10-2019

ReachNN: Reachability Analysis of Neural-Network Controlled Systems

Chao Huang

Jiameng Fan

Wenchao Li

Xin Chen

Qi Zhu

ReachNN: Reachability Analysis of Neural-Network Controlled Systems

CHAO HUANG, Northwestern University
JIAMENG FAN and WENCHAO LI, Boston University
XIN CHEN, University of Dayton
QI ZHU, Northwestern University

Applying neural networks as controllers in dynamical systems has shown great promises. However, it is critical yet challenging to verify the safety of such control systems with neural-network controllers in the loop. Previous methods for verifying neural network controlled systems are limited to a few specific activation functions. In this work, we propose a new reachability analysis approach based on Bernstein polynomials that can verify neural-network controlled systems with a more general form of activation functions, i.e., as long as they ensure that the neural networks are Lipschitz continuous. Specifically, we consider abstracting feedforward neural networks with Bernstein polynomials for a small subset of inputs. To quantify the error introduced by abstraction, we provide both theoretical error bound estimation based on the theory of Bernstein polynomials and more practical sampling based error bound estimation, following a tight Lipschitz constant estimation approach based on forward reachability analysis. Compared with previous methods, our approach addresses a much broader set of neural networks, including heterogeneous neural networks that contain multiple types of activation functions. Experiment results on a variety of benchmarks show the effectiveness of our approach.

CCS Concepts: • **Theory of computation → Machine learning theory**; • **Computer systems organization → Embedded and cyber-physical systems**; • **Software and its engineering → Formal methods**;

Additional Key Words and Phrases: Neural network controlled systems, reachability, verification, Bernstein polynomials

ACM Reference format:
Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. 2019. ReachNN: Reachability Analysis of Neural-Network Controlled Systems. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 106 (October 2019), 22 pages.
https://doi.org/10.1145/3358228

This article appears as part of the ESWEEK-TECS special issue and was presented at the International Conference on Embedded Software (EMSOFT) 2019.

Authors' addresses: C. Huang and Q. Zhu, Northwestern University, Evanston, Illinois; emails: {chao.huang, qzhu}@northwestern.edu; J. Fan and W. Li, Boston University, Boston, Massachusetts; emails: {jmfan, wenchao}@bu.edu; X. Chen, University of Dayton, Dayton, Ohio; email: xchen4@udayton.edu.

1539-9087/2019/10-ART106 $15.00
https://doi.org/10.1145/3358228

1 INTRODUCTION

Data-driven control systems, especially neural-network-based controllers [27, 32, 33], have recently become the subject of intense research and demonstrated great promises. Formally verifying the safety of these systems however still remains an open problem. A Neural-Network Controlled System (NNCS) is essentially a continuous system controlled by a neural network, which produces control inputs at the beginning of each control step based on the current values of the state variables and feeds them back to the continuous system. Reachability of continuous or hybrid dynamical systems with traditional controllers has been extensively studied in the last decades. It has been proven that reachability of most nonlinear systems is undecidable [2, 20]. Recent approaches mainly focus on the overapproximation of reachable sets [13, 17, 22, 29, 34, 40]. The main difficulty impeding the direct application of these approaches to NNCS is the hardness of formally characterizing or abstracting the input-output mapping of a neural network.

Some recent approaches considered the problem of computing the output range of a neural network. Given a neural network along with a set of the inputs, these methods seek to compute an interval or a box (vector of intervals) that contains the set of corresponding outputs. These techniques are partly motivated by the study of robustness [16] of neural networks to adversarial examples [38]. Katz et al. [25] propose an SMT-based approach called Reluplex by extending the simplex algorithm to handle ReLU constraints. Huang et al. [23] use a refinement-by-layer technique to prove the absence or show the presence of adversarial examples around the neighborhood of a specific input. General neural networks with Lipschitz continuity are then considered by Ruan et al. [36], where the authors show that a large number of neural networks are Lipschitz continuous and the Lipschitz constant can help in estimating the output range which requires solving a global optimization problem. Dutta et al. [16] propose an efficient approach using mixed integer linear programming to compute the exact interval range of a neural network with only ReLU activation functions.

However, these existing methods cannot be directly used to analyze the reachability of dynamical systems controlled by neural networks. As the behavior of these systems is based on the interaction between the continuous dynamics and the neural-network controller, we need to not only compute the output range but also describe the input-output mapping for the controller. More precisely, we need to compute a tractable function model whose domain is the input set of the controller and its output range contains the set of the controller's outputs. We call such a function model a higher-order set, to highlight the distinction from intervals which are 0-order sets. Computing a tractable function model from the original model can also be viewed as a form of *knowledge distillation* [21] from the verification perspective, as the function model should be able to produce comparable results or replicate the outputs of the target neural network on specific inputs.

There have been some recent efforts on computing higher-order sets for the controllers in NNCS. Ivanov et al. [24] present a method to equivalently transform a system to a hybrid automaton by replacing a neuron in the controller with an ordinary differential equation (ODE). This method is however only applicable to differentiable neural-network controllers – ReLU neural networks are thus excluded. Dutta et al. [15] use a flowpipe construction scheme to compute overapproximations for reachable set segments. A piecewise polynomial model is used to provide an approximation of the input-output mapping of the controller and an error bound on the approximation. This method is however limited to neural networks with ReLU activation functions. We will discuss technical features of these related works in more detail in Section 2 when we introduce the problem formally.

Neural network controllers in practical applications could involve multiple types of activation functions [4, 27]. The approaches discussed above for specific activation function may not be able to handle such cases, and a more general approach is thus needed.

In this paper, we propose a new reachability analysis approach for verifying NNCS with general neural-network controllers called ReachNN based on Bernstein polynomial. More specifically, given an input space and a degree bound, we construct a polynomial approximation for a general neural-network controller based on Bernstein polynomials. For the critical step of estimating the approximation error bound, inspired by the observation that most neural networks are Lipschitz continuous [36], we present two techniques – a priori theoretical approach based on existing results on Bernstein polynomials and a posteriori approach based on adaptive sampling. By applying these two techniques together, we are able to capture the behavior of a neural-network controller during verification via Bernstein polynomial with tight error bound estimation. Based on the polynomial approximation with the bounded error, we can iteratively compute an overapproximated reachable set of the neural-network controlled system via flowpipes [42]. By the Stone-Weierstrass theorem [12], our Bernstein polynomial based approach can approximate most neural networks with different activation functions (e.g., ReLU, sigmoid, tanh) to arbitrary precision. Furthermore, as we will illustrate later in Section 3, the approximation error bound can be conveniently calculated.

Our paper makes the following contributions.

- We proposed a Bernstein polynomial based approach to generate high-order approximations for the input-output mapping of general neural-network controllers, which is much tighter than the interval based approaches.
- We developed two techniques to analyze the approximation error bound for neural networks with different activation functions and structures based on the Lipschitz continuity of both the network and the approximation. One is based on the theory of Bernstein polynomials and provides a priori insight of the theoretical upper bound of the approximation error, while the other achieves a more accurate estimation in practice via adaptive sampling.
- We demonstrated the effectiveness of our approach on multiple benchmarks, showing its capability in handling dynamical systems with various neural-network controllers, including heterogeneous neural networks with multiple types of activation functions. For homogeneous networks, compared with state-of-the-art approaches Sherlock and Verisig, our ReachNN approach can achieve comparable or even better approximation performance, albeit with longer computation time.

The rest of the paper is structured as follows. Section 2 introduces the system model, the reachability problem, and more details on the most relevant works. Section 3 presents our approach, including the construction of polynomial approximation and the estimation of error bound. Section 4 presents the experimental results. Section 5 provides further discussion of our approach and Section 6 concludes the paper.

2 PROBLEM STATEMENT

In this section, we describe the reachability of NNCS and a solution framework that computes overapproximations for reachable sets. In the paper, a set of ordered variables $x_1, x_2, \ldots, x_m$ is collectively denoted by x. For a vector x, we denote its i-th component by x_i.

A NNCS is illustrated in the Figure 1. The plant is the formal model of a physical system or process, defined by an ODE in the form of $\dot{x} = f(x, u)$ such that x are the n state variables and u are the m control inputs. We require that the function $f : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz continuous in x and continuous in u, in order to guarantee the existence of a unique solution of the ODE from a single initial state (see [31]).

The controller in our system is implemented as a feed-forward neural network, which can be defined as a function κ that maps the values of x to the control inputs u. It consists of S layers, where

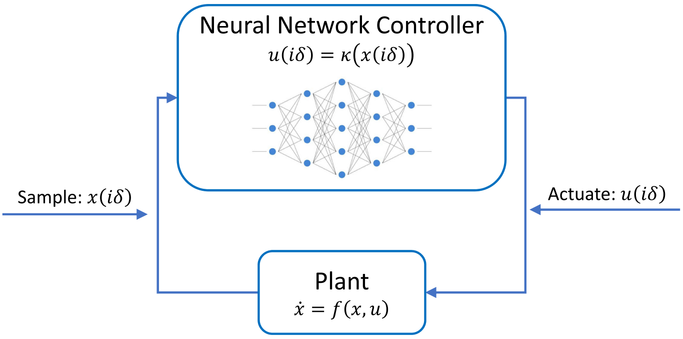


Fig. 1. Neural-network controlled system (NNCS).

the first $S-1$ layers are referred as "hidden layers" and the S-th layer represents the network's output. Specifically, we have

$$\kappa(x) = \kappa_S(\kappa_{S-1}(\ldots\kappa_1(x; W_1, b_1); W_2, b2); W_S, b_S)$$

where W_s and b_s for $s = 1, 2, \ldots, S$ are learnable parameters as linear transformations connecting two consecutive layers, which is then followed by an element-wise nonlinear activation function. $\kappa_i(z_{s-1}; W_{s-1}, b_{s-1})$ is the function mapping from the output of layer $s-1$ to the output layer s such that z_{s-1} is the output of layer $s-1$. An illustration of a neural network is given in Figure 1.

A NNCS works in the following way. Given a control time stepsize $\delta_c > 0$, at the time $t = i\delta_c$ for $i = 0, 1, 2, \ldots$, the neural network takes the current state $x(i\delta_c)$ as input, computes the input values $u(i\delta_c)$ for the next time step and feeds it back to the plant. More precisely, the plant ODE becomes $\dot{x} = f(x, u(i\delta_c))$ in the time period of $[i\delta_c, (i+1)\delta_c]$ for $i = 0, 1, 2, \ldots$. Notice that the controller does not change the state of the system but the dynamics. The formal definition of a NNCS is given as below.

Definition 2.1 (Neural-Network Controlled System). A *neural-network controlled system (NNCS)* can be denoted by a tuple $(\mathcal{X}, \mathcal{U}, F, \kappa, \delta_c, X_0)$, where $\mathcal{X}$ denotes the state space whose dimension is the number of state variables, $\mathcal{U}$ denotes the control input set whose dimension is the number of control inputs, F defines the continuous dynamics $\dot{x} = f(x, u)$, $\kappa : \mathcal{X} \to \mathcal{U}$ defines the input/output mapping of the neural-network controller, δ_c is the control stepsize, and $X_0 \subseteq \mathcal{X}$ denotes the initial state set.

Notice that a NNCS is deterministic when the continuous dynamics function f is Lipschitz continuous. The behavior of a NNCS can be defined by its flowmap. The *flowmap* of a system $(\mathcal{X}, \mathcal{U}, F, \kappa, \delta_c, X_0)$ is a function $\varphi : X_0 \times \mathbb{R}_{\geq 0} \to \mathcal{X}$ that maps an initial state x_0 to the state $\varphi(x_0, t)$, which is the system state at the time t from the initial state x_0. Given an initial state x_0, the flowmap has the following properties for all $i = 0, 1, 2, \ldots$: (a) φ is the solution of the ODE $\dot{x} = f(x, u(i\delta_c))$ with the initial condition $x(0) = \varphi(x_0, i\delta_c)$ in the time interval $t \in [t - i\delta_c, t - i\delta_c + \delta_c]$; (b) $u(i\delta_c) = \kappa(\varphi(x_0, i\delta_c))$.

We call a state x *reachable* at time $t \geq 0$ on a system $(\mathcal{X}, \mathcal{U}, F, \kappa, \delta_c, X_0)$, if and only if there is some $x_0 \in X_0$ such that $x = \varphi(x_0, t)$. Then, the set of all reachable states is called the *reachable set* of the system.

Definition 2.2 (Reachability Problem). The *reachability problem* on a NNCS is to decide whether a given state is reachable or not at time $t \geq 0$.

In the paper, we focus on the problem of computing the reachable set for a NNCS. Since NNCSs are at least as expressive as nonlinear continuous systems, the reachability problem on NNCSs is undecidable. Although there are numerous existing techniques for analyzing the reachability of linear and nonlinear hybrid systems [1, 9, 14, 18, 26], none of them can be directly applied to NNCS, since equivalent transformation from NNCS to a hybrid automaton is usually very costly due to the large number of locations in the resulting automaton. Even an on-the-fly transformation may lead to a large hybrid automaton in general. Hence, we compute flowpipe overapproximations (or flowpipes) for the reachable sets of NNCS.

Similar to the flowpipe construction techniques for the reachability analysis of hybrid systems, we also seek to compute overapproximations for the reachable segments of NNCS. A continuous dynamics can be handled by the existing tools such as SpaceEx [18] when it is linear, and Flow* [10] or CORA [1] when it is nonlinear. The challenge here is to compute an accurate overapproximation for input/output mapping of the neural-network controller in each control step, and we will do it in the following way.

Given a bounded input interval X_I, we compute a model $(g(x), \epsilon)$ where $\epsilon \geq 0$ such that for any $x \in X_I$, the control input $\kappa(x)$ belongs the set $\{g(x) + z \mid z \in B_\epsilon\}$. g is a function of x, and B_ϵ denotes the box $[-\epsilon, \epsilon]$ in each dimension. We provide a summary of the existing works which are close to ours.

Interval overapproximation. The methods described in [36, 39] compute intervals as neural-network input/output relation and directly feed these intervals in the reachability analysis. Although they can be applied to more general neural-network controllers, using interval overapproximation in the reachability analysis cannot capture the dependencies of state variables for each control step, and it is reported in [16].

Exact neural network model. The approach presented in [24] equivalently transforms the neural-network controller to a hybrid system, then the whole NNCS becomes a hybrid system and the existing analysis methods can be applied. The main limitations of the approach are: (a) the transformation could generate a model whose size is prohibitively large, and (b) it only works on neural networks with sigmoid and tanh activation functions.

Polynomial approximation with error bound. In [15], the authors describe a method to produce higher-order sets for neural-network outputs. It is the closest work to ours. In their paper, the approximation model is a piecewise polynomial over the state variables, and the error bound can be well limited when the degrees or pieces of the polynomials are sufficiently high. The main limitation of the method is that it only applies to the neural networks with ReLU activation functions.

3 OUR APPROACH

Our approach exploits high-order set approximation of the output of general types neural network in NNCS reachability analysis via Bernstein polynomials and approximation error bound estimation. The reachable set of NNCS is overapproximated by a finite set of Taylor model flowpipes in our method. The main framework of flowpipe construction is presented in Algorithm 1. Given a NNCS and a bounded time horizon $[0, N\delta_c]$, the algorithm computes Nk flowpipes, each of which is an overapproximation of the reachable set in a small time interval, and the union of the flowpipes is an overapproximation of the reachable set in the time interval of $[0, N\delta_c]$. As stated in Theorem 3.1, this fact is guaranteed by (a) the flowpipes are reachable set overapproximations for the continuous dynamics in each iteration, and (b) the set U_i is an overapproximation of the neural-network controller output. Notice that this framework is also used in [15]. However, we present a technique to compute U_i for more general neural networks.

Let us briefly revisit the technique of computing Taylor model flowpipes for continuous dynamics.

ALGORITHM 1: Flowpipe construction for NNCS

```
Data: NNCS $(\mathcal{X}, \mathcal{U}, F, \kappa, \delta_c, X_0)$, time horizon $N\delta_c$
Result: Flowpipes
Flowpipes ← $\emptyset$;
for $i \leftarrow 0$ to $N - 1$ do
    Compute a set $U_i$ which contains the value of $\kappa(x)$ for all $x \in X_i$;
    Compute the flowpipes $\mathcal{F}_1, \ldots, \mathcal{F}_k$ for the continuous dynamics $\dot{x} = f(x, y)$ with
      $x(0) \in X_i$ and $y \in U_i$;
    Evaluate the flowpipe $X_{i+1}$ based on $\mathcal{F}_k$ for the reachable set at $t = (i + 1)\delta_c$;
    Flowpipes ← Flowpipes $\cup \{\mathcal{F}_1, \ldots, \mathcal{F}_k\}$;
end
return Flowpipes;
```

Taylor model. Taylor models are introduced to provide higher-order enclosures for the solutions of nonlinear ODEs (see [5]), and then extended to overapproximate reachable sets for hybrid systems [9] and solve optimization problems [30]. A *Taylor Model (TM)* is denoted by a pair (p, I) where p is a (vector-valued) polynomial over a set of variables x and I is an (vector-valued) interval. A continuous function $f(x)$ can be overapproximated by a TM $(p(x), I)$ over a domain D in the way that $f(x) \in p(x) + I$ for all $x \in D$. When the approximation p is close to f, I can be made very small.

TM flowpipe construction. The technique of TM flowpipe construction is introduced to compute overapproximations for the reachable set segments of continuous dynamics. Given an ODE $\dot{x} = f(x)$, an initial set X_0 and a time horizon $[0, T]$, the method computes a finite set of TMs $\mathcal{F}_1, \ldots, \mathcal{F}_k$ such that for each $i = 1, \ldots, k$, the flowpipe $\mathcal{F}_i$ contains the reachable set in the time interval of $[t_i, t_{i+1}]$, and $\bigcup_{i=1}^{k} [t_i, t_{i+1}] = [0, T]$, i.e., the union of the flowpipes is an overapproximation of the reachable set in the given time horizon. The standard TM flowpipe construction is described in [5], and it is adapted in [8, 11] for better handling the dynamical systems.

The main contribution of our paper is a novel approach to compute a higher-order overapproximation U_i for the output set of neural networks with a more general form of activation functions, including such as ReLU, sigmoid, and tanh. We show that this approach can generate accurate reachable set overapproximations for NNCSs, in combination with the TM flowpipe construction framework. Our main idea can be described as follows.

Given a neural-network controller with a single output, we assume that its input/output mapping is defined by a function κ and its input interval is defined by a set X_i. In the i-th (control) step, we seek to compute a TM $U_i = P(x) + [-\bar{\varepsilon}, \bar{\varepsilon}]$ such that

$$\kappa(x) \in P(x) + [-\bar{\varepsilon}, \bar{\varepsilon}] \quad \text{for all} \quad x \in X_i. \tag{1}$$

Hence, the TM is an overapproximation of the neural network output. In the paper, we compute P as a Bernstein polynomial with bounded degrees. Since κ is a continuous function that can be approximated by a Bernstein polynomial to arbitrary precision, according to the Stone-Weierstrass theorem [12], we can always ensure that such P exists.

In our reachability computation, the set X_i is given by a TM flowpipe. To obtain a TM for the neural network output set, we use TM arithmetic to evaluate a TM for $P(x) + [-\bar{\varepsilon}, \bar{\varepsilon}]$ with $x \in X_i$. Then, the polynomial part of the resulting TM can be viewed as an approximation of the mapping from the initial set to the neural network output, and the remainder contains the error. Such representation can partially keep the variable dependencies and much better limit the overestimation accumulation than the methods that purely use interval arithmetic.

Theorem 3.1. *The union of the flowpipes computed by Algorithm 1 is an overapproximation of the reachable set of the system in the time horizon of* $[0, N\delta_c]$, *if the flowpipes are overapproximations of the ODE solutions and the TM* U_i *in every step satisfies (1).*

Remark 1. In our framework, Taylor models can also be considered as a candidate of high-order approximation for a neural network's input-output mapping. However, comparing with Bernstein polynomial based approach adopted in this paper, Taylor models suffer from two main limitations: (1) The validity of Taylor models relies on the function differentiability, while ReLU neural networks are not differentiable. Thus Taylor models cannot handle a large number of neural networks; (2) There is no theoretical upper bound estimation for Taylor models, which further limits the rationality of using Taylor models.

Remark 2. To avoid technicalities, we consider the neural-network controller with a single output here. For n-output cases, an intuitive extension is to use Bernstein polynomials to approximate each output respectively.

3.1 Bernstein Polynomials for Approximation

Definition 3.2 (Bernstein Polynomials). Let $d = (d_1, \ldots, d_m) \in \mathbb{N}^m$ and f be a function of $x = (x_1. \ldots, x_m)$ over $I = [0, 1]^m$. The polynomials

$$B_{f,d}(x) \;=\; \sum_{\substack{0 \le k_j \le d_j \\ j \in \{1,\ldots,m\}}} f\left(\frac{k_1}{d_1}, \ldots, \frac{k_m}{d_m}\right) \prod_{j=1}^{m} \left(\binom{d_j}{k_j} x_j^{k_j} (1 - x_j)^{d_j - k_j}\right)$$

are called the *Bernstein polynomials* of f under the degree d.

We then construct $P(x)$ over X by a series of linear transformation based on Bernstein polynomials. Assume that $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$. Let $x' = (x'_1, \ldots, x'_m)$, where

$$x'_j = (x_j - l_j)/(u_j - l_j), \quad j = 1, \ldots, m$$

and

$$\kappa'(x') = \kappa(x) = \kappa\left(\begin{pmatrix} u_1 - l_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & u_m - l_m \end{pmatrix} x' + \begin{pmatrix} l_1 \\ \vdots \\ l_m \end{pmatrix}\right). \tag{2}$$

It is easy to see that κ' is defined over I. For $d = (d_1, \ldots, d_m) \in \mathbb{N}^m$, let $B_{\kappa',d}(x')$ be the Bernstein polynomials of $\kappa'(x')$. We construct the polynomial approximation for κ as:

$$P_{\kappa,d}(x) = B_{\kappa',d}\left(\begin{pmatrix} \frac{1}{u_1 - l_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{u_m - l_m} \end{pmatrix} x - \begin{pmatrix} \frac{l_1}{u_1 - l_1} \\ \vdots \\ \frac{l_m}{u_m - l_m} \end{pmatrix}\right). \tag{3}$$

When we want to compute a Bernstein polynomial over a non-interval domain I, we may consider an interval enclosure of I, since we only need to ensure that the polynomial is valid on the domain and it is sufficient to take its superset. Hence, in Algorithm 1, the Bernstein polynomial(s) in U_i are computed based on an interval enclosure of X_i.

3.2 Approximation Error Estimation

After we obtain the approximation of the neural network controller, a certain question is how to estimate a valid bound for approximation error ε such that Theorem 3.1 holds. Namely, from any given initial state set X, the reachable set of the perturbed system $\dot{x} = f(x, P_{\kappa,d}, \varepsilon)$, $\varepsilon \in [-\bar{\varepsilon}, \bar{\varepsilon}]$ at any time $t \in [0, \delta_c]$ is a superset of the one of the NNCS with ODE $f(x, \kappa)$. A sufficient condition can be derived based on the theory of differential inclusive [37]:

Lemma 3.3. *Given any state set X, let $P_{\kappa,d}$ be the polynomial approximation of κ with respect to the degree d defined as Equation (3). For any time $t \in [0, \delta_c]$, the reachable set of the perturbed system $\dot{x} = f(x, P_{\kappa,d} + \varepsilon)$, $\varepsilon \in [-\bar{\varepsilon}, \bar{\varepsilon}]$ is a superset of the one of the NNCS with ODE $f(x, \kappa)$ from X, if*

$$\kappa(x) \ \in \{\, u \mid u = P_{\kappa,d}(x) + \varepsilon, \ \ \varepsilon \in [-\bar{\varepsilon}, \bar{\varepsilon}]\}, \quad \forall x \in X. \tag{4}$$

Intuitively, the approximation error interval $E = [-\bar{\varepsilon}, \bar{\varepsilon}]$ has a significant impact on the reachable set overapproximation, namely a tighter $\bar{\varepsilon}$ can lead to a more accurate reachable set estimation. In this section, we will introduce two approaches to estimate $\bar{\varepsilon}$, namely theoretical error estimation and sampling-based error estimation. The former gives us a priori insight of how precise the approximation is, while the latter one helps us to obtain a much tighter error estimation in practice.

Compute a Lipschitz constant. We start from computing the Lipschitz constant of a neural network, since Lipschize constant plays a key role in both of our two approaches, which we will see later.

Definition 3.4. A real-valued function $f : X \rightarrow \mathbb{R}$ is called Lipschitz continuous over $X \subseteq \mathbb{R}^m$, if there exists a non-negative real L, such that for any $x, x' \in X$:

$$\|f(x) - f(x')\| \leq L \|x - x'\| .$$

Any such L is called a Lipschitz constant of f over X.

Recent work has shown that a large number of neural networks are Lipschitz continuous, such as the fully-connected neural networks with ReLU, sigmoid, and tanh activation functions and the estimation of Lipschitz constant upper bound for a neural network has been preliminary discussed in [36, 38].

Lemma 3.5 (Lipschitz constant for sigmoid/tanh/ReLU [36, 38]). *Convolutional or fully connected layers with the sigmoid activation function $\mathcal{S}(Wx + b)$, hyperbolic tangent (tanh) activation function $\mathcal{T}(Wx + b)$, and ReLU activation function $\mathcal{R}(Wx + b)$ have $\frac{1}{4}\|W\|$, $\|W\|$, $\|W\|$ as their Lipschitz constants, respectively.*

Based on Lemma 3.5, we further improve the Lipschitz constant upper bound estimation. Specifically, we consider a layer of neural network with n neurons shown in Figure 2, where W and b denote the weight and the bias that are applied on the output of the previous layer. *Input Interval* and *Output Interval* denote the variable space before and after applied by the activation functions of this layer, respectively. Assume $X = [l_1, u_1] \times \cdots \times [l_n, u_n]$ be the Input Interval.

We first discuss layers with sigmoid/tanh activation functions based on the following conclusion:

Lemma 3.6. *[35] Given a function $f : X \rightarrow \mathbb{R}^m$, if $\|\partial f / \partial x\| \leq L$ over X, then f is Lipschitz continuous and L is a Lipschitz constant.*

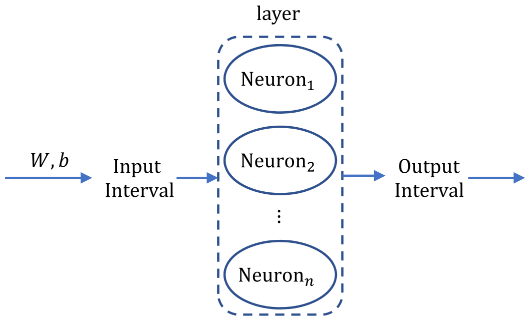


Fig. 2. Schematic diagram of Input Interval and Output Interval of a layer.

Sigmoid. For a layer with sigmoid activation function $\mathcal{S}(y) = 1/(1 + e^{-y})$ with $y = Wx + b$ and $y \in X$, we have

$$\begin{aligned}
&\left\|\frac{\partial \mathcal{S}(x)}{\partial x}\right\| = \left\|\frac{\partial \mathcal{S}(y)}{\partial y}\frac{\partial y}{\partial x}\right\| \le \left\|\frac{\partial \mathcal{S}(y)}{\partial y}\right\|\left\|\frac{\partial y}{\partial x}\right\| \\
&= \|\mathrm{diag}(\mathcal{S}(y_1)(1-\mathcal{S}(y_1)),\ldots,\mathcal{S}(y_n)(1-\mathcal{S}(y_n)))\|\,\|W\| \\
&\le \max_{1\le i\le n}\ \sup_{a_i\le \mathcal{S}(y_i)\le b_i} \{\mathcal{S}(y_i)(1-\mathcal{S}(y_i))\}\,\|W\| \\
&= \max_{1\le i\le n}\left(\frac{1}{4} - \left(\frac{\mathrm{sgn}(\mathcal{S}(a_i)-0.5)+\mathrm{sgn}(\mathcal{S}(b_i)-0.5)}{2}\right)^2\right. \\
&\quad \left.\cdot\ \min\left\{\left(\frac{1}{2}-\mathcal{S}(a_i)\right)^2,\left(\frac{1}{2}-\mathcal{S}(b_i)\right)^2\right\}\right)\|W\|
\end{aligned} \tag{5}$$

Hyperbolic tangent. For a layer with hyperbolic tangent activation function $\mathcal{T}(y) = 2/(1 + e^{-2y}) - 1$ with $y = Wx + b$ and $y \in X$, we have

$$\begin{aligned}
&\left\|\frac{\partial \mathcal{T}(x)}{\partial x}\right\| = \left\|\frac{\partial \mathcal{T}(y)}{\partial y}\frac{\partial y}{\partial x}\right\| \le \left\|\frac{\partial \mathcal{T}(y)}{\partial y}\right\|\left\|\frac{\partial y}{\partial x}\right\| \\
&= \left\|\mathrm{diag}(1-(\mathcal{T}(y_1))^2,\ldots,1-(\mathcal{T}(y_n))^2)\right\|\,\|W\| \\
&= \max_{1\le i\le n}\ \sup_{a_i\le \mathcal{T}(y_i)\le b_i} \{1-(\mathcal{T}(y_i))^2\}\,\|W\| \\
&= \max_{1\le i\le n}\left(1 - \left(\frac{\mathrm{sgn}(\mathcal{T}(a_i))+\mathrm{sgn}(\mathcal{T}(b_i))}{2}\right)^2\right. \\
&\quad \left.\cdot\ \min\{(\mathcal{T}(a_i))^2,(\mathcal{T}(b_i))^2\}\right)\|W\|
\end{aligned} \tag{6}$$

For ReLU networks, we try to derive a Lipschitz constant directly based on its definition:
ReLU. For a layer with ReLU activation function $\mathcal{R}(y) = \max\{0, y\}$ with $y = Wx + b$ and $y \in X$, we have

ALGORITHM 2: Compute a Lipschitz constant for κ

```
   Data: Neural network $\kappa$, input space $X$
   Result: Lipschitz constant $L$
 1 $S \leftarrow$ the number of layers of $\kappa$;
 2 $L \leftarrow 1$;
 3 OutputInterval$_0 \leftarrow X$;
 4 for $s \leftarrow 1$ to $S$ do
 5     $act \leftarrow$ the activation function of the $s$-th layer;
 6     InputInterval$_s \leftarrow$ ComputeInputInterval(OutputInterval$_{s-1}$);
 7     $L_s \leftarrow$ ComputeLipchitzConstant($act$, $W$, InputInterval$_s$);
 8     $L \leftarrow L \cdot L_s$;
 9     OutputInterval$_s \leftarrow$ ComputeOutputInterval(InputInterval$_s$);
10 end
11 return $L$;
```

$$\begin{aligned}
&\sup_{x_1\neq x_2}\frac{\|\mathcal{R}(x_1)-\mathcal{R}(x_2)\|}{\|x_1-x_2\|}\\
=&\sup_{x_1\neq x_2}\frac{\left\|\begin{pmatrix}\max\{0,W_1x_1+u_1\}-\max\{0,W_1x_2+u_1\}\\ \cdots\\ \max\{0,W_nx_1+u_n\}-\max\{0,W_nx_2+u_n\}\end{pmatrix}\right\|}{\|x_1-x_2\|}\\
\leq&\sup_{x_1\neq x_2}\frac{\left\|\begin{pmatrix}\left(\frac{1+\mathrm{sgn}(u_1)}{2}\right)^2 W_1|x_1-x_2|\\ \cdots\\ \left(\frac{1+\mathrm{sgn}(u_n)}{2}\right)^2 W_n|x_1-x_2|\end{pmatrix}\right\|}{\|x_1-x_2\|}\\
\leq&\left\|\left(\left(\frac{1+\mathrm{sgn}(u_1)}{2}\right)^2 W_1,\dots,\left(\frac{1+\mathrm{sgn}(u_n)}{2}\right)^2 W_n\right)\right\|.
\end{aligned}\tag{7}$$

In Algorithm 2, we first do the initialization (line 1–3) by letting the variable denoting Lipschitz constant $L = 1$. Note that OutputInterval$_i$ ($i = 1, \dots, S$) denotes the input interval and output interval of layer i, as shown in Figure 2. For convenience, we let OutputInterval$_0$ be the input space X. Then we do the layer-by-layer interval analysis and compute the corresponding Lipschitz constant (line 4–10). For each layer s, the function **ComputeInputInterval** is first invoked to compute the InputInterval$_s$ (line 6). Then the Lipschitz constant L_s of layer s is evaluated by the function **ComputeLipchitzConstant**, namely Equations (5), (6), (7) in terms of the activation function act of this layer (line 7). L is updated to the current layer by multiplying L_s (line 8). Finally, OutputInterval$_s$ is computed by the function **ComputeOutputInterval** (line 9). Note that the implementation of **ComputeInputInterval** and **ComputeOutputInterval** is a typical interval analysis problem of neural networks, which have been adequately studied in [16, 36, 39]. We do not go into details here due to the space limit.

Naive theoretical error (T-error) estimation. After obtaining a Lipschize constant of κ, we can directly leverage the existing result on Bernstein polynomials for Lipschitz continuous functions to derive the error of our polynomial approximation.

Lemma 3.7. *[28] Assume f is a Lipschitz continuous function of $x = (x_1. \ldots, x_m)$ over $I = [0,1]^m$ with a Lipschitz constant L. Let $d = (d_1, \ldots, d_m) \in \mathbb{N}^m$ and $B_{f,d}$ be the Bernstein polynomials of f under the degree d. Then we have*

$$\left\| B_{f,d}(x) - f(x) \right\| \leq \frac{L}{2} \left(\sum_{j=1}^{m} (1/d_j) \right)^{\frac{1}{2}}, \quad \forall x \in I. \tag{8}$$

Theorem 3.8 (T-Error Estimation). *Assume κ is a Lipschitz continuous function of $x = (x_1, \ldots, x_m)$ over $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$ with a Lipschitz constant L. Let $P_{\kappa,d}$ be the polynomial approximation of κ that is defined as Equation (3) with the degree $d = (d_1, \ldots, d_m) \in \mathbb{N}^m$. Let*

$$\bar{\varepsilon}_t = \frac{L}{2} \left(\sum_{j=1}^{m} \frac{1}{d_j} \right)^{\frac{1}{2}} \max_{j \in \{1,\ldots,m\}} \{u_j - l_j\}. \tag{9}$$

then $\bar{\varepsilon}_t$ satisfies (4), namely,

$$\kappa(x) \in \{ u \mid u = P_{\kappa,d}(x) + \varepsilon, \ \ \varepsilon \in [-\bar{\varepsilon}_t, \bar{\varepsilon}_t] \}, \quad \forall x \in X.$$

Proof. First, by Equation (2) we know that

$$\| \partial x / \partial x' \| = \max_{j \in \{1,\ldots,m\}} \{u_j - l_j\}$$

is a Lipschitz constant $L_{x(x')}$ of the function $x(x')$. Note that $\kappa' = \kappa \circ x$, then we can obtain the Lipschitz constant of $\kappa'(x')$ by

$$L_{\kappa'(x')} = L_{\kappa(x)} \cdot L_{x(x')} = L \max_{j \in \{1,\ldots,m\}} \{u_j - l_j\}.$$

By Equation (8), $\forall x \in X$, we have:

$$\| P_{\kappa,d}(x) - \kappa(x) \| = \| B_{\kappa',d}(x') - \kappa'(x') \| \leq \frac{L_{\kappa'(x')}}{2} \left(\sum_{j=1}^{m} \frac{1}{d_j} \right)^{\frac{1}{2}}.$$

□

Adaptive sampling-based error (S-error) estimation. While Theorem 3.8 can help derive an approximation error bound easily, such bounds are often over-conservative in practice. Thus we propose an alternative sampling-based approach to estimate the approximation error. For a given box $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$, we perform a grid-based partition based on an integer vector $p = (p_1, \ldots, p_m)$. That is, we partition X into a set of boxes $X = \bigcup_{0 \leq k \leq p-1} B_k$, where $0 \leq k \leq p - 1$ is the abbreviation for $k = (k_1, \ldots, k_m), 0 \leq k_j \leq p_j - 1, \ 1 \leq j \leq m$, and for any k,

$$B_k = \left[l_1 + \frac{k_1}{p_1}(u_1 - l_1), l_1 + \frac{k_1 + 1}{p_1}(u_1 - l_1) \right] \times \cdots \times \left[l_m + \frac{k_m}{p_m}(u_m - l_m), l_j + \frac{k_m + 1}{p_m}(u_m - l_m) \right].$$

It is easy to see that the largest error bound of all the boxes is a valid error bound over X.

Lemma 3.9. *Assume κ is a continuous function of $x = (x_1, \ldots, x_m)$ over $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$. Let $P_{\kappa,d}$ be the polynomial approximation of κ that is defined as Equation (3) with the degree $d = (d_1, \ldots, d_m) \in \mathbb{N}^m$. Let $\{B_k\}$ be the box partition of X in terms of p, and $\bar{\varepsilon}_k$ be the approximation error bound of $P_{\kappa,d}$ over the box B_k. We then have $\forall x \in X$,*

$$\kappa(x) \in \left\{ u \mid u = P_{\kappa,d}(x) + \varepsilon, \ \varepsilon \in \left[- \max_{0 \leq k \leq p-1} \bar{\varepsilon}_k, \max_{0 \leq k \leq p-1} \bar{\varepsilon}_k \right] \right\}.$$

Leveraging the Lipschitz continuity of κ, we can estimate the local error bound $\bar{\varepsilon}_k$ for box B_k by sampling the value of κ and $P_{\kappa,d}$ at the box center.

Lemma 3.10. *Assume κ is a Lipschitz continuous function of $x = (x_1, \ldots, x_m)$ over box B_k with a Lipschitz constant L. Let $P_{\kappa,d}$ be the polynomial approximation of κ that is defined as Equation (3) with the degree $d = (d_1, \ldots, d_m) \in \mathbb{N}^m$. Let*

$$c_k = \left(l_1 + \frac{2k_1+1}{2p_1}(u_1 - l_1), \ldots, l_m + \frac{2k_m+1}{2p_m}(u_m - l_m)\right) \tag{10}$$

be the center of B_k. For $x \in B_k$, we have

$$\|P_{\kappa,d}(x) - \kappa(x)\| \le L\sqrt{\sum_{j=1}^{m}\left(\frac{u_j - l_j}{p_j}\right)^2} + \|P_{\kappa,d}(c_k) - \kappa(c_k)\|. \tag{11}$$

Proof. By [6], we know the Bernstein-based approximation $P_{\kappa,d}$ is also Lipschitz continuous with the Lipschitz constant L. Then for $x \in B_k$, we have

$$\begin{aligned}\|P_{\kappa,d}(x) - \kappa(x)\| &\le \|P_{\kappa,d}(x) - P_{\kappa,d}(c_k)\| + \|P_{\kappa,d}(c_k) - \kappa(c_k)\| + \|\kappa(c_k) - \kappa(x)\| \\ &\le L\max_{x\in B_k}\|x - c\| + \|P_{\kappa,d}(c_k) - \kappa(c_k)\| + L\max_{x\in B_k}\|x - c_k\| \\ &= L\sqrt{\sum_{j=1}^{m}\left(\frac{u_j - l_j}{p_j}\right)^2} + \|P_{\kappa,d}(c_k) - \kappa(c_k)\|.\end{aligned}$$

□

Combining Lemma 3.9 and Lemma 3.10, we can derive the sampling-based error bound over X.

Theorem 3.11 (S-Error Estimation). *Assume κ is a Lipschitz continuous function of $x = (x_1, \ldots, x_m)$ over $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$ with a Lipschitz constant L. Let $P_{\kappa,d}$ be the polynomial approximation of κ that is defined as Equation (3) with the degree $d = (d_1, \ldots, d_m) \in \mathbb{N}^m$, and $X_c = \{c_k\}_{0\le k\le p-1}$ be the sampling set with any given positive integer vector $p = (p_1, \ldots, p_m)$. Let*

$$\bar{\varepsilon}_s(p) = L\sqrt{\sum_{j=1}^{m}\left(\frac{u_j - l_j}{p_j}\right)^2} + \max_{0\le k\le p-1}\|P_{\kappa,d}(c_k) - \kappa(c_k)\|. \tag{12}$$

Then, $\bar{\varepsilon}_s(p)$ satisfies (4), namely,

$$\kappa(x) \in \{\, u \mid u = P_{\kappa,d}(x) + \varepsilon,\ \ \varepsilon \in [-\bar{\varepsilon}_s(p), \bar{\varepsilon}_s(p)]\},\quad \forall x \in X.$$

Theorem 3.12 (Convergence of $\bar{\varepsilon}_s$). *Assume κ is a Lipschitz continuous function of $x = (x_1, \ldots, x_m)$ over $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$ with a Lipschitz constant L. Let $P_{\kappa,d}$ be the polynomial approximation of κ that is defined as Equation (3) with the degree $d = (d_1, \ldots, d_m) \in \mathbb{N}^m$. Let $\bar{\varepsilon}_{best} = \max_{x\in X}\|P_{\kappa,d}(x) - \kappa(x)\|$ be the exact error, we have*

$$\lim_{p\to\infty}\bar{\varepsilon}_s(p) \to \bar{\varepsilon}_{best}.$$

Proof. Let $\delta(p) = L\sqrt{\sum_{j=1}^{m}(\frac{u_j - l_j}{p_j})^2}$, we have

$$|\bar{\varepsilon}_s(p) - \bar{\varepsilon}_{best}| \le \left|\bar{\varepsilon}_s(p) - \max_{0\le k\le p-1}\|P_{\kappa,d}(c_k) - \kappa(c_k)\|\right| = \delta(p)$$

Since $\delta(p) \to 0$, when $p \to \infty$, the theorem holds. □

Note that $\delta(p)$ actually specifies the difference between the exact error and S-error. Thus we call it sampling error precision.

[Adaptive sampling] By Theorem 3.12, we can always make $\delta(p)$ arbitrarily small by increasing p. Then the error bound is mainly determined by the sample difference over X_c. Thus, if our polynomial approximation $P_{\kappa,d}$ regresses κ well, we can expect to obtain a tight error bound estimation. To bound the impact of $\delta(p)$, we set a hyper parameter $\bar{\delta}$ as its upper bound. Specifically, given a box $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$, we can adaptively change p to sample fewer times while ensuring $\delta(p) \leq \bar{\delta}$.

Proposition 1. *Given a box $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$ and a positive number $\bar{\delta}$, let*

$$p_j = \left\lceil L(u_j - l_j)\sqrt{m}/\bar{\delta} \right\rceil, \quad j = 1, \ldots, m.$$

Then we have $\delta(p) \leq \bar{\delta}$.

In practice, we can directly use $\bar{\varepsilon}_s$ as $\bar{\varepsilon}$ by specifying a small $\bar{\delta}$, since S-error is more precise. It is worthy noting that a small $\bar{\delta}$ may lead to a large number of sampling points and thus can be time consuming. In our implementation, we mitigate this runtime overhead by computing the sampling errors at c_k for each B_k in parallel.

4 EXPERIMENTS

We implemented a prototype tool and used it in cooperation with Flow*. In our experiments, we first compare our approach ReachNN with the interval overapproximation approach on an example with a heterogeneous neural-network controller that has ReLU and tanh as activation functions. Then we compare our approach with Sherlock [15] and Verisig [24] on multiple benchmarks collected from the related works. All our experiments were run on a desktop, with 12-core 3.60 GHz Intel Core i7 [1].

4.1 Illustrating Example

Consider the following nonlinear control system [19]:

$$\dot{x}_1 = x_2, \quad \dot{x}_2 = ux_2^2 - x_1,$$

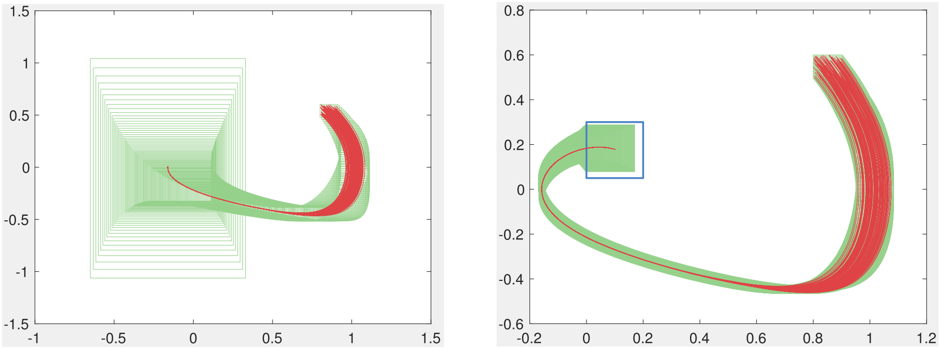
where u is computed from a heterogeneous neural-network controller κ that has two hidden layers, twenty neurons in each layer, and ReLU and tanh as activation functions. Given a control stepsize $\delta_c = 0.2$, we hope to verify whether the system will reach $[0, 0.2] \times [0.05, 0.3]$ from the initial set $[0.8, 0.9] \times [0.5, 0.6]$. Since the state-of-art verification tools focus on NNCSs with single-type activation function and interval overapproximation is the only presented work that can be extended to heterogeneous neural networks, we compare our method with interval overapproximation in the illustrating example.

Figure 3(a) shows the result of the reachability analysis with interval overapproximation, while ReachNN's result is shown in Figure 3(b). Red curves denote the simulation results of the system evolution from 100 different initial states, which are randomly sampled from the initial set. Green rectangles are the constructed flowpipes as the overapproximation of the reachable state set. The blue rectangle is the goal area. We conduct the reachability analysis from the initial set by combining our neural network approximation approach with Flow*. As shown in Figure 3(a), interval overapproximation based approach yields over-loose reachable set approximation, which makes the flowpipes grow uncontrollably and quickly exceeds the tolerance (25 steps). On the contrary, ReachNN can provide a much tighter estimation for all control steps and thus successfully prove the reachability property.

4.2 Benchmarks

Table 1 shows the benchmarks settings, while Table 2 shows the experiment results of ReachNN, Sherlock [15] and Verisig [24]. We can first find that our approach can verify most examples, with

[1] The experiments are not memory bounded.



(a) Interval (b) ReachNN

Fig. 3. Flowpipes computed by interval overapproximation approach and ReachNN on a NNCS with a heterogeneous neural-network controller that has ReLU and tanh activation functions.

Table 1. Benchmark Setting: For Each Example #, *ODE* Denotes the Ordinary Differential Equation of the Example, V Denotes the Dimension of the State Variable, δ_c is the Discrete Control Stepsize, N Denotes the Number of Control Step, *Init* Denotes the Initial State Set, *Goal* Denotes the Goal Set that we Hope to Verify the System will Enter After N Steps

#	ODE	V	δ_c	Init	Goal
1	$\dot{x}_1 = x_2,$ $\dot{x}_2 = ux_2^2 - x_1.$	2	0.2	$x_1 \in [0.8, 0.9]$, $x_2 \in [0.5, 0.6]$	$x_1 \in [0, 0.2]$, $x_2 \in [0.05, 0.3]$
2	$\dot{x}_1 = x_2 - x_1^3,$ $\dot{x}_2 = u.$	2	0.2	$x_1 \in [0.7, 0.9]$, $x_2 \in [0.7, 0.9]$	$x_1 \in [-0.3, 0.1]$, $x_2 \in [-0.35, 0.5]$
3	$\dot{x}_1 = -x_1(0.1 + (x_1 + x_2)^2),$ $\dot{x}_2 = (u + x_1)(0.1 + (x_1 + x_2)^2).$	2	0.1	$x_1 \in [0.8, 0.9]$, $x_2 \in [0.4, 0.5]$	$x_1 \in [0.2, 0.3]$, $x_2 \in [-0.3, -0.05]$
4	$\dot{x}_1 = -x_1 + x_2 - x_3,$ $\dot{x}_2 = -x_1(x_3 + 1) - x_2,$ $\dot{x}_3 = -x_1 + u$	3	0.1	$x_1, x_3 \in [0.25, 0.27]$, $x_2 \in [0.08, 0.1]$	$x_1 \in [-0.05, 0.05]$, $x_2 \in [-0.05, 0]$
5	$\dot{x}_1 = x_1^3 - x_2,$ $\dot{x}_2 = x_3, \dot{x}_3 = u$	3	0.2	$x_1 \in [0.38, 0.4]$, $x_2 \in [0.45, 0.47]$, $x_3 \in [0.25, 0.27]$	$x_1 \in [-0.4, -0.28]$, $x_2 \in [0.05, 0.22]$
6	$\dot{x}_1 = x_2, \dot{x}_2 = -x_1 + 0.1\sin(x_3),$ $\dot{x}_3 = x_4, \dot{x}_4 = u$	4	0.5	$x_1 \in [-0.77, -0.75]$, $x_2 \in [-0.45, -0.43]$, $x_3 \in [0.51, 0.54]$, $x_4 \in [-0.3, -0.28]$	$x_1 \in [-0.1, 0.2]$, $x_2 \in [-0.9, -0.6]$

a few exceptions. The simulation trajectories along with overapproximate reachable set computed by ReachNN, Sherlock and Verisig of some selected examples are shown in Figure 4. The reason why we fail in these examples are mainly due to the relatively large approximation error. As shown later in Section 5, an interesting phenomenon is that Bernstein polynomial based approach may perform differently with respect to the type of activation functions, which we will consider in our future work.

Benefiting from its generality, our approach can handle all these examples, while Sherlock and Verisig are only applicable to a few of them. Furthermore, the heterogeneous neural networks that contain multiple types of activation functions, which is common in practice [4, 27], can only be handled by our approach. However we acknowledge that our method costs much more time than Sherlock and Verisig (see Table 2). The main reason is the large number of samples needed in estimating the error for a Bernstein polynomial. Since we only require a neural network to be Lipschitz continuous, the estimation of approximation error is quite conservative. However, such limitation may be overcome by considering more information of the activation functions. We plan to explore it in our future work.

Table 2. Neural Network Controller Setting and Experimental Results: In Each Example #, for a Neural-Network Controller, *Act* Denotes the Applied Activation Functions, l and n Denote the Number of Layers and the Number of Neurons in Each Hidden Layer, Respectively

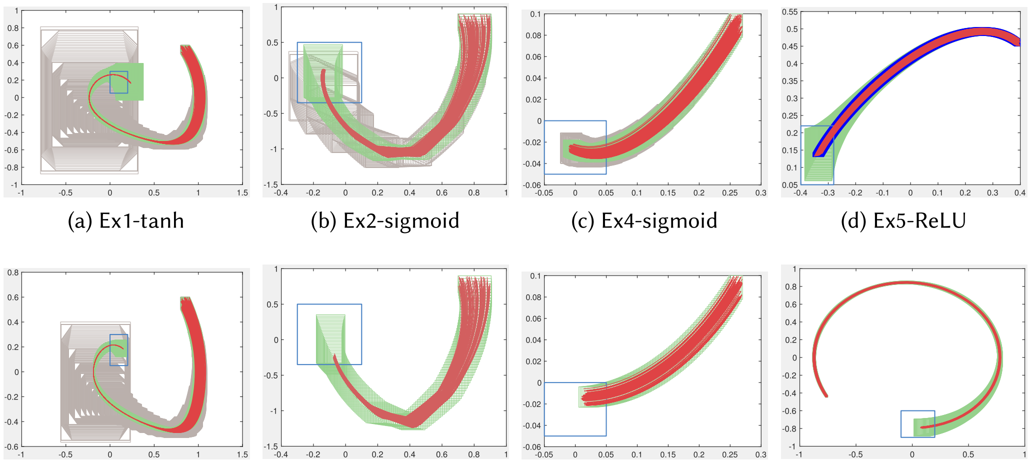
#	NN Controller			ReachNN					Sherlock[15]			Verisig	
	Act	l	n	d	$\bar{\delta}$	$\bar{\varepsilon}$	ifReach	time	d	ifReach	time	ifReach	time
1	ReLU	3	20	[1, 1]	0.001	0.0009995	Yes(35)	3184	2	Yes(35)	41	–	–
	sigmoid	3	20	[3, 3]	0.001	0.0077157	Yes(35)	779	–	–	–	Unknown(22)	–
	tanh	3	20	[3, 3]	0.005	0.0117355	Unknown(35)	–	–	–	–	Unknown(22)	–
	ReLU+tanh	3	20	[3, 3]	0.01	0.0150897	Yes(35)	589	–	–	–	–	–
2	ReLU	3	20	[1, 1]	0.01	0.0090560	Yes(9)	128	2	Yes(9)	3	–	–
	sigmoid	3	20	[3, 3]	0.01	0.0200472	Yes(9)	280	–	–	–	Unknown(7)	–
	tanh	3	20	[3, 3]	0.01	0.0194142	Unknown(7)	–	–	–	–	Unknown(7)	–
	ReLU+tanh	3	20	[3, 3]	0.001	0.0214964	Yes(9)	543	–	–	–	–	–
3	ReLU	3	20	[1, 1]	0.01	0.0205432	Yes(60)	982	2	Yes(60)	139	–	–
	sigmoid	3	20	[3, 3]	0.005	0.0060632	Yes(60)	1467	–	–	–	Yes(60)	27
	tanh	3	20	[3, 3]	0.01	0.0072984	Yes(60)	1481	–	–	–	Yes(60)	26
	ReLU+tanh	3	20	[3, 3]	0.01	0.0230050	Unknown(60)	–	–	–	–	–	–
4	ReLU	3	20	[1, 1, 1]	0.005	0.0048965	Yes(5)	396	2	Yes(5)	19	–	–
	sigmoid	3	20	[2, 2, 2]	0.01	0.0096400	Yes(10)	253	–	–	–	Yes(10)	7
	tanh	3	20	[2, 2, 2]	0.01	0.0095897	Yes(10)	244	–	–	–	Yes(10)	7
	ReLU+sigmoid	3	20	[2, 2, 2]	0.01	0.0096322	Yes(5)	108	–	–	–	–	–
5	ReLU	4	100	[1, 1, 1]	0.004	0.0039809	Yes(10)	5487	2	Yes(10)	12	–	–
	sigmoid	4	100	[2, 2, 2]	0.004	0.0039269	No(10)	8842	–	–	–	Unknown(10)	–
	tanh	4	100	[2, 2, 2]	0.004	0.0038905	Unknown(10)	7051	–	–	–	Unknown(10)	–
	ReLU+tanh	4	100	[2, 2, 2]	0.04	0.0039028	Unknown(10)	7369	–	–	–	–	–
6	ReLU	4	20	[1, 1, 1, 1]	0.001	0.0096789	Yes(10)	7842	2	Yes(10)	33	–	–
	sigmoid	4	20	[1, 1, 1, 1]	0.001	0.0082784	No(7)	32499	–	–	–	Yes(10)	34
	tanh	4	20	[1, 1, 1, 1]	0.001	0.0156596	No(7)	3683	–	–	–	Yes(10)	35
	ReLU+tanh	4	20	[1, 1, 1, 1]	0.001	0.0091648	Yes(10)	10032	–	–	–	–	–

While applying our approach *ReachNN*, d denotes the degree of the polynomial approximation, $\bar{\delta}$ denotes the sampling error precision, $\bar{\varepsilon}$ represents the error bound. For Sherlock, d represents the degree of the Taylor model based approximation. Under general settings of Flow* (the order of taylor models are chosen between 5–12 in terms of the degree d in ReachNN, the stepsizes for flowpipe construction are chosen between 1/10-1/100), the verification results for each approach are evaluated by two metrics: Boolean *ifReach* following by a number indicates whether this approach verifies the reachability or not: $Yes(n)$ means the system is proven to reach the goal set after n steps, $No(n)$means that at nth step the system has not reached the goal set, and *Unknown*(n) means the reachable set at any step $k \leq n$ is not a subset of the goal set and flowpipes start to blow up after n steps. The computation duration *time* (in seconds) indicates the efficiency. We use "–" to represent that a certain approach is not applicable.

5 DISCUSSION AND OPEN CHALLENGES

In this section, we will show further insights into our approach and discuss some remaining challenges.

ReLU v.s. tanh v.s. sigmoid: Approximation performance of Bernstein polynomials. We empirically explore the approximation performance of Bernstein polynomials for different neural networks by sampling. We take Example 2 over $X = [0.7, 0.9] \times [0.7, 0.9]$ in the benchmark for instance. For each neural network controller, we sample a large number of points and plot the function value of the controller and its approximation over X (Figure 5). The orders of



(a) Ex1-tanh (b) Ex2-sigmoid (c) Ex4-sigmoid (d) Ex5-ReLU

(e) Ex1-sigmoid (f) Ex2-ReLU-tanh (g) Ex4-ReLU-tanh (h) Ex6-ReLU-tanh

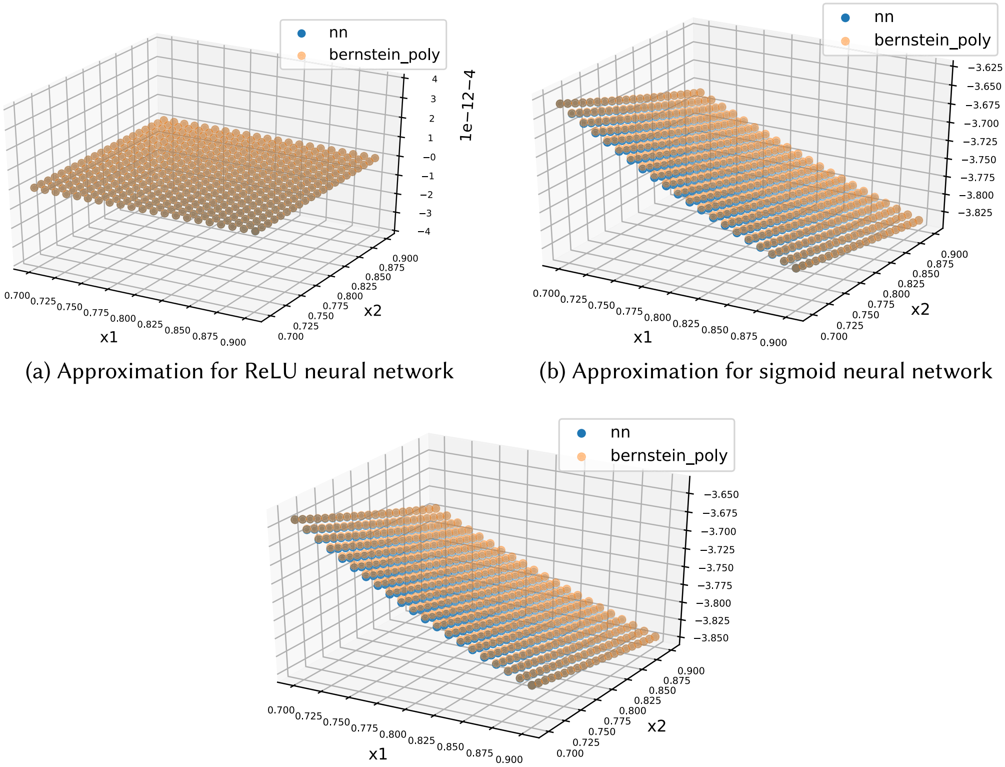
Fig. 4. Flowpipes for the selected examples: Red curves denote the trajectories of x_1 and x_2 of the system simulated from sampled states within the initial set. Green rectangles are the constructed flowpipes as the overapproximation of the reachable state set by our approach, gray rectangles are the flowpipes computed based on Verisig, and Sherlock computes the flowpipes represented as deep blue rectangles. The blue rectangle is the goal area. Neither Verisig nor Sherlock can analyze the networks in (f), (g) or (h) due to the presence of both ReLU and tanh activation functions.

magnitude of the error for ReLU network, sigmoid network and tanh network are 10^{-12}, 10^{-4}, 10^{-4}, respectively. First, the approximation errors for all these three networks are fairly small for the 0.2×0.2 box, which indicates that Bernstein polynomial based approach is promising if more efficient error estimation approaches could be designed. Secondly, we can see that Bernstein polynomials can achieve a higher approximation precision for the ReLU network than the other two. This motivates us to further explore the impact of the inner structure of different neural networks on the approximation performance in future work.

Small Lipschitz constant v.s. Large Lipschitz constant. Given that the approximation error of Bernstein polynomials is upper bounded linearly by the Lipschitz constant of neural network, we conducted preliminary study on the impact of the Lipschitz constant on ReachNN. We consider the dynamical system of an inverted pendulum on a cart:

$$
\begin{aligned}
\dot{x}_1 &= x_2,\\
\dot{x}_2 &= \frac{-mg\sin(x_3)\cos(x_3) + mlx_4^2\sin(x_3) + fmx_4\cos(x_3) + u}{M + (1-\cos(x_3)^2)m},\\
\dot{x}_3 &= x_4,\\
\dot{x}_4 &= \frac{(M+m)*(g\sin(x_3) - fx_4) - (lmx_4^2\sin(x_3) + u)}{l(M + 1 - \cos{x_3}^2)m},
\end{aligned}
$$

where the angular position and velocity of the pendulum are x_1 and x_2, the position and velocity of the cart are x_3 and x_4, the pendulum mass is $m = 0.23$, the cart mass is $M = 2.4$, gravitational acceleration is $g = 9.8$, the length of pendulum is $l = 0.36$, and the coefficient of friction is $f = 0.1$. The goal is to stabilize the pendulum at the upward position and verify whether the cart position remains in $[2, 4]$ after 25 control steps. The system will start randomly from $x_1 \in [0.5, 0.55]$, $x_2 \in [-1, -0.95]$, $x_3 \in [2.5, 2.55]$, $x_4 \in [0, 0.05]$.



(a) Approximation for ReLU neural network

(b) Approximation for sigmoid neural network

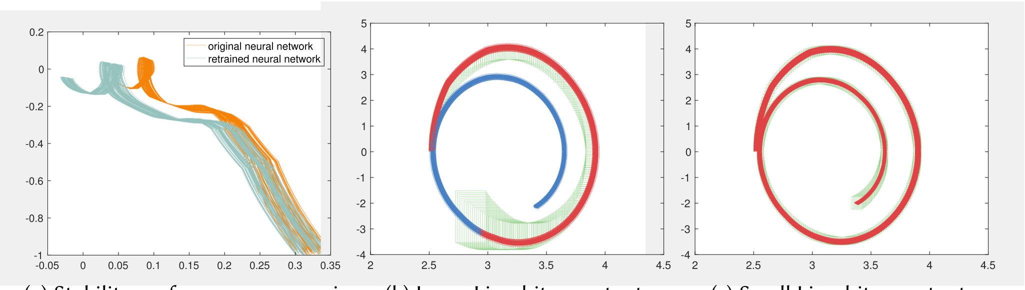
(c) Approximation for tanh neural network

Fig. 5. Bernstein polynomial based approximation for different neural networks. In Figure 5(a), 5(b), 5(c), $x1$-axis and $x2$-axis are x_1 and x_2 respectively, while z axis is the value of neural network/polynomial approximation. The blue point and yellow point are the sample values of the neural network and the approximation polynomial respectively.

Given a trained five-layer ReLU neural network κ_1 with the width of each layer as [100, 1, 2, 1, 2], the computed Lipschitz constant is 874.5 [36]. Although this Lipschitz constant is an upper bound of the best Lipschitz constant, by using the same Lipschitz constant computation method, we can use it as a proxy to estimate the differences between the best Lipschitz constants of different neural networks. Figure 6(b) illustrates that the Lipschitz constant of a network has a significant impact on our Bernstein polynomial-based reachablility analysis. For κ_1 as shown in Figure 6(b), the error bound estimation grows rapidly and end up becoming too large at the 10th step. The blue curves are continuations of the simulation results from 10 to 20 steps after reachability analysis becomes unreliable.

Neural Network Retraining. Thus, to handle neural-network controllers with large Lipschitz constants, we retrained the network by sampling input-output data from the original network and added a penalty term for Lipschitz constant in the loss function. To reduce the Lipschitz constant, L_θ, of the retrained network $\kappa'(x;\theta)$, we consider the following empirical risk minimization problem:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(\kappa'(x;\theta), \kappa(x)) + \lambda L_\theta, \tag{13}$$



(a) Stability performance comparison (b) Large Lipschitz constant (c) Small Lipschitz constant

Fig. 6. ReachNN comparison between large Lipschitz constant neural-network controller and the retrained neural-network controller using Bernstein polynomials of the same degree. In Figure 6(a), x-axis and y-axis are the pendulum angular position and angular velocity respectively. In Figure 6(b) and 6(c), x-axis and y-axis are the cart position and velocity respectively.

where $\lambda \in \mathbb{R}_+$ is a regularization factor and $L_\theta = \prod_{l=1}^{L} \|W_\theta^l\|_2$ as the Lipschitz constant [36]. We refer to the second term as the *Lipschitz constant regularizer*. The gradient of the *Lipschitz constant regularizer* is only related to the largest singular value and corresponding vectors of W_θ^l and projected by weights of other layers. This means each retrained weight matrix, W_θ^l, does not shrink significantly in the directions orthogonal to the first right singular vector and preserves potential important information in the original network. We use classical gradient descent method to do the optimization. A similar approach is also mentioned in [41].

For the inverted pendulum on a cart example, we retrained a three-layer ReLU neural network κ_2 with 50 neurons each layer, which has the Lipschitz constant upper bound of 14.7. In Figure 6(a), we show state evolution of angular position and angular velocity controlled by the original neural network κ_1 and by the retrained neural network κ_2, respectively. In Figure 6(b) and Figure 6(c), we show that the retrained neural network can produce results comparable to the original neural network. For the neural network with a small Lipschitz constant, we postulate that Bernstein polynomials can track the behaviors of the neural network better. In addition, ReachNN provides tighter error bound estimation. As a result, the overapproximation quality is significantly improved. This idea is aligned with *model compression* [7] (or *distillation* [21]), which uses high-performance neural networks to guide the training of shallower [3] or more structured neural networks. One key observation in [3] is that deep and complex neural networks perform better not because of better representation power, but because they are better regularized and therefore easier to train. Here, we consider the retraining process as a form of regularization that effectively maintains the performance of the original network but obtains a smaller Lipschitz constant. We plan to explore the tighter connection between training and verification more thoroughly in future work.

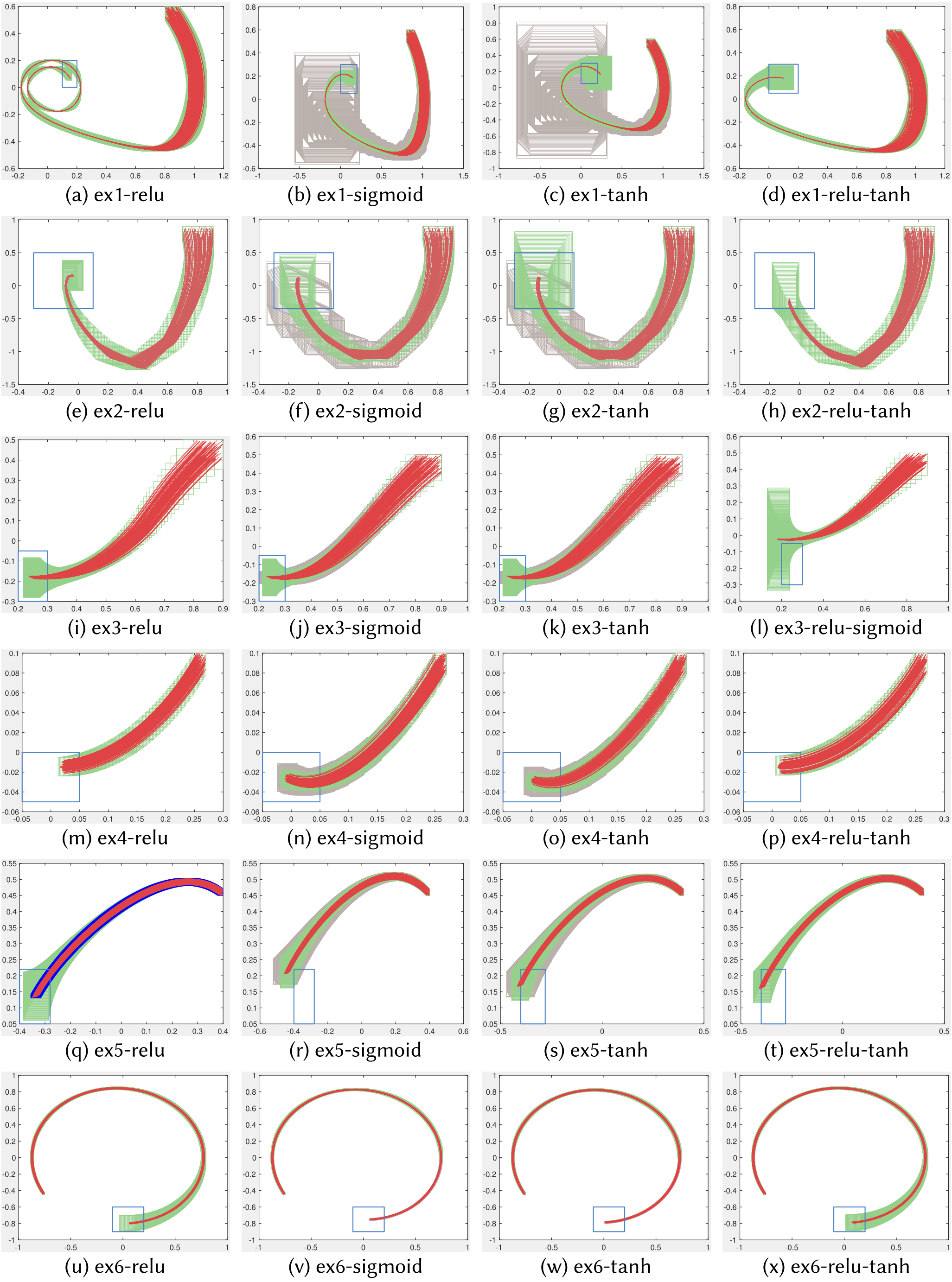
Low input dimension v.s. High input dimension. Thanks to the universal approximation property of Bernstein polynomials, our approach can theoretically approximate any neural network well, if the degree is high enough. However from the the perspective of implementation, we can see that the total order of the generated approximation polynomial $P_{\kappa,d}$ by ReachNN will increase exponentially along with the input dimension. Thus, our current approach may not be efficient enough to handle high dimension inputs in practice. We will investigate methods to address those high-dimensional cases in the future.

6 CONCLUSION

In this paper, we address the reachability analysis of neural-network controlled systems, and present a novel approach ReachNN. Given an input space and a degree bound, our approach constructs a polynomial approximation for a neural-network controller based on Bernstein polynomials and provides two techniques to estimate the approximation error bound. Then, leveraging the off-the-shelf tool Flow*, our approach can iteratively compute flowpipes as over-approximate reachable sets of the neural-network controlled system. The experiment results show that our approach can effectively address various neural-network controlled systems. Our future work includes further tightening the approximation error bound estimation and better addressing high-dimensional cases.

A APPENDIX

We present the plots of flowpipes for each benchmark (see Figure 7). Red curves denote the trajectories of x_1 and x_2 of the system simulated from sampled states within the initial set. Green rectangles are the constructed flowpipes as the overapproximation of the reachable state set by our approach, gray rectangles are the flowpipes computed based on Verisig, and Sherlock computes the flowpipes represented as deep blue rectangles. The blue rectangle is the goal area.



(a) ex1-relu (b) ex1-sigmoid (c) ex1-tanh (d) ex1-relu-tanh

(e) ex2-relu (f) ex2-sigmoid (g) ex2-tanh (h) ex2-relu-tanh

(i) ex3-relu (j) ex3-sigmoid (k) ex3-tanh (l) ex3-relu-sigmoid

(m) ex4-relu (n) ex4-sigmoid (o) ex4-tanh (p) ex4-relu-tanh

(q) ex5-relu (r) ex5-sigmoid (s) ex5-tanh (t) ex5-relu-tanh

(u) ex6-relu (v) ex6-sigmoid (w) ex6-tanh (x) ex6-relu-tanh

Fig. 7. Examples

ACKNOWLEDGMENTS

We gratefully acknowledge the support from the National Science Foundation awards 1834701, 1834324, 1839511, and 1724341. This work is also funded in part by the DARPA BRASS program under agreement number FA8750-16-C-0043 and NSF grant CCF-1646497.

REFERENCES

[1] M. Althoff. 2015. An introduction to CORA 2015. In *Proc. of ARCH'15 (EPiC Series in Computer Science)*, Vol. 34. EasyChair, 120–151.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. 1995. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.* 138, 1 (1995), 3–34.

[3] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep?. In *Advances in Neural Information Processing Systems*. 2654–2662.

[4] Randall D. Beer, Hillel J. Chiel, and Leon S. Sterling. 1989. Heterogeneous neural networks for adaptive behavior in dynamic environments. In *Advances in Neural Information Processing Systems*. 577–585.

[5] M. Berz and K. Makino. 1998. Verified integration of ODEs and flows using differential algebraic methods on high-order taylor models. *Reliable Computing* 4 (1998), 361–369. Issue 4.

[6] B. M. Brown, D. Elliott, and D. F. Paget. 1987. Lipschitz constants for the Bernstein polynomials of a Lipschitz continuous function. *Journal of Approximation Theory* 49, 2 (1987), 196–199.

[7] Cristian Bucilă, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 535–541.

[8] X. Chen. 2015. *Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models*. Ph.D. Dissertation. RWTH Aachen University.

[9] X. Chen, E. Ábrahám, and S. Sankaranarayanan. 2012. Taylor model flowpipe construction for non-linear hybrid systems. In *Proc. of RTSS'12*. IEEE Computer Society, 183–192.

[10] X. Chen, E. Ábrahám, and S. Sankaranarayanan. 2013. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of CAV'13 (LNCS)*, Vol. 8044. Springer, 258–263.

[11] X. Chen and S. Sankaranarayanan. 2016. Decomposed reachability analysis for nonlinear systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE Press, 13–24.

[12] Louis De Branges. 1959. The stone-weierstrass theorem. *Proc. Amer. Math. Soc.* 10, 5 (1959), 822–824.

[13] T. Dreossi, T. Dang, and C. Piazza. 2016. Parallelotope bundles for polynomial reachability. In *HSCC*. ACM, 297–306.

[14] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. 2015. C2E2: A verification tool for stateflow models. In *Proc. of TACAS'15 (LNCS)*, Vol. 9035. Springer, 68–82.

[15] S. Dutta, X. Chen, and S. Sankaranarayanan. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Hybrid Systems: Computation and Control (HSCC)*. ACM Press, 157–168.

[16] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. 2018. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*. Springer, 121–138.

[17] G. Frehse. 2005. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *HSCC*. Springer, 258–273.

[18] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In *Proc. of CAV'11 (LNCS)*, Vol. 6806. Springer, 379–395.

[19] Eduardo Gallestey and Peter Hokayem. 2019. Lecture notes in Nonlinear Systems and Control.

[20] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. 1998. What's decidable about hybrid automata? *Journal of Computer and System Sciences* 57, 1 (1998), 94–124.

[21] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR* abs/1503.02531 (2015).

[22] C. Huang, X. Chen, W. Lin, Z. Yang, and X. Li. 2017. Probabilistic safety verification of stochastic hybrid systems using barrier certificates. *TECS* 16, 5s (2017), 186.

[23] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. 2017. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 3–29.

[24] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2018. Verisig: Verifying safety properties of hybrid systems with neural network controllers. *arXiv preprint arXiv:1811.01828* (2018).

[25] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

[26] S. Kong, S. Gao, W. Chen, and E. M. Clarke. 2015. dReach: δ-reachability analysis for hybrid systems. In *Proc. of TACAS'15 (LNCS)*, Vol. 9035. Springer, 200–205.

[27] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2016).

[28] George G. Lorentz. 2013. *Bernstein Polynomials*. American Mathematical Soc.

[29] J. Lygeros, C. Tomlin, and S. Sastry. 1999. Controllers for reachability specifications for hybrid systems. *Automatica* 35, 3 (1999), 349–370.

[30] K. Makino and M. Berz. 2005. Verified global optimization with taylor model-based range bounders. *Transactions on Computers* 11, 4 (2005), 1611–1618.

[31] J. D. Meiss. 2007. *Differential Dynamical Systems*. SIAM publishers.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[33] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. 2018. Agile autonomous driving using end-to-end deep imitation learning. *Proceedings of Robotics: Science and Systems. Pittsburgh, Pennsylvania* (2018).

[34] S. Prajna and A. Jadbabaie. 2004. Safety verification of hybrid systems using barrier certificates. In *HSCC*. Springer, 477–492.

[35] H. L. Royden. 1968. *Real Analysis*. Krishna Prakashan Media.

[36] W. Ruan, X. Huang, and M. Kwiatkowska. 2018. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242* (2018).

[37] Georgi V. Smirnov. 2002. *Introduction to the Theory of Differential Inclusions*. Vol. 41. American Mathematical Soc.

[38] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[39] W. Xiang and T. T. Johnson. 2018. Reachability analysis and safety verification for neural network control systems. *arXiv preprint arXiv:1805.09944* (2018).

[40] Z. Yang, C. Huang, X. Chen, W. Lin, and Z. Liu. 2016. A linear programming relaxation based approach for generating barrier certificates of hybrid systems. In *FM*. Springer, 721–738.

[41] Yuichi Yoshida and Takeru Miyato. 2017. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941* (2017).

[42] F. Zhao. 1992. *Automatic Analysis and Synthesis of Controllers for Dynamical Systems Based on Phase-Space Knowledge*. Ph.D. Dissertation. Massachusetts Institute of Technology.

Received April 2019; revised June 2019; accepted July 2019