal{N}_u$ and use that to compute the reachset of $\mathcal{N}$, and then use that to compute the reachset for the linear dynamics (linear transformations of star sets can be computed efficiently). We iterate the procedure k times to compute the k-step reachable set of the closed loop linear system and use that for safety analysis.

We have implemented our abstraction based safety analysis procedure in a Python toolbox, and evaluated on two benchmarks, namely, ACAS XU and the Translational Oscillators with Rotating Actuator (TORA). Our experimental results illustrate the trade-off between the size of the abstraction, the verification time, and the safety conclusions.

[1]Ratan Lal is a faculty in the school of computer science and information systems at Northwest Missouri State University, USA rlal@nwmissouri.edu

[2]Pavithra Prabhakar is a faculty in the department of computer science at Kansas State University, USA pprabhakar@ksu.edu

II. Related Work

Verification of neural networks has gained momentum in recent years [2], [3], [23], [33], [32], [15]. In recent decades, neural network has been popularly used for the controller of cyber physical systems. Although many safety verification of neural network controlled dynamical systems [13], [25], [26], [16] have been proposed, the main challenge is the safety verification of neural networks. For the neural networks, satisfiability solving based approaches that encode the verification problem as satisfiability modulo theory (SMT) solving problem, have been investigated. In particular, Reluplex [17], Planet [9], NeVer [24], and VeriDeep/DLV [12] can perform satisfiability checking of SMT formulas with linear constraints and ReLU operations. An SMT based counterexample guided abstraction refinement (CEGAR) approach has been explored in [23], [10]. A reduction to efficient mixed integer linear programming (MILP) for reachability analysis have been explored in [4], [7], [2], [30]. Methods for over-approximation of the reachable set of neural networks, including abstract interpretation [6] based methods that consider abstract domains such as boxes, zonotopes, polyhedra, star sets and Bernstein polynomials have been explored [11], [20], [34], [14], [31]. Approaches based on convex optimization [19], [8], [27], interval analysis [29], [21], [33], and linear approximation [35] have been explored. Network reduction techniques have been explored [22], [10], [1].

III. Preliminaries

a) Numbers and Sets: Let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{IR}$ and $\mathbb{N}$ denote the set of real numbers, the set of positive real numbers, the set of all real intervals, and the set of natural numbers, respectively. We say that an interval $[a, b] \in \mathbb{IR}$ is singular if $a = b$. Given an interval $[a, b]$, we use $\underline{[a, b]}$ to denote a and $\overline{[a, b]}$ to denote b. We use $[n]$ and $(n]$ to denote the set $\{0, 1, \dots, n\}$ and $\{1, 2, \dots, n\}$, respectively. Given a set $\mathcal{S}$, we use $|\mathcal{S}|$ to denote the number of elements in the set.

b) Relations and Functions: Given three sets $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and relations $R_1 \subseteq \mathcal{S}_1 \times \mathcal{S}_2$, $R_2 \subseteq \mathcal{S}_2 \times \mathcal{S}_3$, the composition of R_1 and R_2 denoted by $R_1 \diamond R_2$ is the subset $\{(s_1, s_3) \in \mathcal{S}_1 \times \mathcal{S}_3 \mid \exists\ s_2 \in \mathcal{S}_2,\ (s_1, s_2) \in R_1,\ (s_2, s_3) \in R_2\}$ of $\mathcal{S}_1 \times \mathcal{S}_3$. A function $f : \mathcal{S}_1 \to \mathcal{S}_2$ is bijection if it is one-one and onto function. Given functions $f : \mathcal{S}_1 \to \mathcal{S}_2$ and $g : \mathcal{S}_2 \to \mathcal{S}_3$, we use $g \circ f$ for the composition of the functions f and g, that is, $g \circ f : \mathcal{S}_1 \to \mathcal{S}_3$, where $g \circ f(s_1) = g(f(s_1))$. Given a set $\mathcal{S}$, a valuation on $\mathcal{S}$ is a function $f : \mathcal{S} \to \mathbb{R}$. We will use $Val(\mathcal{S})$ to denote the set of all valuations on $\mathcal{S}$.

c) Convex Hull: Let $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n$ be subsets of $\mathbb{R}^n$. Convex hull of $\mathcal{S}_i$s, $i \in (n]$ denoted by $\mathrm{CH}(\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n)$ is the smallest convex set that contains all $\mathcal{S}_i$s, $i \in (n]$.

d) Partition: A partition $\mathcal{P}$ of a given set $\mathcal{S}$ is a collection of some subsets of $\mathcal{S}$ such that $\mathcal{S} = \bigcup_{P \in \mathcal{P}} P$ and $P \cap P' = \emptyset$ for $P, P' \in \mathcal{P}$, $P \neq P'$.

IV. Interval Neural Networks

Recall that a neural network (NN) is a computational model, that consists of an input layer, one or more hidden layers, and an output layer of neurons (or nodes). Each neuron in the hidden layers and the output layer is labeled with a bias and an activation function. Each edge between two neurons in two consecutive layers is labeled with a weight. In this paper, we consider interval neural networks (INNs), which extend NNs, by allowing interval values for biases and edges, and annotating the neurons in the hidden and output layers with a natural number, that allows to keep relationship between concrete and abstract neurons. Next, we formally define the INN.

Definition 1: An interval neural network (INN) is a tuple $\mathcal{N} = \big(k, \mathcal{A}ct, \{\mathcal{S}_i\}_{i \in [k]}, \{\mathcal{L}_i\}_{i \in [k]}, \{\mathcal{W}_i\}_{i \in (k]}, \{b_i\}_{i \in (k]}, \{\mathcal{A}_i\}_{i \in (k]}\big)$, where

- k is the number of layers;
- $\mathcal{A}ct$ is a set of activation functions;
- For each $i \in [k]$,
  - $\mathcal{S}_i$ is a set of neurons in layer i;
  - $\mathcal{L}_i : \mathcal{S}_i \to \mathbb{N}$ is a function that annotates each neuron in layer i with a natural number;
- For each $i \in (k]$,
  - $\mathcal{W}_i : \mathcal{S}_{i-1} \times \mathcal{S}_i \to \mathbb{IR}$ is a function that assigns a real number interval to each edge between layer $i-1$ and layer i;
  - $b_i : \mathcal{S}_i \to \mathbb{IR}$ is a function that assigns a real number interval to each neuron in layer i;
  - $\mathcal{A}_i : \mathcal{S}_i \to \mathcal{A}ct$ is a function that assigns one of the activation functions from the set $\mathcal{A}ct$ to each neuron in layer i.

In the rest of this paper, we will use $\mathcal{N}$ to denote an interval neural network. We consider the class of activation functions to be the class of continuous functions from $\mathbb{R}$ to $\mathbb{R}$. We list here some of the common activation functions, which can be easily seen to be continuous in Figure 1.

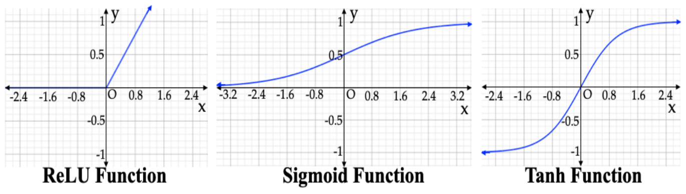

Fig. 1: Activation Functions

A neural network is a subclass of INNs, where weights and biases are singular intervals. Next, we illustrate INN with an example, where we consider ReLU activation function for the purpose of demonstration.

Example 1: Let us consider an INN $\mathcal{N}$ which is shown in Figure 2. $\mathcal{N}$ consists of three layers, that is ($k = 3$), namely an input layer, a hidden layer and an output layer, where there is one input neuron ($s_{0,1}$), four hidden neurons ($s_{1,1}, s_{1,2}, s_{1,3}, s_{1,4}$), and one output neuron ($s_{2,1}$), respectively. Here, $\mathcal{A}ct = \{\mathrm{ReLU}\}$ and each neuron is labeled with natural number 1. The biases associated with the hidden and output neurons are singular intervals $[1, 1]$, which is shown in Figure 2 as 1. Weights for the edges between the input and hidden neurons are singular intervals $[0,0]$, which is shown in Figure 2 as 0, and the weights between the hidden and output neurons are interval $[0,1]$.

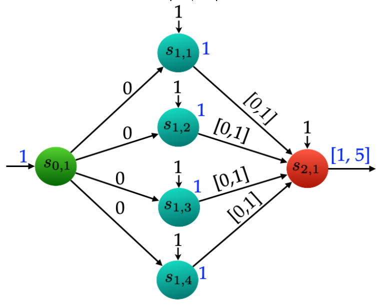

Fig. 2: INN

We capture the semantics of INN as a set of pairs of input-output valuations, where the output valuation corresponds to a possible valuation of the output layer, given the input valuation. The standard semantics of a neural network corresponds to an iterative computation of the values of the nodes layer by layer, wherein, the value of a node s' at a layer i is computed as the sum of the bias at s' and weighted sum of the values at the nodes s in the previous layer, where the weights correspond to the weight of the edge from s to s'. This is followed by an application of the activation function. The semantics of INN differs from that of NN in two ways. First, the edge weights and biases are intervals, so the valuation at s' can choose any weight and bias from the corresponding intervals. Further, weighting has another term, the annotation at node s. More precisely, in the INN, the valuation of a neuron s' in layer i is obtained by the application of the activation function corresponding to s' on the sum of a bias chosen from the bias interval for s' and a weighted sum over the valuation of the nodes s of layer $i-1$, where the weights correspond to the product of the annotation at s and a weight from the edge weight interval for s. The next definition formalizes the semantics of INN for a particular layer, which is then iteratively composed to obtain the input-output relation of the INN. Note that while the NN semantics is deterministic, the interval weights and biases render the semantics of INN non-deterministic. However, this is needed to express over-approximations of neural network semantics.

Definition 2: Given an INN $\mathcal{N}$ and a layer index $i \in (k]$, we define the semantics of $\mathcal{N}$ for the layer i, denoted by $[\![\mathcal{N}]\!]_i$ as follow:

$$[\![\mathcal{N}]\!]_i = \{(v,v') \in Val(\mathcal{S}_{i-1}) \times Val(\mathcal{S}_i) \mid \forall\ s' \in \mathcal{S}_i,$$
$$\exists\ \{w_{s,s'}\}_{s\in\mathcal{S}_{i-1}} \in \{\mathcal{W}_i(s,s')\}_{s\in\mathcal{S}_{i-1}},\quad b_{s'} \in b_i(s') \text{ s.t.}$$
$$v'(s') = \mathcal{A}_i(s')(\left(\sum_{s\in\mathcal{S}_{i-1}} w_{s,s'}\mathcal{L}_{i-1}(s)v(s)\right) + b_{s'})\}$$

We define the semantics of INN $\mathcal{N}$ as a composition of the semantics $[\![\mathcal{N}]\!]_i$s, $i \in (n]$, where $[\![\mathcal{N}]\!]_i$ captures all valid pairs of valuations for the neurons in layer $i-1$ and i.

Definition 3: Given an INN $\mathcal{N}$, its semantics denoted by $[\![\mathcal{N}]\!]$, is defined as:

$$[\![\mathcal{N}]\!] = [\![\mathcal{N}]\!]_1 \diamond [\![\mathcal{N}]\!]_2 \diamond \ldots \diamond [\![\mathcal{N}]\!]_k$$

Next, we illustrate the semantics of INN.

Example 2: We consider the INN shown in Figure 2. We start with valuation v for the input neuron $s_{0,1}$, which is 1. Then, we compute all valuations for the hidden neurons $s_{1,j}$, $j \in (4]$. From Definition 2, we obtain only one valuation v_1 shown in Figure 2 with blue color, where $v_1(s_{1,j}) = 1$, $j \in (4]$ because $v_1(s_{1,j}) = \text{ReLU}(w_{s_{0,1},s_{1,1}}\mathcal{L}_0(s_{0,1})v(s_{0,1}) + b_1(s_{1,1})) = \text{ReLU}(0*1*1+1) = 1$. Next, we compute all valuations for output neuron $s_{2,1}$ as a set given by: $\{v' \mid v'(s_{2,1}) = \text{ReLU}((\sum_{j=1}^{4} w_{s_{1,j},s_{2,1}}\mathcal{L}_1(s_{1,j})v_1(s_{1,j})) + b_i(s_{2,1}))\}$. Since weights lie in the interval $[0,1]$, the set of valuations v' for $s_{2,1}$ is the interval $[1,5]$.

V. Linear Dynamical Systems

In this section, we formalize an n-dimensional discrete-time linear dynamical system given as:

$$x(k+1) = Ax(k) + Bu(k), \tag{1}$$

using explicit state and input variables. Here, $x(k)$ denote the state at time k and $u(k)$ denotes the input at time k, and A and B are matrices. So, the state of time $k+1$ is a linear function of the state at time k and input at time k. Instead of treating state and inputs at time k as vectors, we treat them in the following formalization as valuations for state and input variables.

Definition 4: A linear dynamical system (LDS) is a tuple $\mathcal{D} = (\mathcal{S}_\mathcal{X}, \mathcal{S}_\mathcal{I}, \mathcal{M}_A, \mathcal{M}_B, \mathbb{S}, \mathbb{I})$, where

- $\mathcal{S}_\mathcal{X}$ is a set of state variables;
- $\mathcal{S}_\mathcal{I}$ is a set of input variables;
- $\mathcal{M}_A : \mathcal{S}_\mathcal{X} \times \mathcal{S}_\mathcal{X} \to \mathbb{R}$ is a function that assign a real value for each pair of state variables;
- $\mathcal{M}_B : \mathcal{S}_\mathcal{X} \times \mathcal{S}_\mathcal{I} \to \mathbb{R}$ is a function that assign a real value for each pair of state and input variable;
- $\mathbb{S} \subseteq \{v \mid v : \mathcal{S}_\mathcal{X} \to \mathbb{R}\}$ is a set of initial valuations for the state variables in $\mathcal{S}_\mathcal{X}$.
- $\mathbb{I} \subseteq \{v \mid v : \mathcal{S}_\mathcal{I} \to \mathbb{R}\}$ is a set of valuations for the input variables in $\mathcal{S}_\mathcal{I}$.

Example 3: Consider a 4-dimensional linear dynamics system given as below:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.5 \end{bmatrix} \begin{bmatrix} u(k) \end{bmatrix},$$

where $\mathcal{X}_0 = [0.6, 0.8] \times [-0.8, -0.6] \times [-0.5, -0.3] \times [0.5, 0.7]$, and $\mathcal{I} = [1, 2]$. This can be formally expressed as a 4-dimensional LDS $\mathcal{D} = (\mathcal{S}_\mathcal{X}, \mathcal{S}_\mathcal{I}, \mathcal{M}_A, \mathcal{M}_B, \mathbb{S}, \mathbb{I})$, where

- $\mathcal{S}_\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, $\mathcal{S}_\mathcal{I} = \{u\}$;
- $\mathcal{M}_A(x_1, x_2) = \mathcal{M}_A(x_2, x_3) = \mathcal{M}_A(x_3, x_4) = 1$, $\mathcal{M}_A(x_2, x_1) = -1$, and for other pair of states $(x, y) \in \mathcal{S}_\mathcal{X} \times \mathcal{S}_\mathcal{X}$, $\mathcal{M}_A(x, y) = 0$;
- $\mathcal{M}_B(x_4, u) = 0.5$, $\mathcal{M}_B(x, u) = 0$ for $x = x_1, x_2, x_3$;
- $\mathbb{S} = \{v \mid v(x_1) \in [0.6, 0.8], v(x_2) \in [-0.8, -0.6], v(x_3) \in [-0.5, -0.3], v(x_4) \in [0.5, 0.7]\}$;
- $\mathbb{I} = \{v \mid v(u) \in [1, 2]\}$.

Next, we define the semantics of LDS that captures the next state valuation given an initial state and input valuation.

Definition 5: Given an n-dimensional LDS $\mathcal{D} = (\mathcal{S}_{\mathcal{X}}, \mathcal{S}_{\mathcal{I}}, \mathcal{M}_A, \mathcal{M}_B, \mathbb{S}, \mathbb{I})$, we define the semantics of $\mathcal{D}$ as follow:

$$[\![\mathcal{D}]\!] = \{(v_1, v, v_2) \mid v_1 \in \mathbb{S},\ v \in \mathbb{I},\ \forall\ x \in \mathcal{S}_{\mathcal{X}},$$
$$v_2(x) = \sum_{x' \in \mathcal{S}_{\mathcal{X}}} \mathcal{M}_A(x, x')v_1(x') + \sum_{u \in \mathcal{S}_{\mathcal{I}}} \mathcal{M}_B(x, u)v(u)\}$$

Example 4: Consider the linear dynamical system shown in Example 3. Let v_1 be an initial state valuation, where $v_1(x_1) = 0.5$, $v_1(x_2) = -0.7$, $v_1(x_3) = -0.4$, and $v_1(x_4) = 0.6$, and input valuation v, where $v(u) = 1.5$. Then, the next valuation v_2 is evaluated as $v_2(x_1) = 1*0.7+0*1.5 = 0.7$, $v_2(x_2) = -1*0.5+1*(-0.4)+0*1.5 = -0.9$, $v_2(x_3) = 1*0.6+0*1.5 = 0.6$, $v_2(x_4) = 0+0.5*1.5 = 0.75$.

VI. Linear Dynamical Systems with INN

In this section, we introduce the linear dynamical systems controlled by interval neural networks. Here, the input to the linear dynamical system $x(k+1) = Ax(k) + Bu(k)$ is provided by the output of the neural network $\mathcal{N}$ which in turn takes the state of the linear dynamical system as input as shown in Figure 3. Our broad object is to analyze systems in which neural networks control the dynamical systems. However, the analysis can be expensive when we have large neural network. We will present an abstractions of neural networks into small interval neural network in order to scale the analysis. Next, we formally define syntax and semantics of a linear dynamical system with an INN.

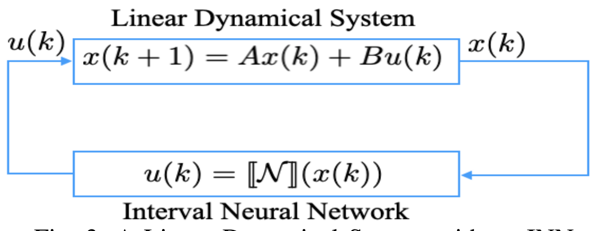

Fig. 3: A Linear Dynamical System with an INN

Definition 6: A linear dynamical system with an interval neural network (LDSINN) is a tuple $\mathcal{C} = (\mathcal{D}, \mathcal{N})$ where

- $\mathcal{D} = (\mathcal{S}_{\mathcal{X}}, \mathcal{S}_{\mathcal{I}}, \mathcal{M}_A, \mathcal{M}_B, \mathbb{S}, \mathbb{I})$ is a linear dynamical system;
- $\mathcal{N} = \big(k, \mathcal{A}ct, \{\mathcal{S}_i\}_{i\in[k]}, \{\mathcal{L}_i\}_{i\in[k]}, \{\mathcal{W}_i\}_{i\in(k]}, \{b_i\}_{i\in(k]}, \{\mathcal{A}_i\}_{i\in(k]}\big)$ is an interval neural network;
- $\mathcal{S}_{\mathcal{X}} = \mathcal{S}_0$ and $\mathcal{S}_{\mathcal{I}} = \mathcal{S}_k$.

Further, we define a semantics of LDSINN.

Definition 7: Given an LDSINN $\mathcal{C} = (\mathcal{D}, \mathcal{N})$, we define the semantics of $\mathcal{C}$ as follows: $[\![\mathcal{C}]\!] =$

$$\{(x_1, x_2) \mid \exists\, u \in Val(\mathcal{S}_k),\ (x_1, u) \in [\![\mathcal{N}]\!],\ (x_1, u, x_2) \in [\![\mathcal{D}]\!]\}.$$

Our broad objective is to perform safety analysis of linear dynamics systems with neural network controllers for a finite time horizon $k \in \mathbb{N}$. This require us to define the semantics of k-composition of a LDSINN $\mathcal{C} = (\mathcal{D}, \mathcal{N})$ denoted by $[\![(\mathcal{C}, k)]\!]$ that captures all pairs of initial state valuation and state valuation at k^{th} iteration of evolution through $\mathcal{D}$ and $\mathcal{N}$. $[\![(\mathcal{C}, k)]\!]$ is formally defined as follows:

$$[\![(\mathcal{C}, k)]\!] = \underset{1}{[\![\mathcal{C}]\!]} \diamond \underset{2}{[\![\mathcal{C}]\!]} \diamond \ldots \diamond \underset{k}{[\![\mathcal{C}]\!]}.$$

Problem 1: [Safety Problem] Given an LDSINN $\mathcal{C} = (\mathcal{D}, \mathcal{N})$, a set of unsafe state valuations $\mathbb{U}$, and number of steps $k \in \mathbb{N}$, verify whether there exist initial state valuation $v \in \mathbb{S}$ and $i \in (k]$ such that for some v', $(v, v') \in [\![(\mathcal{C}, i)]\!]$, and $v' \in \mathbb{U}$.

The crux for the safety analysis lies in computing the all valid pairs of initial state and state valuation for k-composition of LDSINN that consists of two computational steps, namely, (a) computation of control inputs from INN $\mathcal{N}$; (b) computation of the set of next state valuations with respect to the set of current state valuations and the set of control input valuations obtained from $\mathcal{N}$. The step (a) is computationally expensive, specifically, for the large neural networks. Hence, we present an abstraction technique based on a partitioning of the neuron-space that is sound. The abstraction technique reduces the large neural network into a smaller neural network expressed in the form of INN. The step (b) can be performed by linear transformation operation over current state and control input valuations.

VII. Abstraction of INN

In this section, we present an abstraction procedure that reduces a large INN $\mathcal{N}$ (or neural network) into a small INN $\widehat{\mathcal{N}}$ such that $\widehat{\mathcal{N}}$ is an over-approximation of $\mathcal{N}$. The procedure is based on partitioning the nodes in each layer of INN into a finite number of sets, and considering these sets as abstract nodes. We start with some preliminaries.

Definition 8 (Partition): A partition of an INN $\mathcal{N}$ is a sequence $\mathcal{P} = \{\mathcal{P}_i\}_{i\in[k]}$, where $\mathcal{P}_i$ is a partition of $\mathcal{S}_i$.
We allow the neurons in an INN to have different activation functions, however, we want to only merge nodes with the same activation function. Hence, we define a consistent partition.

Definition 9 (Consistent Partition): A partition $\mathcal{P} = \{\mathcal{P}_i\}_{i\in[k]}$ of an INN $\mathcal{N}$ is consistent if the following condition holds. For each $i \in (k]$,

$$\text{for each } \mathcal{S} \in \mathcal{P}_i,\ \text{ if } s_1, s_2 \in \mathcal{S},\ \text{ then } \mathcal{A}_i(s_1) = \mathcal{A}_i(s_2).$$

The broad abstraction procedure consists of merging the nodes of a particular set of a partition into an abstract node and instantiating the biases and the weights such that over-approximation is guaranteed. For the bias on an abstract node, we consider the convex hull of the biases associated with the nodes in the set corresponding to the abstract node. The weights on abstract edge is a bit tricky. As shown in [22], a convex hull of the weight intervals associated with the concrete edges corresponding to the abstract edges does not provide an over-approximation; however, "scaling" the interval by the number of nodes being merged in the source provides an over-approximation. However, such an over-approximation is not conducive to refinement, since, depending on the sequence of partitions used to construct the abstraction (with the "same" final partition), one ends up with different abstractions. Hence, we avoid the scaling, and instead store this information in the annotations, which provides a unique abstraction corresponding to a partition, irrespective of the sequence of node mergings applied to arrive at that abstraction. More precisely, for the abstract edge weight, we consider only the convex hull of the corresponding concrete edge weights (without any scaling) and annotate the abstract node with the number of merged nodes. Note that our semantics for INN has the effect of scaling with respect to the number of merged incoming nodes. Next, we provide the formal definition of the abstraction construction.

Definition 10: Given an INN $\mathcal{N}$ and its consistent partition $\mathcal{P} = \{\mathcal{P}_i\}_{i\in[k]}$, we define an abstract INN $\mathcal{N}/\mathcal{P} = \big(k, \mathcal{A}ct, \{\widehat{\mathcal{S}}_i\}_{i\in[k]}, \{\widehat{\mathcal{L}}_i\}_{i\in[k]}, \{\widehat{\mathcal{W}}_i\}_{i\in(k]}, \{\widehat{b}_i\}_{i\in(k]}, \{\widehat{\mathcal{A}}_i\}_{i\in(k]}\big)$, where

- for each $i \in [k]$,
  - $\widehat{\mathcal{S}}_i = \mathcal{P}_i$;
  - for each $\widehat{s} \in \widehat{\mathcal{S}}_i$, $\widehat{\mathcal{L}}_i(\widehat{s}) = \sum_{s\in\widehat{s}} \mathcal{L}_i(s)$;
- for each $i \in (k]$,
  - for each $\widehat{s} \in \widehat{\mathcal{S}}_{i-1}$, $\widehat{s}' \in \widehat{\mathcal{S}}_i$, $\widehat{\mathcal{W}}_i(\widehat{s}, \widehat{s}') = \mathrm{CH}(\{\mathcal{W}_i(s, s') \mid s \in \widehat{s},\ s' \in \widehat{s}'\})$;
  - for each $\widehat{s} \in \widehat{\mathcal{S}}_i$, $\widehat{b}_i(\widehat{s}) = \mathrm{CH}(\{b_i(s) \mid s \in \widehat{s}\})$;
  - for each $\widehat{s} \in \widehat{\mathcal{S}}_i$, $\widehat{\mathcal{A}}_i(\widehat{s}) = \mathcal{A}_i(s)$ for some $s \in \widehat{s}$.

In the rest of this paper, we use $\widehat{\ }$ to denote the components of the abstract INN $\mathcal{N}/\mathcal{P}$ as defined in Definition 10 in order to distinguish from the components of $\mathcal{N}$. Next, we illustrate with an example.

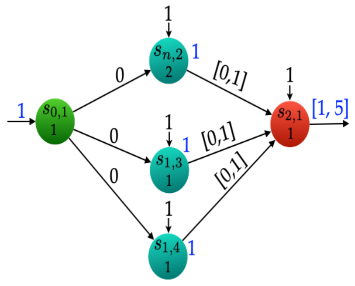

Fig. 4: Abstraction of INN in Fig. 2

Example 5: We consider the INN $\mathcal{N}$ shown in Figure 2 for the illustration. Let $\mathcal{P} = \{\mathcal{P}_i\}_{i\in[2]}$ be a partition of $\mathcal{N}$ where $\mathcal{P}_0 = \{\{s_{0,1}\}\}$, $\mathcal{P}_1 = \{\{s_{1,1}, s_{1,2}\}, \{s_{1,3}\}, \{s_{1,4}\}\}$, and $\mathcal{P}_2 = \{\{s_{2,1}\}\}$. Next, we construct an abstract INN $\mathcal{N}/\mathcal{P}$ according to Definition 10, where we merge neurons $s_{1,1}$ and $s_{1,2}$ into a new neuron $s_{n,2}$ with annotation 2 which is $(\mathcal{L}_1(s_{1,1}) + \mathcal{L}_1(s_{1,2}))$, activation function ReLU, and bias 1. Weight of the edge $(\{s_{0,1}\}, s_{n,2})$ and $(s_{n,2}, \{s_{2,1}\})$ are intervals $[0, 0]$ and $[0, 1]$, respectively because convex hull of two equal intervals is the same interval.

Next, we define an abstraction function for establishing the relation between the valuations of the concrete and abstract systems. The value for an abstract node corresponds to a weighted average of the values for the concrete nodes corresponding to it, which are weighted according to the annotation function values.

Definition 11: Given an INN $\mathcal{N}$ and its abstraction $\mathcal{N}/\mathcal{P}$, we define an abstraction function $\alpha_{(\mathcal{N},\mathcal{P})} = \{\alpha^i_{(\mathcal{N},\mathcal{P})}\}_{i\in[k]}$, where $\alpha^i_{(\mathcal{N},\mathcal{P})} : Val(\mathcal{S}_i) \to Val(\widehat{\mathcal{S}}_i)$ such that

$$\alpha^i_{(\mathcal{N},\mathcal{P})}(v)(\widehat{s}) = \frac{\sum_{s\in\widehat{s}} \mathcal{L}_i(s)v(s)}{\widehat{\mathcal{L}}_i(\widehat{s})}.$$

Remark 1: We avoid subscripts $\mathcal{N}$, $\mathcal{P}$ and superscript i from $\alpha^i_{(\mathcal{N},\mathcal{P})}$, when it is clear from the context.

Next, we show that given an input-output valuation (v, v') of an INN $\mathcal{N}$, its mapping under the abstraction function α, that is, $(\alpha(v), \alpha(v'))$ will be an input-output valuation of the abstract INN $\mathcal{N}/\mathcal{P}$, which is stated in the following theorem.

Theorem 1: Given an INN $\mathcal{N}$ and its abstraction $\mathcal{N}/\mathcal{P}$, we have the following:

$$\text{if } (v, v') \in [\![\mathcal{N}]\!], \text{ then } (\alpha(v), \alpha(v')) \in [\![\mathcal{N}/\mathcal{P}]\!].$$

The proof of Theorem 1 can be found on Github link.

In order to compare the input-output relations of the concrete and abstract systems, we need the neurons in the input and output layers not to be merged. Hence, we define un-merged partition.

Definition 12: A partition $\mathcal{P} = \{\mathcal{P}_i\}_{i\in[k]}$ of an INN $\mathcal{N}$ is un-merged if $\mathcal{P}_0 = \{\{s\} \mid s \in \mathcal{S}_0\}$, $\mathcal{P}_k = \{\{s\} \mid s \in \mathcal{S}_k\}$.

Note that when comparing the input-output relations of arbitrary INNs, we need to be able to map input and output valuations, of the two systems. We do this by creating bijections between the input and the output neurons of the two INNs.

Definition 13: Let $F_I : \mathcal{S}_I \to \mathcal{S}'_I$ and $F_O : \mathcal{S}_O \to \mathcal{S}'_O$ be two bijective functions, and $R \subseteq Val(\mathcal{S}'_I) \times Val(\mathcal{S}'_O)$ be a binary relation. We define $R/(F_I, F_O) = \{(v_1 \circ F_I, v_2 \circ F_O) \mid (v_1, v_2) \in R\}$.

We define an ordering on INNs using over-approximation relationship between two INNs.

Definition 14: Given two INNs $\mathcal{N}_1$ and $\mathcal{N}_2$, we say $\mathcal{N}_2$ is an over-approximation of $\mathcal{N}_1$, denoted by $\mathcal{N}_1 \preccurlyeq \mathcal{N}_2$ if there are bijection functions $F_I : \mathcal{S}^1_0 \to \mathcal{S}^2_0$ and $F_O : \mathcal{S}^1_k \to \mathcal{S}^2_k$ such that $[\![\mathcal{N}_1]\!] \subseteq [\![\mathcal{N}_2]\!]/(F_I, F_O)$.

Given an un-merged partition $\mathcal{P}$, abstract INN $\mathcal{N}/\mathcal{P}$ is always an over-approximation of $\mathcal{N}$, which is stated in the following theorem.

Theorem 2: Given an INN $\mathcal{N}$ and its un-merged partition $\mathcal{P}$, we have $\mathcal{N} \preccurlyeq \mathcal{N}/\mathcal{P}$.

VIII. CONSTRUCTION OF $\mathcal{N}_l$ AND $\mathcal{N}_u$

In this section, we present a new approach for computing the reachable set of an interval neural network by reducing it the verification of the traditional neural networks. Our broad ideas is that the reach set of an INN can be represented as a convex hull of reach sets of two neural networks, namely N_l and $\mathcal{N}_u$, referred to as the lower and upper neural networks. This holds as long as the inputs are all positive, or when all the inputs for a particular node have the same sign. We decompose the input set into a finite number of sets based on the signs of the inputs to generalize this observation.

For simplicity, first we consider the case where the state valuations are all positive. $\mathcal{N}_l$ is the neural network obtained by considering the lower bound of weights and biases of INN $\mathcal{N}$. $\mathcal{N}_u$ is the neural network constructed by considering the upper bound of weights and biases of INN $\mathcal{N}$.

Definition 15: Given an INN $\mathcal{N}$, lower neural network $\mathcal{N}_l$ is defined as the same structure of $\mathcal{N}$ except:

- For each $i \in (k]$, for $s \in \mathcal{S}_{i-1}^l$, $s' \in \mathcal{S}_i^l$, $\mathcal{W}_i^l(s,s') = [\mathcal{L}_i(s) * \overline{\mathcal{W}_i(s,s')}, i(s) * \mathcal{W}_i(s,s')]$;
- For each $i \in (k]$, for $s \in \mathcal{S}_i^l$, $b_i^l(s) = [\underline{b_i(s)}, \underline{b_i(s)}]$.

Definition 16: Given an INN $\mathcal{N}$, upper neural network $\mathcal{N}_u$ is defined as the same structure of $\mathcal{N}$ except:

- For each $i \in (k]$, for $s \in \mathcal{S}_{i-1}^u$, $s' \in \mathcal{S}_i^u$, $\mathcal{W}_i^u(s,s') = [\mathcal{L}_i(s) * \overline{\mathcal{W}_i(s,s')}, \mathcal{L}_i(s) * \overline{\mathcal{W}_i(s,s')}]$;
- For each $i \in (k]$, for $s \in \mathcal{S}_i^u$, $b_i^u(s) = [\overline{b_i(s)}, b_i(s)]$.

Next, we establish the relationship between output valuation of the INN $\mathcal{N}$, $\mathcal{N}_l$ and $\mathcal{N}_u$ with respect to an input valuation, which is stated in the following theorem.

Theorem 3: Let $\mathcal{N}$ be an INN, and $v \in Val(\mathcal{S}_0)$ such that for all $s \in \mathcal{S}_0$, $v(s) \geq 0$. Then, we have

$$(v,v') \in [\![\mathcal{N}]\!] \text{ iff } \exists\ v^l \in Val(\mathcal{S}_k^l),\ v^u \in Val(\mathcal{S}_k^u) \text{ such that } (v,v^l) \in [\![\mathcal{N}_l]\!], (v,v^u) \in [\![\mathcal{N}_u]\!] \text{ and for all } s \in \mathcal{S}_k, \; v^l(s) \leq v'(s) \leq v^u(s).$$

Next, we provide a construction of $\mathcal{N}_l$ and $\mathcal{N}_u$ for the general case. For the ReLU activation function, we know that all the valuations of neurons in layer 1 or higher are positive. Hence, for N_l and $\mathcal{N}_u$, to compute weights and biases from layer 1, we use Definitions 15, and 16, respectively. To find weights between input neurons and neurons in the first layer for both N_l and N_u, we first divide the input set into regions such that the sign of all the valuations in the same region for any input neuron is same, that is, $\forall\ s \in \mathcal{S}_0, \mathbb{I}(s) \cap \mathbb{R}_{\geq 0} = \emptyset$ or $\forall\ s \in \mathcal{S}_0, \mathbb{I}(s) \cap \mathbb{R}_{<0} = \emptyset$. The upper and lower bound of the weight are chosen based on the sign of the valuation of the incoming node.

Let us define for any $x \in \mathbb{R}$, $sign(x)$ is positive if $x >= 0$ and negative otherwise. We define an equivalence relation on the input set $\mathbb{I}$ as follows. $v_1, v_2 \in \mathbb{I}$ are equivalent if for every input neuron $s \in \mathcal{S}_0$, $sign(v_1(s)) = sign(v_2(s))$. Let $\mathbb{I}_1, \ldots, \mathbb{I}_m$ be the partition of $\mathbb{I}$ corresponding to the equivalence relation. Note that m could be $2^{|\mathcal{S}_0|}$ in the worst case. Next we define for each $\mathbb{I}_j$, upper and lower bound neural networks, $\mathcal{N}_l^j$ and $\mathcal{N}_u^j$. The neural networks $\mathcal{N}_l^j$ and $\mathcal{N}_u^j$ are the same except the weight on layer 1, that is, $\mathcal{W}_1^{l,j}$ and $\mathcal{W}_1^{u,j}$. For $s \in \mathcal{S}_0$, let $sign(\mathbb{I}_j, s)$ be the sign of $v(s)$ for some $v \in \mathbb{I}_j$. Note that for all the valuations $v \in \mathbb{I}_j$, $v(s)$ always have the same sign. Then, we use the following for $\mathcal{W}_1^{l,j}$ and $\mathcal{W}_1^{u,j}$. For $s \in \mathcal{S}_0$, for $s' \in \mathcal{S}_1$,
(a) if $sign(\mathbb{I}_j, s)$ is positive, then $\mathcal{W}_1^{l,j}$ and $\mathcal{W}_1^{u,j}$ are the same as defined in Definition 15 and 16;
(b) if $sign(\mathbb{I}_j, s)$ is negative, then $\mathcal{W}_1^{l,j}(s,s')$ and $\mathcal{W}_1^{u,j}(s,s')$ will be interchanged, that is, lower and upper weight for the edge (s,s') will be $\mathcal{W}_1^{u,j}(s,s')$ and $\mathcal{W}_1^{l,j}(s,s')$, respectively.

IX. Safety Analysis

To perform reachability analysis, we first compute control input u for a given initial state value x through a given control neural network $\mathcal{N}$ for a linear dynamical system $\mathcal{D}$. To compute the set of all control input valuations for a given set of state valuations $\mathbb{S}$, we need to determine the following set, $\mathbb{O}_{\mathcal{N}}^{\mathbb{S}} = \{(v' \mid \exists\ v \in \mathbb{S}, (v,v') \in [\![\mathcal{N}]\!]\}$. $\mathbb{O}_{\mathcal{N}}^{\mathbb{S}}$ can be computed by the Starset based method [28]. Upon computation of $\mathbb{O}_{\mathcal{N}}^{\mathbb{S}}$, we compute reach set of $\mathcal{D}$ for a given set of initial state valuations $\mathbb{S}$, which is denoted by $\mathbb{O}_{\mathcal{D}}^{\mathbb{S},\mathbb{I}}$, where $\mathbb{I} = \mathbb{O}_{\mathcal{N}}^{\mathbb{S}}$, given as: $\mathbb{O}_{\mathcal{D}}^{\mathbb{S},\mathbb{I}} = \mathcal{M}_A\mathbb{S} + \mathcal{M}_B\mathbb{I}$.

$\mathbb{O}_{\mathcal{D}}^{\mathbb{S},\mathbb{I}}$ can be determined by the linear transformation and Minkowski sum operation on the Starset. After that we check safety against an unsafe specification $\mathbb{U}$ via checking emptiness of the intersection between $\mathbb{O}_{\mathcal{D}}^{\mathbb{S},\mathbb{I}}$ and $\mathbb{U}$. If the set $\mathbb{O}_{\mathcal{D}}^{\mathbb{S},\mathbb{I}}$ is exact, then the result will be "Yes" if the intersection is empty and "No" otherwise. However, if the set $\mathbb{O}_{\mathcal{D}}^{\mathbb{S},\mathbb{I}}$ is an over-approximation, then the result will be "Yes" if the intersection is empty and "Unknown" otherwise. We repeat the safety checking up to k times. If the result is "Yes" in all the iterations, then the system is safe against the unsafe specification. Otherwise, the system could be unsafe or unknown based on the exact or over-approximate state valuations computation, respectively.

Our broad goal is to scale the reachability analysis of LDS which is relying on the reachability analysis of NN, which is computationally expensive, specifically for a large neural network. Hence, we perform abstraction procedure explained in Section VII on a large neural network that reduces the network into a smaller interval neural network. To enable the use of neural network reachability tools, we have developed a novel trasformer that transforms an interval neural network into two neural networks, namely lower and upper described in Section VIII. The broad safety analysis framework is shown in Figure 5.

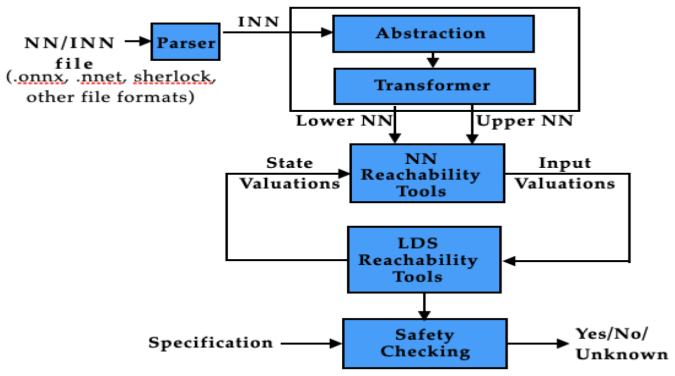

Fig. 5: Safety Analysis Framework

X. Experimental Analysis

In this section, we report our experimental evaluation on two benchmarks, namely ACASXu and the Translational Oscillators with Rotating Actuator (TORA). All the experiments were performed with Ubuntu 18.04 OS, Intel® core™ i7-4870HQ CPU @2.50GHz, 8GB RAM.

ACAS Xu. This is a controller for unmanned aircraft [18] designed for collision avoidance. The system consists of seven state variables, namely, (i) ρ: distance from ownership to intruder; (ii) θ: angle to intruder relative to ownership heading direction; (iii) ψ: heading angle of intruder relative to ownership heading direction; (iv) v_{own}: speed of ownership; (v) v_{int}: speed of intruder; (vi) τ: time until loss of vertical separation; and (vii) a_{prev}: previous advisory, and five input variables which are turning advisories of different

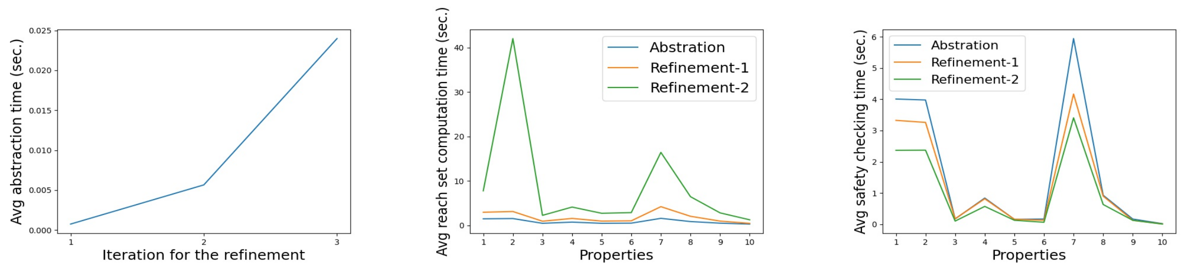

Fig. 6: Average abstraction, reach, and safety time over 45 networks

strengths and directions, namely, Clear of Conflict (COC), Weak Left (WL), Weak Right (WR), Strong Left (SL), Strong Right (SR). Details of the benchmarks can be found in the paper [17].

The Translational Oscillators with Rotating Actuator (TORA). This is an underactuated system which has one actuated rotor and one unactuated translational cart. It consists of 4 state variables and one input variable. The discrete dynamics of TORA is given below:

$$x_1[k+1] = x_2[k], \quad x_2[k+1] = -x_1[k] + x_3[k],$$
$$x_3[k+1] = x_4[k], \quad x_4[k+1] = u.$$

The input u is computed by a neural network that consists of 4 input neurons corresponds to the state variables, 3 hidden layers each with 100 neurons, and 1 output neuron corresponds to the input variable.

Next, we report our experimental evaluation. For the reachability analysis, we adapt Starset based method [28].

ACAS Xu. We have performed safety of 10 properties [17] for 45 networks of ACAS Xu benchmarks. For the experiments, we start with the coarse abstraction of each network, where at each hidden layer, we merged all neurons into an abstract neuron. Then, we perform the safety verification of all 10 properties on the abstract network via computing the reach set. Next, we systemically construct a more precise abstraction to improve the safety results. Hence, we have constructed two more precise abstractions, namely refinement-1 and refinement-2, where for the refinement-1, we have merged sixteen neurons into an abstract neuron, and for refinement-2, four neurons have been merged into an abstract neuron. The experimental results are shown in Table I.

	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7	ϕ_8	ϕ_9	ϕ_{10}
Abstraction	0	0	0	0	0	0	0	0	0	0
Refinement-1	4	7	0	0	0	0	0	0	0	0
Refinement-2	19	17	0	0	0	0	0	0	0	0

TABLE I: Safety Analysis of ACAS Xu Neural Networks

In Table I, we observe that the properties ϕ_1 and ϕ_2 are true for more networks in the first and second refinement. However, for other properties, it does not change because properties $\phi_3 \dots \phi_{10}$ are more sensitive. Hence, we need careful selection of partitioning for better abstraction.

In Figure 6, when we consider more precise abstraction, the following things happen: (a) the average abstraction time grows; (b) the average computation of all output valuations increases. The reason is that the number of neurons grow. However, average time for safety checking reduces when we consider more precise abstraction because the range of output valuations is less, and the domain for checking properties is reduced.

The Translational Oscillators with Rotating Actuator (TORA). We have performed computational analysis for computing the set of states at different number of steps under original, abstract, and refined neural network, where for the abstract neural network, we have merged eight neurons into an abstract neuron at each hidden layer while two neurons are merged into an abstract neuron for the refined network. The experimental results are presented in Table II.

	Original (sec.)			Abstraction (sec.)			refinement (sec.)		
K	T_{NN}	T_{LDS}	T_{total}	T_{NN}	T_{LDS}	T_{total}	T_{NN}	T_{LDS}	T_{total}
2	0.44	0.0002	0.93	0.19	0.0001	0.41	0.26	0.0002	0.57
3	0.60	0.0002	1.95	0.31	0.0002	1.03	0.38	0.0003	1.27
4	1.02	0.0002	4.44	0.62	0.0002	2.74	0.67	0.0002	2.99
5	2.22	0.0003	12.19	1.41	0.0003	7.88	1.67	0.0003	9.22
6	5.76	0.0006	38.46	4.01	0.0006	26.65	5.26	0.0008	35.01
7	18.86	0.004	146.10	12.10	0.001	94.66	12.12	0.01	102.38
8	61.03	0.01	542.18	41.40	0.002	367.46	49.26	0.004	437.43

TABLE II: Computational Analysis over Different Abstractions

In Table II, K shows the number of steps. T_{NN}, T_{LDS} represent average time for computing output valuations for the neural network and linear dynamical system over a given number of steps. T_{total} shows the total average time among all steps. All the times are measured in seconds.

In Table II, we have observed that when we increase the number of steps, T_{NN} increases because the input set is splitted into to multiple sub-input sets and lower and upper neural networks are different for each sub-input set. In each step, for each sub-input set, output range of the neural network is computed. Note that the number of sub-inputs may grow over the number of steps. However, T_{LDS} is almost constant, because output valuation of the linear dynamical system is just one linear operation on a star set.

XI. Conclusions

In this paper, we present an abstraction based safety analysis framework for neural network controlled dynamical systems. We present a novel abstraction technique and a novel reachable set computation algorithm for the annotated interval neural networks. Our experimental analysis demonstrates the benefits of the abstraction based safety analysis approach. We noticed that the success of the approach critically relies on the specific abstractions. Our future work will investigate mechanisms for exploring the abstractions in a systematic manner, for instance, using methods such as counter-example guided abstraction refinement [5].

XII. Acknowledgement

This work was partially supported by NSF CAREER Grant No. 1552668, NSF Grant No. 2008957 and Amazon Research Award.

References

[1] Pranav Ashok, Vahid Hashemi, Jan Křetínskỳ, and Stefanie Mohr. Deepabstract: neural network abstraction for accelerating verification. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2020.

[2] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Piecewise linear neural network verification: A comparative study. *CoRR*, 2017.

[3] Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Piecewise linear neural networks verification: A comparative study. 2018.

[4] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, pages 251–268, 2017.

[5] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proceedings of the International Conference on Computer Aided Verification*, pages 154–169. Springer, 2000.

[6] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, pages 238–252, 1977.

[7] Souradeep Dutta, Susmit Jha, and Ashish Sankaranarayanan, Sriramand Tiwari. Output range analysis for deep feedforward neural networks. In Aaron Dutle, César Muñoz, and Anthony Narkawicz, editors, *NASA Formal Methods*, pages 121–138. Springer International Publishing, 2018.

[8] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI*, pages 550–559, 2018.

[9] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In Deepak D'Souza and K. Narayan Kumar, editors, *Automated Technology for Verification and Analysis*, pages 269–286, Cham, 2017. Springer International Publishing.

[10] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In *Proceedings of the International Conference on Computer Aided Verification*. Springer, 2020.

[11] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP, pages = 3–18, year = 2018,*.

[12] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 1989.

[13] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

[14] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems*, 18:1–22, 10 2019.

[15] Xiaowei Huang, Daniel Kroening, Marta Z. Kwiatkowska, Wenjie Ruan, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. Safety and trustworthiness of deep neural networks: A survey. *CoRR*, abs/1812.08342, 2018.

[16] Kyle D Julian and Mykel J Kochenderfer. A reachability method for verifying dynamical systems with deep neural network controllers. *arXiv preprint arXiv:1903.00520*, 2019.

[17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 2017.

[18] Mykel J Kochenderfer, Jessica E Holland, and James P Chryssanthacopoulos. Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[19] J. Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *CoRR*, abs/1711.00851, 2017.

[20] Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3575–3583, 2018.

[21] Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. Lower bounds on the robustness to adversarial perturbations. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 804–813, 2017.

[22] Pavithra Prabhakar and Zahra Rahimi Afzal. Abstraction based output range analysis for neural networks. 2019.

[23] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 243–257, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[24] Luca Pulina and Armando Tacchella. Never: a tool for artificial neural networks verification. *Annals of Mathematics and Artificial Intelligence*, 2011.

[25] Jagannathan Sarangapani. *Neural network control of nonlinear discrete-time systems*. CRC press, 2018.

[26] 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*, pages 147–156, 2019.

[27] 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*, pages 147–156, 2019.

[28] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-based reachability analysis of deep neural networks. In *International Symposium on Formal Methods*. Springer, 2019.

[29] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1599–1614, 2018.

[30] 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. *arXiv preprint arXiv:1804.09699*, 2018.

[31] Weiming Xiang and Taylor T. Johnson. Reachability analysis and safety verification for neural network control systems. *CoRR*, abs/1805.09944, 2018.

[32] Weiming Xiang, Patrick Musau, Ayana A. Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel A. Rosenfeld, and Taylor T. Johnson. Verification for machine learning, autonomy, and neural networks survey. *CoRR*, abs/1810.01989, 2018.

[33] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multi-layer neural networks. *CoRR*, abs/1708.03322, 2017.

[34] Pengfei Yang, Jiangchao Liu, Jianlin Li, Liqian Chen, and Xiaowei Huang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. *CoRR*, abs/1902.09866, 2019.

[35] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4939–4948. Curran Associates, Inc., 2018.

APPENDIX

A. Proof of Theorem 1

We need the following propositions and lemmas to prove Theorem 1. Since the abstraction function defined in Definition 11 is the weighted average of valuation of concrete neurons, we need the following proposition to bound its value via minimum and maximum of the valuation of concrete neurons corresponding to an abstract neuron.

Proposition 1: Let $V = \{v_1, v_2, \dots, v_n\}$ and $L = \{l_1, l_2, \dots, l_n\}$ be the sets of n real values and positive numbers, respectively. Let $v_{min} = \min_{v \in V} v$, $v_{max} = \max_{v \in V} v$, and $\overline{v} = \sum_{i=1}^{n} l_i v_i / \sum_{i=1}^{n} l_i$. Then, we have

$$v_{min} \leq \overline{v} \leq v_{max}.$$

The proof of Proposition 1 is straight forward. Furthermore, we need the following proposition which states that sum of the multiplication of three entities, namely weights (w_i's), labels (l_i's) and values (v_i's), can be expressed as a single multiplication of weight (w), label (l) and value (v), where v is the weighted average of v_i's with respect to labels l_i's; l is the sum of all labels, and w is some value in the convex hull of $w_i' s$.

Proposition 2: Let $W = \{w_1, w_2, \dots, w_n\}$ and $V = \{v_1, v_2, \dots, v_n\}$ be two sets of real numbers and $L = \{l_1, l_2, \dots, l_n\}$ be a set of natural numbers. Then, there is $w \in \mathrm{CH}(W)$ such that

$$\sum_{i=1}^{n} w_i l_i v_i = w l \overline{v}, \ where, \ \overline{v} = \frac{\sum_{i=1}^{n} l_i v_i}{l}, \ and \ l = \sum_{i=1}^{n} l_i.$$

Proof: Let $W = \{w_1, w_2, \dots, w_n\}$, $V = \{v_1, v_2, \dots, v_n\}$ be two sets of real numbers and $L = \{l_1, l_2, \dots, l_n\}$ be a set of natural numbers. Let $w_{min} = \min(W)$, $w_{max} = \max(W)$. Then, we have

$$w_{min} \sum_{i=1}^{n} l_i v_i \leq \sum_{i=1}^{n} w_i l_i v_i \leq w_{max} \sum_{i=1}^{n} l_i v_i \qquad (2)$$

Let $\overline{v}$ be weighted average of v_i's with respect to weights l_i's, that is, $\overline{v} = \frac{\sum_{i=1}^{n} l_i v_i}{\sum_{i=1}^{n} l_i}$. This provides us $\overline{v} \sum_{i=1}^{n} l_i = \sum_{i=1}^{n} l_i v_i$. Now, Equation 2 can be written as follows.

$$w_{min} l v \leq \sum_{i=1}^{n} w_i l_i v_i \leq w_{max} l v \qquad (3)$$

where $l = \sum_{i=1}^{n} l_i$. Let $x = \sum_{i=1}^{n} w_i l_i v_i$. Then, we have $w_{min} \leq \frac{x}{l\overline{v}} \leq w_{max}$. This implies that there exists $w \in [w_{min}, w_{max}] = \mathrm{CH}(W)$ such that $w = \frac{x}{l\overline{v}}$. Thus, we have $x = w l \overline{v}$, that is, $\sum_{i=1}^{n} w_i l_i v_i = w l \overline{v}$. ■

Next, we state intermediate value theorem to show that a mapping of concrete input-output valuation under abstraction function will be an input-output valuation of the abstract network.

Proposition 3 (Intermediate Value Theorem): Let $\mathcal{S}$ be n-dimensional path connected subset of $\mathbb{R}^n$, and $f : \mathcal{S} \to \mathbb{R}$ be a continuous function. If $\mathbf{a}$ and $\mathbf{b}$ are points in $\mathcal{S}$ and $f(\mathbf{a}) \leq \mathbf{t} \leq \mathbf{f}(\mathbf{b})$, then there exists $\mathbf{c} \in \mathcal{S}$ such that $f(\mathbf{c}) = \mathbf{t}$.

To prove Theorem 1, we first need to prove it for a single layer, which is stated in the following lemma.

Lemma 1: Given an INN $\mathcal{N}$and its abstraction $\mathcal{N}/\mathcal{P}$ and a layer number $i \in [k]$, we have the following:

$$if \ (v, v') \in [\![\mathcal{N}]\!]_i, then \ (\alpha(v), \alpha(v')) \in [\![\mathcal{N}/\mathcal{P}]\!]_i.$$

Proof: Let $\mathcal{S}$, $\mathcal{S}'$ be the set of neurons in layer i and $i+1$ of $\mathcal{N}$, respectively, and $\widehat{\mathcal{S}}$, $\widehat{\mathcal{S}}'$ be the set of neurons in layer i and $i+1$ of $\mathcal{N}/\mathcal{P}$, respectively. Let $(v, v') \in [\![\mathcal{N}]\!]_i$, that is, for each $s' \in \mathcal{S}'$,

$$v'(s') = \mathcal{A}_i(s')(\left(\sum_{s \in \mathcal{S}} w_{s,s'} i-1(s) v(s)\right) + b_{s'})$$

for some $w_{s,s'} \in \mathcal{W}_i(s, s')$ for $s \in \mathcal{S}$ and $b_{s'} \in b_i(s')$. Our goal is to show that $(\alpha(v), \alpha(v')) \in [\![\mathcal{N}/\mathcal{P}]\!]_i$. For simplicity, we use $\widehat{v}$, $\widehat{v}'$ for $\alpha(v)$, $\alpha(v')$, respectively. Since $\widehat{\mathcal{S}}$ is a partition of $\mathcal{S}$, we expand the expression of $v'(s')$ over partition elements of $\widehat{\mathcal{S}}$ as follow:

$$v'(s') = \mathcal{A}_i(s')(\left(\sum_{\widehat{s} \in \widehat{\mathcal{S}}} \left(\sum_{s \in \widehat{s}} w_{s,s'} i-1(s) v(s)\right)\right) + b_{s'}).$$

From Proposition 2, we have the following:

$$\sum_{s \in \widehat{s}} w_{s,s'} i-1(s) v(s) = w_{\widehat{s},s'} l_{\widehat{s}} \widehat{v}(\widehat{s}) \qquad (4)$$

where $w_{\widehat{s},s'} \in \mathrm{CH}(\{\mathcal{W}_i(s) \mid s \in \widehat{s}\})$ denoted by $\mathcal{I}_{\widehat{s},s'}$, $l_{\widehat{s}} = \sum_{s \in \widehat{s}} i-1(s)$, and $\widehat{v}(\widehat{s}) = \sum_{s \in \widehat{s}} i-1(s) v(s)$. From Equation 4, we express the expression $v'(s')$ in terms of partition elements of $\widehat{\mathcal{S}}$ as follow:

$$v'(s') = \mathcal{A}_i(s')(\left(\sum_{\widehat{s} \in \widehat{\mathcal{S}}} w_{\widehat{s},s'} l_{\widehat{s}} \widehat{v}(\widehat{s})\right) + b_{s'}) \qquad (5)$$

Let $\widehat{s}' \in \widehat{\mathcal{S}}'$. From Definition 10, we have $\widehat{\mathcal{L}}_{i-1}(\widehat{s}) = \sum_{s \in \widehat{s}} i-1(s)$. This implies that $l_{\widehat{s}} = \widehat{\mathcal{L}}_{i-1}(\widehat{s})$. Also, for any $s' \in \widehat{s}'$, $\mathcal{A}_i(s') = \widehat{\mathcal{A}}_i(\widehat{s}')$. Hence, we have

$$v'(s') = \widehat{\mathcal{A}}_i(\widehat{s}')(\left(\sum_{\widehat{s} \in \widehat{\mathcal{S}}} w_{\widehat{s},s'} \widehat{\mathcal{L}}_{i-1}(\widehat{s}) \widehat{v}(\widehat{s})\right) + b_{s'}) = f(\{w_{\widehat{s},s'}\}_{s \in \mathcal{S}}, b_{s'}) \qquad (6)$$

From Definition 11,

$$\widehat{v}'(\widehat{s}') = \frac{\sum_{s' \in \widehat{s}'} \widehat{\mathcal{L}}_i(s') v'(s')}{\sum_{s' \in \widehat{s}'} \widehat{\mathcal{L}}_i(s')}.$$

Let $\mathcal{I}_b = \mathrm{CH}(\{b_i(s')\}_{s' \in \widehat{s}})$, $v'_{min} = \min_{s' \in \widehat{s}'} v'(s')$ and $v'_{max} = \max_{s' \in \widehat{s}'} v'(s')$, that is, there are $w^1_{\widehat{s},s'}, w^2_{\widehat{s},s'} \in \mathcal{I}_{\widehat{s},s'}$ for all $\widehat{s} \in \widehat{\mathcal{S}}$, and $b^1_{s'}, b^2_{s'} \in \mathcal{I}_b$ such that $v'_{min} = f(\{w^1_{\widehat{s},s'}\}_{s \in \mathcal{S}}, b^1_{s'})$ and $v'_{max} = f(\{w^2_{\widehat{s},s'}\}_{s \in \mathcal{S}}, b^2_{s'})$. Then, we have $v'_{min} \leq \widehat{v}'(\widehat{s}') \leq v'_{max}$ from Proposition 1. We observe that the domain of f is a Cartesian product of intervals $\{\mathcal{I}_{\widehat{s}}\}_{\widehat{s}\in\widehat{\mathcal{S}}}$ and $\mathcal{I}_b$, which is a path connected set, and f is a continuous function because activation functions are considered to be continuous. From intermediate value theorem given in Proposition 3, there exist $w_{\widehat{s},\widehat{s}'} \in \mathcal{I}_{\widehat{s},s'}$ for all $\widehat{s} \in \widehat{\mathcal{S}}$ and $b_{\widehat{s}'} \in \mathcal{I}_b$ such that $f(\{w_{\widehat{s},\widehat{s}'}\}_{\widehat{s}\in\widehat{\mathcal{S}}}, b_{\widehat{s}'}) = \widehat{v}'(\widehat{s}')$. Since $w_{\widehat{s},\widehat{s}'} \in \mathcal{I}_{\widehat{s},s'}$, $w_{\widehat{s},\widehat{s}'} \in \mathrm{CH}(\{\mathcal{I}_{\widehat{s},s'}\}_{s'\in\widehat{s}'})$, we have $(\widehat{v},\widehat{v}') \in [\![\mathcal{N}/\mathcal{P}]\!]_i$, that is, $(\alpha(v),\alpha(v')) \in [\![\mathcal{N}/\mathcal{P}]\!]_i$. ■

Proof of Theorem 1: Let $(v_0, v_k) \in [\![\mathcal{N}]\!]$. This means that there exist $\{v_i\}_{i\in(k-1]} \in Val(\mathcal{S}_i)_{i\in(k-1]}$ such that $(v_{i-1}, v_i) \in [\![\mathcal{N}]\!]_i$ for $i \in (k]$. From Lemma 1, we have $(\alpha(v_{i-1}), \alpha(v_i)) \in [\![\mathcal{N}/\mathcal{P}]\!]_i$ for $i \in (k]$). From Definition 3, $(\alpha(v_0), \alpha(v_k)) \in [\![\mathcal{N}/\mathcal{P}]\!]$.