Verisig 2.0: Verification of Neural Network Controllers Using Taylor Model Preconditioning

Radoslav Ivanov(✉), Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee

University of Pennsylvania, Philadelphia, PA 19104, USA
{rivanov,carptj,weimerj,alur, pappasg,lee}@seas.upenn.edu

Abstract. This paper presents Verisig 2.0, a verification tool for closed-loop systems with neural network (NN) controllers. We focus on NNs with tanh/sigmoid activations and develop a Taylor-model-based reachability algorithm through Taylor model preconditioning and shrink wrapping. Furthermore, we provide a parallelized implementation that allows Verisig 2.0 to efficiently handle larger NNs than existing tools can. We provide an extensive evaluation over 10 benchmarks and compare Verisig 2.0 against three state-of-the-art verification tools. We show that Verisig 2.0 is both more accurate and faster, achieving speed-ups of up to 21x and 268x against different tools, respectively.

1 Introduction

Following their increasing popularity, neural networks (NNs) have been recently introduced to various new domains, including safety-critical systems such as autonomous cars [4] and airborne collision avoidance systems [21]. At the same time, NNs have been shown to be greatly susceptible to input perturbations: minor input changes can cause a NN's outputs to vary drastically, as is the case with adversarial examples [26]. Such issues have emphasized the need to formally analyze NN-based systems and assure their safety before they are deployed.

A number of formal verification approaches have been proposed in the last few years to analyze closed-loop systems with NN components. On the one hand, several techniques have been developed for reachability analysis. These works

This work was supported by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-18-C-0090, and by the Army Research Office (ARO) under Grant Number W911NF-20-1-0080, and by the Office of Naval Research (ONR) award N00014-20-1-2115. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL, ARO, DARPA, ONR, or the Department of Defense, or the United States Government.

A. Silva and K. R. M. Leino (Eds.) CAV 2021, LNCS 12759, pp. 249–262, 2021.
https://doi.org/10.1007/978-3-030-81685-8_11

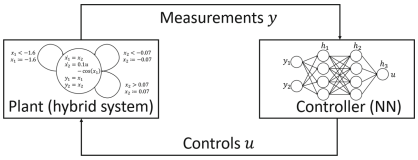


Fig. 1. Overview of the closed-loop system considered in this paper.

handle the NN reachability problem in a variety of ways: by converting the NN into a hybrid system [19]; by casting the problem into a satisfiability modulo convexity problem [25]; by approximating the NN with a Taylor model [8,11,16,20]; or by propagating NN reachable sets using star sets [27,28]. Multiple falsification techniques have been developed as well: these approaches work by adapting existing hybrid-system falsifiers [2,6] to the NN case [7,29,33]; methods for systematic testing through scenario specification languages have been proposed as well [14]. Finally, a number of techniques have been developed to analyze properties of the NN in isolation (e.g., input-output properties) [9,10,12,15,22,30–32], though it is challenging to use these tools in a closed-loop setting as it is unclear what NN specification ensures closed-loop safety in general.

While existing reachability techniques have shown impressive performance, scalability remains an obstacle to applying these tools to realistic systems. In particular, these methods have been evaluated mostly on low-dimensional systems, i.e., systems with several states and at most 41 measurements [18]. The main scalability challenge stems from the fact that reachability is undecidable even for linear hybrid systems [1]. Thus, all approaches overapproximate the true reachable sets using a computationally convenient representation such as polytopes [13] or Taylor models [5]. At the same time, this overapproximation, known as the wrapping effect, leads to quick error accumulation over time, thus making it challenging to verify complex specifications over a longer time horizon.

To address these limitations, we present Verisig 2.0, a scalable tool for verifying safety properties of closed-loop systems with NN controllers. We combine ideas from NN reachability with ideas from hybrid system verification. In particular, we adopt the idea of approximating NNs with Taylor models (TMs) [11,16,20], and we alleviate the wrapping effect through TM preconditioning and shrink wrapping [3,23,24]. Finally, we note that the NN reachability computation can be parallelized since each neuron in a layer can be analyzed independently. We have implemented our tool in conjunction with the hybrid system tool Flow* [5], which enables us to handle general hybrid system models with NN components.

We compare Verisig 2.0 against three tools, namely Verisig [20], NNV [28], and ReachNN* [11]. We use 10 benchmarks that illustrate various challenges, such as hybrid models, non-linear systems and systems with high-dimensional observations. The results indicate that Verisig 2.0 is significantly faster (achieving speed-ups of up to 21x and 268x against Verisig and ReachNN*, respectively) and produces tighter reachable set approximations on all benchmarks.

In summary, this paper has three contributions: 1) a Taylor-model-based NN reachability method using TM preconditioning and shrink wrapping; 2) an efficient implementation that allows for parallel execution; 3) an extensive comparison against existing tools on 10 diverse benchmarks. The source code to reproduce the results is available online (github.com/rivapp/CAV21_repeatability_package) as well as in the main Verisig repository (github.com/Verisig/verisig).

2 Problem Statement

This section outlines the reachability problem addressed in this paper. We consider a closed-loop system, illustrated in Fig. 1, consisting of: a) a plant with states x modeled as a hybrid system; b) measurements y produced as a function of x; c) an NN controller h that takes y as input and produces controls u.

Plant Model. We assume the plant is modeled as a standard hybrid system. In particular, the state space $X = X_D \times X_C$ consists of continuous variables X_C and discrete locations $X_D = \{q_1, \ldots, q_m\}$. When in location $q \in X_D$, the system evolves according to differential equations f_q, i.e., $\dot{x} = f_q(x, u)$, where $x \in X_C$. Each location $q \in X_D$ has an associated invariant $I(q) \subseteq X_C$ that must hold true in that location. Transitions between locations are enabled by guards, which are boolean predicates on the continuous variables. Finally, each continuous variable may be reset to a new value when transitioning to a new location.

Observation Model. The system produces observations $y = g(x)$, where $g : X \to \mathbb{R}^p$. Note that some benchmarks in this paper use state feedback only, i.e., $y = x$.

Controller. The controller h is a fully-connected feedforward NN with sigmoid/tanh activations. Formally, h can be represented as a composition of its L layers:

$$h(y) = h_L \circ h_{L-1} \circ \cdots \circ h_1(y), \tag{1}$$

where each $h_i(y) = a(W_i y + b_i)$ performs a linear function, with parameters W_i and b_i identified during training, followed by a sigmoid/tanh activation a.

Composed System. Although the hybrid system formulation places no restrictions on the controller/plant composition, in the interest of clarity we assume the controller is executed in a time-triggered fashion, with sampling period T, as follows: $u(t) = h(y(t_k))$, for $t \in [t_k, t_k + T)$, where $t_k = kT$ and $k = 0, 1, 2, \ldots$

Closed-Loop Reachability Problem. Let S be a composed system. Given an initial set of states $x(0) \in X_0$, the reachability problem, expressed as property ϕ, is to verify a property ψ of the reachable states of S:

$$\phi(X_0) \equiv (x(0) \in X_0) \Rightarrow \psi(x(t)), \ \forall t \geq 0. \tag{2}$$

3 Background: Neural Networks as Taylor Models

As described in Sect. 1, in this work we adopt a TM-based approach for propagating NN reachable sets. There are two main reasons for this: 1) TMs can approximate any differentiable function over a bounded range given a high enough order; 2) TMs are very effective at approximating hybrid system reachable sets, which allows for a smooth composition between the NN and the rest of the system. The rest of this section formalizes TMs and summarizes the existing approaches to using TMs for NN reachability.

Taylor Model Definition. Intuitively, a TM of a function f is a polynomial approximation p, together with a worst-case error bound I. A j-degree polynomial approximation p of a j times continuously differentiable function f around a point x, written $p(x) \equiv_j f(x)$, is a polynomial p of degree j such that all partial derivatives of f and p coincide at x. Let $\mathbb{I}$ be the set of all intervals $I = [a, b]$ and let $f : D \to \mathbb{R}$ be a function of n variables defined over a domain $D \in \mathbb{I}^n$. Then a Taylor model of f over D of degree j is a pair (p, I) of a polynomial approximation p and an error bound I (also known as a remainder) such that:

$$1) f(c) \equiv_j p(c), \text{where } c \text{ is the center of } D,$$
$$2) \forall x \in D,\ f(x) \in \{p(x) + e \mid e \in I\}.$$

Taylor Model Arithmetic. Let $TM_1 = (p_1, I_1)$ and $TM_2 = (p_2, I_2)$ be two TMs defined over a domain D. Addition and multiplication are defined as follows [5]:

$$TM_1 + TM_2 = (p_1 + p_2, I_1 + I_2)$$
$$TM_1 \times TM_2 = (p_1 \times p_2, \texttt{Int}(p_1)I_2 + \texttt{Int}(p_2)I_1 + I_1 \times I_2),$$

where $\texttt{Int}(p)$ is an interval bound of p over D.

TMs have shown impressive performance in hybrid system reachability problems due to their ability to approximate any continuously differentiable function given a high enough order [5]. Another appealing feature is that TMs can be used to approximate solutions of differential equations through Picard iteration [5]. Thus, it is natural to try to use TMs to approximate NN reachable sets as well.

Two classes of approaches for approximating NNs with TMs have been developed in the literature. The first one is sampling-based: given a TM TM_y of the inputs y to h, these methods sample points Z from TM_y and corresponding outputs $h(Z)$ to perform polynomial regression [8] or approximation [16]. While these approaches work well for systems with several state variables, they cannot handle higher-dimensional problems due to insufficient sampling.

A second approach to using TMs for NN reachability is to propagate the TMs through each neuron in the NN. Specifically, let $TM_y = (p, I)$ be the TM for y and consider a neuron ν that computes the function $\sigma(wy + b)$, where σ denotes the sigmoid. One can use TM arithmetic [5] to obtain $TM_L = (wp + b, wI)$ for the linear map in ν. For the sigmoid TM, TM_σ, one could obtain a Taylor series expansion of σ around the center of TM_L and get remainder

bounds using Taylor's theorem [20]. Thus, the final TM for ν is $TM_\nu = TM_\sigma \circ TM_L$. The benefit of propagating TMs in this fashion is that no sampling is necessary since the NN is approximated directly. On the other hand, scalability challenges manifest in a different way, namely the TM remainders may grow quickly depending on the NN architecture (explained in more detail in Sect. 4).

We adopt the latter approach to approximating NN as TMs due to its improved scalability. The next section describes our approach to reducing the TM remainder size through TM preconditioning and shrink wrapping.

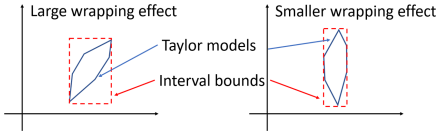


Fig. 2. The wrapping effect for different taylor model orientations.

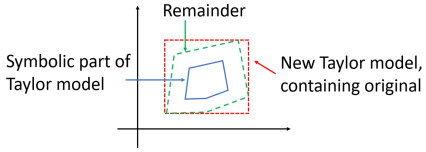


Fig. 3. Illustration of the shrink wrapping method.

4 Taylor Model Preconditioning and Shrink Wrapping

This section presents our approach to limiting the remainder growth as TMs are propagated through the NN. We explore two complementary techniques, namely TM preconditioning and shrink wrapping. Both of these ideas were originally developed for the purpose of reachability analysis of hybrid systems [3,23] – in this paper, we adapt them to the NN case.

4.1 Taylor Model Preconditioning

As noted in Sect. 3, although propagating the TM through the NN is preferred since it captures the functional representation of each neuron, it may suffer from quick remainder growth. The following example illustrates this process.

Example 1. Let y_1 and y_2 be inputs to the NN h with corresponding TMs $TM_{y_1} = (p_1, I_1)$ and $TM_{y_2} = (p_2, I_2)$ over domain $D \in \mathbb{I}^n$. Let ν be a neuron in the first layer implementing the function $\nu(y_1, y_2) = \sigma(w_1 y_1 + w_2 y_2 + b)$. The TM for the linear part of ν is

$$TM_L := (p_L, I_L) = (w_1 p_1 + w_2 p_2 + b, w_1 I_1 + w_2 I_2).$$

Let $TM_\sigma = \sigma(a) + \sigma'(a)(TM_L - a) + \sigma''(a)(TM_L - a)^2/2 + I_\sigma$ be a second-order Taylor series expansion of the sigmoid around point a, with remainder I_σ. Using TM arithmetic [5], the TM for ν is $TM_\nu = (p_\nu, I_\nu)$, where

$$p_\nu = \sigma''(a)p_L^2 + (\sigma'(a) - a\sigma''(a))p_L - (\sigma'(a) - 0.5a\sigma''(a))a + \sigma(a)$$
$$I_\nu = \sigma''(a)(2\texttt{Int}(p_L)I_L + I_L^2) + (\sigma'(a) - a\sigma''(a))I_L + I_\sigma.$$

Remark 1. In order to compute a $TM_\sigma = (p_\sigma, I_\sigma)$ for the sigmoid/tanh, one can follow the procedure described in prior work [20]. In summary, the following three steps need to be performed, assuming the input TM is denoted by TM_L:

1. compute interval bounds, $[a, b]$, for TM_L using interval analysis;
2. obtain a Taylor series approximation, p_σ, of the sigmoid/tanh around the midpoint of $[a, b]$;
3. obtain worst-case error bounds, I_σ, for p_σ using Taylor's theorem.

As shown in Example 1, the remainder is propagated using interval analysis, where a major contributor is the $\texttt{Int}(p_L)$ term, i.e., the interval bounds of p_L. Since this term approximates the (potentially high-dimensional) input TM with a box, it may introduce significant wrapping effect if the input TM is not a box, as illustrated in Fig. 2. The natural way to address this wrapping effect is through rotating the TM in order to align it with the axes [23,24].

Algorithm 1. NN Verification Using Taylor Model Preconditioning

Input: Measurement TM Vector TMV_y, NN h with L layers, and sigmoid activations.

$TMV_0 \leftarrow TMV_y$
for each i in $\{1, \ldots, L\}$ **do**
  $TMV_i^L \leftarrow W_i * TMV_{i-1} + b_i$
  $(Q + c, \mathbf{0}) \circ (R + Q^\top N, Q^\top \mathbf{I}) \leftarrow TaylorModelPreconditioning(TMV_i^L)$
  $TMV_i^\nu \leftarrow TaylorModelForSigmoid((Q, \mathbf{0}))$ //Taylor series approximation
  $TMV_i \leftarrow TMV_i^\nu \circ (R + Q^\top (c + N), \mathbf{I})$
end for
return TMV_L

Since the set represented by a TM is the image of a polynomial over a given domain, it is challenging to choose an appropriate rotation matrix. However, as discussed in prior work [23,24], if one first normalizes the TM so that the domain is $[-1, 1]^n$, then the linear terms become the largest contributors to interval analysis overapproximation (since higher order terms are less than 1 in magnitude). Thus, a good choice for a rotation matrix is the matrix formed by the linear terms of the (normalized) TM.

To formalize the above concept, let us decompose a TM vector $TMV = (\mathbf{p}, \mathbf{I})$ into $TMV = (c + M + N, \mathbf{I})$, where c denotes the constant terms, M denotes the linear terms and N denotes the higher-order terms. The idea of preconditioning is to decompose $M = QR$, where Q is an orthonormal matrix and R is upper-triangular. This is achieved by splitting TMV into a composition of two TM vectors: $TMV = (Q + c, \mathbf{0}) \circ (R + Q^\top N, Q^\top \mathbf{I})$.[1] Then, each neuron's computation is performed on Q only, which alleviates the wrapping effect introduced by $\texttt{Int}(p_L)$ in Example 1 since Q is orthonormal.

[1] Note that the new remainder may need to be enlarged to also include numerical errors due to the computation of Q.

The algorithm is presented in Algorithm 1. Note that preconditioning is performed in each layer, followed by again composing the two parts into the full TM. While it is possible to represent the final TM as a composition of individual layer TMs, the benefits of preconditioning would decrease after the first layer, since most of the variability is captured in the right-most TM.

4.2 Shrink Wrapping

In systems where verification over a longer time horizon is required, avoiding large remainders may be impossible even with effective preconditioning. In such cases, one could use shrink wrapping in order to refactor the TM into one that results in slower remainder accumulation in the future [3,24].

The high-level idea of shrink wrapping is illustrated in Fig. 3. If the remainder becomes a significant part of the set described by the TM, then TM arithmetic degrades into standard interval analysis. In this case, it helps to transform the TM into a new TM that contains the original one but has no remainder. Thus, even if the new TM is slightly larger, it is propagated symbolically using TM arithmetic, which results in smaller error accumulation in the long run.

The choice of new TM is not obvious and is affected by the system in consideration. The standard approach in related work [3,24] is to focus on the linear terms (assuming the TM is normalized so that $D = [-1, 1]^n$). Specifically, suppose that the system's state x is described by the TM vector $TMV_x = (\mathbf{p}, \mathbf{I}) = (c + M + N, \mathbf{I})$. One option for the new TM vector is to premultiply TMV_x by M^{-1}, thereby reducing the linear terms to the identity matrix, $\mathcal{I}$. Then a shrink wrap factor q is chosen such that the image of the higher-order terms contains the remainder of the initial TM vector, i.e., $TMV_x^{new} = (c + \mathcal{I} + qM^{-1}N, \mathbf{0})$.[2]

While it is possible to choose q by finding bounds on the partial derivatives of the higher-order terms $M^{-1}N$ [3], our initial experiments indicated that a more straightforward approach leads to no loss in precision. In particular, we represent the new TM vector as $TMV_x^{new} = (c + \text{diag}(\mathbf{q}), \mathbf{0})$, where $\mathbf{q} = \texttt{Int}(TMV_x)$. The last consideration is when to perform the TM conversion: if it is applied too often, more error could be introduced by the frequent elimination of useful information in the TMs. In our experiments, shrink wrapping is triggered when the remainder is larger than $1e^{-6}$ and larger than 1% of the total TM range.

5 Implementation

We implemented our approach in conjunction with the Flow* tool [5], for easy integration with standard hybrid system models. We provide similar TM functions to the ones existing in Flow*, adapted to the case of NNs. In addition to modified data structures, a main difference in our implementation is the option to parallelize the TM vector propagation, i.e., Line 5 in Algorithm 1. This parallelization is possible since each neuron in a layer only depends on the input

[2] The new remainder may be greater than 0 due to round-off error during the inversion.

TMs, thus each computation can be done on a separate core. As illustrated in Sect. 7, this implementation brings great benefits, especially in the case of larger NNs, where multiple neuron computations can be performed in parallel.

6 Benchmarks

We use 10 benchmarks to evaluate the proposed approach. These benchmarks were compiled from the related literature [17,19,20,28] and were selected in order to cover a wide variety of systems and controllers: 1) continuous and hybrid systems; 2) systems with state feedback and systems with measurements as a function of the states; 3) low-dimensional systems as well as systems with high-dimensional measurements; 4) controllers with both tanh and sigmoid activations and with a number of neurons varying from 16 to 200 per layer.

Table 1 presents the dynamics and the initial set for each benchmark. For simplicity, all properties are reachability properties (i.e., the problem is to verify whether a goal set is reached from all initial states), though safety properties can be handled as well. In particular, the goal regions are as follows:

– $B_1 : x_1 \in [0, 0.2], x_2 \in [0.05, 0.3]$; $B_2 : x_1 \in [-0.3, 0.1], x_2 \in [-0.35, 0.5]$;

Table 1. List of benchmarks. Benchmarks $B_1 - B_5$ and Tora were introduced by Huang et al. [17]; adaptive cruise control (ACC) was presented by Tran et al. [28]; mountain car (MC), quadrotor with model-predictive control (QMPC) and F1/10 were introduced by Ivanov et al. [20]. We use V to denote the measurement dimension. In F1/10, y is a 21-dimensional LiDAR scan.

Name	Dynamics	V	Initial set
B_1	$\dot{x_1} = x_2, \dot{x_2} = ux_2^2 - x_1$	2	$x_1 \in [0.8, 0.9], x_2 \in [0.5, 0.6]$
B_2	$\dot{x_1} = x_2 - x_1^3, \dot{x_2} = u$	2	$x_1 \in [0.7, 0.9], x_2 \in [0.7, 0.9]$
B_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	$x_1 \in [0.8, 0.9]$, $x_2 \in [0.4, 0.5]$
B_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	$x_1, x_3 \in [0.25, 0.27]$, $x_2 \in [0.08, 0.1]$
B_5	$\dot{x_1} = x_1^3 - x_2$, $\dot{x_2} = x_3, \dot{x_3} = u$	3	$x_1 \in [0.38, 0.4], x_2 \in [0.45, 0.47]$ $x_3 \in [0.25, 0.27]$,
Tora	$\dot{x_1} = x_2$, $\dot{x_2} = -x_1 + 0.1\sin(x_3)$, $\dot{x_3} = x_4$, $\dot{x_4} = u$	4	$x_1 \in [-0.77, -0.75]$, $x_2 \in [-0.45, -0.43]$, $x_3 \in [0.51, 0.54]$, $x_4 \in [-0.3, -0.28]$
ACC	$\dot{x_1} = x_2, \dot{x_2} = x_3, \dot{x_3} = -4 - 2x_3 - \frac{x_2^2}{1000}$ $\dot{x_4} = x_5, \dot{x_5} = x_6, \dot{x_6} = 2u - 2x_6 - \frac{x_5^2}{1000}$	5	$x_1 \in [90, 91], x_2 \in [32, 32.05]$ $x_4 \in [10, 11], x_5 \in [30, 30.05]$
MC	$x_1^+ = x_1 + x_2$, $x_2^+ = x_2 + 0.0015u - 0.0025\cos(3x_1)$	2	$x_1 \in [-0.53, -0.5]$
QMPC	$\dot{x_1} = x_4 - 0.25, \dot{x_2} = x_5 + 0.25, \dot{x_3} = x_6$ $\dot{x_4} = 9.81u_1, \dot{x_5} = -9.81u_2, \dot{x_6} = u_3 - 9.81$	6	$x_1 \in [0.025, 0.05]$, $x_2 \in [0, 0.025]$
F1/10	$\dot{x_1} = x_3\cos(x_4), \dot{x_2} = x_3\sin(x_4)$ $\dot{x_3} = -1.633x_3 + 0.3266(u - 4), \dot{x_4} = \frac{x_3\tan(u)}{0.225}$	21	$x_1 \in [-0.0025, 0.0025]$, $x_3 \in [-0.0025, 0.0025]$

- $B_3 : x_1 \in [0.2, 0.3], x_2 \in [-0.3, -0.05]$; $B_4 : x_1 \in [-0.05, 0.05], x_2 \in [-0.05, 0]$;
- $B_5(\text{sig}) : x_1 \in [-0.4, -0.28], x_2 \in [0.05, 0.22]$;
- $B_5(\text{tanh}) : x_1 \in [-0.43, -0.38], x_2 \in [0.16, 0.18]$;
- Tora: $x_1 \in [-0.1, 0.2], x_2 \in [-0.9, -0.6]$;
- ACC: $x_1 \in [22.81, 22.87], x_4 \in [29.88, 30.02]$;
- MC: $x_1 \geq 0.45$; QMPC: $x_1, x_2, x_3 \in [-0.32, 0.32]$; F1/10: no crash [18].

7 Experiments

We compare our tool, named Verisig 2.0, against three state-of-the-art tools, namely Verisig [19,20], ReachNN* [11,17], and NNV [27,28]. We selected these tools because they handle NNs with sigmoid/tanh activations. For each benchmark, we record whether each tool could verify the property (or return Unknown due to large approximation error). In addition, we compare the verification times between the different tools. While Verisig and NNV do not support parallel execution,[3] ReachNN* has been optimized for GPU execution, so a comparison in terms of verification times is fair (all experiments were run on an Intel Xeon Gold 6248 running at 2.5 GHz and with an Nvidia GeForce RTX 2080 Ti GPU). Finally, we provide a comparison in terms of reachable sets.

Verification outcomes and times are reported in Table 2. Multiple controllers were used in some benchmarks in order to test a variety of NNs. We present the

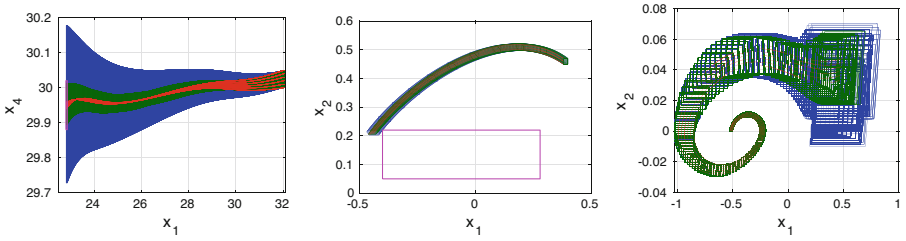
Table 2. Verification evaluation. The notation *tanh/sig ($n \times k$)* indicates a NN with tanh/sig activations, n hidden layers and k neurons per layer. For each tool, we provide the verification time in seconds; if a property could not be verified, it is marked as Unknown. If a tool crashed on a benchmark, it is marked as DNF.

Name	NN setup	Verisig 2.0 (40 cores)	Verisig 2.0 (1 core)	Verisig	ReachNN*	NNV
B_1	tanh (2×20)	**38 s**	48 s	DNF	Unknown	Unknown
	sig (2×20)	**40 s**	49 s	Unknown	69 s	Unknown
B_2	tanh (2×20)	Unknown	Unknown	Unknown	Unknown	Unknown
	sig (2×20)	**6 s**	8 s	12 s	32 s	Unknown
B_3	tanh (2×20)	**32s**	43 s	98 s	128 s	Unknown
	sig (2×20)	**36 s**	47 s	98 s	130 s	Unknown
B_4	tanh (2×20)	**9 s**	11 s	23 s	20 s	DNF
	sig (2×20)	**10 s**	12 s	24 s	20 s	DNF
B_5	tanh (3×100)	**48 s**	168 s	Unknown	Unknown	Unknown
	sig (3×100)	51s	196 s	1063 s	**31 s**	Unknown
Tora	tanh (3×20)	**43 s**	70 s	134 s	2524 s	Unknown
	sig (3×20)	**50 s**	83 s	136 s	3402 s	Unknown
ACC	tanh (3×20)	**529 s**	1512 s	Unknown	DNF	Unknown
MC	sig (2×16)	48 s	52 s	**33 s**	N/A	N/A
	sig (2×200)	**1241 s**	4311 s	Unknown	N/A	N/A
QMPC	tanh (2×20)	**636 s**	697 s	703 s	N/A	N/A
F1/10	tanh (2×64)	**3411 s**	3654 s	2021 s	N/A	N/A

[3] NNV is parallelized in the case of ReLU activations, but not for smooth activations.

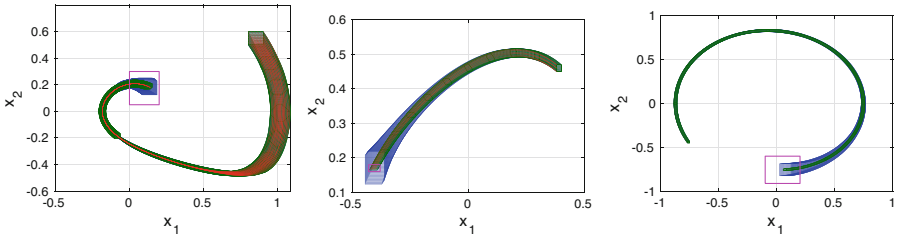
results for Verisig 2.0 as used with one and with 40 cores, in order to illustrate the benefit of parallelization. Note that parallelization helps the most in systems with wider NNs, e.g., the MC benchmark, since a larger part of the computation is devoted to NN computation (relative to plant computation) in these systems.

Comparison with Verisig. Verisig is the closest method to Verisig 2.0, as it also propagates TMs through the NN. Thus, Verisig can be seen as a baseline for our approach, so this comparison illustrates most clearly the benefits of preconditioning and shrink wrapping. Firstly, note that Verisig takes significantly more time to compute reachable sets (21 times slower in the case of the B_5 benchmark) on all but one benchmark – the MC benchmark is peculiar because the NN is very small, hence most of the computation is spent on the plant. Furthermore, Verisig is unable to verify some properties due to increasing error. As shown in Fig. 4, the reachable sets computed by Verisig introduce more approximation error, especially in the challenging ACC benchmark, where preconditioning is particularly useful due to the larger input space.



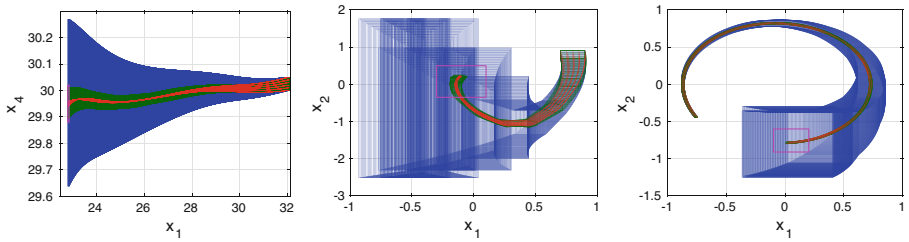
(a) ACC benchmark. (b) B_5 benchmark, sigmoid. (c) MC benchmark, 2×200.

Fig. 4. Comparison between the reachable sets produced by Verisig (blue) and Verisig 2.0 (green). Example simulated trajectories are plotted in red. The goal set is shown in magenta. Note that the goal is not reached in the B_5 benchmark. (Color figure online)



(a) B_1 benchmark, sigmoid. (b) B_5 benchmark, tanh. (c) Tora benchmark, sigmoid.

Fig. 5. Comparison between the reachable sets produced by ReachNN* (blue) and Verisig 2.0 (green). Simulated trajectories are plotted in red (not shown in the Tora benchmark to improve visibility). The goal set is shown in magenta. (Color figure online)



(a) ACC benchmark. (b) B_2 benchmark, sigmoid. (c) Tora benchmark, tanh.

Fig. 6. Comparison between the reachable sets produced by NNV (blue) and the Verisig 2.0 approach (green) on three of the benchmarks from Table 2. Example simulated trajectories are plotted in red. The goal set is shown in magenta. (Color figure online)

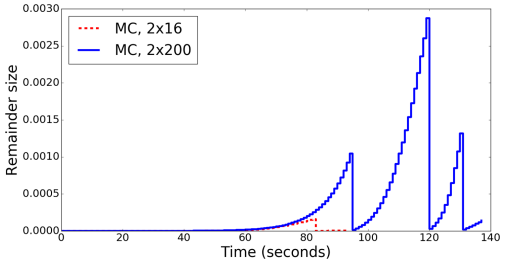


Fig. 7. Verisig 2.0 remainder growth (for position, x_1) on the MC benchmark as we increase the NN size. The remainder is reset to 0 after shrink wrapping.

Comparison with ReachNN.* ReachNN* is a sampling-based approach to NN verification, so it is expected to work well on low-dimensional systems and encounter difficulties as the dimension increases. As can be seen in Table 2, Verisig 2.0 is faster on all but one benchmark, and the difference is especially pronounced on the four-dimensional Tora benchmark, where ReachNN* is 268 times slower. Note that ReachNN* cannot handle hybrid models, so no comparison could be made on those benchmarks. Finally, as shown in Fig. 5, Verisig 2.0 also results in tighter reachable sets – the benefit of shrink wrapping can be observed in Fig. 5a, where the ReachNN* reachable sets eventually start to grow fast whereas Verisig 2.0 is able to maintain low approximation error over time.

Comparison with NNV. Note that NNV is unable to verify any of the properties considered in this paper due to high approximation error. This is mostly due to the fact that NNV is optimized for networks with ReLU activations, where the star set method used in NNV is effective and parallelizable. Figure 6 shows the reachable computed by each tool, where it is clear that Verisig 2.0 maintains tight reachable sets whereas the NNV approximation error grows quickly.

Scalability Evaluation. Finally, we also evaluate the scalability of Verisig 2.0 as we increase the NN size on the MC benchmark. Figure 7 illustrates how the remainder grows over time for the x_1 (position) state. We observe that the larger NN results in significantly larger remainder growth. At the same time, interpreting the remainder growth in isolation may be misleading since it also depends on the size and shape of the true reachable sets. We leave a rigorous analysis of the effect of NN size on scalability for future work.

8 Conclusion

This paper presented Verisig 2.0, a parallelized tool for NN verification. We developed a Taylor-model-based approach in which we reduce the approximation error in reachable sets through Taylor model preconditioning and shrink wrapping. Finally, we provided an extensive evaluation over 10 benchmarks and showed that our method is significantly more accurate and faster than state-of-the-art tools, resulting in 21x and 268x speed-ups on some benchmarks, respectively. For future work, we will investigate which NN architectures are more amenable for verification, both in terms of size and number of layers as well as in terms of weight magnitude and direction.

References

1. Alur, R., et al.: The algorithmic analysis of hybrid systems. Theoret. Comput. Sci. **138**(1), 3–34 (1995)
2. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
3. Berz, M., Makino, K.: Suppression of the wrapping effect by taylor model-based verified integrators: long-term stabilization by shrink wrapping. Int. J. Diff. Eq. Appl **10**, 385–403 (2005)
4. Bojarski, M., Del Testa, D., Dworakowski, D., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
5. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
6. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: International Conference on Computer Aided Verification (2010)
7. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 357–372. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57288-8_26
8. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: 22nd International Conference on Hybrid Systems: Computation and Control, pp. 157–168 (2019)

9. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Dutle, A., Muñoz, C., Narkawicz, A. (eds.) NFM 2018. LNCS, vol. 10811, pp. 121–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77935-5_9
10. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
11. Fan, J., Huang, C., Chen, X., Li, W., Zhu, Q.: ReachNN*: a tool for reachability analysis of neural-network controlled systems. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 537–542. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_30
12. Fazlyab, M., Robey, A., Hassani, H., Morari, M., Pappas, G.J.: Efficient and accurate estimation of lipschitz constants for deep neural networks. arXiv preprint arXiv:1906.04893 (2019)
13. Frehse, G., et al.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
14. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: Conference on Programming Language Design and Implementation (2019)
15. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP) (2018)
16. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: reachability analysis of neural-network controlled systems. ACM Trans. Embed. Comput. Syst. (TECS) **18**(5s), 1–22 (2019)
17. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: International Conference on Computer Aided Verification (2017)
18. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Case study: verifying the safety of an autonomous racing car with a neural network controller. In: International Conference on Hybrid Systems: Computation and Control (2020)
19. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: 22nd ACM International Conference on Hybrid Systems: Computation and Control (2019)
20. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verifying the safety of autonomous systems with neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(1) (2020). https://doi.org/10.1145/3419742
21. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th, pp. 1–10. IEEE (2016)
22. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
23. Makino, K., Berz, M.: Suppression of the wrapping effect by taylor model-based verified integrators: Long-term stabilization by preconditioning. Int. J. Differ. Equ. Appl. **10**(4) (2011)
24. Neher, M., Jackson, K.R., Nedialkov, N.S.: On taylor model based integration of odes. SIAM J. Numer. Anal. **45**(1), 236–262 (2007)

25. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 147–156. ACM (2019)
26. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., et al.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
27. Tran, H.-D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using ImageStars. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 18–42. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_2
28. Tran, H., Cai, F., Lopez, D.M., Musau, P., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control. ACM Trans. Embed. Comput. Syst. **18**(5s), 105 (2019)
29. Tuncali, C.E., Fainekos, G., Ito, H., Kapinski, J.: Simulation-based adversarial test generation for autonomous vehicles with machine learning components. arXiv preprint arXiv:1804.06760 (2018)
30. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in Neural Information Processing Systems (2018)
31. Weng, T., et al.: Towards fast computation of certified robustness for relu networks. In: International Conference on Machine Learning, pp. 5273–5282 (2018)
32. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multi-layer neural networks. arXiv preprint arXiv:1708.03322 (2017)
33. Yaghoubi, S., Fainekos, G.: Gray-box adversarial testing for control systems with machine learning components. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 179–184 (2019)