Reachable Set Estimation for Neural Network Control Systems: A Simulation-Guided Approach

Weiming Xiang, *Senior Member, IEEE*, Hoang-Dung Tran, Xiaodong Yang, and Taylor T. Johnson, *Member, IEEE*

***Abstract*— The vulnerability of artificial intelligence (AI) and machine learning (ML) against adversarial disturbances and attacks significantly restricts their applicability in safety-critical systems including cyber-physical systems (CPS) equipped with neural network components at various stages of sensing and control. This article addresses the reachable set estimation and safety verification problems for dynamical systems embedded with neural network components serving as feedback controllers. The closed-loop system can be abstracted in the form of a continuous-time sampled-data system under the control of a neural network controller. First, a novel reachable set computation method in adaptation to simulations generated out of neural networks is developed. The reachability analysis of a class of feedforward neural networks called multilayer perceptrons (MLPs) with general activation functions is performed in the framework of interval arithmetic. Then, in combination with reachability methods developed for various dynamical system classes modeled by ordinary differential equations, a recursive algorithm is developed for over-approximating the reachable set of the closed-loop system. The safety verification for neural network control systems can be performed by examining the emptiness of the intersection between the over-approximation of reachable sets and unsafe sets. The effectiveness of the proposed approach has been validated with evaluations on a robotic arm model and an adaptive cruise control system.**

***Index Terms*— Neural network control systems, reachability, safety verification, simulation.**

I. Introduction

NEURAL networks have been demonstrated to be effective tools in controlling complex systems in a variety of research activities such as stabilization [1], [2] and adaptive control [3], [4]. In some latest applications, neural networks have been deployed and played a critical role in high-safety-assurance systems such as autonomous systems [5], unmanned vehicles [6], and aircraft collision avoidance systems [7]. However, due to the vulnerability neural networks against adversarial disturbances/attacks and the black-box nature of neural networks, such controllers with a neural network structure, in essence, are only restricted to the control applications with the lowest levels of requirements of safety as there is a short of effective methods to compute the output reachable set of neural networks and further assure the safety of underlying closed-loop systems. It has been frequently observed that even a slight perturbation against the input of a well-trained neural network will produce a completely incorrect and unpredictable output [8]. As we consider a closed-loop system with a feedback channel involving neural networks, the safety issues will inevitably arise since disturbances and uncertainties are unavoidable in measurement and control channels, which may result in undesirable and unsafe system behaviors even instability. Furthermore, with advanced adversarial machine learning (ML) techniques developed recently, such safety matters for safety-critical control systems with neural network controllers only become even much worse. Therefore, to integrate artificial intelligence (AI)/ML components such as neural networks into safety-critical control systems, safety verification for such AI/ML systems is required at all stages for the purpose of safety assurance. However, because of the sensitivity of neural networks against perturbations and the complex structure of neural networks, the verification of neural networks represents extreme difficulties. It has been demonstrated that a simple property verification of a small-scale neural networks is a nondeterministic polynomial (NP) complete problem [9].

A. Related Work

Formal verification of neural networks has been well recognized in recent literature. One of the earliest methods is the abstraction-refinement approach proposed in [10] and [11], which is developed for computing the output set of a feedforward neural network to perform verification. In [12], a satisfiability modulo theories (SMT) solver was proposed for the verification of feedforward neural networks. Some Lyapunov function-based approaches were proposed for dynamical systems with neural network structures [13]–[15], in which invariant sets are constructed to estimate reachable sets. For a special class of neural networks with rectified linear unit (ReLU) neurons, several methods have been

Manuscript received August 26, 2019; revised December 17, 2019 and March 30, 2020; accepted April 24, 2020. Date of publication May 14, 2020; date of current version May 3, 2021. This work was supported in part by the Air Force Office of Scientific Research (AFOSR) under Contract FA9550-18-1-0122, in part by the Defense Advanced Research Projects Agency (DARPA) under Contract FA8750-18-C-0089, and in part by the Research Scholarship and Creative Activity Grant Program of Augusta University. *(Corresponding author: Weiming Xiang.)*

Weiming Xiang is with the School of Computer and Cyber Sciences, Augusta University, Augusta, GA 30912 USA (e-mail: wxiang@augusta.edu).

Hoang-Dung Tran, Xiaodong Yang, and Taylor T. Johnson are with the Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37212 USA (e-mail: dung.h.tran@vanderbilt.edu; xiaodong.yang@vanderbilt.edu; taylor.johnson@vanderbilt.edu).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.2991090

reported in the literature such as mixed-integer linear programming (MILP) approaches [16], [17], linear programming (LP)-based approaches [18], the Reluplex algorithm that stems from the Simplex algorithm [9], and polytope-operation-based approaches [19], [20]. For neural networks with general activation functions, the sensitivity for neural networks was introduced in [21] and [22] and used for various problems, for instance, weight selection [23], learning algorithm improvement [24], and architecture construction [25]. Based on the maximal sensitivity concept, a simulation-based verification approach is introduced in [26]. The output reachable set estimation for feedforward neural networks with general activation functions is formulated in terms of four convex optimization problems. These results are able to compute estimated and exact output sets of a feedforward neural network, and it, therefore, implies the availability of reachable set estimation and safety verification of closed-loop systems equipped with neural network controllers as shown in [27]–[29]. The Verisig approach [30] transforms a neural network controller with sigmoid activation functions to an equivalent nonlinear hybrid system. This is combined with plant dynamics by using ordinary differential equation (ODE) reachability analysis routines for safety verification. All those existing methods were developed mainly based on exploiting the neural network itself such as the piecewise linear feature of ReLU activation functions or transformation of neural networks. In this article, we emphasize that our method focuses on both interval-based derivations of neural networks as well as taking advantage of simulations originated from neural networks for reachable set computation and safety verification of neural network control systems.

B. Contributions

This article focuses on improving the simulation-based approach developed in [26] for the output set overapproximation of feedforward neural networks with general activation functions. A novel adaptive simulation-guided method will be developed and further integrated for safety verification of closed-loop systems with neural network controllers. In this article, we develop a novel simulation-guided approach to perform the output reachable set estimation of feedforward neural networks with general activation functions. The algorithm is formulated in the framework of interval arithmetic and under the guidance of a finite number of simulations. The developed method using the information of simulations is able to provide much less computational cost than the previous article [26]. As shown by a robotic arm model example, it only needs about 3% computational cost of the method proposed in [26] to obtain the same interval-based reachability analysis result. We also extend our reachable set estimation result for safety verification of neural network control systems, in which plants are modeled by ODEs. We develop an algorithm to compute the reachable set of a neural network control system modeled by sampled-data systems. Based on the reachable set estimation, a safety verification algorithm is developed to provide formal safe assurance for neural network control systems, and an adaptive cruise control (ACC) system using a software prototype is proposed to demonstrate our method.

II. System Description and Problem Formulation

A. Neural Network Control Systems

In this article, we consider a class of continuous-time nonlinear systems in the form of

$$\begin{cases} \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) = h(\mathbf{x}(t)) \end{cases} \tag{1}$$

where $\mathbf{x}(t) \in \mathbb{R}^{n_x}$ is the state vector, $\mathbf{u}(t) \in \mathbb{R}^{n_u}$ is the control input, and $\mathbf{y}(t) \in \mathbb{R}^{n_y}$ is the controlled output. The nonlinear controller is considered in its general form of

$$\mathbf{u}(t) = \gamma(\mathbf{y}(t), \mathbf{v}(t), t) \tag{2}$$

where $\mathbf{v}(t) \in \mathbb{R}^{n_v}$ is the reference input. As we know, the controller design problem for nonlinear systems in the general form is quite challenging and still open even if f and h are available. To avoid the difficulties arising in such controller design problems for systems with complex model or even model unavailable, some data-driven approaches which only rely on the input–output data of the system were developed. In this article, we consider a class of feedforward neural network trained by input–output data as the feedback controller of dynamical systems. The feedforward neural network is in the following general form of

$$\mathbf{u}(t) = \Phi(\mathbf{y}(t), \mathbf{v}(t)) \tag{3}$$

where $\Phi : \mathbb{R}^{n_y} \times \mathbb{R}^{n_v} \to \mathbb{R}^{n_u}$ is a neural network trained by data collected during system operations. We can rewrite the neural network controller in a more compact form of

$$\mathbf{u}(t) = \Phi(\boldsymbol{\eta}(t)) \tag{4}$$

where $\boldsymbol{\eta}(t) = [\mathbf{y}^\top(t)\ \mathbf{v}^\top(t)]^\top$.

In practice, it always takes certain amount of time to compute the output signal of the neural network as the control input of the controlled plant. Hence, the neural network controller produces the control signals at every sampling time instant t_k, $k \in \mathbb{N}$, and then the controller maintains its value between two successive sampling instants t_k and t_{k+1}. Due to the sampling mechanism of practical control systems, we can formulate the sampled neural network controller in the form of

$$\mathbf{u}(t) = \Phi(\boldsymbol{\eta}(t_k)),\ \ t \in [t_k, t_{k+1}) \tag{5}$$

and by substituting the above controller into system (1), we can obtain the closed-loop system in the following form:

$$\begin{cases} \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \Phi(\boldsymbol{\eta}(t_k))) \\ \mathbf{y}(t) = h(\mathbf{x}(t)) \end{cases} \qquad t \in [t_k, t_{k+1}) \tag{6}$$

where $\boldsymbol{\eta}(t_k) = [\mathbf{y}^\top(t_k)\ \mathbf{v}^\top(t_k)]^\top$. The mechanism of a sampled-data neural network system in the form of (6) is illustrated in Fig. 1. It worth mentioning that sampled-data model for neural network control systems in the form of (6) can be found in a variety of articles such as [1].

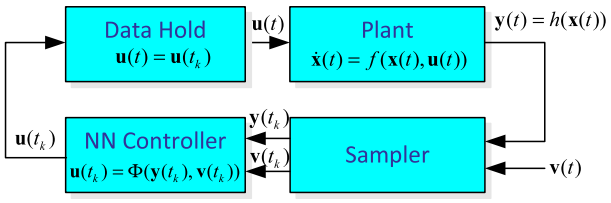


Fig. 1. Block diagram for continuous-time sampled-data neural network control systems.

B. Feedforward Neural Networks

In this article, we consider a class of feedforward neural networks which is called multilayer perceptrons (MLPs). The basic processing elements in MLPs are neurons which are defined by activation functions in the form of

$$u_i = \phi\left(\sum_{j=1}^{n} \omega_{ij}\eta_j + \theta_i\right) \tag{7}$$

where η_j is the jth input of the ith neuron, ω_{ij} is the weight from the jth input to the ith neuron, θ_i is the bias of the ith neuron, u_i is the output of the ith neuron, and $\phi(\cdot)$ is the activation function. There are a variety of activation functions such as ReLU, $\tanh$, and logistic. In this article, our approach is able to deal with activation functions without considering their specific forms.

In this article, we assume that the MLP has L layers, and each layer ℓ, $1 \leq \ell \leq L$, consists of $n^{\{\ell\}}$ neurons. Especially, we use layer $\ell = 0$ to denote the input layer where the input vector is passed to the network, and $n^{\{0\}}$ is the number of the inputs for the network. In addition, $n^{\{L\}}$ is used to denote the output layer. For the layer ℓ, the input vector of the layer ℓ is $\boldsymbol{\eta}^{\{\ell\}}$, and the weight matrix and the bias vector are

$$\mathbf{W}^{\{\ell\}} = \left[\omega_1^{\{\ell\}}, \ldots, \omega_{n^{\{\ell\}}}^{\{\ell\}}\right]^\top \tag{8}$$

$$\boldsymbol{\theta}^{\{\ell\}} = \left[\theta_1^{\{\ell\}}, \ldots, \theta_{n^{\{\ell\}}}^{\{\ell\}}\right]^\top \tag{9}$$

and the output vector of layer ℓ can be written in the form of

$$\mathbf{u}^{\{\ell\}} = \phi_\ell(\mathbf{W}^{\{\ell\}}\boldsymbol{\eta}^{\{\ell\}} + \theta^{\{\ell\}}) \tag{10}$$

where $\phi_\ell(\cdot)$ denotes the activation function of layer ℓ.

As the output of layer ℓ equals the input of its successive layer $\ell + 1$, we can obtain the mapping from the input vector of the input layer $\ell = 0$ to the output vector of the output layer $\ell = L$. Namely, the input–output relationship of an MLP can be expressed in the following form:

$$\mathbf{u}^{\{L\}} = \Phi(\boldsymbol{\eta}^{\{0\}}) \tag{11}$$

where $\Phi(\cdot) \triangleq \phi_L \circ \phi_{L-1} \circ \cdots \circ \phi_1(\cdot)$.

C. Problem Formulation

Given an input set, the output set of an MLP is given by the following definition.

Definition 1: Given an input set $\mathcal{H}$, the output set of the MLP in the form of (11) is

$$\mathcal{U} = \left\{\mathbf{u}^{\{L\}} \in \mathbb{R}^{n_u} \mid \mathbf{u}^{\{L\}} = \Phi(\boldsymbol{\eta}^{\{0\}}),\ \boldsymbol{\eta}^{\{0\}} \in \mathcal{H}\right\}. \tag{12}$$

The exact output set of an MLP is extremely difficult to obtain due to the complexity of neural networks. We often resort to compute an over-approximation of $\mathcal{U}$ which would be more feasible and practical.

Definition 2: Given the output set $\mathcal{U}$ of MLP (11), if there exist a set $\mathcal{U}_e$ such as $\mathcal{U} \subseteq \mathcal{U}_e$ holds, then $\mathcal{U}_e$ is an output reachable set estimation of MLP (11).

The first key issue that needs to be addressed in this article is the reachable set estimation for an MLP in the form of (11), which is summarized as follows.

Problem 1: Given an MLP in the form of (11) and a bounded set $\mathcal{H}$ as input set, how does one compute a set $\mathcal{U}_e$ such that $\mathcal{U} \subseteq \mathcal{U}_e$ holds and moreover, the set $\mathcal{U}_e$ is required to be as small as possible?[1]

Then, let us consider the neural network control system (6). The state trajectory of the closed-loop system (6) from a single initial state $\mathbf{x}_0$ is denoted by $\mathbf{x}(t; \mathbf{x}_0, \mathbf{v}(\cdot))$, where $t \in \mathbb{R}_{\geq 0}$ is the time and $\mathbf{v}(\cdot)$ stands for the input trajectory. With an initial set and input set, the reachable set for the closed-loop system (6) is given as follows.

Definition 3: Given a neural network control system in the form of (6), an initial set $\mathcal{X}_0$ and an input set $\mathcal{V}$, the reachable set at time instant t is defined by

$$\mathcal{R}(t) = \left\{\mathbf{x}(t; \mathbf{x}_0, \mathbf{v}(\cdot)) \in \mathbb{R}^{n_x} \mid \mathbf{x}_0 \in \mathcal{X}_0, \mathbf{v}(t) \in \mathcal{V}\right\} \tag{13}$$

and the reachable set of system (6) over time interval $[t_0, t_f]$ is defined by

$$\mathcal{R}([t_0, t_f]) = \bigcup_{t \in [t_0, t_f]} \mathcal{R}(t). \tag{14}$$

Similarly, for most of the system classes, the exact reachable set cannot be computed. Instead, we resort to derive over-approximations for the purpose of safety verification.

Definition 4: Given system (6) and its reachable set $\mathcal{R}(t)$, $\mathcal{R}_e(t)$ is an over-approximation of $\mathcal{R}(t)$ at time t if $\mathcal{R}(t) \subseteq \mathcal{R}_e(t)$ holds. Moreover, $\mathcal{R}_e([t_0, t_f]) = \bigcup_{t \in [t_0, t_f]} \mathcal{R}_e(t)$ is an over-approximation of $\mathcal{R}([t_0, t_f])$ over interval $[t_0, t_f]$.

The main problem, the reachable set estimation problem for neural network control system (6), is summarized as below.

Problem 2: Given closed-loop system (6), a bounded initial set $\mathcal{X}_0$ and an input set $\mathcal{V}$, how does one find a set $\mathcal{R}_e(t)$ such that $\mathcal{R}(t) \subseteq \mathcal{R}_e(t)$ holds?

Based on the reachable set estimation of neural network control systems, the safety verification for such dynamical systems can be performed. The safety specification is defined by a set the state space, which describes the safety requirement for the system.

Definition 5: Given neural network control system (6) and a safety specification set $\mathcal{S}$ which formalizes the safety requirements. The closed-loop system (6) is safe during time interval $[t_0, t_f]$ if and only if the following condition holds:

$$\mathcal{R}([t_0, t_f]) \cap \neg\mathcal{S} = \emptyset \tag{15}$$

where $\neg$ is the logical negation symbol.

[1]For a set $\mathcal{U}$, its over-approximation $\mathcal{U}_{e,1}$ is smaller than another over-approximation $\mathcal{U}_{e,2}$ if $d_{\mathrm{H}}(\mathcal{U}_{e,1}, \mathcal{U}) < d_{\mathrm{H}}(\mathcal{U}_{e,2}, \mathcal{U})$ holds, where d_{H} stands for the Hausdorff distance.

Therefore, the safety verification problem for neural network control system (6) is as follows.

Problem 3: Given a neural network control system in the form of (6), a bounded initial set $\mathcal{X}_0$, an input set $\mathcal{V}$, and a safety specification set $\mathcal{S}$, how does one examine if the safety requirement (15) holds?

Before ending this section, a useful lemma is presented, which implies that the safety verification of the neural network control system (6) can be relaxed with the help of the reachable set estimation $\mathcal{R}_e$, instead of using the exact reachable set $\mathcal{R}$.

Lemma 1: Given a neural network control system in the form of (6) and a safety specification set $\mathcal{S}$, the closed-loop system (6) is safe over time interval $[t_0, t_f]$ if the following condition holds:

$$\mathcal{R}_e([t_0, t_f]) \cap \neg\mathcal{S} = \emptyset \tag{16}$$

where $\mathcal{R}([t_0, t_f]) \subseteq \mathcal{R}_e([t_0, t_f])$.

Proof: Because of $\mathcal{R}([t_0, t_f]) \subseteq \mathcal{R}_e([t_0, t_f])$, condition (16) implies $\mathcal{R}([t_0, t_f]) \cap \neg\mathcal{S} = \emptyset$. The proof is complete.

III. Simulation-Guided Reachability Analysis for Neural Networks

A. Preliminaries

Let $[x] = [\underline{x}, \overline{x}]$, $[y] = [\underline{y}, \overline{y}]$ be real compact intervals and $\circ$ denote one of the basic operations including addition, subtraction, multiplication, and division, respectively, for real numbers, that is $\circ \in \{+, -, \cdot, /\}$, where it is assumed that $0 \notin [b]$ in the case of division. We define these operations for intervals $[x]$ and $[y]$ by $[x] \circ [y] = \{x \circ y \mid x \in [y], x \in [y]\}$. The width of an interval $[x]$ is defined and denoted by $w([x]) = \overline{x} - \underline{x}$. The set of compact intervals in $\mathbb{R}$ is denoted by $\mathbb{IR}$. We say $[\phi] : \mathbb{IR} \to \mathbb{IR}$ is an interval extension of function $\phi : \mathbb{R} \to \mathbb{R}$, if for any degenerate interval arguments, $[\phi]$ agrees with ϕ such that $[\phi]([x, x]) = \phi(x)$. In order to consider multidimensional problems where $\mathbf{x} \in \mathbb{R}^n$ is taken into account, we denote $[\mathbf{x}] = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_n, \overline{x}_n] \in \mathbb{IR}^n$, where $\mathbb{IR}^n$ denotes the set of compact interval in $\mathbb{R}^n$. The width of an interval vector $\mathbf{x}$ is the largest of the widths of any of its component intervals $w([\mathbf{x}]) = \max_{i=1,\ldots,n}(\overline{x}_i - \underline{x}_i)$. A mapping $[\Phi] : \mathbb{IR}^n \to \mathbb{IR}^m$ denotes the interval extension of a function $\Phi : \mathbb{R}^n \to \mathbb{R}^m$. An interval extension is inclusion monotonic if, for any $[\mathbf{x}_1], [\mathbf{x}_2] \in \mathbb{IR}^n$, $[\mathbf{x}_1] \subseteq [\mathbf{x}_2]$ implies $[\Phi]([\mathbf{x}_1]) \subseteq [\Phi]([\mathbf{x}_2])$. A fundamental property of inclusion monotonic interval extensions is that $\mathbf{x} \in [\mathbf{x}] \Rightarrow \Phi(\mathbf{x}) \in [\Phi]([\mathbf{x}])$, which means the value of Φ is contained in the interval $[\Phi]([\mathbf{x}])$ for every $\mathbf{x}$ in $[\mathbf{x}]$.

Definition 6 [31]: Piecewise monotone functions, including absolute value, exponential, logarithm, rational power, and trigonometric functions, are standard functions.

Lemma 2 [31]: A function Φ composed by finitely many standard functions with elementary operations $\{+, -, \cdot, /\}$ is inclusion monotone.

Definition 7 [31]: Given an interval extension $[\Phi]([\mathbf{x}])$, if there is a constant ξ such that $w([\Phi]([\mathbf{x}])) \leq \xi w([\mathbf{x}])$ for every $[\mathbf{x}] \subseteq [\mathbf{x}_0]$, then $[\Phi]([\mathbf{x}])$ is said to be Lipschitz in $[\mathbf{x}_0]$.

Lemma 3 [31]: If a function $\Phi(\mathbf{x})$ satisfies a Lipschitz condition in $[\mathbf{x_0}]$,

$$\|\Phi(\mathbf{x}_2) - \Phi(\mathbf{x}_1)\| \leq \xi\|\mathbf{x}_2 - \mathbf{x}_1\|, \ \mathbf{x}_1, \mathbf{x}_2 \in [\mathbf{x}_0] \tag{17}$$

then the interval extension $[\Phi]([\mathbf{x}])$ is a Lipschitz interval extension in $[\mathbf{x}_0]$,

$$w([\Phi]([\mathbf{x}])) \leq \xi w([\mathbf{x}]), \ [\mathbf{x}] \subseteq [\mathbf{x_0}]. \tag{18}$$

Assumption 1: The activation function ϕ considered in this article is composed of standard functions with finitely many elementary operations.

The above assumption allows that the reachability analysis of MLP can be conducted in the framework of interval arithmetic, and to our knowledge, popular activation functions such as tanh, sigmoid, and ReLU satisfy this assumption.

B. Interval Analysis

First, we consider a single layer $\mathbf{u} = \phi(\mathbf{W}\boldsymbol{\eta} + \boldsymbol{\theta})$. Given an interval input $[\boldsymbol{\eta}]$, the interval extension is $[\phi](\mathbf{W}[\boldsymbol{\eta}] + \boldsymbol{\theta}) = [\underline{u}_1, \overline{u}_1] \times \cdots \times [\underline{u}_n, \overline{u}_n] = [\mathbf{u}]$, where

$$\underline{u}_i = \min_{\boldsymbol{\eta} \in [\boldsymbol{\eta}]} \phi\left(\sum_{j=1}^{n} \omega_{ij}\eta_j + \theta_i\right) \tag{19}$$

$$\overline{u}_i = \max_{\boldsymbol{\eta} \in [\boldsymbol{\eta}]} \phi\left(\sum_{j=1}^{n} \omega_{ij}\eta_j + \theta_i\right). \tag{20}$$

According to (19) and (20), the minimum and maximum values of the output of nonlinear function ϕ are required to compute the interval extension $[\phi]$. However, the optimization problems are still challenging for general nonlinear functions. We propose the following monotonic assumption for activation functions.

Assumption 2: For any two scalars $\eta_1 \leq \eta_2$, the activation function satisfies $\phi(\eta_1) \leq \phi(\eta_2)$.

It worth mentioning that Assumption 2 can be satisfied by a variety of activation functions such as logistic, tanh, and ReLU, all satisfy Assumption 2. Taking advantage of the monotonic property of ϕ, we have interval extension $[\phi]([\eta]) = [\phi(\underline{\eta}), \phi(\overline{\eta})]$. Therefore, $\underline{u}_i$ and $\overline{u}_i$ in (19) and (20) can be explicitly written out as

$$\underline{u}_i = \phi\left(\textstyle\sum_{j=1}^{n} \underline{p}_{ij} + \theta_i\right) \tag{21}$$

$$\overline{u}_i = \phi\left(\textstyle\sum_{j=1}^{n} \overline{p}_{ij} + \theta_i\right) \tag{22}$$

with $\underline{p}_{ij}$ and $\overline{p}_{ij}$ defined by

$$\underline{p}_{ij} = \begin{cases} \omega_{ij}\underline{\eta}_j, & \omega_{ij} \geq 0 \\ \omega_{ij}\overline{\eta}_j, & \omega_{ij} < 0 \end{cases} \tag{23}$$

$$\overline{p}_{ij} = \begin{cases} \omega_{ij}\overline{\eta}_j, & \omega_{ij} \geq 0 \\ \omega_{ij}\underline{\eta}_j, & \omega_{ij} < 0. \end{cases} \tag{24}$$

From (21)–(24), the output interval of a single layer can be efficiently computed with these explicit expressions. Then, we consider the MLP $\mathbf{u}^{\{L\}} = \Phi(\boldsymbol{\eta}^{\{0\}})$ with multiple layers,

the interval extension $[\Phi]([\boldsymbol{\eta}^{\{0\}}])$ can be computed by the following layer-by-layer computation.

Theorem 1: Given an MLP in the form of (11) with activation functions satisfying Assumption 2 and an interval input $[\boldsymbol{\eta}^{\{0\}}]$, an interval extension can be determined by

$$[\Phi]([\boldsymbol{\eta}^{\{0\}}]) = [\hat{\phi}_L] \circ \cdots \circ [\hat{\phi}_1] \circ [\hat{\phi}_0]([\boldsymbol{\eta}^{\{0\}}]) \tag{25}$$

where $[\hat{\phi}_\ell]([\boldsymbol{\eta}^{\{\ell\}}]) = [\phi_\ell](\mathbf{W}^{\{\ell\}}[\boldsymbol{\eta}^{\{\ell\}}] + \boldsymbol{\theta}^{\{\ell\}}) = [\mathbf{u}^{\{\ell\}}]$ in which

$$\underline{u}_i^{\{\ell\}} = \phi_\ell\left(\sum_{j=1}^{n^{\{\ell\}}} \underline{p}_{ij}^{\{\ell\}} + \theta_i^{\{\ell\}}\right) \tag{26}$$

$$\overline{u}_i^{\{\ell\}} = \phi_\ell\left(\sum_{j=1}^{n^{\{\ell\}}} \overline{p}_{ij}^{\{\ell\}} + \theta_i^{\{\ell\}}\right) \tag{27}$$

with $\underline{p}_{ij}^{\{\ell\}}$ and $\overline{p}_{ij}^{\{\ell\}}$ defined by

$$\underline{p}_{ij}^{\{\ell\}} = \begin{cases} \omega_{ij}^{\{\ell\}} \underline{\eta}_j^{\{\ell\}}, & \omega_{ij}^{\{\ell\}} \geq 0 \\ \omega_{ij}^{\{\ell\}} \overline{\eta}_j^{\{\ell\}}, & \omega_{ij}^{\{\ell\}} < 0 \end{cases} \tag{28}$$

$$\overline{p}_{ij}^{\{\ell\}} = \begin{cases} \omega_{ij}^{\{\ell\}} \overline{\eta}_j^{\{\ell\}}, & \omega_{ij}^{\{\ell\}} \geq 0 \\ \omega_{ij}^{\{\ell\}} \underline{\eta}_j^{\{\ell\}}, & \omega_{ij}^{\{\ell\}} < 0. \end{cases} \tag{29}$$

Proof: We denote $\hat{\phi}_\ell(\boldsymbol{\eta}^{\{\ell\}}) = \phi_\ell(\mathbf{W}^{\{\ell\}}\boldsymbol{\eta}^{\{\ell\}} + \boldsymbol{\theta}^{\{\ell\}})$. Given an MLP, it essentially has $\boldsymbol{\eta}^{\{\ell\}} = \hat{\phi}_{\ell-1}(\boldsymbol{\eta}^{\{\ell-1\}})$, $\ell = 1, \ldots, L$ which leads to (25). Then, for each layer, the interval extension $[\mathbf{u}^{\{\ell\}}]$ computed by (26)–(29) can be obtained by (21)–(24).

According to the explicit expressions (25)–(29), the computation on interval extension $[\Phi]$ can be performed in a fast manner. In the next step, we should discuss the conservativeness for the computation outcome of (25)–(29).

Theorem 2: The interval extension $[\Phi]$ of neural network Φ composed by activation functions satisfying Assumption 1 is inclusion monotonic and Lipschitz such that

$$w([\Phi]([\boldsymbol{\eta}])) \leq \xi^L \prod_{\ell=1}^{L} \left\|\mathbf{W}^{\{\ell\}}\right\| w([\boldsymbol{\eta}]), \ [\boldsymbol{\eta}] \subseteq \mathbb{IR}^{n^{\{0\}}} \tag{30}$$

where ξ is a Lipschitz constant for activation functions in Φ.

Proof: Under Assumption 1, the inclusion monotonicity can be obtained directly based on Lemma 2. Then, we denote $\hat{\phi}_\ell(\boldsymbol{\eta}^{\{\ell\}}) = \phi_\ell(\mathbf{W}^{\{\ell\}}\boldsymbol{\eta}^{\{\ell\}} + \boldsymbol{\theta}^{\{\ell\}})$. For any $\boldsymbol{\eta}_1, \boldsymbol{\eta}_2$, it has

$$\begin{aligned} &\left\|\hat{\phi}_\ell(\boldsymbol{\eta}_2^{\{\ell\}}) - \hat{\phi}_\ell(\boldsymbol{\eta}_1^{\{\ell\}})\right\| \\ &\quad \leq \xi \left\|\mathbf{W}^{\{\ell\}}\boldsymbol{\eta}_2^{\{\ell\}} - \mathbf{W}\boldsymbol{\eta}_1^{\{\ell\}}\right\| \\ &\quad \leq \xi \left\|\mathbf{W}^{\{\ell\}}\right\| \left\|\boldsymbol{\eta}_2^{\{\ell\}} - \boldsymbol{\eta}_1^{\{\ell\}}\right\|. \end{aligned}$$

Due to $\boldsymbol{\eta}^{\{\ell\}} = \hat{\phi}_{\ell-1}(\boldsymbol{\eta}^{\{\ell-1\}})$, $\ell = 1, \ldots, L$, $\xi^L \prod_{\ell=1}^{L} \left\|\mathbf{W}^{\{\ell\}}\right\|$ becomes the Lipschitz constant for Φ, and (30) can be established by Lemma 3.

We denote the set image for neural network Φ as follows:

$$\Phi([\boldsymbol{\eta}^{\{0\}}]) = \{\Phi(\boldsymbol{\eta}^{\{0\}}) : \boldsymbol{\eta}^{\{0\}} \in [\boldsymbol{\eta}^{\{0\}}]\}. \tag{31}$$

Since $[\Phi]$ is inclusion monotonic according to Theorem 2, one has $\Phi([\boldsymbol{\eta}^{\{0\}}]) \subseteq [\Phi]([\boldsymbol{\eta}^{\{0\}}])$. We have $[\Phi]([\boldsymbol{\eta}^{\{0\}}]) = \Phi([\boldsymbol{\eta}^{\{0\}}]) + E([\boldsymbol{\eta}^{\{0\}}])$ for some interval-valued function $E([\boldsymbol{\eta}^{\{0\}}])$ such that $w([\Phi]([\boldsymbol{\eta}^{\{0\}}])) = w(\Phi([\boldsymbol{\eta}^{\{0\}}])) + w(E([\boldsymbol{\eta}^{\{0\}}]))$.

Definition 8: $w(E([\boldsymbol{\eta}^{\{0\}}])) = w([\Phi]([\boldsymbol{\eta}^{\{0\}}])) - w(\Phi([\boldsymbol{\eta}^{\{0\}}]))$ is the excess width of interval extension of neural network $\Phi([\boldsymbol{\eta}^{\{0\}}])$.

Explicitly, the excess width measures the conservativeness of interval extension $[\Phi]$ regarding its corresponding function Φ. The following theorem gives the upper bound of the excess width $w(E([\boldsymbol{\eta}^{\{0\}}]))$.

Theorem 3: Given an MLP in the form of (11) with an interval input $[\boldsymbol{\eta}^{\{0\}}]$, the excess width $w(E([\boldsymbol{\eta}^{\{0\}}]))$ satisfies

$$w(E([\boldsymbol{\eta}^{\{0\}}])) \leq \gamma \, w([\boldsymbol{\eta}^{\{0\}}]) \tag{32}$$

where $\gamma = \xi^L \prod_{\ell=1}^{L} \left\|\mathbf{W}^{\{\ell\}}\right\|$.

Proof: We have $[\Phi]([\boldsymbol{\eta}^{\{0\}}]) = \Phi([\boldsymbol{\eta}^{\{0\}}]) + E([\boldsymbol{\eta}^{\{0\}}])$ for some $E([\boldsymbol{\eta}^{\{0\}}])$ and

$$\begin{aligned} w(E([\boldsymbol{\eta}^{\{0\}}])) &= w([\Phi]([\boldsymbol{\eta}^{\{0\}}])) - w(\Phi([\boldsymbol{\eta}^{\{0\}}])) \\ &\leq w([\Phi]([\boldsymbol{\eta}^{\{0\}}])) \\ &\leq \xi^L \prod_{\ell=1}^{L} \left\|\mathbf{W}^{\{\ell\}}\right\| w([\boldsymbol{\eta}^{\{0\}}]) \end{aligned}$$

which means (32) holds.

Given a neural network Φ which means $\mathbf{W}^{\{\ell\}}$ and ξ are fixed, Theorem 3 implies that a less conservative result can be only obtained by reducing the width of input interval $[\boldsymbol{\eta}^{\{0\}}]$. On the other hand, a smaller $w([\boldsymbol{\eta}^{\{0\}}])$ means more subdivisions of an input interval which will bring more computational cost. Therefore, how to generate appropriate subdivisions of an input interval is the key issue for the reachability analysis of neural networks in the framework of interval arithmetic. In the next section, an efficient simulation-guided method is proposed to address this key problem.

C. Simulation-Guided Reachability Analysis

Inspired by the Moore–Skelboe algorithm [32], we propose a reachable set computation algorithm under the guidance of a finite number of simulations. It proposes an adaptive input interval partitioning scheme with the help of simulations. The simulation-guided algorithm shown in Algorithm 1 checks the emptiness of the intersection between the computed output set and the over-approximation interval for simulations, within a predefined tolerance ε. This algorithm is able to avoid unnecessary partition for the input interval to get a tight output range. The tightness of reachable set estimation is accomplished by dividing and checking the initial input interval into increasingly smaller subintervals, as seen in Algorithm 1.

1) *Initialization:* Perform N simulations for neural network Φ to get N output points $\mathbf{u}_{\text{sim},n}$, $n = 1, \ldots, N$ and compute an interval $[\mathbf{u}_{\text{sim}}]$ such that $\mathbf{u}_{\text{sim},n} \in [\mathbf{u}_{\text{sim}}]$, $\forall n$. The N simulations can be generated either randomly or by gridding input set. Since our approach is based on interval analysis, convert input set $\mathcal{H}$ to an interval $[\boldsymbol{\eta}]$ such that $\mathcal{H} \subseteq [\boldsymbol{\eta}]$. Compute the initial output interval $[\mathbf{u}] = [\Phi]([\boldsymbol{\eta}])$ by (25)–(29). Initialize set $\mathcal{M} = \{([\boldsymbol{\eta}], [\mathbf{u}])\}$. Set a tolerance $\varepsilon > 0$, which will be used to terminate algorithm.

Algorithm 1 Simulation-Guided Reachable Set Estimation

Input : Feedforward neural network Φ ; Input set $\mathcal{H}$; Tolerance ε; Number of simulations N.
Output: Output set estimation $\mathcal{U}_e$.

Function *reachMLP*
  /* Initialization */
  Compute interval $[\boldsymbol{\eta}]$ such that $\mathcal{H} \subseteq [\boldsymbol{\eta}]$;
  $[\mathbf{u}] \leftarrow [\Phi]([\boldsymbol{\eta}])$; // Using (25)-(29)
  $\mathcal{M} \leftarrow \{([\boldsymbol{\eta}], [\mathbf{u}])\}$;
  Compute N simulations $\mathbf{u}_{\text{sim},n} = \Phi(\boldsymbol{\eta}_{\text{sim},n})$, $n = 1, \ldots, N$;
  Compute interval $[\mathbf{u}_{\text{sim}}]$ such that $\mathbf{u}_{\text{sim},n} \in [\mathbf{u}_{\text{sim}}]$, $\forall n$;
  $\mathcal{U}_e \leftarrow \emptyset$;
  /* Simulation-guided bisection */
  while $\mathcal{M} \neq \emptyset$ **do**
    Select and remove an element $([\boldsymbol{\eta}], [\mathbf{u}])$ from $\mathcal{M}$;
    if $[\mathbf{u}] \subseteq [\mathbf{u}_{\text{sim}}]$ **then**
      $\mathcal{U}_e \leftarrow \mathcal{U}_e \cup [\mathbf{u}]$;
      Continue ;
    else
      if $w([\boldsymbol{\eta}]) > \varepsilon$ **then**
        Bisect $[\boldsymbol{\eta}]$ to obtain $[\boldsymbol{\eta}_1]$ and $[\boldsymbol{\eta}_2]$;
        $[\mathbf{u}_1] \leftarrow [\Phi]([\boldsymbol{\eta}_1])$; // Using (25)-(29)
        $[\mathbf{u}_2] \leftarrow [\Phi]([\boldsymbol{\eta}_2])$; // Using (25)-(29)
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{([\mathbf{u}_1], [\boldsymbol{\eta}_1])\} \cup \{([\mathbf{u}_2], [\boldsymbol{\eta}_2])\}$;
      else
        Break ; // Bisection terminates
      end
    end
  end
  return $\mathcal{U}_e \leftarrow \mathcal{U}_e \cup \left(\bigcup_{\{[\boldsymbol{\eta}],[\mathbf{u}]\} \in \mathcal{M}} [\mathbf{u}]\right)$

2) *Simulation-Guided Bisection:* This is the key step in the algorithm. Select an element $([\boldsymbol{\eta}], [\mathbf{u}])$ for simulation-guided bisection. If the output interval $[\mathbf{u}]$ satisfies $[\mathbf{u}] \subseteq [\mathbf{u}_{\text{sim}}]$, we can discard this subinterval for the subsequent dividing and checking since it has been proven to be included in the output range. Otherwise, the bisection action will be activated to produce finer subdivisions to be added to $\mathcal{M}$ for subsequent checking. The bisection process is guided by simulations since the bisection actions are totally determined by the nonemptiness of the intersection between output interval sets and the interval for simulations. This distinguishing feature leads to finer subdivisions when the output set is getting close to the boundary of interval for simulations, and on the other hand, coarse subdivisions are sufficient for interval reachability analysis when the output set is included in the interval for simulations. Therefore, unnecessary computational cost can be avoided.
3) *Termination:* The simulation-guided bisection continues until the width of subdivisions becomes less than the predefined tolerance ε. Generally, a smaller tolerance ε will lead to a tighter output interval computation result.

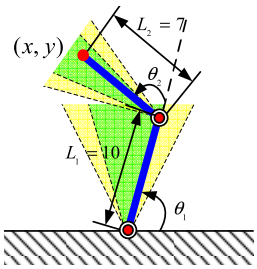


Fig. 2. Robotic arm with two joints. The normal working zone of (θ_1, θ_2) is colored in green $\theta_1, \theta_2 \in [(5\pi/12), (7\pi/12)]$. The buffering zone is in yellow $\theta_1, \theta_2 \in [(\pi/3), (5\pi/12)] \cup [(7\pi/12), (2\pi/3)]$. The forbidden zone is $\theta_1, \theta_2 \in [0, (\pi/3)] \cup [(2\pi/3), 2\pi]$.

D. Reachability Analysis of a Robotic Arm Model

In [26], a *learning forward kinematics* of a robotic arm model with two joints is proposed, shown in Fig. 2. The learning task is using a feedforward neural network to predict the position (x, y) of the end with knowing the joint angles (θ_1, θ_2). The input space $[0, 2\pi] \times [0, 2\pi]$ for (θ_1, θ_2) is classified into three zones for its operations: normal working zone $\theta_1, \theta_2 \in [(5\pi/12), (7\pi/12)]$, buffering zone $\theta_1, \theta_2 \in [(\pi/3), (5\pi/12)] \cup [(7\pi/12), (2\pi/3)]$, and forbidden zone $\theta_1, \theta_2 \in [0, (\pi/3)] \cup [(2\pi/3), 2\pi]$. The detailed formulation for this robotic arm model and neural network training can be found in [26].

In [26], a uniform partition of input interval, which is the union of normal working and buffering zones $(\theta_1, \theta_2) \in [(\pi/3), (2\pi/3)] \times [(\pi/3), (2\pi/3)]$, is used to compute an over-approximation for safety verification. The safety specification for the position (x, y) is an interval set $\mathcal{S} = \{(x, y) \mid -14 \leq x \leq 3 \text{ and } 1 \leq y \leq 17\}$. To illustrate the advantages of simulation-guided approach, we aim to compute a tight output interval using both uniform partition method in [26] and Algorithm 1. The precision/tolerance for both methods is chosen the same, $\varepsilon = 0.01$. The number of simulations used in Algorithm 1 is set to be 1000. The computed output ranges are shown in Figs. 3 and 4. It can be clearly observed that two methods can produce same output range estimations, that is $\mathcal{U}_e = \{(x, y) \mid -12.0258 \leq x \leq 1.1173 \text{ and } 2.8432 \leq y \leq 14.8902\}$ which is sufficient to ensure the safety due to $\mathcal{U}_e \subseteq \mathcal{S}$. Though both methods can achieve same output range analysis results, the computation costs are significantly different as shown in Table I. In [26], a uniform partition for input space is used, and it results in 16 384 intervals with precision $\varepsilon = 0.01$ and the computation takes 4.4254 s. Using our simulation-guided approach, the safety can be guaranteed by partitioning the input space into 397 intervals (2.42% of those by uniform partition method in [26]) with tolerance $\varepsilon = 0.01$. The simulation-guided partition of the input interval $[(\pi/3), (2\pi/3)] \times [(\pi/3), (2\pi/3)]$ is shown in Fig. 5. Along with the less number of intervals involved in the computation process, the computational time is 0.1423 s (3.22% of that by uniform partition method in [26]) for simulation-guided approach.[2]

[2]The source code is available at: https://github.com/xiangweiming/ignnv

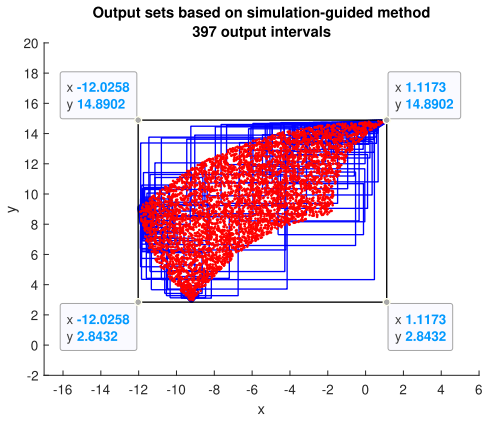


Fig. 3. Output intervals obtained by simulation-guided methods. The 397 output intervals (blue rectangles) are generated and the output range is $\mathcal{U}_e = \{(x, y) \mid -12.0258 \leq x \leq 1.1173 \text{ and } 2.8432 \leq y \leq 14.8902\}$ (black rectangle). Red points are 5000 random outputs which are all included in output intervals.

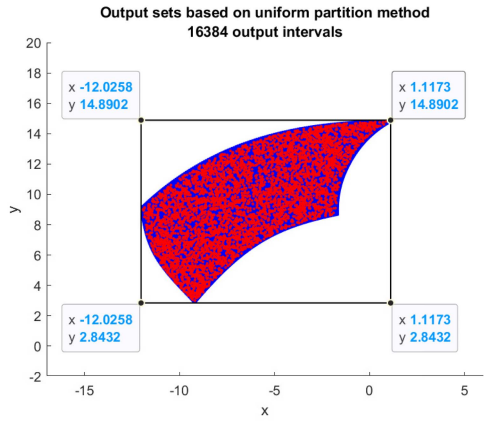


Fig. 4. Output intervals obtained by uniform partition method in [26]. The 16384 output intervals (blue rectangles) are generated and the output range is $\mathcal{U}_e = \{(x, y) | -12.0258 \leq x \leq 1.1173 \text{ and } 2.8432 \leq y \leq 14.8902\}$ (black rectangle). Red points are 5000 random outputs which are all included in output intervals.

TABLE I

COMPARISON ON NUMBER OF INTERVALS AND COMPUTATIONAL TIME BETWEEN SIMULATION-GUIDED METHOD AND UNIFORM PARTITIONING METHOD

	Intervals	Computational Time
Algorithm 1	397	0.1423 seconds
Xiang *et al.* 2018 [26]	16384	4.4254 seconds

IV. REACHABILITY ANALYSIS FOR NEURAL NETWORK CONTROL SYSTEMS

A. Reachability Analysis

The reachable set estimation for a sampled-data neural network control system in the form of (6) involves two essential portions. First, an over-approximation of the output set of the underlying neural network controllers is supposed to be computed in the employment of the aforementioned output set computation result of neural networks, Algorithm 1. Then, the reachable set and output set of the controlled plant (1) needs to be computed accordingly. There are a variety of existing approaches and tools for reachable set computation of systems modeled by ODEs such as those well-developed in [33]–[37].

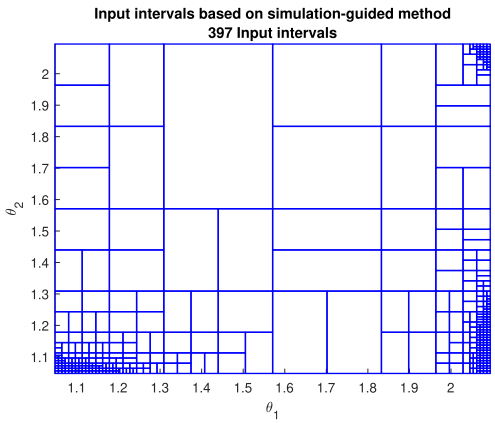


Fig. 5. Simulation-guided bisections of input interval by Algorithm 1. Guided by the outputs of simulations, finer partitions are generated when the output intervals are close to the boundary of the interval of simulations, and coarse partitions are generated when the output intervals are in the interval of simulations.

Due to the existence of those reachable set estimation of ODE models, we shall not develop new methods or tools for ODE models. We use the following functions to denote the reachable set estimation that is obtained by using reachable set computation tools for sampled data ODE models during $[t_k, t_{k+1}]$

$$\mathcal{R}_e([t_k, t_{k+1}]) = \texttt{reachODEx}(f, \mathcal{U}(t_k), \mathcal{R}_e(t_k)) \quad (33)$$

$$\mathcal{Y}_e(t_k) = \texttt{reachODEy}(h, \mathcal{R}_e(t_k)) \quad (34)$$

where $\mathcal{U}(t_k)$ is the input set for sampling interval $[t_k, t_{k+1}]$. $\mathcal{R}_e(t_k)$ and $\mathcal{R}_e([t_k, t_{k+1}])$ are the estimated reachable sets for state $\mathbf{x}(t)$ at sampling instant t_k and interval $[t_k, t_{k+1}]$, respectively. $\mathcal{Y}_e(t_k)$ is the estimated reachable set for output $\mathbf{y}(t_k)$.

Combining `reachODEx`, `reachODEy` with `reachMLP` proposed by Algorithm 1, an over-approximation of the reachable set of a closed-loop system in the form of (6) can be obtained. The computation process is a recursive algorithm, which is summarized by Algorithm 2 and Proposition 1. The general steps can be illustrated as below:

1) *Reachable Set Estimation of Neural Network Controller:* Compute the output reachable set estimation for the neural network controller using Algorithm 1 at each beginning sampling instant t_k, by which an over-approximation of the output set is obtained.
2) *Reachable Set Estimation of Plant:* As the output generated by the neural network controller holds its value unchanged in $[t_k, t_{k+1}]$, perform the reachable set estimation for the nonlinear continuous-time system using sophisticated methods or tools such as [33]–[37].
3) *Return for Next Sampling Interval Computation:* Return to the first step of reachable set estimation of neural network controller for the next sampling period $[t_{k+1}, t_{k+2}]$.

Proposition 1: Given a neural network control system in the form of (6), an initial set $\mathcal{X}_0$ and an input set $\mathcal{V}$, an estimated reachable set $\mathcal{R}_e([t_0, t_f])$ can be obtained by Algorithm 2 such that $\mathcal{R}([t_0, t_f]) \subseteq \mathcal{R}_e([t_0, t_f])$, where $\mathcal{R}([t_0, t_f])$ is the reachable set of system (6).

Algorithm 2 Reachable Set Estimation for Sampled-Data Neural Network Control Systems

Input : System dynamics f, h; Feedforward neural network Φ ; Initial Set $\mathcal{X}_0$; Input set $\mathcal{V}$; Tolerance ε; Number of simulations N; Sampling sequence t_k, $k = 0, 1, \ldots, K$; Termination time t_f.

Output: Reachable set estimation $\mathcal{R}_e([t_0, t_f])$.

Function *reachNNCS*

/* Initialization */

$k \leftarrow 0$;

$t_{K+1} \leftarrow t_f$;

$\mathcal{R}_e(t_0) \leftarrow \mathcal{X}_0$;

/* Iteration for all sampling intervals */

while $k \leq K$ **do**

$\mathcal{Y}_e(t_k) \leftarrow \texttt{reachODEy}(h, \mathcal{R}_e(t_k))$;

$\mathcal{H} \leftarrow \mathcal{Y}_e(t_k) \times \mathcal{V}$;

$\mathcal{U}_e(t_k) \leftarrow \texttt{reachMLF}(\Phi, \mathcal{H}, \varepsilon, N)$;

// Algorithm 1

$\mathcal{R}_e([t_k, t_{k+1}]) \leftarrow \texttt{reachODEx}(f, \mathcal{U}_e, \mathcal{R}_e(t_k))$;

$k \leftarrow k + 1$;

end

return $\mathcal{R}_e([t_0, t_f]) \leftarrow \bigcup_{k=0,1\ldots,K} \mathcal{R}_e([t_k, t_{k+1}])$

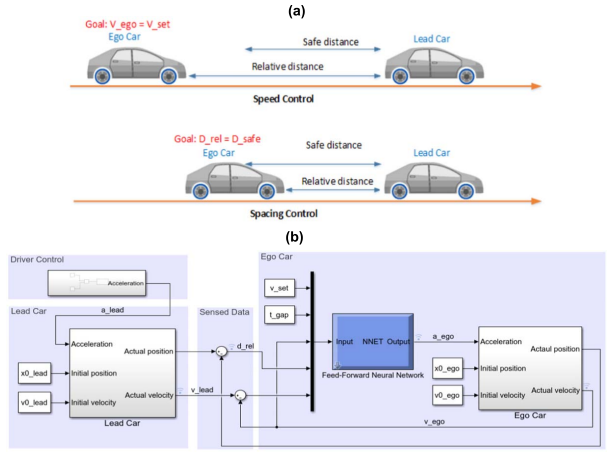


Fig. 6. Illustration of ACC systems and simulink block diagram of the closed-loop system. (a) Adaptive cruise control. (b) Simulink model of adaptive cruise control system.

The safety specification can be examined by checking the emptiness of the intersection between the proposed unsafe regions $\neg\mathcal{S}$ and the reachable set estimation outcome produced by Algorithm 2.

Proposition 2: Given a neural network control system in the form of (6) and a safety specification $\mathcal{S}$, if $\mathcal{R}_e([t_0, t_f]) \cap \neg\mathcal{S} = \emptyset$, where $\mathcal{R}_e([t_0, t_f])$ is a reachable set estimation obtained by Algorithm 2, then the closed-loop system (6) is safe over time interval $[t_0, t_f]$.

B. Safety Verification of ACC Systems

In this section, our approach will be evaluated by the safety verification of an ACC system equipped with a neural network controller as depicted in Fig. 6. The ACC system consists of two cars, the ego car with ACC module that has a radar sensor to measure the distance to the lead car which is denoted by d_{rel}, and the relative velocity against the lead car denoted by v_{rel}. There are two system operating modes including speed control and spacing control. In the speed control mode, the ego car travels at a speed v_{set}. In the spacing control mode, the ego car's safety control goal is to maintain a safe distance from the leading car, d_{safe}. In summary, the system dynamics is in the form of

$$\begin{cases} \dot{x}_l(t) = v_l(t) \\ \dot{v}_l(t) = \gamma_l(t) \\ \dot{\gamma}_l(t) = -2\gamma_l(t) + 2\alpha_l(t) - \mu v_l^2(t) \\ \dot{x}_e = v_e(t) \\ \dot{v}_e(t) = \gamma_e(t) \\ \dot{\gamma}_e(t) = -2\gamma_e(t) + 2\alpha_e(t) - \mu v_e^2(t) \end{cases} \tag{35}$$

where $x_l(x_e)$, $v_l(v_e)$, and $\gamma_l(\gamma_e)$ are the position, velocity, and actual acceleration of the lead (ego) car, respectively. $\alpha_l(\alpha_e)$ is the acceleration control input applied to the lead (ego) car, and $\mu = 0.001$ is the friction parameter. The ACC controller we considered here is a 2×20 feedforward neural network with `tanh` as its activation functions. The sampling scheme is considered as a periodic sampling every 0.2 s, that is $t_{k+1} - t_k = 0.2$ s.

The inputs to the neural network ACC control module are:

1) driver-set velocity v_{set};
2) time gap t_{gap};
3) velocity of the ego car v_e;
4) relative distance to the lead car $d_{\text{rel}} = x_l - x_e$;
5) relative velocity to the lead car $v_{\text{rel}} = v_l - v_e$.

The output for the neural network ACC controller is the acceleration of the ego car, α_e. In summary, the sampled-data neural network controller for the acceleration control of the ego car is in the form of

$$\alpha_e(t) = \Phi(v_{\text{set}}(t_k), t_{\text{gap}}, v_e(t_k), d_{\text{rel}}(t_k), v_{\text{rel}}(t_k)), \quad t \in [t_k, t_{k+1}].$$

The threshold of the safe distance between the lead car and the ego car can be considered as a function of the ego car velocity v_e. The safety specification is defined as

$$d_{\text{safe}} > d_{\text{thold}} = d_{\text{def}} + t_{\text{gap}} \cdot v_e \tag{36}$$

where d_{def} is the standstill default spacing and t_{gap} is the time gap between the vehicles. The safety verification scenario is that the lead car decelerates with $\alpha_l = -2$ to reduce its speed as an emergency braking occurs. We expect that the ego car is able to maintain a safe relative distance to the lead car to avoid collision. The safety specification is defined by (36) with $t_{\text{gap}} = 1.4$ s and $d_{\text{def}} = 10$. The time horizon that we want to verify is 5 s after the emergency braking comes into play. The initial intervals are $[x_l(0)] = [94, 96]$, $[v_l(0)] = [30, 30.2]$, $[\gamma_l(0)] = 0$, $[x_e(0)] = [10, 11]$, $[v_e(0)] = [30, 30.2]$, and $[\gamma_e(0)] = 0$.

We apply Algorithm 2 to perform the reachable set estimation for the closed-loop system. The tolerance is chosen as $\varepsilon = 0.1$ and number of simulations is 1×10^5. For this

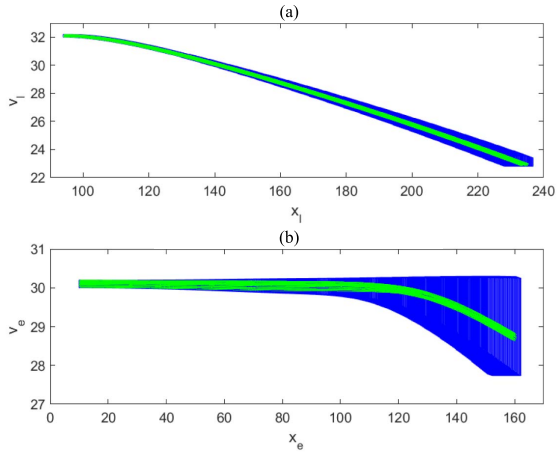


Fig. 7. Reachable set estimation for both lead car (a) and ego car (b). The over-approximation of reachable set (blue boxes) includes all 100 randomly generated system trajectories (green lines).

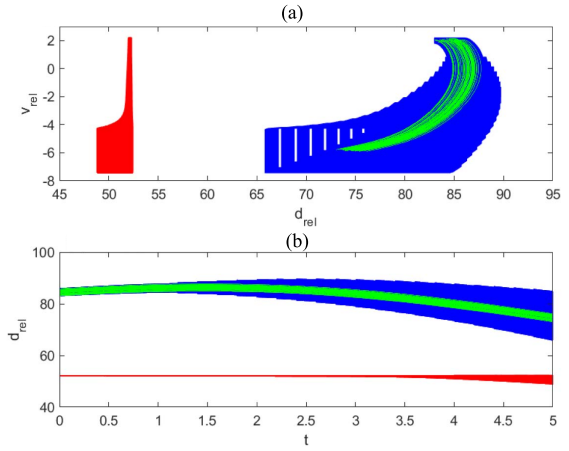


Fig. 8. Reachable set estimation for relative distance and velocity between lead and ego cars (blue boxes), there is no intersection between relative distance set and safe distance threshold (red boxes) in (a). In (b), the flowpipe of relative distance (blue) has no intersection with the safe distance threshold area (red) which also implies the safety of the ACC system. The green lines are 100 randomly generated system trajectories.

neural network controller, we use a simulation-guided method to compute the output set of the control signal. Meanwhile, for the continuous-time nonlinear dynamics, we use CORA [33] to do the reachability analysis for the time interval between two sampling instants. The reachable set estimations for both lead car and ego car are shown in Fig. 7. In order to verify the safety property, we compute the reachable set estimation of relative distance based on the reachable sets of the lead car and ego car. In Fig. 8, the reachable set of relative distance does not violate the threshold of safe distance which is defined by (36), so it can be concluded that the ACC system is safe during the time interval [0, 5] s in this safety verification scenario of interest.[3]

V. Conclusion

This article investigated the reachable set estimation and safety verification problems for a class of neural network control systems which can be modeled as sampled data continuous-time dynamical systems. A novel simulation-guided approach is developed to soundly over-approximate the output set of a class of feedforward neural networks called MLP. Based on the interval analysis of neural networks and guidance of simulations generated from neural networks, the output reachable set can be efficiently over-approximated upon avoidance of unnecessary computation cost. Compared with the other simulation-based approach in [26], the approach developed in this article can reduce the computational cost significantly. Furthermore, in a combination of reachable set computation methods and tools for ODE models, a recursive algorithm is developed to perform reachable set estimation and safety verification of neural network control systems. Beyond the initial results derived in this article, other modeling and reachability analysis approaches for the plant and neural network controllers, as well as broader classes of neural networks, should be considered in the future study.

[3]The source code is available at: https://github.com/xiangweiming/ignnv

Acknowledgment

The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR or DARPA.

References

[1] Z.-G. Wu, P. Shi, H. Su, and J. Chu, "Exponential stabilization for sampled-data neural-network-based control systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 12, pp. 2180–2190, Dec. 2014.

[2] S.-H. Yu and A. M. Annaswamy, "Stable neural controllers for nonlinear dynamic systems," *Automatica*, vol. 34, no. 5, pp. 641–650, May 1998.

[3] S. S. Ge, C. C. Hang, and T. Zhang, "Adaptive neural network control of nonlinear systems by state and output feedback," *IEEE Trans. Syst., Man, Cybern. B. Cybern.*, vol. 29, no. 6, pp. 818–828, Dec. 1999.

[4] K. J. Hunt, D. Sbarbaro, R. Z. Bikowski, and P. J. Gawthrop, "Neural networks for control systems: A survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[5] K. D. Julian and M. J. Kochenderfer, "Neural network guidance for UAVs," in *Proc. AIAA Guid., Navigat., Control Conf.*, Jan. 2017, p. 1743.

[6] M. Bojarski *et al.*, "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*. [Online]. Available: http://arxiv.org/abs/1604.07316

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

[8] C. Szegedy *et al.*, "Intriguing properties of neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–16.

[9] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Proc. Int. Conf. Comput. Aided Verification*, 2017, pp. 97–117.

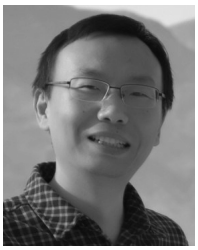
[10] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *Proc. Int. Conf. Comput. Aided Verification*, 2010, pp. 243–257.

[11] L. Pulina and A. Tacchella, "Challenging SMT solvers to verify neural networks," *AI Commun.*, vol. 25, no. 2, pp. 117–135, 2012.

[12] N. Narodytska, "Formal verification of deep neural networks," in *Proc. Formal Methods Comput. Aided Design (FMCAD)*, Oct. 2018, pp. 3–29.

[13] Z. Xu, H. Su, P. Shi, R. Lu, and Z.-G. Wu, "Reachable set estimation for Markovian jump neural networks with time-varying delays," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3208–3217, Oct. 2017.

[14] Z. Zuo, Z. Wang, Y. Chen, and Y. Wang, "A non-ellipsoidal reachable set estimation for uncertain neural networks with time-varying delay," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 19, no. 4, pp. 1097–1106, Apr. 2014.
[15] M. V. Thuan, H. M. Tran, and H. Trinh, "Reachable sets bounding for generalized neural networks with interval time-varying delay and bounded disturbances," *Neural Comput. Appl.*, vol. 29, no. 10, pp. 783–794, May 2018.
[16] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward ReLU neural networks," 2017, *arXiv:1706.07351*. [Online]. Available: http://arxiv.org/abs/1706.07351
[17] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep neural networks," 2017, *arXiv:1709.09130*. [Online]. Available: http://arxiv.org/abs/1709.09130
[18] R. Ehlers, "Formal verification of piecewise linear feed-forward neural networks," in *Proc. Int. Symp. Automated Technol. Verification Anal.*, 2017, pp. 269–286.
[19] W. Xiang, H.-D. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with ReLU activations," 2017, *arXiv:1712.08163*. [Online]. Available: http://arxiv.org/abs/1712.08163
[20] H.-D. Tran *et al.*, "Star-based reachability analysis for deep neural networks," in *Proc. Int. Symp. Formal Methods*, 2019, pp. 670–689.
[21] X. Zeng and D. S. Yeung, "Sensitivity analysis of multilayer perceptron to input and weight perturbations," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1358–1366, Oct. 2001.
[22] X. Zeng and D. S. Yeung, "A quantified sensitivity measure for multilayer perceptron to input perturbation," *Neural Comput.*, vol. 15, no. 1, pp. 183–212, Jan. 2003.
[23] S. W. Piche, "The selection of weight accuracies for madalines," *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 432–445, Mar. 1995.
[24] W. Xi-zhao, S. Qing-yan, M. Qing, and Z. Jun-hai, "Architecture selection for networks trained with extreme learning machine using localized generalization error model," *Neurocomputing*, vol. 102, pp. 3–9, Feb. 2013.
[25] D. Shi, D. S. Yeung, and J. Gao, "Sensitivity analysis applied to the construction of radial basis function networks," *Neural Netw.*, vol. 18, no. 7, pp. 951–957, Sep. 2005.
[26] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5777–5783, Nov. 2018.
[27] W. Xiang, H.-D. Tran, J. A. Rosenfeld, and T. T. Johnson, "Reachable set estimation and safety verification for piecewise linear systems with neural network controllers," 2018, *arXiv:1802.06981*. [Online]. Available: http://arxiv.org/abs/1802.06981
[28] W. Xiang, H. Tran, and T. T. Johnson, "Reachability analysis and safety verification for neural network control systems," in *Proc. AAAI Spring Symp. Verification Neural Netw.*, 2019, pp. 1–10.
[29] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *Proc. 22nd ACM Int. Conf. Hybrid Syst., Comput. Control*, New York, NY, USA, Apr. 2019, pp. 157–168.
[30] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: Verifying safety properties of hybrid systems with neural network controllers," in *Proc. 22nd ACM Int. Conf. Hybrid Syst., Comput. Control*, New York, NY, USA, Apr. 2019, pp. 169–178.
[31] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, vol. 110. Philadelphia, PA, USA: SIAM, 2009.
[32] S. Skelboe, "Computation of rational interval functions," *BIT Numer. Math.*, vol. 14, no. 1, pp. 87–95, Mar. 1974.
[33] M. Althoff, "An introduction to Cora 2015," in *Proc. Workshop Appl. Verification Continuous Hybrid Syst.*, 2015, pp. 1–8.
[34] G. Frehse *et al.*, "Spaceex: Scalable verification of hybrid systems," in *Proc. Int. Conf. Comput. Aided Verification*, 2011, pp. 379–395.
[35] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow: An analyzer for non-linear hybrid systems," in *Proc. Int. Conf. Comput. Aided Verification*, 2013, pp. 258–263.
[36] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok, "C2E2: A verification tool for stateflow models," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.*, 2015, pp. 68–82.
[37] S. Bak and P. S. Duggirala, "HyLAA: A tool for computing simulation-equivalent reachability for linear systems," in *Proc. 20th Int. Conf. Hybrid Syst., Comput. Control*, Apr. 2017, pp. 173–178.



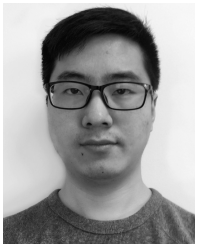
Weiming Xiang (Senior Member, IEEE) received the Ph.D. degree from Southwest Jiaotong University, Chengdu, China, in 2014, with his Ph.D. thesis on formal methods for hybrid traffic systems awarded the Outstanding Ph.D. dissertation of Southwest Jiaotong University in 2014.

Before joining Augusta University, he worked as a Post-Doctoral Research Scholar with the Department of Electrical Engineering and Computer Sciences, Vanderbilt University, Nashville, TN, USA, from September 2016 to July 2019, a Post-Doctoral Research Associate with the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX, USA, from November 2015 to August 2016, a Research Associate with the Department of Mechanical Engineering, The University of Hong Kong, Pok Fu Lam, Hong Kong, from May 2015 to October 2015. He is currently an Assistant Professor with the School of Computer and Cyber Sciences, Augusta University, Augusta, GA, USA. His research interest is developing formal synthesis and verification techniques and software tools for cyber-physical systems (CPS). His current research centers on formal methods on safety, security, and reliability of learning-enabled CPS. He is also broadly interested in methods and applications across CPS domains, such as control synthesis, stability analysis, reachable set computation, hybrid systems, power and energy, transportation, fuzzy logic, and neural networks.

Dr. Xiang was an Associate Editor of *Neurocomputing* (2014–2019), and he was the Leading Guest Editor of the Special Issue: Recent Advances in Control and Verification for Hybrid Systems in *IET Control Theory and Applications*, and he is currently on the Editorial Board of *IET Control Theory and Applications*.

Hoang-Dung Tran is a Ph.D. candidate in electrical engineering and computer science with the School of Engineering, Vanderbilt University, Nashville, TN, USA.

His research interests are in formal verification of autonomous cyber-physical systems with learning-enabled components, safe artificial intelligence, hybrid, and switching systems, distributed control system. He is also interested in robust control, stability analysis of nonlinear control systems, and networked control systems.



Xiaodong Yang is a Ph.D. candidate in electrical engineering and computer science with the School of Engineering, Vanderbilt University, Nashville, TN, USA.

His research interests are in formal verification of neural networks, system dynamics learning, and hybrid systems.

Taylor T. Johnson (Member, IEEE) received the M.Sc. and Ph.D. degrees from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2010 and 2013, respectively, both in electrical and computer engineering.

He is currently an Assistant Professor of electrical engineering and computer science at the Vanderbilt University, Nashville, TN, USA. His research interests include developing algorithmic techniques and software tools to improve the reliability of cyber-physical systems.

Dr. Johnson received the Air Force Office of Scientific Research Young Investigator Program Award in 2016 and the National Science Foundation Computer and Information Science and Engineering Research Initiation Initiative Award in 2015.