Verifying the Safety of Autonomous Systems with Neural Network Controllers

RADOSLAV IVANOV, TAYLOR J. CARPENTER, JAMES WEIMER, RAJEEV ALUR, GEORGE J. PAPPAS, and INSUP LEE, University of Pennsylvania

This article addresses the problem of verifying the safety of autonomous systems with neural network (NN) controllers. We focus on NNs with sigmoid/tanh activations and use the fact that the sigmoid/tanh is the solution to a quadratic differential equation. This allows us to convert the NN into an equivalent hybrid system and cast the problem as a hybrid system verification problem, which can be solved by existing tools. Furthermore, we improve the scalability of the proposed method by approximating the sigmoid with a Taylor series with worst-case error bounds. Finally, we provide an evaluation over four benchmarks, including comparisons with alternative approaches based on mixed integer linear programming as well as on star sets.

CCS Concepts: • **Software and its engineering** → **Formal methods**; • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Neural networks**;

Additional Key Words and Phrases: Neural network verification, safe autonomy, hybrid systems with neural network controllers

ACM Reference format:
Radoslav Ivanov, Taylor J. Carpenter, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2020. Verifying the Safety of Autonomous Systems with Neural Network Controllers. *ACM Trans. Embed. Comput. Syst.* 20, 1, Article 7 (December 2020), 26 pages.
https://doi.org/10.1145/3419742

1 INTRODUCTION

Following the success of neural networks (NNs) in traditional learning tasks such as image classification [44] and natural language processing [8], learning-enabled components have been introduced to a variety of new domains, including safety-critical systems such as autonomous vehicles [4] and air traffic collision avoidance systems [23]. Assuring the safety of such systems, however, has proven challenging due to the brittle nature of modern NNs. For example, slight

Preliminary versions of some of the results presented in this article appeared in References [21, 22].

This material is based upon work supported by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-18-C-0090. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the AFRL, DARPA, the Department of Defense, or the United States Government. This research is supported in part by grant awards ONR N00014-17-1-2012, ONR N00014-20-1-2115 and ARO MURI W911NF2010080.

Authors' address: R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, University of Pennsylvania, 3330 Walnut Street, Philadelphia, Pennsylvania 19104; emails: {rivanov, carptj, weimerj, alur, pappasg, lee}@seas.upenn.edu.

1539-9087/2020/12-ART7 $15.00
https://doi.org/10.1145/3419742

input perturbations can cause the NN to switch its output to an arbitrary new class, as is the case with adversarial examples [43]. Such issues have highlighted the need to formally verify the safety of NN-based systems for different scenarios and for a range of possible inputs to the NN.

One approach to ensuring the safety of these systems is to analyze the NN in isolation and verify safety properties of the NN's output for given sensitive inputs [12, 13, 15, 18, 24, 49, 50]. Most of these methods work by exploiting the NN structure, e.g., the piecewise linearity of the rectified linear unit (ReLU), and by transforming the verification problem into an optimization problem such as a satisfiability modulo theory (SMT) program [13, 24], mixed integer linear program (MILP) [12], semi-definite program (SDP) [15], or a relaxed linear program [50]. There also exist techniques that approach the problem from a reachability point of view and approximate the reachable NN output sets using interval analysis [49] or zonotopes [18].

Analyzing the NN in isolation does not immediately imply the safety of the entire autonomous system, however. To analyze the whole system, one would need to reason about the interaction between the NN and the physical plant (e.g., a car) and verify that all reachable plant states are safe. Several works have been developed to verify the safety of autonomous systems with NN controllers [11, 20, 22, 42, 47]. These approaches combine ideas from NN verification, e.g., transforming the NN into an MILP or an SMT program [11, 42], with ideas from classical dynamical system reachability [6, 25], e.g., compute reachable sets for the NN output [22, 47].

In our preliminary work, we developed one of the approaches mentioned in the previous paragraph, namely Verisig [22]. Verisig focuses on NNs with smooth activations, e.g., sigmoid or hyperbolic tangent (tanh), and works by transforming the NN into an equivalent hybrid system. Specifically, since the sigmoid/tanh is the solution to a quadratic differential equation, each neuron could be viewed as a state in a dynamical system, whereas each layer could be mapped to a discrete mode. The NN's hybrid system is then composed with the plant's hybrid system, thereby casting the problem as a hybrid system verification problem that can be solved by an optimized tool such as Flow* [6]. Verisig was originally evaluated on Mountain Car [34] (i.e., a reinforcement learning benchmark) as well as on a quadrotor case study in which the NN is used to approximate a model predictive controller [41] that cannot be executed online. Although Verisig showed promising performance in these case studies, its scalability is limited by the fact that it integrates the sigmoid "dynamics" to obtain the reachable set for each neuron.

In this article, we develop a more scalable verification approach. In particular, we build on the technique used in Verisig but instead of integrating the sigmoid "dynamics," we adopt the Taylor Model (TM) framework used in hybrid system reachability [6]. A TM of a function is a polynomial approximation, together with worst-case error bounds. TMs can effectively approximate the flow of various non-linear hybrid systems and scale well when used with interval analysis. To make use of TMs in NNs, we approximate the NN's activations with a Taylor series and obtain error bounds (using Taylor's Theorem) based on bounds on the inputs to each neuron. In this setting, the NN can be viewed as a simple hybrid system with TM resets and no continuous dynamics.

We compare the TM approach with Verisig both on the original case studies as well as on two additional benchmarks, an automatic cruise control (ACC) system [29] and a more challenging autonomous racing verification case study presented in our second preliminary work [21]. In all cases, the TM approach results in an order of magnitude improvement in scalability at no cost in precision (sometimes with much greater precision).

For further evaluation, we also implemented an alternative verification approach based on an MILP approximation of the sigmoid/tanh, as proposed by Dutta et al. [12]. Specifically, the sigmoid is upper and lower bounded by piecewise linear functions, which allows us to transform the entire NN into an MILP that can be solved by efficient tools such as Gurobi [36]. We observe that for small NNs with few inputs, the MILP formulation is comparable with TM Verisig in terms of speed and

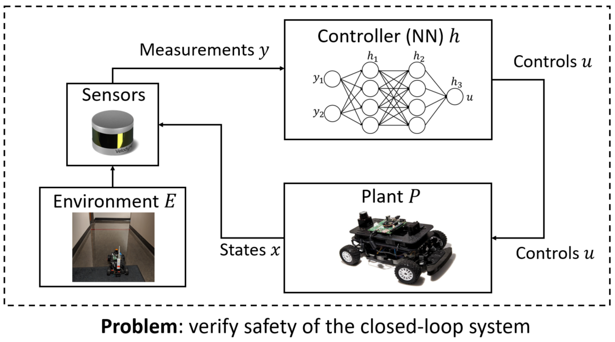


Fig. 1. Overview of the problem considered in this article.

precision. However, as we increase (1) the NN size, (2) the input uncertainty, or (3) the number of inputs, the MILP approach is significantly slower and results in larger error due to the sigmoid approximation. These results highlight the benefit of the TM approach, both in terms of scalability and of approximation of reachable sets.

Finally, we provide a comparison between the TM approach and an alternative closed-loop verification tool, namely NNV [47]. NNV works by approximating the NN's reachable set using star sets, which provide tight approximations while also allowing for efficient computation. For fair comparison, we tune both tools' parameters so as to achieve roughly the same runtimes on the ACC benchmark. For this setup, the TM method results in significantly smaller reachable sets, likely due to the fact that Verisig addresses the problem as a single hybrid system verification instance, whereas NNV performs the plant and NN verification sequentially, thereby potentially introducing additional error due to the composition.

In summary, the contributions of our preliminary work are (1) a verification approach for closed-loop systems with NN controllers using a hybrid system formulation, (2) a theoretical analysis of the decidability of sigmoid-based NN reachability, and (3) evaluation of Verisig on three benchmarks, including a challenging autonomous racing case study. The additional contributions of this article are (1) a TM-based verification approach that builds on the original hybrid system technique; (2) a comparison with the original approach on the first three benchmarks, as well as on the ACC benchmark; and (3) an exhaustive comparison between Verisig and two alternative methods.

This article is organized as follows. Section 2 formulates the verification problems considered in this work. In Section 3, we analyze the decidability of the considered problem, whereas Sections 4 and 5 present the original and improved versions of Verisig, respectively. All verification benchmarks are described in Section 6, and the evaluation is shown in Section 7. A further comparison between the Verisig approaches and an MILP-based approach is given in Section 8, whereas the comparison with NNV is provided in Section 9. Finally, Sections 10 and 11 discuss related work and provide concluding remarks, respectively.

2 PROBLEM FORMULATION

This section presents the verification problem addressed in this article. A high-level overview is shown in Figure 1. We consider a closed-loop system that consists of (1) a physical plant with states x, (2) an environment where the plant operates, (3) measurements y produced as a function of the plant states within the environment, and (4) a NN controller h that maps the measurements to control inputs u. The rest of this section describes each of these components in more detail.

2.1 Plant Model

We assume that the plant dynamics and measurements are described by a hybrid system. A hybrid system consists of a set of discrete modes and a finite number of continuous variables [27]. Within each mode, the continuous states evolve according to differential equations with respect to time. Each mode may have a number of invariants that must hold true while the system is in that mode. Finally, transitions between modes may reset the continuous states and are controlled by guards, which are Boolean expressions of the continuous states. The formal definition is shown next.

Definition 1 (Hybrid System). A hybrid system with inputs u and outputs y is a tuple $H = (X, X_0, F, E, I, G, R, g)$, where

- $X = X_D \times X_C$ is the state space with X_C a manifold and $X_D = \{q_1, \dots, q_m\}$;
- $X_0 \subseteq X$ is the set of initial states;
- $F : X \to TX_C$ assigns to each discrete mode $q \in X_D$ a set of differential equations f_q, i.e., $\dot{x} = f_q(x, u)$, where $x \in X_C$;
- $E \subseteq X_D \times X_D$ is the set of mode transitions;
- $I : X_D \to 2^{X_C}$ assigns to $q \in X_D$ an invariant of the form $I(q) \subseteq X_C$;
- $G : E \to 2^{X_C}$ assigns to each edge $e = (q_1, q_2)$ a guard $U \subseteq I(q_1)$;
- $R : E \to (2^{X_C} \to 2^{X_C})$ assigns to each edge $e = (q_1, q_2)$ a reset $V \subseteq I(q_2)$;
- $g : X \to \mathbb{R}^p$ is the observation model, i.e., $y = g(x)$.

Note that the relationship between the measurements y and the environment is implicitly captured in the observation model g. Without loss of generality, this definition assumes that the environment has no state; if this assumption is wrong, then the environment and plant states can be stacked together in a new combined state. Finally, note that, for a given initial set X_0, the reachable set for a continuous state x at time t is set of all values that $x(t)$ can take when started from X_0.

2.2 NN Controller

As mentioned above, the NN controller h takes measurements y as input and outputs control actions u. For ease of presentation, we assume h is a fully connected NN, although other common NN classes, such as convolutional, residual and recurrent NNs, could also be accommodated by the presented framework. Thus, the controller h could be represented as the composition of its L layers:

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

where each layer h_i consists of a linear map W_i followed by a non-linear activation function $a : \mathbb{R} \to \mathbb{R}$ (the last layer h_L is typically linear but could have an activation as well)[1]:

$$h_i(y) = a(W_i y + b_i). \tag{2}$$

As mentioned in the Introduction, we consider smooth activations, e.g., the sigmoid, $\sigma(x) = 1/(1 + e^{-x})$, and tanh, $tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. Furthermore, we assume that the controller is *already trained* and all parameters $(W_1, b_1, \dots, W_L, b_L)$ are known and fixed.

2.3 Composed System

We assume that the NN controller is executed in a time-triggered fashion.[2]

[1]We abuse notation by allowing a to also take a vector of inputs. In this case, the output of a is a vector in which a is applied separately to each element.

[2]Although an event-triggered formulation could be used as well, we only present the time-triggered version in the interest of clarity.

Definition 2 (Time-triggered Hybrid System). Consider a hybrid system with inputs u and outputs y as in Definition 1. The NN controller h is said to be time-triggered, with period T, if the inputs u are changed every T seconds, i.e.,

$$u(t) = h(y(t_k)), \text{for } t \in [t_k, t_k + T),$$

where $t_k = kT$ and $k = 0, 1, 2, \ldots$

2.4 Problem Statement

Since verifying safety properties of the closed-loop system presented in this section requires a special approach for the NN, we first investigate the problem of verifying input-output properties of the NN in isolation.

Problem 1. *Let h be a NN as described in Section 2.2. The NN verification problem, expressed as property ϕ, is to verify a property ψ on the NN's outputs u given constraints ξ on the inputs y:*

$$\phi(y, u) \equiv (\xi(y) \wedge h(y) = u) \Rightarrow \psi(u). \tag{3}$$

Once Problem 1 has been addressed, we can also state the closed-loop reachability problem.

Problem 2. *Let $S = h \mid\mid H_P$ be the composition of a NN controller h (Section 2.2) and a plant P, modeled with a hybrid system H_P (Section 2.1). Given a set of initial states X_0 of P, the problem, expressed as property ϕ_S, is to verify a property ψ_S of the reachable states of P:*

$$\phi_S(X_0) \equiv (x(0) \in X_0) \Rightarrow \psi_S(x(t)),\ \forall t \geq 0. \tag{4}$$

3 DECIDABILITY OF SIGMOID-BASED NN VERIFICATION

Before presenting our verification approach, we first investigate the decidability of the NN verification problem. Note that the verification of ReLU-based NNs is readily shown to be decidable (for linear constraints on the inputs and outputs), as the problem can be transformed into an MILP. However, sigmoid-based NNs are harder to reason about due to the non-linear and smooth nature of the sigmoid/tanh. In this section, we show that verification is decidable for NNs with one hidden layer, under mild assumptions on the parameters. In the general case, we show that if the inputs and outputs are constrained by a real arithmetic property (defined below), then the NN verification problem can be stated as a real arithmetic property with transcendental functions, which is decidable if Schanuel's conjecture is true [51].[3]

3.1 NNs with Multiple Hidden Layers

We begin our discussion with the general case of NNs with multiple hidden layers. As stated in Section 2, the NN verification problem has the following form:

$$\phi(y, u) \equiv (\xi(y) \wedge h(y) = u) \Rightarrow \psi(u), \tag{5}$$

where ξ and ψ are properties on the real numbers. Since verifying general properties of the reals is undecidable, we focus on a special class of properties, called real arithmetic properties. These are first-order logic formulas over $(\mathbb{R}, <, +, -, \cdot, 0, 1)$, i.e., the language where $<$ is the relation, $+$, $-$, and $\cdot$ are functions, and 0 and 1 are the constants [45]; we denote such formulas by $\mathcal{R}$-formulas. Example $\mathcal{R}$-formulas are $\forall x\ \forall y\ :\ xy > 0$, $\exists x\ :\ x^2 - 2 = 0$, and $\exists w\ :\ xw^2 + yw + z = 0$. Intuitively, $\mathcal{R}$-formulas are first-order logic statements over polynomial constraints with integer coefficients. Tarski was the first to provide a decision procedure for verifying real arithmetic properties [45].

[3]Note that all results presented in this section hold for NN sigmoid activations, but similar results can be derived for tanh.

A related class of properties contains real arithmetic properties with transcendental functions, denoted by $\mathcal{R}_{\exp}$-formulas. This is the language $(\mathbb{R}, <, +, -, \cdot, \exp, 0, 1)$, which also includes exponentiation. Although it is open whether verifying $\mathcal{R}_{\exp}$-formulas is decidable, it is known that decidability is connected to Schanuel's conjecture [51]. Schanuel's conjecture concerns the transcendence degree of certain field extensions of the rational numbers and, if true, would imply that verifying $\mathcal{R}_{\exp}$-formulas is decidable [51].

We investigate the case where ξ and ψ are $\mathcal{R}$-formulas. However, the exponentiation in the sigmoid means that ϕ is not a $\mathcal{R}$-formula. The next proposition shows that ϕ can in fact be stated as a $\mathcal{R}_{\exp}$-formula, which implies that NN verification is decidable if Schanuel's conjecture is true [51].

Proposition 1. *Let $h : \mathbb{R}^p \to \mathbb{R}^q$ be a sigmoid-based NN with $L-1$ hidden layers (with N neurons each), a linear last layer and rational parameters. The property $\phi(y,u) \equiv (\xi(y) \wedge h(y) = u) \Rightarrow \psi(u)$, where ξ and ψ are $\mathcal{R}$-formulas, is an $\mathcal{R}_{exp}$-formula.*

Proof. Since ψ is an $\mathcal{R}$-formula, it suffices to show that $\phi_0(y,u) \equiv \xi(y) \wedge h(y) = u$ can be expressed as an $\mathcal{R}_{\exp}$-formula. Note that

$$\begin{aligned}\phi_0(y,u) \equiv \xi(y) \wedge h_1^1 = {} & \frac{1}{1+\exp\left\{-\left(w_1^1\right)^\top y - b_1^1\right\}} \wedge \cdots \wedge h_1^N = \frac{1}{1+\exp\left\{-\left(w_1^N\right)^\top y - b_1^N\right\}} \wedge \ldots \\ \wedge\, h_{L-1}^1 = {} & \frac{1}{1+\exp\left\{-\left(w_{L-1}^1\right)^\top h_{L-2} - b_{L-1}^1\right\}} \wedge \cdots \wedge h_{L-1}^N = \frac{1}{1+\exp\left\{-\left(w_{L-1}^N\right)^\top h_{L-2} - b_{L-1}^N\right\}} \\ \wedge\, u = {} & W_L[h_{L-1}^1, \ldots, h_{L-1}^N]^\top + b_L,\end{aligned}$$

where $(w_i^j)^\top$ is row j of W_i and $h_l = [h_l^1, \ldots, h_l^N]^\top, l \in \{1, \ldots, L-1\}$. The last constraint, call it $p(u)$, is already an $\mathcal{R}$-formula. Let $[W_i]_{jk} = p_{jk}^i / q_{jk}^i$, with p_{jk}^i and $q_{jk}^i > 0$ integers, and let $d_0 = q_{11}^1 q_{12}^1 \cdots q_{Np}^{L-1}$. To remove fractions from the exponents, we add extra variables z_i and v_i^j and arrive at an equivalent property $\phi_{\mathbb{Z}}$, which is an $\mathcal{R}_{\exp}$-formula, since all denominators are $\mathcal{R}_{\exp}$-formulas:

$$\begin{aligned}\phi_{\mathbb{Z}}(y,u) \equiv \xi(y) \wedge z_0 d_0 = y \wedge h_1^1 = {} & \frac{1}{1+\exp\left\{-\left(r_1^1\right)^\top z_0 - v_1^1\right\}} \wedge \cdots \wedge h_1^N = \frac{1}{1+\exp\left\{-\left(r_1^N\right)^\top z_0 - v_1^N\right\}} \\ \wedge\, v_1^1 = b_1^1 \wedge \cdots \wedge v_1^N = {} & b_1^N \wedge \cdots \wedge z_{L-2} d_0 = h_{L-2} \\ \wedge\, h_{L-1}^1 = {} & \frac{1}{1+\exp\left\{-\left(r_{L-1}^1\right)^\top z_{L-2} - v_{L-1}^1\right\}} \wedge \cdots \wedge h_{L-1}^N = \frac{1}{1+\exp\left\{-\left(r_{L-1}^N\right)^\top z_{L-2} - v_{L-1}^N\right\}} \\ \wedge\, v_{L-1}^1 = b_{L-1}^1 \wedge \cdots \wedge v_{L-1}^N = {} & b_{L-1}^N \wedge p(u),\end{aligned}$$

where $r_i^j = w_i^j d_0$ are vectors of integers; $v_i^j = b_i^j$ are $\mathcal{R}$-formulas, since b_i^j are rational. $\square$

Corollary 1 ([51]). *If Schanuel's conjecture holds, then verifying the property $\phi(y,u) \equiv (\xi(y) \wedge h(y) = u) \Rightarrow \psi(u)$ is decidable under the conditions stated in Proposition 1.*

3.2 NNs with a Single Hidden Layer

In the case of NNs with one hidden layer, one could show that verification is in fact decidable, assuming interval constraints on the inputs. In particular, the following theorem shows that the NN verification problem can be stated as an $\mathcal{R}$-formula, thereby implying decidability.

Theorem 1. *Let $h : \mathbb{R}^p \to \mathbb{R}^q$ be a sigmoid-based NN with rational parameters and with one hidden layer (with N neurons), i.e., $h(x) = W_2(\sigma(W_1 x + b_1)) + b_2$. Let $[W_1]_{ij} = p_{ij}/q_{ij}$ and let $d_0 = q_{11} q_{12} \cdots q_{Np}$. Consider the property*

$$\phi(y,u) \equiv \exists y\ (y \in I_y \wedge u = h(y)) \Rightarrow \psi(u),$$

where $y = [y_1, \dots, y_p]^\top \in \mathbb{R}^p$, $u = [u_1, \dots, u_q]^\top \in \mathbb{R}^q$, ψ is an $\mathcal{R}$-formula, and $I_y = [\alpha_1, \beta_1] \times \cdots \times [\alpha_p, \beta_q] \subseteq \mathbb{R}^p$, i.e., the Cartesian product of p one-dimensional intervals. Then verifying $\phi(y, u)$ is decidable if, for all $i \in \{1, \dots, N\}$ and $j \in \{1, \dots, p\}$, $e^{b_1^i}$, e^{α_j / d_0}, and e^{β_j / d_0} are rational, i.e., $b_1^i = \ln(b_r^i)$, $\alpha_j = d_0 \ln(\alpha_r^j)$ and $\beta_j = d_0 \ln(\beta_r^j)$ for some rational numbers b_r^i, α_r^j, and β_r^j.

Proof. The proof technique borrows ideas from Reference [27]. It suffices to show that $\phi(y, u)$ is an $\mathcal{R}$-formula. Since $\psi(u)$ is an $\mathcal{R}$-formula, we focus on the left-hand side of the implication, call it $\phi_0(y, u)$:

$$\phi_0(y,u) \equiv y \in I_y \wedge h_1^1 = \frac{1}{1 + \exp\left\{ - \left(w_1^1\right)^\top y - b_1^1 \right\}} \wedge \cdots \wedge h_1^N = \frac{1}{1 + \exp\left\{ - \left(w_1^N\right)^\top y - b_1^N \right\}} \wedge$$
$$\wedge\, u = W_2 \left[h_1^1, \dots, h_1^N\right]^\top + b_2,$$

where $(w_1^i)^\top$ is row i of W_1. The last constraint in $\phi_0(y, u)$, call it $p(u)$, is an $\mathcal{R}$-formula. To remove fractions from the exponentials, we change the limits of y. Consider the property

$$\phi_{\mathbb{Z}}(y,u) \equiv y \in I_y^{\mathbb{Z}} \wedge h_1^1 = \frac{1}{1 + \exp\left\{ - \left(r_1^1\right)^\top y - b_1^1 \right\}} \wedge \cdots \wedge h_1^N = \frac{1}{1 + \exp\left\{ - \left(r_1^N\right)^\top y - b_1^N \right\}} \wedge p(u),$$

where $I_y^{\mathbb{Z}} = [\alpha_1/d_0, \beta_1/d_0] \times \cdots \times [\alpha_p/d_0, \beta_p/d_0]$ and each $r_1^i = d_0 w_1^i$ is a vector of integers. Note that $\phi_0(y, u) \equiv \phi_{\mathbb{Z}}(y, u)$, since a change of variables $z = y/d_0$ implies that $z \in I_y^{\mathbb{Z}}$ iff $y \in I_y$. To remove exponentials from the constraints, we use their monotonicity property and transform $\phi_{\mathbb{Z}}(x, y)$ into an equivalent property $\phi_e(x, y)$:

$$\phi_e(y,u) \equiv y \in I_y^e \wedge h_1^1 = \frac{1}{1 + y_1^{r_{11}^1} \cdots y_p^{r_{1p}^1} \exp\left\{ -b_1^1 \right\}} \wedge \cdots \wedge h_1^N = \frac{1}{1 + y_1^{r_{11}^N} \cdots y_p^{r_{1p}^N} \exp\left\{ -b_1^N \right\}} \wedge p(u),$$

where $I_y^e = [e^{-\beta_1/d_0}, e^{-\alpha_1/d_0}] \times \cdots \times [e^{-\beta_p/d_0}, e^{-\alpha_p/d_0}]$, and r_{1j}^i is element j of r_1^i. To see that $\phi_e(y, u) \equiv \phi_{\mathbb{Z}}(y, u)$, take any $y \in I_y^{\mathbb{Z}}$ and note that $\exp\{-r_{1j}^i y_j\} = z_j^{r_{1j}^i}$, with $z_j = e^{-y_j}$; thus, $z \in I_x^e$.

The final step transforms the property $\phi_e(y, u)$ into an equivalent property $\nu(y, u)$ to eliminate negative integers r_{1j}^i in the exponents:

$$\nu(y,u) \equiv y \in I_y^e\ \exists z \in I_y^{e-}\ y_1 z_1 = 1 \wedge \cdots \wedge y_p z_p = 1 \wedge h_1^1 = \frac{1}{1 + \prod_{j \in \mathcal{I}_1^+} y_j^{r_{1j}^1} \prod_{j \in \mathcal{I}_1^-} z_j^{-r_{1j}^1} \exp\left\{ -b_1^1 \right\}} \wedge \dots$$
$$\wedge\, h_1^N = \frac{1}{1 + \prod_{j \in \mathcal{I}_N^+} y_j^{r_{1j}^N} \prod_{j \in \mathcal{I}_N^-} z_j^{-r_{1j}^N} \exp\left\{ -b_1^N \right\}} \wedge p(u),$$

where $I_y^{e-} = [e^{\alpha_1/d_0}, e^{\beta_1/d_0}] \times \cdots \times [e^{\alpha_p/d_0}, e^{\beta_p/d_0}]$, $\mathcal{I}_i^+ = \{k \mid r_{1k}^i \geq 0\}$, and $\mathcal{I}_i^- = \{k \mid r_{1k}^i < 0\}$. Note that $\phi_e(y, u) \equiv \nu(y, u)$, since for $r_{1j}^i < 0$ the constraint $z_j y_j = 1$ implies $y_j^{r_{1j}^i} = z_j^{-r_{1j}^i}$.

Thus, if $e^{b_1^j}$, e^{α_i/d_0}, and e^{β_i/d_0} are rational for all $i \in \{1, \dots, p\}, j \in \{1, \dots, N\}$, then one can show that $\nu(y, u)$ is an $\mathcal{R}$-formula by multiplying all h_1^i constraints by their denominators. All denominators are positive, since y_i and z_i are constrained to be positive. □

Note that the technique used to prove Theorem 1 cannot be applied to multiple hidden layers, since the NN becomes an $\mathcal{R}_{\exp}$-formula in that case. Also, although the assumption on the NN's weights may appear contrived, it can be easily satisfied by conservatively increasing the interval constraints on the inputs by a small number.

4 NN VERIFICATION USING HYBRID SYSTEM REACHABILITY

Section 3 analyzed the decidability of NN verification to provide intuition on the problem's difficulty. In this section, we present the verification approach used in Verisig, namely converting the NN into an equivalent hybrid system. Given this hybrid system, one can use existing hybrid system verification tools to verify the desired property by computing the hybrid system's reachable sets. The rest of this section presents the verification approach in detail, beginning with the dynamical-system interpretation of the sigmoid.[4]

4.1 Sigmoids as Solutions to Differential Equations

The key idea that enables the transformation of a NN into a hybrid system is the fact that the sigmoid derivative can be expressed in terms of the sigmoid itself:[5]

$$\frac{d\sigma}{dx}(x) = \sigma(x)(1-\sigma(x)). \tag{6}$$

Equation (6) has the flavor of a dynamical system, the main difference being that the partial derivative is not taken with respect to time. We could obtain a true dynamical system by introducing a proxy "time" variable as follows:

$$g(t,x) = \sigma(tx) = \frac{1}{1+e^{-xt}}, \tag{7}$$

such that $g(1,x) = \sigma(x)$ and, by the chain rule,

$$\frac{\partial g}{\partial t}(t,x) = \dot{g}(t,x) = xg(t,x)(1-g(t,x)). \tag{8}$$

Thus, if x lies in a set X, then the image $\sigma(X)$ could be obtained by computing the reachable set of g at time $t = 1$, starting from an initial condition $g(0,x) = 0.5$ (as can be verified from Equation (7)) and following the dynamics in Equation (8). While the intermediate values of g are not important, continuously tracing the sigmoid "dynamics" allows us to iteratively construct the sigmoid's reachable set, i.e., the image $g(1,X)$. Section 5 presents a more direct method to obtain this reachable set.

4.2 NNs as Hybrid Systems

Given the dynamical-system interpretation of the sigmoid presented in the previous subsection, we now show how to transform the entire NN into a hybrid system. For ease of presentation, we assume that all hidden layers have the same number of neurons, N, although the formulation can be easily adapted to the more general case.

First note that each neuron h_{ij} in hidden layer h_i can be written as:

$$h_{ij}(x) = \sigma\left(\left(w_i^j\right)^\top x + b_i^j\right), \tag{9}$$

[4]Note that this section focuses on the case of sigmoid activations; the treatment of tanh activations is almost identical—the differences are noted in the relevant places in the section.
[5]The corresponding differential equation for tanh is $(d\tanh/dx)(x) = 1 - \tanh^2(x)$.

where $(w_i^j)^\top$ is row j of W_i and b_i^j is element j of b_i. Given h_{ij}, the corresponding proxy function g_{ij} is defined as follows:

$$g_{ij}(t,x) = \frac{1}{1 + \exp\Big\{-t \cdot \Big(\big(w_i^j\big)^\top x + b_i^j\Big)\Big\}},$$

where, once again, $g_{ij}(1,x) = h_{ij}(x)$. Also, by the chain rule,

$$\dot{g}_{ij}(t,x) = \Big(\big(w_i^j\big)^\top x + b_i^j\Big) g_{ij}(t,x)(1 - g_{ij}(t,x)). \tag{10}$$

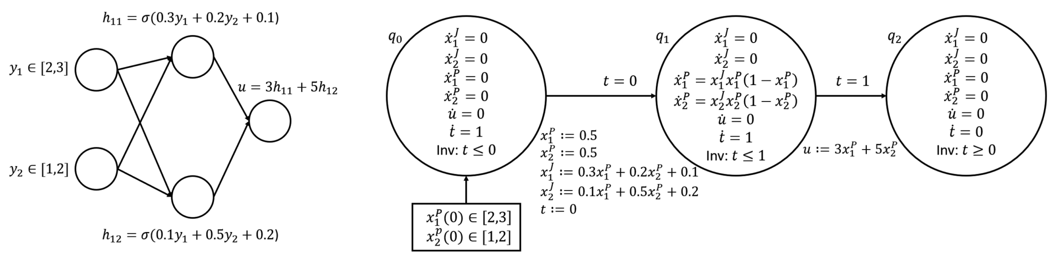
Thus, each neuron h_{ij} can be captured by a corresponding g_{ij} with initial condition $g_{ij}(0,x) = 0.5$ and dynamics as shown in Equation (10).

To transform the entire NN, note that the linear mappings between layers in the NN can be thought of as discrete resets in a hybrid system. Thus, each NN layer corresponds to a mode in the hybrid system that computes the sigmoid dynamics in Equation (10) until $t = 1$. At time $t = 1$, a reset occurs that resets all states according to the linear map, after which we continue processing the following layer/mode. Formally, we use N continuous states, $x_1^P, \dots, x_N^P$, to represent the proxy variables for each layer; when in mode i, each $x_j^P, j \in \{1, \dots, N\}$, represents neuron h_{ij} in the DNN. The linear resets are stored in states $x_1^J, \dots, x_N^J$. The x_i^J states are necessary, because the inputs to each neuron are functions of the x_i^P states reached in the previous mode.

The full transformation is described in Proposition 2 below. Note that there is an additional mode q_0, which is used to reset the x_i^P states to 0.5 and the x_i^J states to their corresponding values in q_1. For simplicity, Proposition 2 focuses on the case of one NN output, stored in state u—the case of additional outputs can be handled by adding more u states. Note that $\odot$ denotes Hadamard (element-wise) product.

Proposition 2. *Let $h : \mathbb{R}^p \to \mathbb{R}^1$ be a sigmoid-based NN with $L-1$ hidden layers (with N neurons each) and a linear last layer with one output. The image under h of a given set I_y is exactly the reachable set for u in mode q_L of the following hybrid system:*

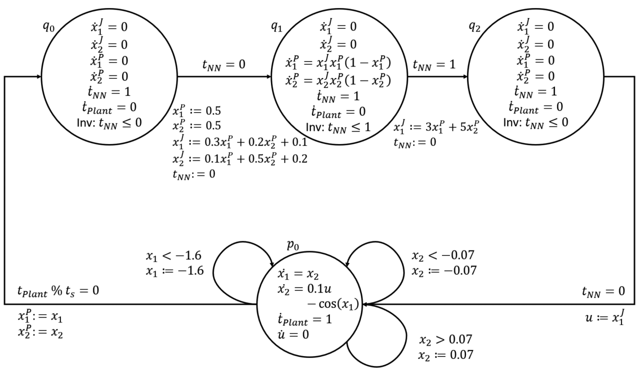
- *Continuous states:* $x^P = [x_1^P, \dots, x_N^P]^\top, x^J = [x_1^J, \dots, x_N^J]^\top, u, t$;
- *Discrete states (modes):* $q_0, q_1, \dots, q_L$;
- *Initial states:* $x^P \in I_y, x^J = 0, u = 0, t = 0$;
- *Flow:*
  —$F(q_0) = [\dot{x}^P = 0, \dot{x}^J = 0, \dot{u} = 0, \dot{t} = 1]$;
  —$F(q_i) = [\dot{x}^P = x^J \odot x^P \odot (1 - x^P), \dot{x}^J = 0, \dot{u} = 0, \dot{t} = 1]$ *for* $i \in \{1, \dots, L-1\}$;
  —$F(q_L) = [\dot{x}^P = 0, \dot{x}^J = 0, \dot{u} = 0, \dot{t} = 0]$;
- *Transitions:* $E = \{(q_0, q_1), \dots, (q_{L-1}, q_L)\}$;
- *Invariants:*
  —$I(q_0) = \{t \leq 0\}$;
  —$I(q_i) = \{t \leq 1\}$ *for* $i \in \{1, \dots, L-1\}$;
  —$I(q_L) = \{t \leq 0\}$;
- *Guards:*
  —$G(q_0, q_1) = \{t = 0\}$;
  —$G(q_i, q_{i+1}) = \{t = 1\}$ *for* $i \in \{1, \dots, L-1\}$;
- *Resets:*
  —$R(q_i, q_{i+1}) = \{x^P = 0.5, x^J = W_i x^P + b_i, t = 0\}$ *for* $i \in \{0, \dots, L-2\}$;
  —$R(q_{L-1}, q_L) = \{u = W_L x^P + b_L\}$.



$h_{11} = \sigma(0.3y_1 + 0.2y_2 + 0.1)$
$y_1 \in [2,3]$
$u = 3h_{11} + 5h_{12}$
$y_2 \in [1,2]$
$h_{12} = \sigma(0.1y_1 + 0.5y_2 + 0.2)$
q_0
$\dot{x}_1^I = 0$, $\dot{x}_2^I = 0$, $\dot{x}_1^P = 0$, $\dot{x}_2^P = 0$, $\dot{u} = 0$, $\dot{t} = 1$, Inv: $t \leq 0$
$t = 0$
$x_1^P := 0.5$, $x_2^P := 0.5$, $x_1^I := 0.3x_1^P + 0.2x_2^P + 0.1$, $x_2^I := 0.1x_1^P + 0.5x_2^P + 0.2$, $t := 0$
q_1
$\dot{x}_1^I = 0$, $\dot{x}_2^I = 0$, $\dot{x}_1^P = x_1^I x_1^P(1 - x_1^P)$, $\dot{x}_2^P = x_2^I x_2^P(1 - x_2^P)$, $\dot{u} = 0$, $\dot{t} = 1$, Inv: $t \leq 1$
$t = 1$
$u := 3x_1^P + 5x_2^P$
q_2
$\dot{x}_1^I = 0$, $\dot{x}_2^I = 0$, $\dot{x}_1^P = 0$, $\dot{x}_2^P = 0$, $\dot{u} = 0$, $\dot{t} = 0$, Inv: $t \geq 0$
$x_1^P(0) \in [2,3]$
$x_2^P(0) \in [1,2]$

(a) Example NN. (b) Equivalent hybrid system.

Fig. 2. Small example illustrating the transformation from a NN to a hybrid system.



q_0
$\dot{x}_1^I = 0$, $\dot{x}_2^I = 0$, $\dot{x}_1^P = 0$, $\dot{x}_2^P = 0$, $\dot{t}_{NN} = 1$, $\dot{t}_{Plant} = 0$, Inv: $t_{NN} \leq 0$
$t_{NN} = 0$
$x_1^P := 0.5$, $x_2^P := 0.5$, $x_1^I := 0.3x_1^P + 0.2x_2^P + 0.1$, $x_2^I := 0.1x_1^P + 0.5x_2^P + 0.2$, $t_{NN} := 0$
q_1
$\dot{x}_1^I = 0$, $\dot{x}_2^I = 0$, $\dot{x}_1^P = x_1^I x_1^P(1 - x_1^P)$, $\dot{x}_2^P = x_2^I x_2^P(1 - x_2^P)$, $\dot{t}_{NN} = 1$, $\dot{t}_{Plant} = 0$, Inv: $t_{NN} \leq 1$
$t_{NN} = 1$
$x_1^I := 3x_1^P + 5x_2^P$, $t_{NN} := 0$
q_2
$\dot{x}_1^I = 0$, $\dot{x}_2^I = 0$, $\dot{x}_1^P = 0$, $\dot{x}_2^P = 0$, $\dot{t}_{NN} = 1$, $\dot{t}_{Plant} = 0$, Inv: $t_{NN} \leq 0$
p_0
$\dot{x}_1 = x_2$, $\dot{x}_2 = 0.1u - \cos(x_1)$, $\dot{t}_{Plant} = 1$, $\dot{u} = 0$
$x_1 < -1.6$, $x_1 := -1.6$
$x_2 < -0.07$, $x_2 := -0.07$
$x_2 > 0.07$, $x_2 := 0.07$
t_{Plant} % $t_s = 0$
$x_1^P := x_1$, $x_2^P := x_2$
$t_{NN} = 0$
$u := x_1^I$

Fig. 3. Composition of the NN in Figure 2 with a toy plant hybrid system.

Proof. First, note that the reachable set of x^P in mode q_1 at time $t = 1$ is exactly the image of I_y under h_1, the first hidden layer. This is true, because at $t = 1$, x^P takes the value of the sigmoid function. Applying this argument inductively, the reachable set of x^P in mode q_{L-1} at time $t = 1$ is exactly the image of I_y under $h_{L-1} \circ \cdots \circ h_1$. Finally, u is a linear function of x^P with the same parameters as the last linear layer of h. Thus, the reachable set for u in mode q_L is the image of I_y under $h_L \circ \cdots \circ h_1 = h$. □

4.3 Illustrative Example

To illustrate the transformation described in the previous subsection, we present an example in Figure 2. In this example, a two-layer NN is transformed into a three-mode hybrid system. According to Proposition 2, the NN and the corresponding hybrid system are equivalent in the sense that the image of the set $y_1 \in [2, 3], y_2 \in [1, 2]$ under the NN is the same as the reachable set for u in mode q_2 of the hybrid system. In particular, since all weights are positive, the output u is in the range $[h(2, 1), h(3, 2)]$—the same conclusion can be reached about the state u in the hybrid system.

4.4 Composing the NN and the Plant

Once the NN is converted into a hybrid system, we can compose the NN's hybrid system with the plant's hybrid system. The composed hybrid system thus describes the entire closed-loop system. An example is shown in Figure 3, where the NN from Figure 2 is composed with a toy plant hybrid system. We emphasize that the "time" in the sigmoid dynamics, t_{NN}, is local to the NN, whereas

t_{Plant} captures the global physical time (note that t_{Plant} does not progress inside the NN modes).[6] Note that the controller in the example in Figure 3 is time triggered but that guard can be changed depending on the condition for triggering the control in a given system.

4.5 Hybrid System Verification Tools

Given the composed hybrid system, we can now use existing tools to address the hybrid system verification problem. Multiple tools have been developed in the literature, depending on the system class. SpaceEx [16] was designed for linear hybrid systems and can scale up to a few thousand states. Although verification is undecidable for general non-linear hybrid systems [3, 27], several approaches have been proposed that scale well in different scenarios. Flow* works by constructing flowpipe approximations using Taylor Models. Alternatively, dReach [25] casts the verification problem as an SMT formula and provides δ-decidability guarantees. Several other approaches have been proposed as well, e.g., CORA [2] and C2E2 [10], that rely on different computationally convenient approximations of the reachable sets such as zonotopes or simulation-based approximations.

In this article, we use Flow* due to its scalability on the considered case studies. Furthermore, the TM framework used in Flow* also suggests a natural way to extend the approach presented in this section. In particular, the next section describes a modified method that avoids the sigmoid integration and provides an order of magnitude improvement in scalability at no cost in precision.

5 TAYLOR MODEL APPROXIMATION OF THE SIGMOID

The previous section presented the hybrid-system-based approach used in Verisig. In this section, we build on these ideas and develop a more scalable technique that provides an order of magnitude improvement in scalability. We begin with a brief explanation of the TM framework adopted in Flow*, which serves us a starting point for our improved NN verification approach.

5.1 Hybrid System Reachability Using TMs

Intuitively, a TM of a given function f is a polynomial approximation p of f together with worst-case error bounds. In what follows, we use $\mathbb{I}$ to denote the set of all intervals $I = [l, b]$, and for any $S = I_1 \times \cdots \times I_n$, the center of S is a vector $c = [c_1, \ldots, c_n]$ such that each c_i is the midpoint of I_i.

To define a TM, we first introduce the concept of polynomial approximation. Let $f : D \to \mathbb{R}$ be a function over n variables, with $D \in \mathbb{I}^n$, and suppose f is j times continuously differentiable. A polynomial p of degree j is said to *approximate* f at a point $x \in D$, written $f(x) \equiv_j p(x)$, if all $0 < m \leq j$ partial derivatives of f and p coincide at x.

The TM of a function f is defined as follows. Let $f : D \to \mathbb{R}$ be a function over n variables, where $D \in \mathbb{I}^n$. A *Taylor Model* of order $j > 0$ for f over D is a pair (p, I) of a polynomial p of degree at most j and a remainder interval I such that:

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

TMs have shown great promise in hybrid system reachability due to the fact that they can approximate the reachable sets of various non-linear systems and scale well when used with interval analysis [5]. Specifically, given a set of differential equations and corresponding initial conditions, one can use Picard iteration to obtain a polynomial approximation with error bounds, i.e., a TM approximation, of the reachable set. Once a TM is available, one can use domain contraction [5] to check whether the reachable set intersects a given unsafe set, i.e., whether the safety property

[6]The example in Figure 3 assumes that the NN computation occurs instantaneously. If one would like to also model the time it takes to execute the NN, then the composition would need to also include a zero-order hold for the NN output.

ALGORITHM 1: NN Verification Using Taylor Models

```
Input: Measurement Taylor Model $TM_y$, NN controller $h$ with $L$ layers, and sigmoid/tanh activations.
1: $TM_0 \leftarrow TM_y$
2: for each $i$ in $\{1, \dots, L\}$ do
3:     $TM_i^L \leftarrow W_i * TM_{i-1} + b_i$
4:     $I_i^{TM} \leftarrow intervalApproximation(TM_i^L)$
5:     $TM_i^{\sigma} \leftarrow TaylorsTheoremForSigmoid(I_i^{TM})$
6:     $TM_i \leftarrow TM_i^{\sigma} \circ TM_i^L$
7: end for
8: return $TM_L$
```

to be verified is true or false. Finally, multiple techniques have also been proposed to keep the TM remainder small, such as preconditioning and suppression of the wrapping effect [31].

5.2 Neural Networks as Taylor Models

We begin with a general outline of the proposed TM-based approach for NN reachability, followed by the specific treatment of the sigmoid and tanh activations. The main idea of the new approach is to approximate every neuron with a TM, thereby avoiding the sigmoid integration used in the original method. This way, we not only drastically reduce the computation, but we also obtain better approximations, since we can analytically derive tight remainder bounds.

To get a TM approximation of each neuron, we use Taylor's Theorem, which allows us to construct a polynomial approximation with bounded error over a given interval. For completeness, we first state Taylor's Theorem (with the Lagrange form of the remainder) to show how it can be used to derive a TM for a given function.

Theorem 2 (Taylor's Theorem.). *Let $k \geq 1$ be an integer, let $I \in \mathbb{I}$ and let $f : \mathbb{R} \to \mathbb{R}$ be $k+1$ times differentiable at the point $a \in I$. Then for any $x \in I$*

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + R_k(x),$$

where (for $x \geq a$)

$$q\frac{(x-a)^{k+1}}{(k+1)!} \leq R_k(x) \leq Q\frac{(x-a)^{k+1}}{(k+1)!},$$

where q and Q satisfy $q \leq f^{(k+1)}(x) \leq Q$ for all $x \geq a$. Similar bounds can be derived for the case when $x < a$.

Taylor's Theorem can be used to derive a TM (with the bounds on R_k as the error bounds) for any function with bounded inputs and derivatives. Thus, to obtain a TM for each neuron n of the NN, one needs to find interval bounds I on the inputs to n, compute a Taylor series approximation of the sigmoid around the center of I, and bound the sigmoid derivative on I.

Using this intuition, Algorithm 1 illustrates how to transform the entire NN into a composition of TMs. The input to the algorithm is a TM for the measurements, call it TM_y. To obtain TM_y, we use the fact that the plant itself is part of the reachability problem, i.e., TMs are available for the reachable set of the measurements y. From TM_y and the initial condition, one first performs the linear part of each layer (Line 3). Given the resulting TM, can then obtain interval bounds using interval analysis (Line 4), thereby obtaining interval bounds on the inputs to the sigmoid that are used in Taylor's Theorem (Line 5). Taylor's Theorem provides a TM for each neuron's activation, which is composed with the linear TM to produce the final TM for that layer (Line 6). This process repeats as TMs are propagated through the NN layers. Note that the interval analysis in Line 4 is only used to obtain bounds on the TM input range so that we can bound the sigmoid Taylor series approximation error; however, each layer is still represented by a TM that is propagated using standard TM reachability techniques [5].

It is important to emphasize that Algorithm 1 not only allows us to avoid the sigmoid integration involved in the original version of Verisig, but it also enables us to analytically control the approximation error by choosing a sufficiently high TM order that achieves the desired error. The NN can now be seen as a simple hybrid system, with no dynamics and with TM resets between modes. The composition with the plant is performed in the same manner as before, as illustrated in Figure 3.

5.3 TMs for Sigmoid and Tanh

Since sigmoid and tanh are infinitely differentiable, Taylor series are easily derived for these functions. Furthermore, as shown in prior work [32], the sigmoid (and tanh) derivative coefficients can be mapped to Eulerian numbers, which can be precomputed offline; thus, high derivatives can be computed quickly using table look-ups. Derivative bounds over a given interval I can be obtained by finding all local optima[7] and checking where the endpoints of I lie. Two illustrative examples are shown next.

5.4 TM Examples

This subsection provides two examples to illustrate the technical aspects of Algorithm 1.

Example 1 (Obtaining Interval Bounds for a Given TM). This example illustrates how one could use interval analysis to obtain interval bounds for the inputs to the NN. Consider a system with two continuous states, x_1 and x_2, and initial condition $x_1 \in [0, 0.1], x_2 \in [-0.1, 0.5]$. Suppose the system has access to one measurement y, and suppose a second-order TM for y is available:

$$TM_y(x_1, x_2) = 2x_1^2 + 3x_2^2 - 0.4x_1x_2 + x_1 + [-0.01, 0.01].$$

Using interval analysis, one obtains the following interval bounds for y:

$$I_y = [0, 0.02] + [0, 0.75] - [-0.004, 0.02] + [0, 0.1] + [-0.01, 0.01] = [-0.03, 0.884].$$

Example 2 (Using Taylor's Theorem to Obtain a TM for Each Neuron). Consider the above example again, and suppose that the measurement y is sent as input to a single-neuron sigmoid layer: $h_1(y) = \sigma(0.2y + 0.1)$. To compute a TM for h_1, we use interval analysis in a similar way and obtain $I_1^{TM} = [0.094, 0.2768]$. Given I_1^{TM}, we obtain a second-order Taylor series approximation of the sigmoid around the center of I_1^{TM}, $c = 0.1854$:

$$TM_1^{\sigma}(y) = \sigma(c) + \sigma'(c)(y - c) + \sigma''(c)\frac{(y - c)^2}{2} + R_2(y),$$

where $\sigma'(c) = 0.2479, \sigma''(c) = -0.0229,\ q = -0.1208,\ Q = -0.1157,\ q(0.0914)^3/(3!) \leq R_2(y) \leq Q(0.0914)^3/(3!)$, and 0.0914 is the largest deviation from c to any point in I_1^{TM}. Thus, the final TM for h_1 is $TM_1^{\sigma}(0.2TM_y + 0.1)$.

[7]Local optima for sigmoid and tanh can be found analytically up to the fourth derivative. For higher derivatives, conservative bounds can be obtained numerically.

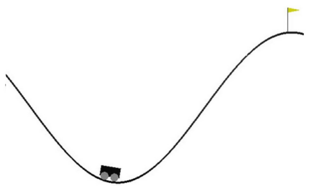


Fig. 4. Mountain Car problem [35]. The car needs to drive up the left hill first to gather enough momentum.

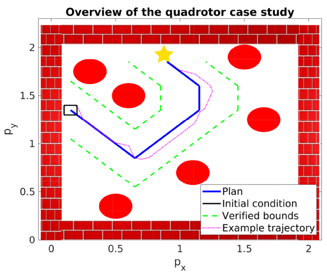


Fig. 5. Quadrotor case study, as projected to the (p_x, p_y)-plane. The quadrotor follows its plan to reach the goal without colliding into obstacles.

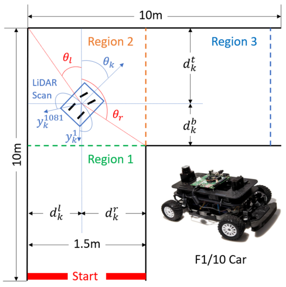


Fig. 6. Autonomous racing car navigation scenario. There are three different regions depending on how many walls can be reached using LiDAR.

5.5 Implementation

We implemented the TM approach on top of Flow*. Since Flow* provides an interface for dynamical system reachability using TMs, it is a logical choice as a starting point for our tool. The source code is available at github.com/verisig.

6 EVALUATION BENCHMARKS

We evaluate our approaches on four different benchmarks. These benchmarks were chosen to expose different challenges in terms of scalability and approximation. In particular, these are systems with the following diverse properties: (1) a hybrid plant model with a continuous-valued controller, (2) a continuous plant model with a discrete-valued controller, (3) a continuous plant model with a continuous-valued controller, and (4) a continuous plant model with a hybrid high-dimensional observation model and a continuous-valued end-to-end controller. These benchmarks also illustrate different ways in a which a NN could be trained: In benchmarks 1 and 4, the NN was trained using reinforcement learning; in benchmark 2, the NN was trained using a safe learning algorithm based on model predictive control; in benchmark 3, the NN was trained using supervised learning. The remainder of this section describes each benchmark in more detail.

6.1 Mountain Car

Mountain Car (MC) is a reinforcement learning benchmark in which the task is to train a controller to drive an underpowered car up a hill [34], as shown in Figure 4. The car does not have enough power to accelerate up the hill, so it needs to drive up the left hill first and gather momentum. The car has two states, position and velocity, that evolve in discrete time as follows:

$$\begin{aligned} p_{k+1} &= p_k + v_k \\ v_{k+1} &= v_k + 0.0015u_k - 0.0025 * cos(3p_k), \end{aligned}$$

where u_k is the control and p_k and v_k are the car's position and velocity, respectively, with p_0 chosen uniformly at random from $[-0.6, -0.4]$ and $v_0 = 0$. The car states are constrained as follows: $v_k \in [-0.07, 0.07], p_k \in [-1.2, 0.6]$, which introduces a hybrid mode switch if these constraints are violated. The control u_k takes a continuous value in the range $[-1, 1]$.

During training, a reward of $-0.1u_k^2$ is received after each step, which forces a control policy that applies as little thrust as possible. A reward of 100 is received upon reaching the goal. In our preliminary work [22], we used deep deterministic policy gradient (DDPG) reinforcement learning [28] to train a NN controller. The controller has two hidden layers with sigmoid activations with 16 neurons per layer and a tanh output layer with one neuron. The controller takes p_k and v_k as input and produces u_k. The machine learning task is considered solved if an average reward above 90 is obtained over 100 random runs. We can now strengthen the definition of a "solved" task and verify that the reward is above 90 for *any* initial condition, i.e., $p_0 \in [-0.6, -0.4]$.

6.2 Quadrotor with a NN Controller

In the second benchmark a NN is trained to approximate a model predictive controller (MPC) with safety guarantees [41]. In particular, one can train the NN to follow a piecewise-linear plan and bound the deviation from the plan by simulating against a worst-case planner from a given point [41]. To guarantee a bounded deviation over sets of initial conditions, however, one would need to formally verify such a bound, which is the problem considered in this benchmark.

We consider an unmanned quadrotor with a NN controller that is following a piecewise-linear plan avoiding obstacles, as illustrated in Figure 5. The quadrotor dynamics are modeled as a six-dimensional control-affine system, whereas the planner dynamics are piecewise linear, as follows:

$$\dot{q} := \begin{bmatrix} \dot{p}_x^q \\ \dot{p}_y^q \\ \dot{p}_z^q \\ \dot{v}_x^q \\ \dot{v}_y^q \\ \dot{v}_z^q \end{bmatrix} = \begin{bmatrix} v_x^q \\ v_y^q \\ v_z^q \\ g\tan\theta \\ -g\tan\phi \\ \tau - g \end{bmatrix}, \quad \dot{p} := \begin{bmatrix} \dot{p}_x^p \\ \dot{p}_y^p \\ \dot{p}_z^p \\ \dot{v}_x^p \\ \dot{v}_y^p \\ \dot{v}_z^p \end{bmatrix} = \begin{bmatrix} b_x \\ b_y \\ b_z \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{11}$$

where p_x^q, p_y^q, p_z^q and p_x^p, p_y^p, p_z^p are the quadrotor and planner's positions, respectively; v_x^q, v_y^q, v_z^q and v_x^p, v_y^p, v_z^p are the quadrotor and planner's velocities, respectively; θ, ϕ and τ are control inputs (for pitch, roll and thrust); $g = 9.81$ m/s^2 is gravity; b_x, b_y, b_z are piecewise constant functions of time. The control inputs have constraints $\phi, \theta \in [-0.1, 0.1]$ and $\tau \in [7.81, 11.81]$; the planner velocities have constraints $b_x, b_y, b_z \in [-0.25, 0.25]$. The control task is to ensure the quadrotor follows the planner as closely as possible.

As described in prior work [41], the optimal controller for the model in Equation (11) is a "bang-bang" controller, i.e., it is effectively a classifier mapping plant states to a finite set of control actions. To train the NN controller, we follow the approach described in prior work, i.e., we sample multiple points from the state space over a bounded horizon and train a sequence of controllers, one for each control sampling step. When two consecutive NNs have similar error, we interrupt training and pick the last one as the final controller. We trained a NN with two hidden layers with 20 neurons each (with tanh activations) and a linear last layer with eight neurons (i.e., the number of possible control actions). The property to be verified is that the quadrotor does not deviate by more than 0.32 m from the planner, for initial conditions $(p_x^r(0), p_y^r(0)) \in [-0.05, 0.05] \times [-0.05, 0.05]$ (the other states are initialized at 0), where we define the vector of relative states as $r := [p_x^r, p_y^r, p_z^r, v_x^r, v_y^r, v_z^r]^\top = q - p$.

6.3 Adaptive Cruise Control

In the ACC benchmark [29], there are two vehicles, the ego and the lead vehicle, going in a straight line. The control task is to make sure the ego vehicle follows the lead vehicle at a safe distance.

The two cars' dynamics are the same, modulo the control inputs, and are given as follows:

$$\begin{bmatrix} \dot{x}_c \\ \dot{v}_c \\ \dot{\gamma}_c \end{bmatrix} = \begin{bmatrix} v_c \\ \gamma_c \\ -2\gamma_c + 2a_c - uv_c^2 \end{bmatrix}, \tag{12}$$

where x_c is position, v_c is velocity, γ_c is acceleration, a_c is the control thrust applied to the car, $u = 0.0001$ is the friction control, and the subscript $c \in \{ego, lead\}$. The lead car has an MPC as described in prior work [29], with a target velocity $V_{set} = 30$ m/s. The ego vehicle has a NN controller that takes five inputs: V_{set}, $T_{gap} = 1.4s$, v_{ego}, $x_{lead} - x_{ego}$, $v_{lead} - v_{ego}$. The NN was trained using supervised learning [29] and has three hidden layers with 20 neurons per layer (with tanh activations) and a linear last layer with one neuron. In the considered scenario, the lead car applies a sudden brake (with $a_{lead} = -2$), and the problem is to verify that a safety distance is maintained over the next 5 s. Specifically, the safety property to be verified is $x_{lead} - x_{ego} \geq D_{default} + T_{gap} * v_{ego}$, where $D_{default} = 10m, T_{gap} = 1.4s$, for initial conditions $x_{lead}(0) \in [90, 110], v_{lead}(0) = [32, 32.05], \gamma_{lead}(0) = 0, x_{ego} \in [10, 11], v_{ego}(0) \in [30, 30.05], \gamma_{ego}(0) = 0$.

6.4 Autonomous Racing Car

The last benchmark is a more challenging case study in which an autonomous car must navigate a structured environment using LiDAR measurements. Specifically, we consider the scenario illustrated in Figure 6, in which the car starts in the middle of a hallway and must safely make a right turn. This case study is motivated by the F1/10 autonomous racing competition [1]. A more detailed description of this case study is provided in our preliminary work [21].

The car is modeled with a kinematic bicycle model [38, 39], which is a standard model for cars with front steering. The car dynamics are given as follows:

$$\begin{aligned} \dot{x} &= v cos(\theta + \beta) \\ \dot{y} &= v sin(\theta + \beta) \\ \dot{v} &= -c_a v + c_a c_m (u - c_h) \\ \dot{\theta} &= \frac{V cos(\beta)}{l_f + l_r} tan(\delta) \\ \beta &= tan^{-1}\left(\frac{l_r tan(\delta)}{l_f + l_r}\right), \end{aligned} \tag{13}$$

where v is the car's linear velocity, θ is the car's orientation, β is the car's slip angle, and x and y are the car's position; u is the throttle input, and δ is the heading input; c_a is an acceleration constant, c_m is a car motor constant, c_h is a hysteresis constant, and l_f and l_r are the distances from the car's center of mass to the front and rear, respectively. Since tan^{-1} is not supported by Flow*, we assume that $\beta = 0$; this is not a limiting assumption in the considered case study as the slip angle is typically fairly small at low speeds. The parameter values were identified as follows: $c_a = 1.633, c_m = 0.2, c_h = 4, l_f = 0.225m, l_r = 0.225m$. The throttle is constant at $u = 16$ (resulting in a top speed of roughly 2.4 m/s), i.e., the controller only controls heading.

The car has access to 21 laser imaging, detection and ranging (LiDAR) rays, as illustrated in Figure 6. Each ray can be modeled as a function of the car's position and orientation within the hallway. As shown in the figure, there are three regions the car can be in, depending on how many walls can be reached by LiDAR. We present the model for Region 2 only, as the other regions are special cases. We consider a LiDAR scan with a 230° field of view and a 5-m range. Let $\alpha_1, \ldots, \alpha_{21}$ denote the relative angles for each ray with respect to the car's heading, i.e., $\alpha_1 = -115, \alpha_2 = -103.5, \ldots, \alpha_{21} = 115$. One can determine which wall each LiDAR ray hits by comparing the α_i for that ray with the relative angles to the two corners of that turn, θ_l and θ_r in Figure 6. The model for each ray $y_k^i, i \in \{1, \dots, 21\}$ is given as follows:

$$y_k^i = \begin{cases} d_k^r / cos(90 + \theta_k + \alpha_i) & \text{if } \theta_k + \alpha_i \leq \theta_r \\ d_k^b / cos(180 + \theta_k + \alpha_i) & \text{if } \theta_r < \theta_k + \alpha_i \leq -90 \\ d_k^t / cos(\theta_k + \alpha_i) & \text{if } -90 < \theta_k + \alpha_i \leq \theta_l \\ d_k^l / cos(90 - \theta_k - \alpha_i) & \text{if } \theta_l < \theta_k + \alpha_i, \end{cases} \tag{14}$$

where k is the sampling step (the sampling rate is assumed to be 10 Hz) and $d_k^t, d_k^b, d_k^l, d_k^r$ are distances to the four walls, as illustrated in Figure 6, and can be derived from the car's position (x, y).[8] Note that this observation model is quite challenging, both due to its non-linear and hybrid nature. In particular, if the reachable set for the car's state is large, then a given ray might reach different walls, which would mean that different paths through the hybrid system are taken—in this case, all paths need to be verified separately, thereby introducing a combinatorial aspect to the verification task.

As described in our preliminary work [21], we trained multiple controllers using both DDPG and a twin delayed deep deterministic policy gradient (TD3) algorithm [17]. For comparison, we only use one of the controllers, namely a NN with two hidden layers (with tanh activations) with 64 neurons per layer and a single-neuron output layer with a tanh activation. The safety property to be verified is that the car does not get within 0.3 m of either wall for 7s (which is enough to make the right turn and get roughly to the middle of the next hallway). The initial condition is $x(0) \in [-0.1, 0.1]$, i.e., a 0.2-m interval in the middle of the first hallway.

7 VERIFICATION RESULTS

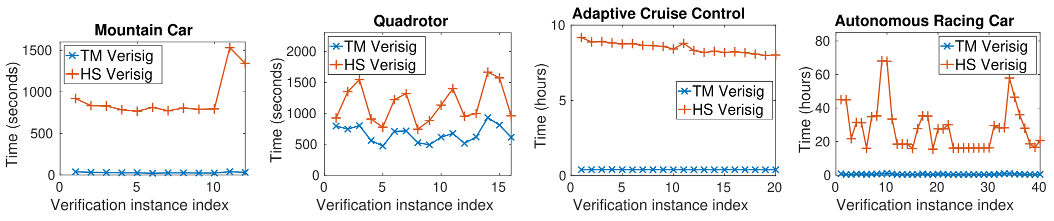
This section presents the comparison between the hybrid-system based version of Verisig and the improved, TM-based, version. The next section provides a separate comparison between both versions of Verisig and an MILP approach to verification.

The comparison in this section is purely based on the time it takes to verify the safety properties in the different benchmarks. The next section will also explore the quality of approximation of reachable sets obtained using the different algorithms. A common theme across all benchmarks is that the initial conditions need to be subdivided into smaller sets to maintain a small approximation error. If the initial condition is too large, then the uncertainty can magnify over time, thus making it impossible to verify the safety property (even if it is true). Thus, for each benchmark, we (manually) partition the initial set and present the verification times for each instance. Exploring an automated refinement procedure of the initial set is left for future work.

7.1 Mountain Car

As noted in Section 6, the initial condition for MC is $p_0 \in [-0.6, -0.4]$. During the verification, we found a counter-example at $p_0 = -0.6$, i.e., the car goes up the hill with a reward of less than 90. That is why, we only verify the property over the initial set $p_0 \in [-0.59, -0.4]$, which is partitioned as follows: [−0.59, −0.58], [−0.58, −0.57], [−0.57, −0.55], [-0.55, -0.53], [−0.53,−0.5], [−0.5, −0.48], [−0.48, −0.45], [−0.45, −0.43], [−0.43, −0.42], [−0.42, −0.415], [−0.41, −0.4]. The uneven partitioning is caused by the fact that the NN takes quite different actions from different initial conditions. Notably, when started from the right end of the initial set, the car hits the left boundary of the environment, thereby causing a branch in the hybrid model such that each branch needs to be verified separately.

[8]If $\theta_k + \alpha_i \notin [-180, 180]$, then $\theta_k + \alpha_i$ needs to be normalized by adding/subtracting 360.



(a) Mountain Car. (b) Quadrotor. (c) Adaptive Cruise Control. (d) F1/10 Car.

Fig. 7. Comparison in terms of verification times between the two versions of Verisig, the hybrid-system based one (HS Verisig), and the TM-based one (TM Verisig).

The verification times for all the instances are presented in Figure 7(a). The TM version of Verisig is roughly an order of magnitude faster for all instances—most instances take about half a minute to verify with the TM approach and more than 10 minutes with the original approach. The last two instances took considerably longer to verify due to the branch in the hybrid system.

7.2 Quadrotor

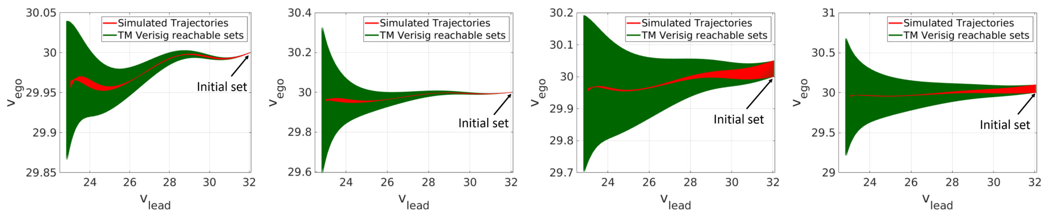
The initial condition for this benchmark, $(p_x^r(0), p_y^r(0)) \in [-0.05, 0.05] \times [-0.05, 0.05]$, was subdivided into 16 equal-size subintervals, by offsets of 0.025 along each axis. Note that the "bang-bang" nature of the controller means that there is an exponentially increasing number of paths (in the number of control sampling steps) that need to be verified. This is true, because, depending on the uncertainty, at each step the controller might take a different discrete action. Thus, this benchmark is challenging not only due to the presence of the NN, but also due to the hybrid plant model.

The verification times are presented in Figure 7(b). Once again, the TM approach is significantly faster for all instances. The differences are less pronounced in this case, since verifying the plant dynamics, which is the same for both approaches, takes a bigger fraction of the entire computation.

7.3 Adaptive Cruise Control

The initial set for ACC is subdivided into 20 instances, by offsets of 1 along the x_{lead} axis. The verification times are shown in Figure 7(c). Similarly to the MC case study, the TM approach is an order of magnitude faster, with verification times of roughly 25 minutes, as opposed to over 9 hours for the original approach. Note that this model is not hybrid, so the verification times do not vary greatly across instances for each method.

The higher dimensionality of the ACC benchmark's initial set also allows us to explore another issue related to scalability, namely how much uncertainty can be tolerated along different dimensions of the state-space. Answering this question is important, as it would suggest better ways to partition the initial set than the uniform partitioning used in this work. In particular, we compare how much initial uncertainty can be tolerated along the lead car position, p_{lead}, dimension vs. the ego car velocity, v_{ego}, dimension. Figure 8 provides the reachable sets, as projected to the $v_{lead} - v_{ego}$ plain, computed by TM Verisig for different combinations of initial conditions for these two variables (all other states have the same initial conditions as before). As can be seen in the figure, while the true reachable sets do not vary greatly across setups, the reachable sets computed by Verisig are much more affected by velocity uncertainty, with Figure 8(d) having the largest degree of overapproximation error. This suggests that higher-order terms (e.g., velocity) have a much bigger effect on the accuracy of reachable sets than lower-order terms (e.g., position).



(a) $p_{lead}(0) \in [90, 91]$, $v_{ego}(0) = 30$. (b) $p_{lead}(0) \in [90, 93]$, $v_{ego}(0) = 30$. (c) $p_{lead}(0) = 90$, $v_{ego} \in [30, 30.05]$. (d) $p_{lead}(0) = 90$, $v_{ego} \in [30, 30.1]$.

Fig. 8. Reachable sets produced by TM Verisig for various initial conditions for the ACC benchmark chosen to illustrate the relative importance of velocity uncertainty over position uncertainty. Note that the reachable sets progress from right to left, i.e., v_{lead} decreases with time.

7.4 Autonomous Racing Car

The initial condition for the car, $x(0) \in [-0.1, 0.1]$, is subdivided into 40 regions, by offsets of 0.005. We emphasize again that this case study is not only challenging due to the larger NNs but also due to the hybrid LiDAR model. Please consult our preliminary work [21] for an exhaustive discussion of the various challenges presented by this case study.

The comparison is presented in Figure 7(d). We observe the same trend of the TM approach being significantly faster, with verification times of less than an hour, as compared with some instances taking more than a day with the original approach. It is also worth noting that different instances can take drastically different times to verify, which is due to the LiDAR model challenges.

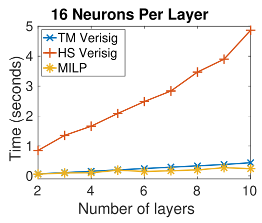
In summary, the TM version of Verisig is able to verify the same properties as the original version, but with an order of magnitude improvement in scalability. We observe similar trends across different plant models and NN architectures. The next section demonstrates that TM Verisig also results in better approximations than the original version, in addition to being faster.

8 COMPARISON WITH AN MILP APPROACH

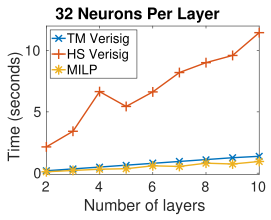
For better evaluation, this section provides a comparison with an alternative verification approach for sigmoid/tanh NNs, namely an MILP-based approach. The MILP approach was suggested in prior work [12] as a possible way to verify sigmoid/tanh NNs, similar to Sherlock, i.e., the MILP approach used for ReLUs [12]. A comparison with another recently released tool for closed-loop verification, NNV [47], is provided in Section 9.

The main idea of the approach is to transform the NN into an MILP. Since sigmoids are not linear, they could be bounded from above and below by piece-wise linear functions. Once such functions are obtained, one could formulate the MILP by adding a binary variable for each linear piece of each piece-wise linear function and use the big M method, as described in prior work [12]. Finally, one could use an optimized solver such as Gurobi [36] to verify safety about the NN's outputs given (linear) bounds on the inputs. Note that this approach can only be used on the NN in isolation, so we only provide a comparison in terms of NNs, ignoring the plant. To verify the closed-loop system, one could combine the output of Gurobi with polynomial regression to obtain a TM for the NN, similar to the way Sherlock was extended to closed-loop systems [11].

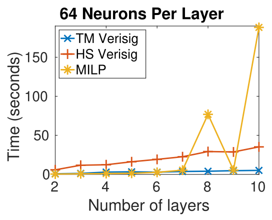
The main consideration when implementing the MILP approach is how many linear pieces to use to approximate the sigmoid. Adding more pieces reduces the approximation error but it significantly complicates the MILP (since the solver may need to search over all combinations of binary variables). In this comparison, we use roughly 100 pieces, which results in an approximation error of around 10^{-4} per sigmoid. As the rest of this section indicates, however, while this approximation



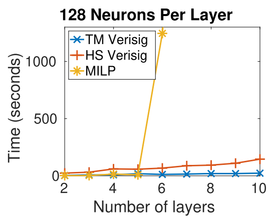
(a) 16 neurons per layer, 2 – 10 layers.



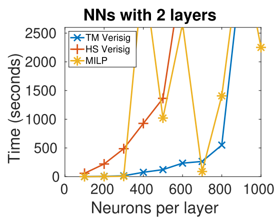
(b) 32 neurons per layer, 2 – 10 layers.



(c) 64 neurons per layer, 2 – 10 layers.



(d) 128 neurons per layer, 2 – 10 layers.



(e) 100-1000 neurons per layer, 2 layers.

Fig. 9. Comparison in terms of verification times between the two versions of Verisig and the MILP-based approach. In 9(a)–(d), the number of neurons is fixed and number of layers varies from two to 10. In (e), the number of layers is fixed to two, and the number of neurons varies from 100 to 1,000.

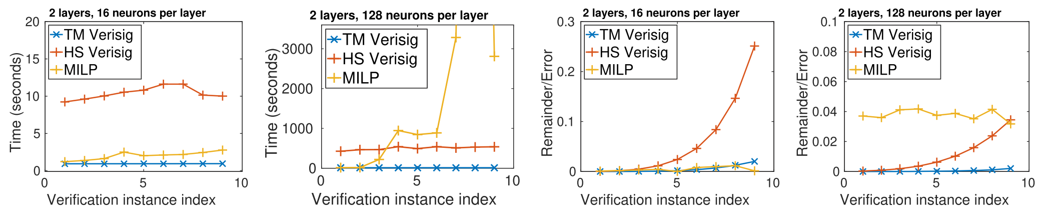
is good enough for small uncertainties and small NNs, it becomes insufficient for larger NNs and larger initial conditions, as compared with Verisig.

For fair comparison, we compare the different approaches in terms of both verification time and approximation error. We also compare them on two different benchmarks, MC and ACC. As discussed at the end of the section, the MILP approach cannot be directly applied to the autonomous racing car, which is another limitation of approaches of this type.

8.1 Comparison on MC

In the first comparison, we train NNs of increasing size on the MC benchmark and compare the verification times of the different approaches for a fixed initial set, namely $p_0 \in [-0.5, -0.48]$. In particular, we train NNs with 16, 32, 64, and 128 neurons per layer, respectively, and vary the number of layers from 2 to 10. Furthermore, we train networks of increasing width, where we fix the number of layers to two but vary the number of neurons per layer from 100 to 1,000.

The comparison is shown in Figure 9. As can be seen in the figure, the MILP approach achieves comparable times with TM Verisig for small NNs. However, as the NN size increases, both in terms of number of neurons and depth, the MILP approach suffers from the exponential complexity of the problem and is eventually even slower than the original version of Verisig. In contrast, both versions of Verisig scale linearly with the number of layers, since the same computation is performed for each layer/mode. Finally, the runtimes for all approaches increase super-linearly as the NN width is increased, as shown in Figure 9(e). While, the MILP approach can sometimes handle even the biggest NNs that were tested, TM Verisig is in general faster and more predictable. It is worth noting that, although not shown in the graphs, the memory requirements increase substantially as well, as NNs with 1,000 neurons per layer have more than one million parameters that need to be encoded.



(a) NN with two layers and 16 neurons per layer. (b) NN with two layers and 128 neurons per layer. (c) Approximation error for NN from Figure 10a. (d) Approximation error for NN from Figure 10b.

Fig. 10. Comparison in terms of verification times (Figure 10(a) and (b)) and approximation error (Figure 10(c) and (d)) between the two versions of Verisig and the MILP-based approach. In each figure, the initial set uncertainty is gradually increased with the instance index.

We also compare verification times as the initial uncertainty is increased. Specifically, we vary the initial uncertainty as follows: $p_0 \in [-0.5 \pm i * 0.005]$, $v_0 \in [\pm i * 0.001]$ for $i \in \{1, \dots, 9\}$, where the notation $[a \pm b]$ is shorthand for $[a - b, a + b]$. Figure 10(a) and (b) shows the verification times of the three approaches for two of the NNs with two layers from Figure 9, one with 16 neurons per layer and one with 128 neurons per layer. Similarly to the first comparison, the MILP approach is comparable with TM Verisig on the small NN but is greatly affected by the uncertainty on the big one and is eventually slower than the original version of Verisig as well. As before, the Verisig approaches are not greatly affected by the initial uncertainty.

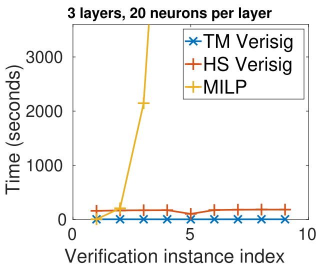
Finally, we also consider the approximation error incurred by the different methods. Performing a fair comparison is not easy, since the MILP approach only outputs bounds for the NN outputs, whereas the Verisig approaches output TMs. The metrics we use are as follows: (1) for the Verisig approaches we use the TM remainder in the NN's output, and (2) for the MILP approach, we report the approximation error as compared with ground truth (estimated numerically by sampling the two-dimensional NN input space). Using these metrics, the comparison is shown in Figure 10(c) and (d). As can be seen in these figures, the MILP error is comparable with the TM Verisig remainder for the small NN, but is significantly larger than either Verisig version for the larger NN. Note also that the TM version of Verisig results in a much smaller remainder than the original version.

8.2 Comparison on ACC

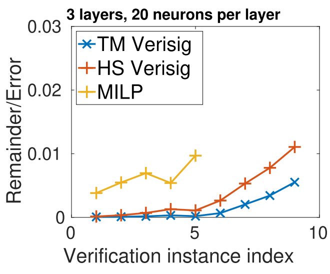
For further comparison, we also consider the ACC case study, where the NN inputs are not just states but rather functions of states. This benchmark is more challenging for the MILP approach, since interval ranges for all inputs need to be obtained, thereby losing the connection between the inputs (which is retained in a TM). We present the same comparison as in Figure 10, the main difference being that we use only one NN for comparison, namely the NN that was used in Section 7. We vary the initial condition as follows: $x_{lead}(0) \in [90 \pm i * 0.2]$, $v_{lead}(0) = [32.05 \pm i * 0.01]$, $x_{ego} \in [10.5 \pm i * 0.1]$, $v_{ego}(0) \in [30.05 \pm i * 0.01]$, for $i \in \{1, \dots, 9\}$.

The comparison is presented in Figure 11. Similarly to Figure 10, the MILP approach quickly becomes infeasible and takes more than an hour for most instances (some were terminated after 24 hours). Furthermore, the approximation error of the MILP approach is an order of magnitude larger than the TM approach (for the instances that finished execution). As in the other comparisons, the Verisig runtimes are not significantly affected by the initial uncertainty. Furthermore, the TM approach once again results in a significantly smaller remainder than the original version.

In summary, we observe similar trends in both benchmarks: The MILP approach is greatly affected by the initial uncertainty, both in terms of runtime and approximation error. In contrast, the Verisig approaches have similar runtimes regardless of the initial condition. While the Verisig



(a) Verification times.



(b) Approximation error/remainder.

Fig. 11. Comparison on the ACC benchmark in terms of verification times and approximation error between the two versions of Verisig and the MILP-based approach. In each figure, the initial set uncertainty is gradually increased with the instance index. The MILP instances were terminated after 24 hours.

approaches also result in higher remainders as the uncertainty increases, TM Verisig consistently produces much lower remainders than the other techniques.

Finally, it is worth mentioning that the MILP approach cannot be directly applied to benchmarks with observations such as the autonomous racing one. This is due to the fact that, as the number of inputs to the NN increases, simply using interval ranges for each input (without considering the relationship between inputs) will result in a vast overapproximation of the output set. This is another benefit of the TM method, as it maintains these relationships at all times.

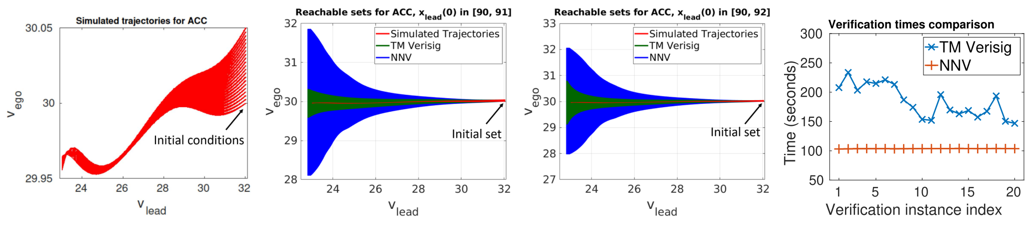
9 COMPARISON WITH NNV

For further evaluation of the TM approach, this section provides a comparison with a recently released closed-loop verification tool, namely NNV [47]. NNV computes NN reachable sets using a star set approach [46]. The star set allows one to compute exact reachable sets for NNs with ReLU activations; for NNs with smooth activations, the star sets provide tighter approximations than other convex sets such as zonotopes, as discussed in prior work [46]. Using star sets is also appealing as they can be computed in parallel, thereby enabling a great speed-up on multi-core machines. Finally, NNV uses CORA [2] to compute the plant reachable sets.

Since NNV only supports continuous (non-hybrid) plant models, we perform a comparison on the ACC benchmark. Note that both tools have parameters, e.g., the flowpipe step size in Flow* and CORA, that can be tuned to improve the approximation performance at the expense of runtime. For fair comparison, we tune both tools to have similar runtimes and compare the resulting reachable sets. Specifically, we use a 0.005-s step size in Flow* and a 0.001-s step size in CORA.

Recall from Section 6 that the initial set for the ACC benchmark is $x_{lead}(0) \in [90, 110]$, $v_{lead}(0) = [32, 32.05], \gamma_{lead}(0) = 0, x_{ego} \in [10, 11], v_{ego}(0) \in [30, 30.05], \gamma_{ego}(0) = 0$. As before, we split this initial set into subsets along the x_{lead} axis. We use the notation $S_{[a,b]}$ to denote the initial subset where $x_{lead}(0) \in [a, b]$, and the other states have the same initial conditions as before.

Figure 12 illustrates the comparison between the two tools. We first compare the reachable sets computed by the two tools for the initial set $S_{[90,91]}$. Figure 12(a) provides a number of simulated trajectories, projected to the plane of the two cars' velocities, v_{lead} and v_{ego}. As can be seen, the true reachable sets (progressing from right to left) initially contract but eventually expand again; this behavior in general makes it difficult for reachability tools to compute tight approximations. This is clearly shown in Figure 12(b), which shows the reachable sets computed by each tool. Figure 12(b) shows that both tools eventually result in large overapproximation errors, due to the complexity of the verification task. Yet, the reachable sets computed by Verisig are significantly tighter, with the NNV reachable sets being more than three times larger at the end of the scenario



(a) Simulated trajectories for the set $S_{[90,91]}$. (b) Reachable sets for the set $S_{[90,91]}$. (c) Reachable sets for the set $S_{[90,92]}$. (d) Verification times. Verification index i corresponds to initial set $S_{[89+i,90+i]}$.

Fig. 12. Comparison between TM Verisig and NNV in terms of reachable sets and verification times. Note that the reachable sets progress from right to left, i.e., v_{lead} decreases with time.

(left side of Figure 12(b)). To illustrate the effect of larger initial uncertainty, Figure 12(c) shows reachable sets for the initial set $S_{[90,92]}$. Again, Verisig produces tighter reachable sets (with NNV reachable sets being two times larger at the end of the scenario as shown on the left side of the figure), although both tools have an even bigger approximation error. Finally, Figure 12(d) shows the runtimes for both tools over all initial instances used in Section 7, i.e., $S_{[90,91]}, \ldots, S_{[109,110]}$. The figure shows that Verisig is between 50% and 100% slower per instance (for the tool parameters described above). However, it is important to emphasize we were not able to obtain significantly tighter reachable sets for NNV for any other parameter setting that we tried.

In summary, despite the challenging nature of the ACC benchmark, Verisig results in significantly tighter reachable sets (with NNV reachable sets being two-to-three times larger at the end of the scenario) at a small cost in verification time. The difference in approximation error is likely due to the fact that Verisig treats the system as a single hybrid system and propagates reachable sets in a symbolic fashion through TMs. In contrast, NNV switches between NN reachable sets, represented as star sets, and plant reachable sets, represented as polytopes in CORA, which introduces additional error.

10 RELATED WORK

Multiple techniques have been proposed to evaluate and improve a NN's robustness, both during training and post training [52]. During training, researchers have developed adversarial training [30, 33] and robust training [7, 19] methods to alleviate the NN's vulnerability to adversarial examples. Once a NN is trained, approaches exist to test the NN's sensitivity to input perturbations [37], generate new adversarial examples [26], or perform physical attacks [14]. Finally, several formal verification and robustness works have been recently proposed to verify safety properties of a *trained* NN's outputs given constraints on the inputs [12, 13, 15, 18, 24, 49, 50, 53]. These techniques exploit the specific form of NNs' activation functions, e.g., ReLUs, by adapting existing SMT [13, 24] and MILP [12] solvers or by otherwise overapproximating the NN's output set [18, 53].

A few techniques have also been developed to analyze properties of closed-loop systems with NN components. Dreossi et al. [9] and Tuncali et al. [48] develop falsification approaches by adapting existing falsifiers to the case of NN components. Verisig [22] can verify safety properties of closed-loop systems with sigmoid- and tanh-based NN controllers by transforming the NN into an equivalent hybrid system. Sun et al. [42] propose an SMT-based approach to verify properties of linear systems with ReLU-based NN controllers. Dutta et al. [11] extend their MILP-based tool [12] to the closed-loop case by approximating the NN using polynomial regression. Furthermore, NNV [47] computes NN reachable sets using a star set formulation that is exact for ReLU activations and allows for parallelization. Finally, ReachNN [20] is similar to the work by Dutta et al. [11] as it approximates the NN locally with a polynomial; however, ReachNN uses Bernstein polynomials, which allows it to achieve arbitrary approximation precision. Although ReachNN is similar to Verisig, since it is based on TMs, Verisig provides an analytic approximation of the sigmoid/tanh, whereas ReachNN (being agnostic of the specific activation function used) relies on sampling methods that may require large amounts of sampling to achieve the desired precision.

11 CONCLUSION

This article presented Verisig, a method to verify safety properties of autonomous systems with NN controllers. Verisig transforms the NN into an equivalent hybrid system and casts the problem as a hybrid system verification problem. We improved the approach by approximating each neuron with a TM such that no integration of the sigmoid "dynamics" is necessary. We analyzed the decidability of NN verification and provided conditions under which the problem is decidable. Finally, we presented an exhaustive evaluation over four benchmarks.

The novelty of the area presents multiples avenues for future work. First, it is important to develop verification approaches for image-based systems, as this is where NNs greatly outperform other learning methods. Furthermore, scalability improvements are necessary, especially to handle convolutional NNs, which are very effective for analyzing images. Finally, it would be interesting to identify verification problems in other safety-critical domains, e.g., cyber security applications.

Finally, it would be interesting to explore hybrid-system-based techniques for verifying ReLU-based NNs, since the ReLU is arguably the most common activation function in use today. One approach would be to approximate the ReLU with a smooth function (e.g., the Swish function [40]) and then apply the TM method described in this article.

REFERENCES

[1] [n.d.]. F1/10 Autonomous Racing Competition. Retrieved from http://f1tenth.org.

[2] M. Althoff. 2015. An introduction to CORA 2015. In *Proceedings of the Workshop on Applied Verification for Continuous and Hybrid Systems*. 120–151.

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

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, and J. Zhao. 2016. End to end learning for self-driving cars. *ArXiv:1604.07316*. Retrieved from https://arxiv.org/abs/1604.07316.

[5] Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. 2012. Taylor model flowpipe construction for non-linear hybrid systems. In *Proceedings of the 2012 IEEE 33rd Real-Time Systems Symposium (RTSS'12)*. IEEE, 183–192.

[6] X. Chen, E. Ábrahám, and S. Sankaranarayanan. 2013. Flow*: An analyzer for non-linear hybrid systems. In *Proceedings of the International Conference on Computer Aided Verification*. Springer, 258–263.

[7] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier. 2017. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70. 854–863.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* 12 (August 2011), 2493–2537.

[9] T. Dreossi, A. Donzé, and S. A. Seshia. 2017. Compositional falsification of cyber-physical systems with machine learning components. In *Proceedings of the NASA Formal Methods Symposium*. Springer, 357–372.

[10] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. 2015. C2E2: A verification tool for stateflow models. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 68–82.

[11] S. Dutta, X. Chen, and S. Sankaranarayanan. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM, 157–168.

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

[13] R. Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *Proceedings of the International Symposium on Automated Technology for Verification and Analysis*. Springer, 269–286.

[14] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. 2018. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1625–1634.

[15] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. 2019. Efficient and accurate estimation of lipschitz constants for deep neural networks. *ArXiv:1906.04893*. Retrieved from https://arxiv.org/abs/1906.04893.

[16] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In *Proceedings of the International Conference on Computer Aided Verification*. 379–395.

[17] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *ArXiv:1802.09477*. Retrieved from https://arxiv.org/abs/1802.09477.

[18] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. 2018. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'18)*.

[19] Matthias Hein and Maksym Andriushchenko. 2017. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems*. 2266–2276.

[20] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. 2019. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Trans. Embedd. Comput. Syst.* 18, 5s (2019), 1–22.

[21] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. 2020. Case study: Verifying the safety of an autonomous racing car with a neural network controller. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. ACM, 1–7.

[22] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. 2019. Verisig: Verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM, 169–178.

[23] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *Proceedings of the 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC'16)*. IEEE, 1–10.

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

[25] S. Kong, S. Gao, W. Chen, and E. Clarke. 2015. dReach: δ-reachability analysis for hybrid systems. In *Proceedings of the International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems*. Springer, 200–205.

[26] A. Kurakin, I. Goodfellow, and S. Bengio. 2016. Adversarial examples in the physical world. *ArXiv:1607.02533*. Retrieved from https://arxiv.org/abs/1607.02533.

[27] G. Lafferriere, G. J. Pappas, and S. Yovine. 1999. A new class of decidable hybrid systems. In *Proceedings of the International Workshop on Hybrid Systems: Computation and Control*. 137–151.

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. Continuous control with deep reinforcement learning. *ArXiv:1509.02971*. Retrieved from https://arxiv.org/abs/1509.02971.

[29] D. M. Lopez, P. Musau, H. Tran, S. Dutta, T. J. Carpenter, R. Ivanov, and T. T. Johnson. 2019. ARCH-COMP19 category report: Artificial intelligence/neural network control systems (AINNCS) for continuous and hybrid systems plants. *EPiC Ser. Comput.* 61 (2019), 103–119.

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *ArXiv:1706.06083*. Retrieved from https://arxiv.org/abs/1706.06083.

[31] Kyoko Makino and Martin Berz. 2007. Suppression of the wrapping effect by Taylor model-based verified integrators: The single step. *Int. J. Pure Appl. Math.* 36, 2 (2007), 175.

[32] Ali A. Minai and Ronald D. Williams. 1993. On the derivatives of the sigmoid. *Neural Netw.* 6, 6 (1993), 845–853.

[33] Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *Proceedings of the International Conference on Machine Learning*. 3575–3583.

[34] Andrew William Moore. 1990. *Efficient Memory-based Learning for Robot Control*. Ph.D. Dissertation. University of Cambridge.

[35] OpenAI. [n.d.]. OpenAI Gym. Retrieved from https://gym.openai.com.

[36] Gurobi Optimization. [n.d.]. Gurobi Optimizer. Retrieved from https://gurobi.com.

[37] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 1–18.

[38] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle. 2017. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?. In *Proceedings of the Intelligent Vehicles Symposium*. IEEE, 812–818.

[39] Rajesh Rajamani. 2011. *Vehicle Dynamics and Control*. Springer Science & Business Media.

[40] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for activation functions. ArXiv:1710.05941 (2017). Retrieved from https://arxiv.org/pdf/1710.05941.

[41] V. R. Royo, D. Fridovich-Keil, S. Herbert, and C. J. Tomlin. 2018. Classification-based approximate reachability with guarantees applied to safe trajectory tracking. *ArXiv:1803.03237*. Retrieved from https://arxiv.org/abs/1803.03237.

[42] X. Sun, H. Khedr, and Y. Shoukry. 2019. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM, 147–156.

[43] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, et al. 2013. Intriguing properties of neural networks. *ArXiv:1312.6199*. Retrieved from https://arxiv.org/abs/1312.6199.

[44] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1701–1708.

[45] A. Tarski. 1998. A decision method for elementary algebra and geometry. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, 24–84.

[46] H. Tran, S. Bak, W. Xiang, and T. T. Johnson. 2020. Verification of deep convolutional neural networks using ImageStars. In *Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV'20)*. Springer.

[47] H. Tran, F. Cai, D. M. Lopez, P. Musau, T. T. Johnson, and X. Koutsoukos. 2019. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Trans. Embedd. Comput. Syst.* 18, 5s (2019), 105.

[48] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. 2018. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. *ArXiv:1804.06760*. Retrieved from https://arxiv.org/abs/1804.06760.

[49] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. 2018. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*. 6367–6377.

[50] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. Boning, and I. Dhillon. 2018. Towards fast computation of certified robustness for ReLU networks. In *Proceedings of the International Conference on Machine Learning*. 5273–5282.

[51] A. J. Wilkie. 1997. Schanuel's conjecture and the decidability of the real exponential field. In *Algebraic Model Theory*. Springer, 223–230.

[52] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. Rosenfeld, and T. T Johnson. 2018. Verification for machine learning, autonomy, and neural networks survey. *ArXiv:1810.01989*. Retrieved from https://arxiv.org/abs/1810.01989.

[53] W. Xiang, H. D. Tran, and T. T. Johnson. 2017. Output reachable set estimation and verification for multi-layer neural networks. *ArXiv:1708.03322*. Retrieved from https://arxiv.org/abs/1708.03322.

Received February 2020; revised July 2020; accepted August 2020