POLAR: A Polynomial Arithmetic Framework for Verifying Neural-Network Controlled Systems

Chao Huang[*]
University of Liverpool
chao.huang2@liverpool.ac.uk

Jiameng Fan
Boston University
jmfan@bu.edu

Xin Chen
University of Dayton
xchen4@udayton.edu

Wenchao Li
Boston University
wenchao@bu.edu

Qi Zhu
Northwestern University
qzhu@northwestern.edu

Abstract

We propose POLAR [2], a **pol**ynomial **ar**ithmetic framework that leverages polynomial overapproximations with interval remainders for bounded-time reachability analysis of neural network-controlled systems (NNCSs). Compared with existing arithmetic approaches that use standard Taylor models, our framework uses a novel approach to iteratively overapproximate the neuron output ranges layer-by-layer with a combination of Bernstein polynomial interpolation for continuous activation functions and Taylor model arithmetic for the other operations. This approach can overcome the main drawback in the standard Taylor model arithmetic, i.e. its inability to handle functions that cannot be well approximated by Taylor polynomials, and significantly improve the accuracy and efficiency of reachable states computation for NNCSs. To further tighten the overapproximation, our method keeps the Taylor model remainders symbolic under the linear mappings when estimating the output range of a neural network. We show that POLAR can be seamlessly integrated with existing Taylor model flowpipe construction techniques, and demonstrate that POLAR significantly outperforms the current state-of-the-art techniques on a suite of benchmarks.

1 Introduction

Machine learning models, especially deep neural networks, have been increasingly used as general decision makers [1, 2, 3]. This has given birth to a category of autonomous systems known as neural-network controlled systems (NNCSs). NNCSs are becoming pervasive in areas such as cyber-physical systems (CPS) for a variety of tasks in which the controllers are difficult to synthesize using traditional approaches. However, using a neural network controller also gives rise to a new challenge which is to verify the correctness of the resulting closed-loop control system.

It is difficult to formally verify an NNCS in general due to complex system behaviors involving continuous evolution and neural network feedback. Although a lot of effort has been devoted to developing techniques for computing or estimating neural network output ranges [4, 5, 6, 7, 8], very few of them can be applied to verifying NNCSs due to their limitation in coping with continuous system evolution. New techniques have been developed for NNCS verification via reachable set computation (overapproximation), however they are often confined by either the category of continuous dynamics, or the type of the activation functions used in the neural network

[*]Part of the work was done when the author was in Northwestern University, US.
[2]The tool is now available at: `https://github.com/ChaoHuang2018/POLAR`.

controller. In [9], the authors propose to first translate an NNCS to a hybrid system and then use the existing tool Flow* [10] to verify the later one. The method is later improved to directly use Taylor model arithmetic [11] and preconditioned Taylor models [12] to compute more accurate reachable sets. Due to the limitation of Taylor approximations, these methods cannot handle ReLU functions. In [13], the authors present an approach of tracking the dependency of state variables under a neural network input-output mapping using piecewise linear polynomials for improving the accuracy of reachable set computation. However, the technique can only handle ReLU neural networks. The use of Bernstein polynomials as abstractions of neural networks has been considered in [14, 15]. Although these methods can in theory handle all continuous activation functions, the estimation requires performing subdivisions and sampling in a multidimensional state space which leads to an exponential blowup in the number of the state variables in the worst case. An effective technique that uses star sets is described in [16], but it can only handle linear continuous dynamics and piecewise linear activation functions in the absence of additional linearization techniques.

In this paper, we propose POLAR, a **pol**ynomial **ar**ithmetic framework to iteratively overapproximate the neuron output ranges layer-by-layer. It adopts univariate Bernstein polynomial interpolation to approximate continuous activation functions, and Taylor model arithmetic for the other operations. Our approach also tries to keep Taylor model remainder symbolic under linear mappings simultaneously. Compared with [13], our approach can handle a wider spectrum of neural networks beyond ReLU networks. Compared with [11, 12], the use of Bernstein polynomial interpolation for activation function along with symbolic remainder makes our approach much more efficient and accurate. Compared with multivariate Bernstein polynomial interpolation [14], our neuron-wise approximation shows a significant advantage on efficiency. Compared with symbolic star set technique [16], our approach can handle more general systems with nonlinear dynamics.

In summary, our work makes the following novel contributions.

- We propose a novel polynomial arithmetic framework for bounded time reachability analysis of NNCSs. Our approach has the advantage of being able to handle NN controllers with general and heterogeneous activation functions.
- We propose neuron-wise Bernstein polynomial interpolation and show that it can be seamlessly integrated with Taylor model approximation for physical plant dynamics. We further use symbolic remainder to tighten the overapproximation of neural network behavior under linear mappings.
- We conduct a comprehensive comparison of our approach with state-of-the-art techniques, including evaluation on the full set of benchmarks published in these works and an additional complex multi-input-multi-output system, showing the overwhelming advantage of our proposed approach.

2 Preliminaries

In this section, we first give formal definitions of neural-network controlled systems and the associated verification problem. Then we revisit the set-based arithmetic techniques using intervals and Taylor models. We use $\mathbb{R}$ to denote the set of real numbers.

2.1 Neural-Network Controlled Systems

A *neural-network controlled system (NNCS)* is a continuous plant governed by a neural network controller, as shown in Figure 1. We denote an NNCS by a tuple $(X, U, F, \kappa, \delta_c, \mathcal{X}_0)$ wherein $X = \{x_1, \ldots, x_n\}$ is the set of finitely many state variables, $U = \{u_1, \ldots, u_m\}$ is the set of finitely many control variables (or inputs), F is the ODE defining the continuous dynamics of the plant, $\kappa(\cdot)$ denotes the input-output mapping defined by the neural network controller, δ_c is a positive number denoting the control stepsize, and $\mathcal{X}_0 \subset \mathbb{R}^n$ is the set of the initial values of the state variables.

We consider ODEs in the form of $\dot{\vec{x}} = f(\vec{x}, \vec{u})$ as the formal definition of plant dynamics. The vector $\vec{x}$ is a collective denotation for the variables $x_1, \ldots, x_n$ and $\vec{u}$ is the denotation for the variables $u_1, \ldots, u_m$. We assume that the function f is at least locally Lipschitz continuous such that its solution w.r.t. an initial state and constant control inputs is unique [17].

An *execution (trajectory)* of an NNCS can be generated in the following way. Starting from an initial state $\vec{x}(0) {\in} \mathcal{X}_0$, the controller senses the system state at the beginning of every control step $t{=}k\delta_c$ for $k{=}0, 1, \ldots$, and update the control inputs to $\vec{v}_k{=}\kappa(\vec{x}(k\delta_c))$. The system's dynamics in that control

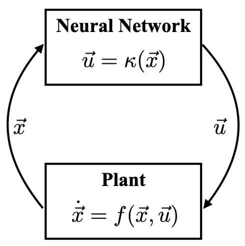

Neural Network
$\vec{u} = \kappa(\vec{x})$
$\vec{x}$
$\vec{u}$
Plant
$\dot{\vec{x}} = f(\vec{x}, \vec{u})$

Figure 1: Neural-network controlled system.

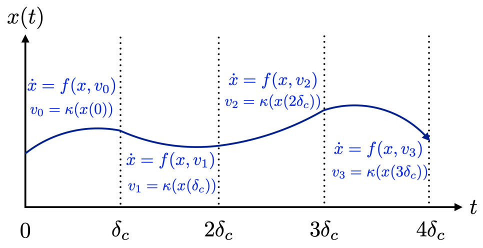


Figure 2: An Execution over 4 control steps.

step is governed by the ODE $\dot{\vec{x}}{=}f(\vec{x}, \vec{v}_k)$. Figure 2 illustrates an execution of NNCS. Because of the Lipschitz continuity of the plant dynamics, the execution from an initial state is always unique.

All executions of an NNCS can be formally defined by a *flowmap* function $\varphi_\mathcal{N} : \mathcal{X} \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, such that the system state at any time $t \geq 0$ from any initial state $\vec{x}(0) \in \mathcal{X}_0$ is the function output $\varphi_\mathcal{N}(\vec{x}(0), t)$. We call a state $\vec{x}' \in \mathbb{R}^n$ *reachable* if there exists $\vec{x}(0) \in \mathcal{X}_0$ and $t \geq 0$ such that $\vec{x}' = \varphi_\mathcal{N}(\vec{x}(0), t)$. The *reachability problem* on NNCS is to decide whether a state is reachable in a given NNCS, and it is *undecidable* due to their higher expressiveness than the two-counter machines on which the reachability problem is already undecidable [18].

Formal Verification. Many formal verification problems can be reduced to the reachability problem. For example, the safety verification problem can be reduced to the reachability problem of checking whether an unsafe state is reachable. In the paper, we verify a safety or reachability property on an NNCS by computing its reachable set over a given number of control steps. Since the actual reachable set is hard to compute, we focus on computing a tight overapproximation of it.

2.2 Taylor Model Arithmetic

Our reachable set overapproximation approach uses the set-based computation methods on intervals and Taylor models. We briefly revisit these techniques.

An *interval* $[a, b]$ wherein $a \leq b$ denotes the set of all real values ranging from a to b. Intervals are used as overapproximate representations for reals in numerical computation. They can also be organized as vectors or matrices. The basic operations on reals can be extended to intervals, e.g., the interval addition and multiplication are defined by $[a, b] + [c, d] = [a + c, b + d]$ and $[a, b] \cdot [c, d] = [\min\{a \cdot c, a \cdot d, b \cdot c, b \cdot d\}, \max\{a \cdot c, a \cdot d, b \cdot c, b \cdot d\}]$ respectively. and such extension is called *interval arithmetic* which is very useful in computing conservative ranges for the results of various computation tasks in which approximation or roundoff errors need to be taken into account.

Taylor models are originally proposed to compute higher-order overapproximations for the ranges of continuous functions (see [19]). A *Taylor model (TM)* is a pair (p, I) wherein p is a polynomial of degree k over a finite group of variables $x_1, \ldots, x_n$ ranging in an interval domain $D \subset \mathbb{R}^n$, and I is the remainder interval. The range of a TM is the Minkowski sum of the range of its polynomial and the remainder interval. Thereby we sometimes intuitively denote a TM (p, I) by $p + I$ in the paper.

Comparing to intervals, TMs often provide more accurate overapproximations. Given a continuous function $f(\vec{x})$ with $\vec{x} \in D$ such that f has partial derivatives upto the order of $k > 0$. An interval overapproximation for the range of f can only be as accurate as an interval whose lower bound is $\min\{f(\vec{x}) \,|\, \vec{x} \in D\}$ and upper bound is $\max\{f(\vec{x}) \,|\, \vec{x} \in D\}$, such that the dependency between the value of f and the value of $\vec{x}$ is not characterized in the overapproximation. However, a TM overapproximation (p_f, I_f) can be obtained by computing p_f as the order k Taylor expansion of f at the center of D and I_f as an interval including the remainder error. Such a model provides a point-wise overapproximation: $\forall \vec{x} \in D.(f(\vec{x}) \in p_f(\vec{x}) + I_f)$ wherein $+$ denotes the operation that adding the interval I_f onto the point $p_f(\vec{x})$. Although the range of a TM overapproximation is not always smaller than the interval overapproximation range for a function, the accumulation of overestimation in a computation task on TMs is often much smaller than that on intervals due to the characteristics of the dependencies. This fact has already been demonstrated in verified integration and global optimization tasks [20, 21].

Algorithm 1 Flowpipe Construction for NNCS

```
Input: NNCS $(X, U, F, \kappa, \delta_c, \mathcal{X}_0)$, number of control steps $K$.
Output: Overapproximation of the reachable set over the time interval of $[0, K\delta_c]$ such that $\delta_c$ is the
    control stepsize.
1: $\mathcal{R} \leftarrow \emptyset$;                                   # the resulting overapproximation set
2: $\mathcal{X}_i \leftarrow \mathcal{X}_0$;                             # the initial set in every control step
3: for $i = 1$ to $K$ do
4:     Computing the an overapproximation $\mathcal{U}_i$ for the range $\kappa(\mathcal{X}_i)$;
5:     Computing the reachable set overapproximation $\mathcal{R}_i$ over the time interval of $[0, \delta_c]$ under the
       continuous dynamics $\dot{\vec{x}} = f(\vec{x}, \vec{u}), \dot{\vec{u}} = 0$ from the initial set $\vec{x}(0) \in \mathcal{X}_i$, $\vec{u}(0) \in \mathcal{U}_i$;
6:     $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_i$;
7:     Evaluating a set for the last flowpipe in $\mathcal{R}_i$ at the end of the time step and assigning it to $\mathcal{X}_i$;
8: end for
9: return $\mathcal{R}$.
```

TM Arithmetic. TMs are closed under operations such as addition, multiplication, and integration (see [22]). Given functions f, g that are overapproximated by TMs (p_f, I_f) and (p_g, I_g) respectively. A TM for $f + g$ can be computed as $(p_f + p_g, I_f + I_g)$, and an order k TM for $f \cdot g$ can be computed as $(\, p_f \cdot p_g - r_k\, ,\, I_f \cdot B(p_g) + B(p_f) \cdot I_g + I_f \cdot I_g + B(r_k)\,)$ wherein $B(p)$ denotes an interval enclosure of the range of p, and the *truncated part* r_k consists of the terms in $p_f \cdot p_g$ of degrees $> k$. Similar to reals and intervals, TMs can also be organized as vectors and matrices to overapproximate the functions whose ranges are multidimensional. For a vector of TMs $((p_1, I_1), \ldots, (p_n, I_n))^T$ such that $p_1, \ldots, p_n$ are over the same variables, we collectively denote it by (p, I) such that p is the polynomial vector $(p_1, \ldots, p_n)^T$ and I is interval vector $(I_1, \ldots, I_n)^T$. As an example, given the TMs $(1 - 0.5x^2, [-0.1, 0.1])$ and $(x + 0.1x^4, [-0.2, 0.2])$ over the domain $x \in [-1, 1]$. The order 4 TM for the sum is $(1 + x - 0.5x^2 + 0.1x^4, [-0.3, 0.3])$, and the order 4 TM for the product is $(x - x^3 + x^4, [-0.38, 0.38])$.

3 Verification Framework Using Polynomial Arithmetic

We present our framework for computing NNCS reachable set overapproximations using polynomial arithmetic. Our approach has three main components: (a) flowpipe construction under ODE dynamics, (b) computing a TM output range for a neural network controller for a given TM input, and (c) a technique of symbolically representing interval remainders in (b). The tasks of (a) can be handled by the existing TM flowpipe construction technique in [23], whereas (b) and (c) constitute our main contributions which allow a seamless integration of the three components to produce state-wise overapproximations for NNCS reachable sets. In this section, we first present our main framework, briefly recall the TM flowpipe construction method, and then describe (b) and (c) in detail.

3.1 Flowpipe Construction for NNCS

We verify a property on a given NNCS by checking its reachable set over a bounded number of control steps. Since the exact set of the reachable states cannot be computed, we use an adaption of the classic flowpipe construction scheme for NNCS, and it is shown in Algorithm 1.

The algorithm basically performs an overapproximated set-based simulation on the given NNCS for a bounded time horizon which is defined by K control steps. At the beginning of every control step, it computes an overapproximated set for the control inputs which are obtained from the NN controller based on the current system state set $\mathcal{X}_i$ (Line 4). Then the algorithm performs the flowpipe construction technique (Line 5) to compute the reachable set overapproximation over the current control step under the continuous dynamics defined by the ODE: $\dot{\vec{x}} = f(\vec{x}, \vec{u}), \dot{\vec{u}} = 0$, which is an extension of the original ODE by treating the constant control inputs as new static state variables. Notice that such an extension has the same behavior at the state variables $\vec{x}$ as the original dynamics $\dot{\vec{x}} = f(\vec{x}, \vec{u})$ with time-invariant $\vec{u} \in \mathcal{U}_i$ (see [23]).

Property Verification. A property such as safety or reachability of the original NNCS can be verified based on the computed reachable set overapproximations (see [24, 14]). (i) *Safety:* If $\mathcal{R}$ does not violate the safety property, then the system is safe. If we are able to find a flowpipe which completely violates the safety property, then the system is unsafe. Otherwise, the safety is unknown and we need to refine the overapproximations. (ii) *Reachability:* The unreachability proof is similar to the safety proof, and the reachability proof is similar to the unsafety proof.

In our framework, the sets $\mathcal{X}_i, \mathcal{U}_i$ are computed as TMs, and $\mathcal{R}_i$ is a union of TMs. We will briefly revisit the flowpipe construction technique for ODEs and introduce our higher-order overapproximation technique to compute the output range of an NN controller in the rest of the section.

Flowpipe Construction for ODEs. We recall the TM flowpipe construction technique for computing reachable set overapproximations under the dynamics defined by ODEs. More details are described in [19, 24, 23]. Here, we only consider the ODE defined in the form of $\dot{\vec{x}} = f(\vec{x})$, since the dynamics in Algorithm 1 can be equivalently translated to this form by treating $\vec{x}, \vec{u}$ as state variables.

Given a continuous dynamics $\dot{\vec{x}} = f(\vec{x})$ and an initial set $\vec{x}(0) \in X_0$ such that X_0 is represented by a TM or interval, the method of TM integration is to compute a finite set of TM flowpipes $(p_1, I_1), \dots, (p_N, I_N)$ with a given flowpipe time stepsize $\delta > 0$ such that $(p_i(\vec{z}, t), I_i)$ is an overapproximation of the exact system reachable state set $\varphi_f(\vec{z}, t + (i-1)\delta)$ with $\vec{z} \in X_0$, $t \in [0, \delta]$, for $1 \leq i \leq N$, i.e., all of the reachable set over the time interval from $(i-1)\delta$ to $i\delta$. To do so, the flowpipes are computed iteratively. In the i-th iteration which is also called integration step the *local initial set* is X_0 if $i = 1$, and is computed as a TM $X_{i-1} = (p_{i-1}(\vec{z}, \delta), R_{i-1})$ if $i \geq 2$. Then, the i-th TM flowpipe can be obtained from the following steps.
(1) Computing the order k Taylor expansion $\Phi_i(\vec{y}, t)$ at t=0 for the ODE solution $\varphi_f(\vec{y}, t)$ with $\vec{y} {\in} X_{i-1}$.
(2) Finding a proper remainder R_i such that $\varphi_f(\vec{y}, t)$ is overapproximated by $(\Phi_i(\vec{y}, t), R_i)$ with $t \in [0, \delta]$. It can be done by verifying the contractiveness (the output TM is contained in the input TM) of the Picard operator on $(\Phi_i(\vec{y}, t), I_i)$ (see [19, 23]).
(3) Computing the i-th TM flowpipe (p_i, I_i) by evaluating $(\Phi_i(X_{i-1}, t), R_i)$ using TM arithmetic.

The TM flowpipes computed by the above method provide state-wise (point-wise) overapproximations for the exact reachable set, i.e., for any initial state $\vec{x}(0) \in X_0$ and $t \in [0, N\delta]$, the exact reachable state $\vec{x}(t) = \varphi_f(\vec{x}(0), t)$ is guaranteed to be contained in the TM $p_i(\vec{x}(0), t - (i-1)\delta) + I_i$ such that $1 \leq i \leq N$ and $t \in [(i-1)\delta, i\delta]$.

TM flowpipe construction can be applied to the task in Line 5 of Algorithm 1. The flowpipe time stepsize is $\delta = \delta_c / N$, and the resulting TMs overapproximate all reachable states in the control step. The local initial set $\mathcal{X}_i$ for the next control step (Line 7) can be obtained from evaluating the last flowpipe in $\mathcal{R}_i$ at the end of the flowpipe step time, i.e., the TM $(p_N(\vec{z}, \delta), I_N)$. Notice that the time variable t in each flowpipe are independent and only ranges in a single flowpipe time step: $[0, \delta]$.

3.2 Polynomial Arithmetic for Overapproximating Neural Network Outputs

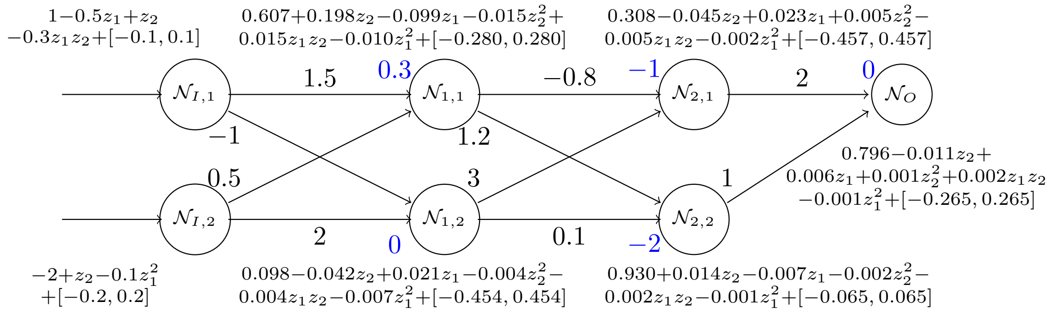
In this section, we describe our key contribution – a polynomial arithmetic to compute a TM overapproximation of a feedforward neural network which is only required to have continuous activation functions. Given a TM input set for a neural network, our approach combines the use of TM arithmetic and Bernstein polynomial interpolation to produce a point-wise overapproximation set for the output of a neural network. The high-level steps are described in Algorithm 2, in which we assume that the neural network has M hidden layers, the neurons in the same layer have the same type of activation functions, and the output layer's neurons also have activation functions. We collectively use $\sigma(\vec{y})$ to denote applying the activation function σ to each element of $\vec{y}$.

Computing the Bernstein polynomial $\boldsymbol{p_\sigma}$. Given the computed TM (p_t, I_t) in Line 3, p_σ is a vector of univariate Bernstein polynomials [25] each component of which is expressed in a variable y_j which denotes the range of the j-th component (dimension) of (p_t, I_t). Then the j-th component of p_σ is computed as $p_\sigma^j(y_j) = \sum_{s=0}^{k} \left(\sigma(\frac{\bar{Y}_j - \underline{Y}_j}{k} s + \underline{Y}_j) \binom{k}{s} \frac{(y_j - \underline{Y}_j)^s (\bar{Y}_j - y_j)^{k-s}}{(\bar{Y}_j - \underline{Y}_j)^k} \right)$ which is an approximation of the $\sigma(y_j)$ such that $\bar{Y}_j$ and $\underline{Y}_j$ denotes the upper and lower bounds respectively of the range in the j-th dimension in (p_t, I_t), and they can be obtained by interval evaluating the TM. Here, the positive integer k is a user given Bernstein approximation order.

Evaluating a conservative remainder $\boldsymbol{I_\sigma}$ for $\boldsymbol{p_\sigma}$. The remainder I_σ is a vector of intervals, each component of which is a conservative remainder for the corresponding Bernstein polynomial in p_σ

Algorithm 2 Polynomial arithmetic for overapproximating neural network outputs

```
Input: Input TM $(p(\vec{z}), I)$ with $\vec{z} \in D$, the $M+1$ matrices $W_1, \dots, W_{M+1}$ of the weights on the
    incoming edges of the hidden and the output layers, the $M+1$ vectors $B_1, \dots, B_{M+1}$ of the
    neurons' bias in the hidden and the output layers, the $M+1$ activation functions $\sigma_1, \dots, \sigma_{M+1}$
    of hidden and output layers.
Output: a TM $(p_r(\vec{z}), I_r)$ that contains the set $\kappa((p(\vec{z}), I))$.
1: $(p_r, I_r) \leftarrow (p, I)$;
2: for $i = 1$ to $M+1$ do
3:     $(p_t, I_t) \leftarrow W_i \cdot (p_r, I_r) + B_i$;                      # Using TM arithmetic
4:     Computing an order $k_B$ Bernstein polynomial $p_\sigma$ for the activation function over the range
       defined by $(p_t, I_t)$, i.e., $p_\sigma(\vec{y}) \approx \sigma_i(\vec{y})$ with $\vec{y} \in (p_t, I_t)$;
5:     Evaluating a safe remainder $I_\sigma$ for the polynomial $p_\sigma$ such that $\forall \vec{y} \in (p_t, I_t).(\sigma_i(\vec{y}) \in$
       $p_\sigma(\vec{y}) + I_\sigma)$ holds;
6:     $(p_r, I_r) \leftarrow p_\sigma(p_t + I_t) + I_\sigma$;                      # Using TM arithmetic
7: end for
8: return $(p_r, I_r)$.
```



$1-0.5z_1+z_2$ $-0.3z_1z_2+[-0.1,0.1]$
$0.607+0.198z_2-0.099z_1-0.015z_2^2+$ $0.015z_1z_2-0.010z_1^2+[-0.280,0.280]$
$0.308-0.045z_2+0.023z_1+0.005z_2^2-$ $0.005z_1z_2-0.002z_1^2+[-0.457,0.457]$
1.5
0.3
−0.8
−1
2
0
$\mathcal{N}_{I,1}$
$\mathcal{N}_{1,1}$
$\mathcal{N}_{2,1}$
$\mathcal{N}_O$
−1
1.2
$0.796-0.011z_2+$ $0.006z_1+0.001z_2^2+0.002z_1z_2$ $-0.001z_1^2+[-0.265,0.265]$
0.5
3
1
$\mathcal{N}_{I,2}$
$\mathcal{N}_{1,2}$
$\mathcal{N}_{2,2}$
2
0
0.1
−2
$-2+z_2-0.1z_1^2$ $+[-0.2,0.2]$
$0.098-0.042z_2+0.021z_1-0.004z_2^2-$ $0.004z_1z_2-0.007z_1^2+[-0.454,0.454]$
$0.930+0.014z_2-0.007z_1-0.002z_2^2-$ $0.002z_1z_2-0.001z_1^2+[-0.065,0.065]$

Figure 3: Demonstration of TM Output Computation

such that the approximation error for $\sigma((p_t, I_t))$ is contained. To do so, the j-th interval in I_σ is evaluated as $[-\epsilon_j, \epsilon_j]$ such that

$$\epsilon_j = \max_{i=1,\cdots,m} \left(\left| p_\sigma^j\left(\frac{\bar{Y}_j - \underline{Y}_j}{m}(i - \frac{1}{2}) + \underline{Y}_j\right) - \sigma\left(\frac{\bar{Y}_j - \underline{Y}_j}{m}(i - \frac{1}{2}) + \underline{Y}_j\right) \right| + \frac{\bar{Y}_j - \underline{Y}_j}{m} \right)$$

wherein m is the number of samples which are uniformly selected to estimate the remainder.

Unlike using multivariate Bernstein polynomials in [14], our framework only computes univariate Bernstein polynomials such that the computation complexity is always linear in the approximation order k, and the remainder is much easier to be evaluated. In contrast, the size of a multivariate Bernstein polynomial grows exponentially in the order k.

Example 1 *We demonstrate the computation of a TM output for a small neural network shown in Figure 3. The neural network has* 2 *inputs,* 1 *output and* 2 *hidden layers each of which has* 2 *neurons. The neurons* $\mathcal{N}_{I,1}$, $\mathcal{N}_{I,2}$ *in the input layer has identity activation function, and the activation functions of all of the other neurons are sigmoid. The weights are given along with their edges, and the bias are show in blue along with the neurons. In the figure, the TMs* $1 - 0.5z_1 + z_2 - 0.3z_1z_2 + [-0.1, 0.1]$ *and* $-2 + z_2 - 0.1z_1^2 + [-0.2, 0.2]$ *are the inputs of* $\mathcal{N}_{I,1}$ *and* $\mathcal{N}_{I,2}$ *respectively, and all of the other TMs are the output of the corresponding neurons.*

Advantages of Bernstein polynomials. We briefly discuss the advantages of using Bernstein polynomials over Taylor expansions in the approximation of activation functions. (i) Bernstein approximations are essentially polynomial interpolations which are more accurate approximation forms than Taylor expansion at the same order for the a function (see [26]). (ii) Taylor approximation requires the approximated function to be differentiable, however Bernstein approximation only requires the function to be continuous. This fact makes Taylor approximation not applicable to ReLU functions.

If the input TM (p, I) is a state-wise overapproximation for the reachable state at the beginning of i-th control step, i.e., $p(\vec{z}) + I$ contains the reachable state at the time $t = (i-1)\delta_c$ from $\vec{z}$ for all $\vec{z} \in \mathcal{X}_0$, then the returned TM $p_r(\vec{z}) + I_r$ contains the exact i-th control input in the execution starting from $\vec{z}$ for all $\vec{z} \in \mathcal{X}_0$, i.e., the output TM (p_r, I_r) is a state-wise overapproximation of the control input. Hence, we have the important property given in Theorem 1, and then the set $\mathcal{R}$ computed in Algorithm 1 is an overapproximation of the system reachable set in K control steps.

Theorem 1 *If $(p(\vec{z}, t), I)$ is the i-th flowpipe computed in the j-th control step, then for any initial state $\vec{x}_0 \in \mathcal{X}_0$, the box $(p(\vec{x}_0, t), I)$ is guaranteed to contain the reachable state $\varphi_{\mathcal{N}}(\vec{x}_0, (j-1)\delta_c + (i-1)\delta + t)$, i.e., the TM flowpipes computed by our approach are point-wise overapproximations for the reachable sets.*

3.3 Symbolic Remainders in Neural Network Analysis

We describe the symbolic remainder method which is a modification of the one described in [27] to reduce the overestimation produced in Algorithm 2. Since the TM flowpipes produced by our framework are state-wise overapproximations, the overestimation size in a flowpipe is directly reflected by the width of the remainder interval. The main idea of the symbolic method is to treat the remainders during the computation of the output TM remainder symbolic by only computing their matrix coefficients. For example, an interval for $A_k \cdot \dots \cdot A_1 \cdot I$ can be more accurately evaluated by first computing the matrix coefficients $\mathcal{A} = A_k \cdot \dots \cdot A_1$ and then evaluating the interval $\mathcal{A} \cdot I$ than performing the iterations $I_i = A_i \cdot I_{i-1}$ from $I_0 = I$ by k times using interval arithmetic (see [28]). The overestimation accumulation in the latter method is called *wrapping effect* [28]. The purpose of our symbolic remainder method is to avoid the wrapping effect in purely linear mappings.

We denote the resulting TM (p_r, I_r) and the Bernstein overapproximation (p_σ, I_σ) in the i-th iteration in Algorithm 2 by $(p_{r,i}, I_{r,i})$ and $(p_{\sigma_i}, I_{\sigma_i})$ respectively. Then $(p_{r,i}(\vec{z}), I_{r,i})$ for $i = 1, \dots, M+1$ is computed as $p_{r,i}(\vec{z}) + I_{r,i} = p_{\sigma_i}(W_i \cdot (p_{r,i-1}(\vec{z}) + I_{r,i-1}) + I_{\sigma_i}$ with $(p_{r,0}(\vec{z}), I_{r,0}) = (p(\vec{z}), I)$ which is the input TM. If we use $q_i(\vec{y})$ to denote the function $p_{\sigma_i}(W_i \cdot \vec{y})$, the above expression can be simplified to $p_{r,i}(\vec{z}) + I_{r,i} = q_i(p_{r,i-1}(\vec{z}) + I_{r,i-1}) + I_{\sigma_i}$. By decomposing the linear and the remaining part in q_i, the function can be expressed as $q_i(\vec{y}) = Q_i\vec{y} + q_i^R(\vec{y})$ such that Q_i is the constant matrix consists of the coefficients of the linear terms in q_i, and q_i^R is the nonlinear and the constant part in q_i. Hence, the output TM can be further expressed as

$$p_{r,i}(\vec{z}) + I_{r,i} = Q_i(p_{r,i-1}(\vec{z}) + I_{r,i-1}) + \underbrace{q_i^R(p_{r,i-1}(\vec{z}) + I_{r,i-1}) + I_{\sigma_i}}_{\phi_i(\vec{z}) + J_i}$$

and then by unfolding the recurrence relation, we have

$$\begin{aligned} p_{r,i}(\vec{z}) + I_{r,i} &= Q_i(p_{r,i-1}(\vec{z}) + I_{r,i-1}) + \phi_i(\vec{z}) + J_i \\ &= Q_i(Q_{i-1}(p_{r,i-2}(\vec{z}) + I_{r,i-2}) + \phi_{i-1}(\vec{z}) + J_{i-1}) + \phi_i(\vec{z}) + J_i \\ &= Q_i \cdot \dots \cdot Q_1 \cdot p(\vec{z}) + Q_i \cdot \dots \cdot Q_1 \cdot I + \Phi_i(\vec{z}) + \mathbb{J}_i \end{aligned} \tag{1}$$

wherein $\Phi_i = \phi_i(\vec{z}) + Q_i \cdot \phi_{i-1}(\vec{z}) + \dots + Q_i \cdot \dots \cdot Q_2 \cdot \phi_1(\vec{z})$, and $\mathbb{J}_i = J_i + Q_i \cdot J_{i-1} + \dots + Q_i \cdot \dots \cdot Q_2 \cdot J_1$. Hence, the term $Q_i \cdot \dots \cdot Q_1 \cdot I$ can be evaluated without wrapping effect, and if we keep an array for J_j, the computation of $\mathbb{J}_i$ is also free from wrapping effect. Algorithm 3 shows the improvement of Algorithm 2 in accuracy using the symbolic remainder method such that we additionally use two arrays: $\mathcal{Q}[j]$ is $Q_i \cdot \dots \cdot Q_j$ and $\mathcal{J}[j]$ is J_i for $1 \le j \le i$.

Time and space complexity. Although Algorithm 3 always produces a TM with a smaller remainder than Algorithm 2 because of the symbolic treatment of the remainder intervals I and J_i under linear mappings, it requires (1) two extra arrays to keep the matrices $Q_M, Q_M \cdot Q_{M-1}, \dots, Q_M \cdot \dots \cdot Q_1$, and the remainder $J_{M+1}, \dots, J_1$, (2) two extra inner loops which perform $i-1$ and $i-2$ iterations in the i-th outer iteration. The size of $Q_i \cdot \dots \cdot Q_j$ is determined by the rows in W_i and the columns in W_j, and hence the maximum number of neurons in a layer determines the maximum size of the matrices in Q. Similarly, the maximum dimension of J_i is also bounded by the maximum number of neurons in a layer. Because of the two inner loops, time complexity of Algorithm 3 is quadratic in M, whereas Algorithm 2 is linear in M.

Algorithm 3 TM output computation using symbolic remainders, input and output are the same as those in Algorithm 2

```
1: $(p_r, I_r) \leftarrow (p, I)$;
2: Setting $\mathcal{Q}$ as an empty array which can keep $M+1$ matrices;
3: Setting $\mathcal{J}$ as an empty array which can keep $M+1$ multidimensional intervals;
4: for $i = 1$ to $M+1$ do
5:     Computing the composite function $q_i$ and the remainder interval $I_{\sigma_i}$ using Bernstein overapproximation and TM arithmetic (similar to Line 3 to 5 in Algorithm 2);
6:     Constructing $Q_i$ using the coefficients of the linear terms in $q_i$;
7:     Computing $q_i^R$ from $q_i$ by eliminating the linear terms;
8:     $(\phi_i, J_i) \leftarrow q_i^R(p_r + I_r) + I_{\sigma_i}$;        # Using TM arithmetic
9:     $\mathbb{J} \leftarrow J_i$;
10:    for $j = 1$ to $i-1$ do
11:        $\mathcal{Q}[j] \leftarrow Q_i \cdot \mathcal{Q}[j]$;
12:    end for
13:    Adding $Q_i$ to $\mathcal{Q}$ as the last element;
14:    for $j = 2$ to $i$ do
15:        $\mathbb{J} \leftarrow \mathbb{J} + \mathcal{Q}[j] \cdot \mathcal{J}[j-1]$;
16:    end for
17:    Adding $J_i$ to $\mathcal{J}$ as the last element;
18:    Computing $(p_r, I_r)$ according to (1);        # Using TM arithmetic
19: end for
20: return $\mathcal{R}$.
```

4 Experiments

In this section, we first present an illustrating example of attitude control with 6 state variables and 3 control inputs. We then present a comprehensive comparison to the state-of-the-art tools over the full benchmarks in the related work [14]. Finally, we remark on the observed limitations of our approach. All our experiments were run on a machine with 6-core 2.90 GHz Intel Core i5 and 8GB of RAM.

4.1 Illustrating Example: Attitude Control.

We consider the attitude control of a rigid body with six states and three inputs as a physically illustrating example [29]. The complexity of this example lies in the combination of the numbers of the state variables and control inputs. The system dynamics is

$$\begin{cases} \dot\omega_1 = 0.25(u_0 + \omega_2\omega_3), \qquad \dot\omega_1 = 0.5(u_1 - 3\omega_1\omega_3), \qquad \dot\omega_3 = u_2 + 2\omega_1\omega_2, \\ \dot\psi_1 = 0.5\left(\omega_2(\psi_1^2 + \psi_2^2 + \psi_3^2 - \psi_3) + \omega_3(\psi_1^2 + \psi_2^2 + \psi_2 + \psi_3^2) + \omega_1(\psi_1^2 + \psi_2^2 + \psi_3^2 + 1)\right), \\ \dot\psi_2 = 0.5\left(\omega_1(\psi_1^2 + \psi_2^2 + \psi_3^2 + \psi_3) + \omega_3(\psi_1^2 - \psi_1 + \psi_2^2 + \psi_3^2) + \omega_2(\psi_1^2 + \psi_2^2 + \psi_3^2 + 1)\right), \\ \dot\psi_3 = 0.5\left(\omega_1(\psi_1^2 + \psi_2^2 - \psi_2 + \psi_3^2) + \omega_2(\psi_1^2 + \psi_1 + \psi_2^2 + \psi_3^2) + \omega_3(\psi_1^2 + \psi_2^2 + \psi_3^2 + 1)\right). \end{cases}$$

wherein the state $\vec{x}{=}(\omega, \psi)$ consists of the angular velocity vector in a body-fixed frame $\omega{\in}\mathbb{R}^3$, and the Rodrigues parameter vector $\psi{\in}\mathbb{R}^3$. The control torque $\vec{u}{\in}\mathbb{R}^3$ is updated every 0.1 seconds by a neural network with 3 hidden layers each of which has 64 neurons. The activation of the hidden layers are sigmoid and identity respectively. The initial state set is: $\omega_1{\in}[-0.45, -0.44], \omega_2{\in}[-0.55, -0.54], \omega_3{\in}[0.65, 0.66], \psi_1{\in}[-0.75, -0.74], \psi_2 \in [0.85, 0.86], \psi_3 \in [-0.65, -0.64]$. POLAR computed the TM flowpipes for 30 control steps in **313** seconds without symbolic remainder techniques, and in **201** seconds with symbolic remainder. The symbolic remainder method surprisingly uses less time, since tighter (smaller) flowpipes are easier to compute. Figure 4 shows the plot of the octagonal enclosures of the flowpipes. It can be seen that only the flowpipes at the end have a slight swelling. Also the symbolic remainder technique better tightens the overapproximations. For a comparison, although Verisig 2.0 [12] can also handle such system in theory, its remainder explodes very quickly and the tool crashes after a few steps.

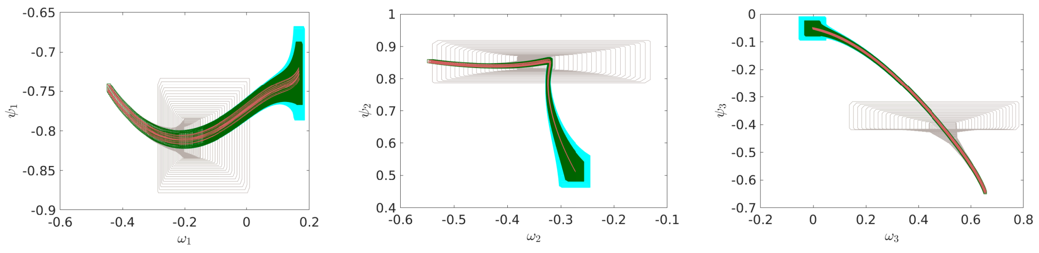


Figure 4: Comparison between reachable sets of the 6-dimensional attitude control benchmark produced by Verisig 2.0 (grey), POLAR without using symbolic remainder (blue) and POLAR using symbolic remainder (green). Verisig 2.0 can only produce reachable sets up to the 6^{th} step (after which the tool crashes due to high uncertainty in its estimation of the reachable set), whereas both POLAR approaches can verify the system for at least 30 steps. The red curves are simulated trajectories.

4.2 Comparison to state-of-the-arts.

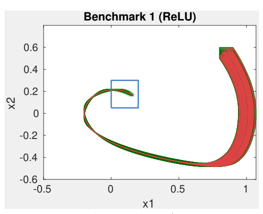
We compare POLAR to the state-of-the-art tools on the full benchmarks in [14], including Sherlock [13] (only for ReLU) , Verisig 2.0 [12] (only for sigmoid and tanh), NNV [16], and ReachNN*[15][3]. For each benchmark, the goal is to check whether the system will reach a given target set. For each tool and in each test, if the computed reachable set overapproximation for the last control step lies entirely in the target set, we consider the tool to have successfully verified the reachability property. If the overapproximation of the reachable set does not intersect with the target set, the tool would have successfully disproved the reachability property. Otherwise, we consider the verification result to be unknown. We refer to [14, 12] for the details of the benchmarks.

Plots of reachable sets computed by different techniques are shown in 5. The red trajectories are sample system executions and should be contained entirely by the flowpipes computed by each tool. The dark green sets are the flowpipes computed by POLAR. The light green sets are the flowpipes computed by ReachNN* [14, 15]. The blue sets are the flowpipes computed by Sherlock [13]. The grey sets are the flowpipes computed by Verisig 2.0 [11]. In some benchmarks, the reachable sets computed by Verisig 2.0 are almost overlapping with the reachable sets computed by POLAR. However, POLAR takes much less time to compute the reachable sets compared to Verisig 2.0 as shown later in Table 1. We also show results from NNV [16] in yellow for some of the benchmarks. For the rest, NNV used up all of the system memory (8GB) and could not finish the computation. Our observations are consistent with those in [11] where NNV is not able to verify any of these benchmarks. The blue box represents the target set in each test. Except for ex2-sigmoid, POLAR produces the tightest reachable set estimation and successfully proves or disproves the reachability property for all the examples.

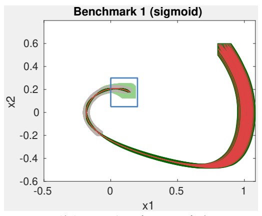
The computation time are presented in Table 1, where NNV is not included since we were not able to successfully use it to prove any of the benchmarks. We can see that POLAR finishes all cases within seconds, and achieves the best performance among all the tools simultaneously. In addition, POLAR scales better with the size of the neural network controller compared to ReachNN* and Verisig 2.0. This demonstrates the potential of handling larger scale systems such as the attitude control example. We discuss the only exception $\#2$ with sigmoid NN controller (ex2-sigmoid) in detail below.

Limitations. POLAR did not prove the property on test # 2 with sigmoid activation but Verisig 2.0 and ReachNN* did. The main reason is the combined use of Bernstein and Taylor approximations. Although a lower-order Taylor approximation can be derived from a higher-order one by truncating the higher-order terms, it is not the case for a Bernstein approximation since the term coefficients often need to be recomputed if the order is changed. Thus, simply using truncation to simplify a TM may produce a large remainder. Secondly, TM-based techniques can be sensitive to the choice of TM settings (stepsizes, orders, etc.). In our experiments, we use the same TM settings for all the tools.

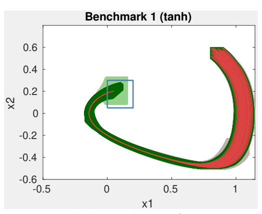
[3]The results of ReachNN* are based on multi-core acceleration.



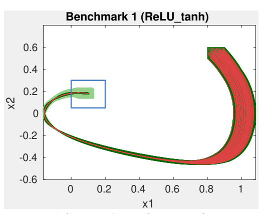
(a) ex1-relu



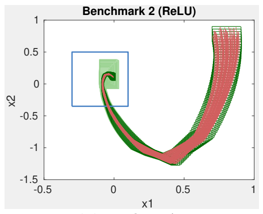
(b) ex1-sigmoid



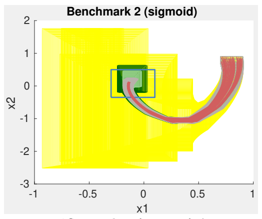
(c) ex1-tanh



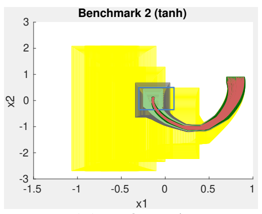
(d) ex1-relu-tanh



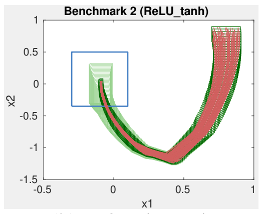
(e) ex2-relu



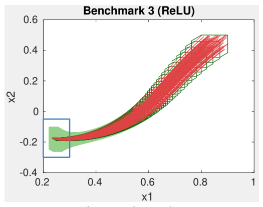
(f) ex2-sigmoid



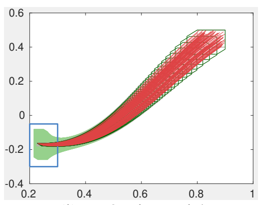
(g) ex2-tanh



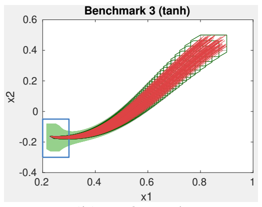
(h) ex2-relu-tanh



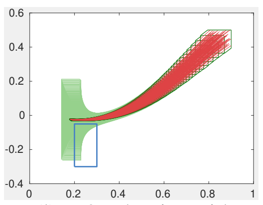
(i) ex3-relu



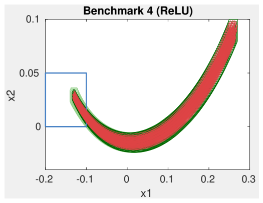
(j) ex3-sigmoid



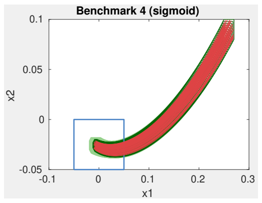
(k) ex3-tanh



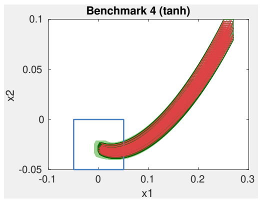
(l) ex3-relu-sigmoid



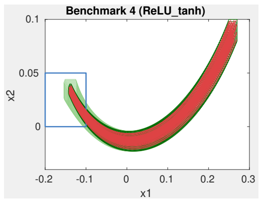
(m) ex4-relu



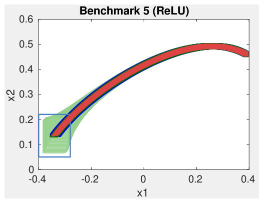
(n) ex4-sigmoid



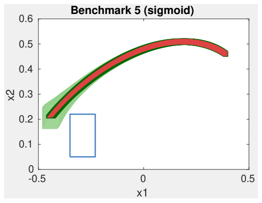
(o) ex4-tanh



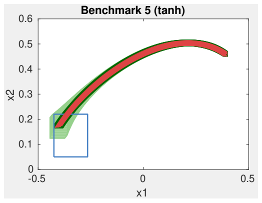
(p) ex4-relu-tanh



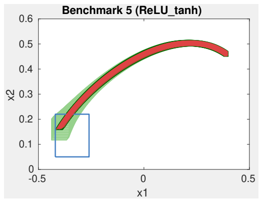
(q) ex5-relu



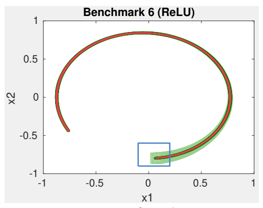
(r) ex5-sigmoid



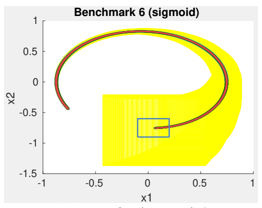
(s) ex5-tanh



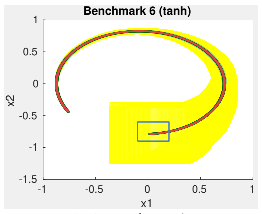
(t) ex5-relu-tanh



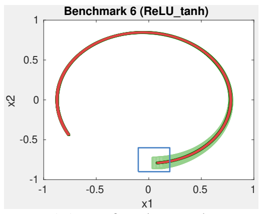
(u) ex6-relu



(v) ex6-sigmoid



(w) ex6-tanh



(x) ex6-relu-tanh

Figure 5: Results of Benchmarks. We can see that except for ex2-sigmoid, POLAR produces the tightest reachable set estimation (dark green sets) and successfully proves or disproves the reachability property for all the examples. This is in comparison with other state-of-the-art tools including ReachNN* [14, 15] (light green sets), Sherlock [13] (blue sets), Verisig 2.0 [11] (grey sets), and NNV [16] (yellow sets).

Table 1: V: number of state variables, σ: activation functions, M: number of hidden layers, n: number of neurons in each hidden layer. For each approach (POLAR, ReachNN*, Sherlock, Verisig 2.0), we give the runtime in seconds if it successfully verifies the property. 'Unknown': the property could not be verified. '–': the approach cannot be applied due to the type of σ.

#	V	NN Controller			**POLAR**	ReachNN* [15]	Sherlock [13]	Verisig 2.0 [12]
		σ	M	n				
1	2	ReLU	3	20	**22**	26	42	–
		sigmoid	3	20	**20**	75	–	47
		tanh	3	20	**18**	Unknown	–	46
		ReLU+tanh	3	20	**11**	71	–	–
2	2	ReLU	3	20	**2**	5	3	–
		sigmoid	3	20	Unknown	13	–	**7**
		tanh	3	20	**3**	73	–	Unknown
		ReLU+tanh	3	20	**2**	8	–	–
3	2	ReLU	3	20	**13**	94	143	–
		sigmoid	3	20	**24**	146	–	44
		tanh	3	20	**22**	137	–	38
		ReLU+sigmoid	3	20	**14**	Unknown	–	–
4	3	ReLU	3	20	**1**	8	21	–
		sigmoid	3	20	**3**	22	–	11
		tanh	3	20	**3**	21	–	10
		ReLU+tanh	3	20	**2**	12	–	–
5	3	ReLU	4	100	**7**	103	15	–
		sigmoid	4	100	**15**	27	–	190
		tanh	4	100	**16**	Unknown	–	179
		ReLU+tanh	4	100	**6**	Unknown	–	–
6	4	ReLU	4	20	**4**	1130	35	–
		sigmoid	4	20	**6**	13350	–	83
		tanh	4	20	**6**	2416	–	70
		ReLU+tanh	4	20	**4**	1413	–	–

5 Conclusion

In this paper, we propose POLAR, a polynomial arithmetic framework, which integrates TM flowpipe construction, Bernstein overapproximation, and symbolic remainder method to efficiently compute reachable set overapproixmations for NNCS. Empirical comparison over a suite of benchmarks show that POLAR performs significantly better than state-of-the-art techniques with respect to both computation efficiency and reachable set estimation accuracy. Future work includes the automatic design of the hyper-parameters in POLAR to further improve performance and accessibility.

References

[1] 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. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 1

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR'16 (Poster)*, 2016. 1

[3] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A. Theodorou, and Byron Boots. Agile autonomous driving using end-to-end deep imitation learning. In *Proc. of RSS'18*, 2018. 1

[4] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Proc. of CAV'17*, volume 10426 of *LNCS*, pages 3–29. Springer, 2017. 1

[5] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proc. of CAV'17*, volume 10426 of *LNCS*, pages 97–117. Springer, 2017. 1

[6] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *Proc. of NFM'18*, volume 10811 of *LNCS*, pages 121–138. Springer, 2018. 1

[7] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *Proc. of USENIX Security (USENIX)*, pages 1599–1614, 2018. 1

[8] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. Beyond the single neuron convex barrier for neural network certification. In *Proc. of NeurIPS'19*, pages 15072–15083, 2019. 1

[9] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proc. of HSCC'18*, pages 169–178. ACM, 2019. 2

[10] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pages 258–263, 2013. 2

[11] Radoslav Ivanov, Taylor J. Carpenter, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verifying the safety of autonomous systems with neural network controllers. *ACM Trans. Embed. Comput. Syst.*, 20(1):7:1–7:26, 2021. 2, 9, 10

[12] Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *Proc. of CAV'21*, To Appear. 2, 8, 9, 11

[13] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proc. of HSCC'19*, pages 157–168. ACM, 2019. 2, 9, 10, 11

[14] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Trans. Embed. Comput. Syst.*, 18(5s): 106:1–106:22, 2019. 2, 4, 6, 8, 9, 10

[15] Jiameng Fan, Chao Huang, Xin Chen, Wenchao Li, and Qi Zhu. ReachNN*: A tool for reachability analysis of neural-network controlled systems. In *Proc. of ATVA'20*, volume 12302 of *LNCS*, pages 537–542. Springer, 2020. 2, 9, 10, 11

[16] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Proc. of CAV'20*, volume 12224 of *LNCS*, pages 3–17. Springer, 2020. 2, 9, 10

[17] James D. Meiss. *Differential Dynamical Systems*. SIAM publishers, 2007. 2

[18] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical computer science*, 126 (2):183–235, 1994. 3

[19] Martin Berz and Kyoko Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable computing*, 4:361–369, 1998. 3, 5

[20] Markus Neher, Kenneth R. Jackson, and Nedialko S. Nedialkov. On Taylor model based integration of ODEs. *SIAM Journal on Numerical Analysis*, 45:236–262, 2006. 3

[21] Martin Berz and Kyoko Makino. Rigorous global search using taylor models. In *Proc. of SNC'09*, pages 11–20. ACM, 2009. 3

[22] Kyoko Makino and Martin Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003. 4

[23] Xin Chen. *Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models*. PhD thesis, RWTH Aachen University, 2015. 4, 5

[24] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Proc. of RTSS'12*, pages 183–192, 2012. 4, 5

[25] George G. Lorentz. *Bernstein Polynomials*. American Mathematical Society, 2013. 5

[26] George M. Phillips. *Interpolation and Approximation by Polynomials*. Springer Science & Business Media, 2003. 6

[27] Xin Chen and Sriram Sankaranarayanan. Decomposed reachability analysis for nonlinear systems. In *Proc. of RTSS'16*, pages 13–24, 2016. 7

[28] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Éric Walter. Interval analysis. In *Applied Interval Analysis*. Springer, 2001. 7

[29] Stephen Prajna, Pablo A Parrilo, and Anders Rantzer. Nonlinear control synthesis by convex optimization. *IEEE Transactions on Automatic Control*, 49(2):310–314, 2004. 8