Verisig: verifying safety properties of hybrid systems with neural network controllers

Radoslav Ivanov
University of Pennsylvania, rivanov@cis.upenn.edu

James Weimer
University of Pennsylvania, weimerj@cis.upenn.edu

Rajeev Alur
University of Pennsylvania, alur@cis.upenn.edu

George J. Pappas
University of Pennsylvania, pappasg@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Follow this and additional works at: https://repository.upenn.edu/cis_papers

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2019), Montreal, Canada, April, 16-18, 2019

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_papers/859
For more information, please contact repository@pobox.upenn.edu.
Verisig: verifying safety properties of hybrid systems with neural network controllers

Abstract
This paper presents Verisig, a hybrid system approach to verifying safety properties of closed-loop systems using neural networks as controllers. We focus on sigmoid-based networks and exploit the fact that the sigmoid is the solution to a quadratic differential equation, which allows us to transform the neural network into an equivalent hybrid system. By composing the network's hybrid system with the plant's, we transform the problem into a hybrid system verification problem which can be solved using state-of-the-art reachability tools. We show that reachability is decidable for networks with one hidden layer and decidable for general networks if Schanuel's conjecture is true. We evaluate the applicability and scalability of Verisig in two case studies, one from reinforcement learning and one in which the neural network is used to approximate a model predictive controller.

Keywords
Neural Network Verification, Hybrid Systems with Neural Network Controllers, Learning-Enabled Components

Disciplines
Computer Engineering | Computer Sciences

Comments
22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2019), Montreal, Canada, April, 16-18, 2019
Verisig: verifying safety properties of hybrid systems with neural network controllers

Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, Insup Lee
University of Pennsylvania
Philadelphia, Pennsylvania
{rivanov,weimerj,alur,pappasg,lee}@seas.upenn.edu

ABSTRACT

This paper presents Verisig, a hybrid system approach to verifying safety properties of closed-loop systems using neural networks as controllers. We focus on sigmoid-based networks and exploit the fact that the sigmoid is the solution to a quadratic differential equation, which allows us to transform the neural network into an equivalent hybrid system. By composing the network’s hybrid system with the plant’s, we transform the problem into a hybrid system verification problem which can be solved using state-of-the-art reachability tools. We show that reachability is decidable for networks with one hidden layer and decidable for general networks if Schanuel’s conjecture is true. We evaluate the applicability and scalability of Verisig in two case studies, one from reinforcement learning and one in which the neural network is used to approximate a model predictive controller.

CCS CONCEPTS
• Theory of computation → Timed and hybrid models; • Software and its engineering → Formal methods; • Computing methodologies → Neural networks;

KEYWORDS

Neural Network Verification, Hybrid Systems with Neural Network Controllers, Learning-Enabled Components

1 INTRODUCTION

In recent years, deep neural networks (DNNs) have been successfully applied to multiple challenging tasks such as image processing [29], reinforcement learning [20], learning model predictive controllers (MPCs) [26], and games such as Go [27]. These results have inspired system developers to use DNNs in safety-critical Cyber-Physical Systems (CPS) such as autonomous vehicles [3] and air traffic collision avoidance systems [14]. At the same time, several recent incidents (e.g., Tesla [1] and Uber [3] autonomous driving crashes) have underscored the need to better understand DNNs and verify properties of the composed system’s reachable space by encoding the DNN as an input to efficient SMT/MILP solvers. These approaches exploit the piecewise-linear nature of the rectified linear units (ReLUs) used in many DNNs and scale well by encoding the DNN as an input to efficient SMT/MILP solvers. As a result, existing tools can be used on reasonably sized DNNs, i.e., DNNs with several layers and a few hundred neurons per layer.

Although the SMT- and MILP-based approaches work well for the verification of properties of the DNN itself, these techniques cannot be straightforwardly extended to closed-loop systems using DNNs as controllers. Specifically, the non-linear dynamics of a typical CPS plant cannot be captured by these frameworks except for special cases such as discrete-time linear systems. While it is in theory possible to also approximate the plant dynamics with a ReLU-based DNN and verify properties about it, it is not clear how to relate properties of the approximating system to properties of the actual plant. As a result, it is challenging to use existing techniques to reason about the safety of the overall system.

To overcome this limitation, we investigate an alternative approach, named Verisig, that allows us to verify properties of the closed-loop system. In particular, we consider CPS using sigmoid-based DNNs instead of ReLU-based ones and use the fact that the sigmoid is the solution to a quadratic differential equation. This allows us to transform the DNN into an equivalent hybrid system such that a DNN with $L$ layers and $N$ neurons per layer can be represented as a hybrid system with $L + 1$ modes and $2N$ states. In turn, we compose the DNN’s hybrid system with the plant’s and verify properties of the composed system’s reachable space by using existing reachability tools such as dReach [17] and Flow* [4].
To analyze the feasibility of the proposed approach, we show that the DNN reachability problem (i.e., checking whether the DNN’s outputs lie in some set given constraints on the inputs) can be transformed into a real-arithmetic property with transcendental functions, which is decidable if Schanuel’s conjecture is true [34]. We also prove that reachability is decidable for DNNs with one hidden layer, given interval constraints on the inputs. Finally, by casting the problem in the dReach framework, we also show that reachability is $\delta$-decidable for general DNNs [10].

To evaluate the applicability of Verisig, we consider two case studies, one from reinforcement learning (RL) and one where a DNN is used to approximate an MPC with safety guarantees. DNNs are increasingly being used in these domains, so it is essential to be able to verify properties of interest about such systems. We trained a DNN on a benchmark RL problem, Mountain Car (MC), and verified that the DNN achieves its control task (i.e., drive an underpowered car up a hill) within the problem constraints. In the MPC approximation setting, we used an existing technique to approximate an MPC with a DNN [26] and verified that a DNN-controlled quadrotor reaches its goal without colliding into obstacles.

Finally, we evaluate Verisig’s scalability, as used with Flow*, by training DNNs of increasing size on the MC problem. For each DNN, we record the time it takes to compute the output’s reachable set. For comparison, we implemented a piecewise-linear approach to approximate each sigmoid as suggested in prior work [7]; in this setting, the problem is cast as an MILP that can be solved by an optimizer such as Gurobi [24]. We observe that, at similar levels of approximation, the MILP-based approach is faster than Verisig+Flow* for small DNNs and DNNs with few layers. However, the MILP-based approach’s runtimes increase exponentially for deeper networks whereas Verisig+Flow* scales linearly with the number of layers since the same computation is run for each layer. This is another positive feature of our technique since deeper networks are known to learn more efficiently than shallow ones [31].

In summary, this paper has three contributions: 1) we develop an approach to transform a DNN into a hybrid system, which allows us to cast the closed-loop system verification problem into a hybrid system verification problem; 2) we show that the DNN reachability problem is decidable for DNNs with one hidden layer and decidable for general DNNs if Schanuel’s conjecture holds; 3) we evaluate both the applicability and scalability of Verisig using two case studies.

The rest of this paper is organized as follows. Section 2 states the problem addressed in this work. Section 3 analyzes the decidability of the verification problem, and Section 4 describes Verisig. Sections 5 and 6 present the case study evaluations in terms of applicability and scalability. Section 7 provides concluding remarks.

2 PROBLEM FORMULATION

This section formulates the problem considered in this paper. We consider a closed-loop system, as shown in Figure 1, with states $x$, measurements $y$, and a controller $h$. The states and measurements are formalized in the next subsection, followed by the (DNN) controller description and the problem statement itself.

2.1 Plant Model

We assume the plant dynamics are given as a hybrid system. A hybrid system’s state space consists of a finite set of discrete modes and a finite number of continuous variables [18]. Within each mode, continuous variables evolve according to differential equations with respect to time. Furthermore, each mode contains a set of invariants that hold true while the system is in that mode. Transitions between modes are controlled by guards, i.e., conditions on the continuous variables. Finally, continuous variables can be reset during each mode transition. The formal definition is provided below.

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

2.2 DNN Controller Model

A DNN controller maps measurements $y$ to control inputs $u$ and can be defined as a function $h$ as follows: $h : \mathbb{R}^p \rightarrow \mathbb{R}^q$. To simplify the presentation, we assume the DNN is a fully connected feedforward neural network. However, the proposed technique applies to all common classes such as convolutional, residual or recurrent DNNs. As illustrated in Figure 1, a typical DNN has a layered architecture and can be represented as a composition of its $L$ layers:

$$h(y) = h_L \circ h_{L-1} \circ \cdots \circ h_1(y),$$
where each hidden layer $h_i$, $i \in \{1, \ldots, L-1\}$, has an element-wise (with each element called a neuron) non-linear activation function:

$$h_i(y) = a(W_i y + b_i).$$

Each $h_i$ is parameterized by a weight matrix $W_i$ and an offset vector $b_i$. The most common types of activation functions are:

- RelU: $a(y) := \text{ReLU}(y) = \max(0, y)$,
- sigmoid: $a(y) := \sigma(y) = 1/(1 + e^{-y})$,
- hyperbolic tangent: $a(y) := \tanh(y) = (e^y - e^{-y})/(e^y + e^{-y}).$

As argued in the introduction, and different from most existing works that assume ReLU activation functions, this work considers sigmoid and tanh activation functions (which also fall in the broad class of sigmoidal functions). Finally, the last layer $h_L$ is linear:

$$h_L(y) = W_L y + b_L,$$

which is parameterized by a matrix $W_L$ and a vector $b_L$.

During training, the parameters $(W_1, b_1, \ldots, W_L, b_L)$ are learned through an optimization algorithm (e.g., stochastic gradient descent [11]) used on a training set. In this paper, we assume the DNN is already trained, i.e., all parameters are known and fixed.

### 2.3 Problem Statement

Given the plant model and the DNN controller model described in this section, we identify two verification problems. The first one is the reachability problem for the DNN itself.

**Problem 1.** Let $h$ be a DNN as described in Section 2.2. The DNN verification problem, expressed as property $\phi_{\text{dnn}}$, is to verify a property $\psi_{\text{dnn}}$ on the DNN’s outputs $u$ given constraints $\xi_{\text{dnn}}$ on the inputs $x$:

$$\phi_{\text{dnn}}(y, u) \equiv (\xi_{\text{dnn}}(y) \land h(y) = u) \Rightarrow \psi_{\text{dnn}}(u). \quad (1)$$

Problem 2 is to verify a property of the closed-loop system.

**Problem 2.** Let $S = h \sqcap H_P$ be the composition of a DNN controller $h$ (Section 2.2) and a plant $P$, modeled with a hybrid system $H_P$ (Section 2.1). Given a property $\xi$ on the initial states $x_0$ of $P$, the problem, expressed as property $\phi$, is to verify a property $\psi$ of the reachable states of $P$:

$$\phi(x_0, x(t)) \equiv (\xi(x_0) \Rightarrow \psi(x(t))), \forall t \geq 0. \quad (2)$$

Our approach to Problem 1, namely transforming the DNN into an equivalent hybrid system, also presents a solution to Problem 2 since we can compose the DNN’s hybrid system with the plant’s and can use existing hybrid system verification tools.

**Approach.** We approach Problem 1 by transforming $h$ into a hybrid system $H_h$ such that if $x_0$ is an initial condition of $H_h$, then the only reachable state in the last mode of $H_h$ is $h(x_0)$. Problem 2 is addressed by verifying safety for the composed hybrid system $H_h \sqcap H_P$.

### 3 ON THE DECIDABILITY OF SIGMOID-BASED DNN REACHABILITY

Before describing our approach to the problems stated in Section 2, a natural question to ask is whether these problems are decidable. The answer is not obvious due to the non-linear nature of the sigmoid. This section shows that if the DNN’s inputs and outputs are given as a real-arithmetic property, then reachability can be stated as a real-arithmetic property with transcendental functions, which is decidable if Schanuel’s conjecture is true [34]. Furthermore, we prove decidability for the case of NNS with a single hidden layer, under mild assumptions on the DNN parameters. Finally, we argue that by casting the DNN verification problem into a hybrid system verification problem, we obtain a $\delta$-decidable problem [10].

#### 3.1 DNNs with multiple hidden layers

As formalized in Section 2, the reachability property of a DNN $h$ with inputs $y$ and outputs $u$ has the general form:

$$\phi(y, u) \equiv (\xi(y) \land h(y) = u) \Rightarrow \psi(u), \quad (3)$$

where $\xi$ and $\psi$ are properties on the real numbers. Verifying properties on the real numbers is undecidable in general. A notable exception is first-order logic formulas over $(\mathbb{R}, <, +, - , \times, 0, 1)$, i.e., the language where $<$ is the relation, $+$, $-$, and $\times$ are functions, and 0 and 1 are the constants [30]; we denote such formulas by $\mathcal{R}$-formulas. Intuitively, $\mathcal{R}$-formulas are first-order logic statements whose variables are polynomial functions of the variables with integer coefficients. Example $\mathcal{R}$-formulas are $\forall x \forall y : xy > 0$, $\exists x : x^2 = 2 - 0$, and $\exists w : w + z = 0$.

Another relevant language is $(\mathbb{R}, <, +, - , \times, 0, 1)$, which also includes exponentiation; we denote these formulas by $\mathcal{R}_{\text{exp}}$-formulas. Although it is an open question whether verifying $\mathcal{R}_{\text{exp}}$-formulas is decidable, it is known that decidability is connected to Schanuel’s conjecture [34]. Schanuel’s conjecture concerns the transcendence degree of certain field extensions of the rational numbers and, if true, would imply that verifying $\mathcal{R}_{\text{exp}}$-formulas is decidable [34].

We focus on the case where $\xi$ and $\psi$ are $\mathcal{R}$-formulas. The exponential in the sigmoid means that $\phi$, however, is not a $\mathcal{R}$-formula. We show below that $\phi$ is in fact an $\mathcal{R}_{\text{exp}}$-formula, which implies that DNN reachability is decidable if Schanuel’s conjecture is true [34].

**Proposition 3.1.** Let $h : \mathbb{R}^P \rightarrow \mathbb{R}^B$ be a sigmoid-based DNN with $L - 1$ hidden layers (with $N$ neurons each) and rational parameters. The property $\phi(y, u) \equiv (\xi(y) \land h(y) = u) \Rightarrow \psi(u)$, where $\xi$ and $\psi$ are $\mathcal{R}$-formulas, is an $\mathcal{R}_{\text{exp}}$-formula.

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

$$\phi_0(y, u) \equiv (\xi(y) \land h_1^1 = \frac{1}{1 + \exp(-(w_1^1)^T y - b_1^1)} \wedge \ldots$$

$$\land h_1^N = \frac{1}{1 + \exp(-(w_1^N)^T y - b_1^N)} \wedge \ldots$$

$$\land h_{L-1}^1 = \frac{1}{1 + \exp(-(w_{L-1}^1)^T h_{L-2} - b_{L-1}^1)} \wedge \ldots$$

$$\land h_{L-1}^N = \frac{1}{1 + \exp(-(w_{L-1}^N)^T h_{L-2} - b_{L-1}^N)} \wedge \ldots$$

$$\land u = W_L [h_{L-1}^1, \ldots, h_{L-1}^N]^\top + b_L,$$

where $(w_j^i)^\top$ 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 $[W_l]_{jk} = p_{jk} / q_{jk}$, with $p_{jk}$ and $q_{jk} > 0$ integers, and let $d_0 =$

---

1The last layer is by convention a linear layer, although it could also have a non-linear activation, as shown in the Mountain Car case study.

2Note that the results presented in this section hold for DNNs with sigmoid activation functions, but similar results can be shown for tanh.
For removing fractions from the exponents, we add extra variables \( z_i \) and \( v_i' \) and arrive at an equivalent property \( \phi_{2a} \), which is an \( R_{\exp} \)-formula since all denominators are \( R_{\exp} \)formulas:

\[
\phi_{2a}(y,u) \equiv \left( \xi(y) \land z_0 d_0 = y \land h_1 = \frac{1}{1 + \exp(-r_1^N)^{T}(z_0 - v_1)} \right) \land \ldots \\
\land h_1^N = \frac{1}{1 + \exp(-(r_1^N)^T z_0 - v_1)} \land v_1^1 = b_1^1 \land \ldots \land v_1^N = b_1^N \land \ldots \\
\land z_{L-2} d_0 = h_{L-2} \land h_{L-1}^1 = \frac{1}{1 + \exp(-(r_{L-1}^N)^T z_{L-2} - v_{L-1}^1)} \land \ldots \\
\land h_{L-1}^N = \frac{1}{1 + \exp(-(r_{L-1}^N)^T z_{L-2} - v_{L-1}^N)} \land v_{L-1}^1 = b_{L-1}^1 \land \ldots \land v_{L-1}^N = b_{L-1}^N \land \ldots
\]

where \( r_i' = w_i'd_0 \) are vectors of integers; \( v_i' = b_i' \) are \( R \)-formulas since \( b_i' \) are rational. \( \square \)

**Corollary 3.2** ([34]). If Schanuel’s conjecture holds, then verifying the property \( \phi(y,u) \equiv \left( \xi(y) \land h(y) = u \right) \Rightarrow \psi(u) \) is decidable under the conditions stated in Proposition 3.1.

**Remark.** Note that by transforming the DNN into an equivalent hybrid system (as described in Section 4), we show that DNN reachability is \( \delta \)-decidable as well [10]. Intuitively, \( \delta \)-decidability means that relaxing all constraints by a rational \( \delta \) results in a decidable problem; as shown in prior work [10], reachability is \( \delta \)-decidable for hybrid systems with dynamics given by Type 2 computable functions, which is large class of functions that contains the sigmoid.

### 3.2 Neural Networks with a single hidden layer

Regardless of whether Schanuel’s conjecture holds, we can show that DNN reachability is decidable for networks with a single hidden layer. In particular, assuming interval bounds are given for each input, it is possible to transform the reachability property into an \( R \)-formula, thus showing that verifying reachability is decidable.

**Theorem 3.3.** Let \( h: \mathbb{R}^p \rightarrow \mathbb{R}^q \) be a sigmoid-based neural network with rational parameters and with one hidden layer (with \( N \) neurons), i.e., \( h(x) = W_0(\sigma(W_1 x + b_1)) + b_2 \). Let \( [W_1]_{ij} = p_{ij}/q_{ij} \) and let \( d_0 = q_1 q_2 \cdots q_{Np} \). Consider the property

\[
\phi(y,u) \equiv \exists y \in I_y \land u = h(y) \Rightarrow \psi(u),
\]

where \( y = \begin{bmatrix} y_1, \ldots, y_p \end{bmatrix}^T \in \mathbb{R}^p, u = \begin{bmatrix} u_1, \ldots, u_q \end{bmatrix}^T \in \mathbb{R}^q, \psi \) is an \( R \)-formula, and \( I_y = [a_1, b_1] \times \cdots \times [a_p, b_p] \subseteq \mathbb{R}^p \), i.e., the Cartesian product of \( p \) one-dimensional intervals. Then verifying \( \phi(y,u) \) is decidable if, for all \( i \in \{1, \ldots, N\} \) and \( j \in \{1, \ldots, p\} \), \( e_i, e_i^T, e_i^T, e_i^{T,j} \) are rational, i.e., \( b_i = \ln(b_i^1), a_j = d_0 \ln(a_j'), \beta_j = d_0 \ln(\beta_j') \) for some rational numbers \( b_i^1, a_j', \beta_j' \).

**Proof.** The proof technique borrows ideas from [18]. It suffices to show that \( \phi(y,u) \) is an \( R \)-formula. Since \( \psi(u) \) is an \( R \)-formula, we focus on the left-hand side of the implication, call it \( \phi_0(y,u) \):

\[
\phi_0(y,u) \equiv y \in I_y \land h_1 = \frac{1}{1 + \exp(-(-w_1)^T y - b_1^1)} \land \ldots \\
\land h_1^N = \frac{1}{1 + \exp(-(-w_1)^T y - b_1^N)} \land u = W_2[h_1^1, \ldots, h_1^N]^T + b_2,
\]

where \( (w_1')^T \) is row \( i \) of \( W_1 \). Note that the last constraint in \( \phi_0(y,u) \), call it \( p(u) \), is an \( R \)-formula. To remove fractions from the exponents, we change the limits of \( y \). Consider the property

\[
\phi_2(y,u) \equiv y \in I_y^0 \land h_1 = \frac{1}{1 + \exp(-r_1^0)^T y - b_1^0} \land \ldots \\
\land h_1^N = \frac{1}{1 + \exp(-r_1^0)^T y - b_1^N} \land p(u),
\]

where \( I_y^0 = \{a_1/d_0, \beta_1/d_0\} \times \cdots \times \{a_p/d_0, \beta_p/d_0\} \) and each \( r_1^i = d_0 w_i^i \) is a vector of integers. Note that \( \phi_0(y,u) \equiv \phi_{2a}(y,u) \), since a change of variables \( z = y/d_0 \) implies that \( z \in I_y^0 \) if \( y \in I_y \). To remove exponentials from the constraints, we use their monotonicity property and transform \( \phi_2(x,y) \) into an equivalent property \( \phi_e(x,y) \):

\[
\phi_e(y,u) \equiv y \in I_y^0 \land h_1 = \frac{1}{1 + \exp(-r_1^0)^T y - b_1^0} \land \ldots \\
\land h_1^N = \frac{1}{1 + \exp(-r_1^0)^T y - b_1^N} \land p(u),
\]

where \( I_y^0 = \{e^{-\beta_1/d_0}, e^{-\alpha_1/d_0}\} \times \cdots \times \{e^{-\beta_p/d_0}, e^{-\alpha_p/d_0}\} \), and \( r_1^{ij} \) is element \( j \) of \( r_1^i \). To see that \( \phi_e(y,u) \equiv \phi_2(y,u) \), take any \( y \in I_y^0 \) and note that \( \exp(-r_1^{ij} y) = z_{r_1^{ij}}^i \), with \( z_j = e^{-\nu_j}; \) thus, \( z \in I_y^0 \).

The final step transforms the property \( \phi_e(y,u) \) into an equivalent property \( \nu(y,u) \) to eliminate negative integers \( r_1^{ij} \) in the exponents:

\[
u(y,u) \equiv y \in I_y^0 \land y_1 z_1 = 1 \land \ldots \land y_p z_p = 1 \\
\land h_1 = \frac{1}{1 + \prod_{j \in I_1^+} r_1^{ij} i \in I_1^- z_j^{-r_1^{ij}} \exp(-b_1^1)} \land \ldots \\
\land h_1^N = \frac{1}{1 + \prod_{j \in I_1^+} r_1^{ij} i \in I_1^- z_j^{-r_1^{ij}} \exp(-b_1^N)} \land p(u),
\]

where \( I_y^+ = \{e^{\alpha_1/d_0}, e^{\beta_1/d_0}\} \times \cdots \times \{e^{\alpha_p/d_0}, e^{\beta_p/d_0}\}, I_y^- = \{k \mid r_1^{ij} \geq 0\} \), and \( I_1^- = \{k \mid r_1^{ij} < 0\} \). Note that \( \phi_e(y,u) \equiv \nu(y,u) \) since for \( r_1^{ij} < 0 \), the constraint \( z_j y_j = 1 \) implies \( y_j = z_j^{-r_1^{ij}} \).

Thus, if \( e_i', e_i^{T,j}, e_i^{T,j} \) are rational for all \( i \in \{1, \ldots, p\}, j \in \{1, \ldots, N\} \), one can show that \( \nu(y,u) \) is an \( R \)-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. \( \square \)

The single-layer assumption in Theorem 3.3 is not too restrictive since DNNs with one hidden layer are still universal approximators. At the same time, the technique used to prove Theorem 3.3 cannot be applied to multiple hidden layers since the DNN becomes an \( R_{\exp} \)-formula in that case. Note that it might be possible to show more general versions of Theorem 3.3 by relaxing the interval constraints or the real-arithmetic constraints. Finally, note that the assumption on the DNN’s weights is mild since a DNN’s weights can be altered in such a way that they are arbitrarily close to the original weights while also satisfying the theorem’s requirements.
4 DNN REACHABILITY USING HYBRID SYSTEMS

Having analyzed the decidability of DNN reachability in Section 3, in this section we investigate an approach to computing the DNN’s reachable set. In particular, we transform the DNN into an equivalent hybrid system, which allows us to use existing hybrid system reachability tools such as Flow*. Sections 4.1 and 4.2 explain the transformation technique, and Section 4.3 provides an illustrative example. Finally, Section 4.4 discusses existing hybrid system reachability tools. 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.

4.1 Sigmoids as solutions to differential equations

The main observation that allows us to transform a DNN into an equivalent hybrid system is the fact that the sigmoid derivative can be expressed in terms of the sigmoid itself:

\[
\frac{d\sigma}{dx}(x) = \sigma(x)(1 - \sigma(x)).
\]

(4)

Thus, the sigmoid can be treated as a quadratic dynamical system. Since we would like to know the possible values of the sigmoid for a given set of inputs, we introduce a “time” variable \( t \) that is multiplied by the inputs. In particular, consider the proxy function

\[
g(t, x) = \sigma(tx) = \frac{1}{1 + e^{-xt}},
\]

(5)

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

\[
\frac{dg}{dt}(t, x) = \sigma(t) = xg(t, x)(1 - g(t, x)).
\]

(6)

Thus, by tracing the dynamics of \( g \) until time \( t = 1 \), we obtain exactly the value of \( \sigma(x) \); the initial condition is \( g(0, x) = 0.5 \), as can be verified from (5). While the intermediate values of the sigmoid states are not considered, the integration allows us to iteratively construct the sigmoid’s reachable set. To avoid the integration, one needs to find a computationally cheap, yet expressive, representation of this reachable set. We leave investigating this approach for future work. Since each neuron in a sigmoid-based DNN is a sigmoid function, we can use the proxy function \( g \) to transform the entire DNN into a hybrid system, as described next.

4.2 Deep Neural Networks as Hybrid Systems

Given the proxy function \( g \) described in Section 4.1, we now show how to transform a DNN into a hybrid system. Let \( N_l \) be the number of neurons in hidden layer \( h_l \) and let \( h_{ij} \) denote neuron \( j \) in \( h_l \), i.e.,

\[
h_{ij}(x) = \sigma((w^j_l)\top x + b^j_l),
\]

(7)

where \( (w^j_l)\top \) is row \( j \) of \( W_l \) and \( b^j_l \) is element \( j \) of \( b_l \). Given \( h_{ij} \), the corresponding proxy function \( g_{ij} \) is defined as follows:

\[
g_{ij}(t, x) = \sigma(t \cdot ((w^j_l)\top x + b^j_l)) = \frac{1}{1 + \exp(-t \cdot ((w^j_l)\top x + b^j_l))},
\]

where, once again, \( g_{ij}(1, x) = h_{ij}(x) \). Note that, by the chain rule,

\[
\frac{dg_{ij}}{dt}(t, x) = g_{ij}(t, x) = ((w^j_l)\top x + b^j_l)g_{ij}(t, x)(1 - g_{ij}(t, x)).
\]

(8)

Thus, for a given \( x \), the value of hidden layer \( h_i(x) \) can be obtained by tracing all \( g_{ij}(t, x) \) until \( t = 1 \) (initialized at \( g_{ij}(0, x) = 0.5 \)). This suggests that each hidden layer can be represented as a set of differential equations \( \dot{g}_{ij}(t, x) \), where \( g_{ij} \) can be considered a state.

With the above intuition in mind, we now show how to transform the DNN into an equivalent hybrid system. To simplify notation, we assume \( N = N_l \) for all \( i \in \{1, \ldots, L - 1\} \); we also assume the DNN has only one output. The proposed approach can be extended to the more general case by adding more states in the hybrid system.

The hybrid system has one mode for each DNN layer. To ensure the hybrid system is equivalent to the DNN, in each mode we trace \( g_{ij}(t, x) \) until \( t = 1 \) by using the differential equations \( \dot{g}_{ij}(t, x) \) (8). Thus, we use \( N \) continuous states, \( x^i_1, \ldots, x^i_{N} \), to represent the proxy variables for each layer; when in mode \( i \), each \( x^i_j \), \( j \in \{1, \ldots, N\} \), represents neuron \( h_{ij} \) in the DNN. We also introduce \( N \) additional continuous states (one per neuron), \( x^i_j, \ldots, x^i_N \), to track the set of the linear functions within each neuron. The \( x^i_j \) states are necessary because the inputs to each neuron are functions of the \( x^i_j \) states reached in the previous mode.

The hybrid system description is formalized in Proposition 4.1. The extra mode \( q \) is used to reset the \( x^i_j \) states to 0.5 and the \( x^i_j \) states to their corresponding values in \( q \). The two extra states, \( u \) and \( q \), are used to store the “time” and the DNN’s output, respectively. Note that \( \odot \) denotes Hadamard (element-wise) product.

**Proposition 4.1.** Let \( h : \mathbb{R}^P \rightarrow \mathbb{R}^1 \) be a sigmoid-based DNN 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 \( q \) of the following hybrid system:

- **Continuous states:** \( x^P = [x^1_1, \ldots, x^1_{N}], \ldots, [x^L_1, \ldots, x^L_{N}] \),

- **Discrete states (modes):** \( q_0, q_1, \ldots, q_L \);

- **Initial states:** \( x^P \in I_y, x^L_1 = 0, u = 0, t = 0 \);

- **Flow:**
  - \( F(q_0) = [x^P = 0, x^L_1 = 0, u = 0, t = 0] \);
  - \( F(q_i) = [x^P = x^P \odot x^L_1 \odot (I - x^P), x^L_1 = 0, u = 0, i = 1] \)
    for \( i \in \{1, \ldots, L - 1\} \);
  - \( F(q_L) = [x^P = 0, x^L_1 = 0, u = 0, t = 0] \);

- **Transitions:** \( E = \{(q_0, q_1), \ldots, (q_{L - 1}, q_L)\} \);

- **Invariants:**
  - \( I(q_0) = \{t \leq 0\} \);
  - \( I(q_i) = \{t \leq 1\} \) for \( i \in \{1, \ldots, L - 1\} \);
  - \( I(q_L) = \{t \leq 0\} \);

- **Guards:**
  - \( G(q_0, q_1) = \{t = 0\} \);
  - \( G(q_i, q_{i+1}) = \{t = 1\} \) for \( i \in \{1, \ldots, L - 1\} \);

- **Resets:**
  - \( R(q_i, q_{i+1}) = [x^P = 0.5, x^L_1 = W_{Li}x^P + b_i, t = 0] \)
    for \( i \in \{0, \ldots, L - 2\} \);
  - \( R(q_{L - 1}, q_L) = [u = W_{Lx}x^P + b_L] \).

**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.
The two-layer DNN is transformed into an equivalent three-mode hybrid system. To illustrate the transformation process from a DNN to a hybrid system, this subsection presents a small example, shown in Figure 2. Depending on the hybrid system model and the desired precision, multiple reachability methods have been developed that have proven useful in specific scenarios. In particular, Flow* [4] works by constructing flowpipe overapproximations of the dynamics in each mode using Taylor Models; although Flow* provides no decidability claims, it scales well in practice. Alternatively, dReach [17] provides \( \delta \)-decidability guarantees for Type 2 computable functions; at the same time, dReach is not as scalable and could not handle more than a few dozen variables in the examples tried in this paper. Finally, one can also use SMT solvers such as z3 [22]; yet, SMT solvers are not optimized for non-linear arithmetic and do not scale well either.

4.3 Illustrative Example

To illustrate the transformation process from a DNN to a hybrid system, this subsection presents a small example, shown in Figure 2. The two-layer DNN is transformed into an equivalent three-mode hybrid system. Since all the weights are positive and the sigmoids are monotonically increasing, the maximum value for the DNN’s output \( u \) is achieved at the maximum values of the inputs, whereas the minimum value for \( u \) is achieved at the minimum values of the inputs, i.e., \( u \geq 3\sigma(0.3 \cdot y_2 + 0.3 \cdot y_1 + 0.1) + 5\sigma(0.1 \cdot 2 + 0.5 \cdot 1 + 0.2) \) and \( u \leq 3\sigma(0.3 \cdot y_2 + 0.3 \cdot y_1 + 0.1) + 5\sigma(0.1 \cdot 3 + 0.5 \cdot 2 + 0.2) \). The same conclusion can be reached about state \( u \) in the hybrid system.

4.4 Hybrid System Verification Tools

Depending on the hybrid system model and the desired precision, there are multiple tools one might use. In the case of linear hybrid systems, there are powerful tools that scale up to a few thousand states [9]. For non-linear systems, reachability is undecidable in general, except for specific subclasses [2, 18]. Despite this negative result, multiple reachability methods have been developed that have proven useful in specific scenarios. In particular, Flow* [4] works by constructing flowpipe overapproximations of the dynamics in each mode using Taylor Models; although Flow* provides no decidability claims, it scales well in practice. Alternatively, dReach [17] provides \( \delta \)-decidability guarantees for Type 2 computable functions; at the same time, dReach is not as scalable and could not handle more than a few dozen variables in the examples tried in this paper. Finally, one can also use SMT solvers such as z3 [22]; yet, SMT solvers are not optimized for non-linear arithmetic and do not scale well either.

5 CASE STUDY APPLICATIONS

This section presents two case studies in order to illustrate possible use cases for the proposed verification approach. These case studies were chosen in domains where DNNs are used extensively as controllers, with weak worst-case guarantees about the trained network. This means it is essential to verify properties about these closed-loop systems in order to assure their functionality. The first case study, presented in Section 5.1, is Mountain Car, a benchmark problem in RL. Section 5.2 presents the second case study in which a DNN is used to approximate an MPC with safety guarantees.
5.1 A Reinforcement Learning Case Study

This subsection illustrates how Verisig could be used on a benchmark RL problem, namely Mountain Car (MC). In MC, an underpowered car must drive up a steep hill, as shown in Figure 3. Since the car does not have enough power to accelerate up the hill, it needs to drive up the opposite hill first in order to gather enough momentum. The learning task is to learn a controller that takes as input the car’s position and velocity and outputs an acceleration command. The car has the following discrete-time dynamics:

\[
P_{k+1} = P_k + v_k \\
v_{k+1} = v_k + 0.0015u_k - 0.0025 + \cos(3p_k),
\]

where \( u_k \) is the controller’s input, 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 \( v_0 = 0 \). Note that \( v_k \) is constrained to be within \([-0.07, 0.07]\) and \( P_k \) is constrained to be within \([-1.2, 0.6]\), thereby introducing (hybrid) mode switches when these constraints are violated. We consider the continuous version of the problem such that \( u_k \) is a real number between -1 and 1.

During training, the learning algorithm tries different control actions and observes a reward. The reward associated with a control action \( u_k \) is \(-0.1u_k^2\), i.e., larger control inputs are penalized more so as to avoid a “bang-bang” strategy. A reward of 100 is received when the car reaches its goal. The goal of the training algorithm is to maximize the car’s reward. The training stage typically occurs over multiple episodes (if not solved, an episode is terminated after 1000 steps) such that various behaviors can be observed. MC is considered “solved” if, during testing, the car goes up the hill with an average reward of at least 90 over 100 consecutive trials.

Using Verisig, one can strengthen the definition of a “solved” task and verify that the car will go up the hill with a reward of at least 90 starting from any initial condition. To illustrate this, we trained a DNN controller for MC in OpenAI Gym [23], a toolkit for developing and comparing algorithms on benchmark RL problems. We utilized a standard actor/critic approach for deep RL problems [19]. This is a two-DNN setting in which one DNN (the critic) learns the reward function, whereas the other one (the actor) learns the control. Once training is finished, the actor is deployed as the DNN controller for the closed-loop system. We trained a two-hidden-layer sigmoid-based DNN with 16 neurons per layer; the last layer has a tanh activation function in order to scale the output to be between -1 and 1. Note that larger networks were also trained in order to evaluate scalability, as discussed in Section 6.

To verify that the car will go up the hill with a reward of at least 90, we transform the DNN into an equivalent hybrid system using Verisig and compose it with the car’s hybrid system. We use Verisig+Flow* to verify the desired property on the composed system, given any initial position in \([-0.6, -0.4]\). Note that we split the initial condition into subsets and verify the property for each subset separately. This is necessary because the DNN takes very different actions from different initial conditions, e.g., large negative inputs when the car is started from the leftmost position and small negative inputs for larger initial conditions. This variability introduces uncertainty in the dynamics and causes large approximation errors. As part of future work, we will investigate a refinement procedure in order to optimize the partitioning of the initial condition.

<table>
<thead>
<tr>
<th>Initial condition</th>
<th>Verified</th>
<th>Reward</th>
<th># steps</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>([-0.41, -0.40])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>1336s</td>
</tr>
<tr>
<td>([-0.415, -0.41])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>1424s</td>
</tr>
<tr>
<td>([-0.42, -0.415])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>812s</td>
</tr>
<tr>
<td>([-0.43, -0.42])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>886s</td>
</tr>
<tr>
<td>([-0.45, -0.43])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>744s</td>
</tr>
<tr>
<td>([-0.50, -0.48])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>465s</td>
</tr>
<tr>
<td>([-0.53, -0.50])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>694s</td>
</tr>
<tr>
<td>([-0.55, -0.53])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>670s</td>
</tr>
<tr>
<td>([-0.57, -0.55])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>763s</td>
</tr>
<tr>
<td>([-0.58, -0.57])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 100</td>
<td>793s</td>
</tr>
<tr>
<td>([-0.59, -0.58])</td>
<td>Yes</td>
<td>&gt;= 90</td>
<td>&lt;= 112</td>
<td>1307s</td>
</tr>
<tr>
<td>([-0.6, -0.59])</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 1: Verisig+Flow* verification times (in seconds) for different initial conditions of MC. The third column shows the verified lower bound of reward. The fourth column shows the verified upper bound of the number of dynamics steps.
5.2 Using DNNs to Approximate MPCs with Safety Guarantees

To further evaluate the applicability of Verisig, we also consider a case study in which a DNN is used to approximate an MPC with safety guarantees. DNNs are used to approximate controllers for several reasons: 1) the MPC computation is not feasible at runtime [12]; 2) storing the original controller (e.g., as a lookup table) requires too much memory [14]; 3) performing reachability analysis by discretizing the state space is infeasible for high-dimensional systems [26]. We focus on the latter scenario in which the aim is to develop a DNN controller with safety guarantees.

As described in prior work [26], it is possible to train a DNN to approximate an MPC in the case of control-affine systems whose goal is to follow a piecewise-linear plan. In this case, the optimal controller is “bang-bang”, i.e., it is effectively a classifier mapping a system state to one of finitely many control actions. Once the DNN is trained, it can be used on the system instead of the MPC. Although worst-case deviations from the planner can be obtained for specific initial points, it is not known whether the DNN controller is safe for a range of initial conditions. Thus, we use Verisig to verify the safety of such a closed-loop system.

In this case study, we consider a six-dimensional control-affine model for a quadrotor controlled by a DNN and verify that the quadrotor reaches its goal without colliding into nearby obstacles. Specifically, the quadrotor follows a planner, given as a piecewise-linear system, and tries to stay as close to the planner as possible. The setup, as projected to the \( (p_x, p_y) \)-plane, is shown in Figure 5.

![Overview of the quadrotor case study](image)

Figure 5: Overview of the quadrotor case study, as projected to the \( (p_x, p_y) \)-plane. The quadrotor follows its plan in order to reach the goal (star) without colliding into obstacles (red circles). We verify that, starting from any initial condition in the black box, the quadrotor does not deviate from its plan by more than 0.32m and does not collide into obstacles.

The quadrotor and planner dynamics models are as follows:

\[
\dot{q} := \begin{bmatrix} p_x^p \\ p_y^p \\ p_z^p \\ v_x^q \\ v_y^q \\ v_z^q \\ \vdash \vdash \vdash \end{bmatrix} = \begin{bmatrix} v_x^q \\ v_y^q \\ v_z^q \\ \tan \theta \\ -\tan \phi \\ \tau - g \end{bmatrix}, \quad \dot{\theta} := \begin{bmatrix} b_x \\ b_y \\ 0 \\ 0 \end{bmatrix}, \tag{9}
\]

where \( p_x^p, p_y^p, p_z^p \) 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 \( v_x^q, v_y^q, v_z^q \) are the quadrotor and planner’s velocities, respectively; \( \theta, \phi \) and \( \tau \) are control inputs (for pitch, roll and thrust); \( g = 9.81 \text{ m/s}^2 \) 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 controller’s goal is to ensure that the quadrotor is as close to the planner as possible, i.e., stabilize the system of relative states \( r := [p_x^p, p_y^p, p_z^p, v_x^q, v_y^q, v_z^q]^\top = q - p \).

To train a DNN controller for the model in (9), we follow the approach described in prior work [26]. We sample multiple points from the state space over a horizon \( T \) and train a sequence of DNNs, one for each dynamics step (as discretized using the Runge-Kutta method). Once two consecutive DNNs have similar training error, we interrupt training and pick the last DNN as the final controller. The DNN takes a relative state as input and outputs one of eight possible actions (the “bang-bang” strategy implies there are two options per control action). We trained a two-hidden layer tanh-based DNN, with 20 neurons per layer and a linear last layer.

Given the trained DNN controller, we verify the safety property shown in Figure 5. Specifically, the quadrotor is started from an initial condition \( (p_x^q(0), p_y^q(0)) \in [-0.05, 0.05] \times [-0.05, 0.05] \) (the other states are initialized at 0) and needs to stay within 0.32m from the planner in order to reach its goal without colliding into obstacles. Similar to the MC case study, we split the initial condition into smaller subsets and verify the property for each subset.

Table 2: Verisig+Flow* verification times (in seconds) for different initial conditions of the quadrotor case study. All properties were verified. Note that \( r_3 = [p_x^q, p_y^q, p_z^q] \).

<table>
<thead>
<tr>
<th>Initial condition on ( (p_x^q, p_y^q) )</th>
<th>Property</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>([-0.05, -0.025] \times [-0.05, -0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.025, 0] \times [-0.05, -0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([0, 0.025] \times [-0.05, -0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([0.025, 0.05] \times [-0.05, -0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.05, -0.025] \times [-0.025, 0])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.025, 0] \times [-0.025, 0])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([0.025, 0.05] \times [0, 0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.05, -0.025] \times [0, 0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.025, 0] \times [0, 0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([0.025, 0.05] \times [0, 0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.025, 0] \times [0, 0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([0.025, 0.05] \times [0, 0.025])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.05, -0.025] \times [0.025, 0.05])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.025, 0] \times [0.025, 0.05])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([0.025, 0.05] \times [0.025, 0.05])</td>
<td>(</td>
<td>r_3</td>
</tr>
<tr>
<td>([-0.025, 0] \times [0.025, 0.05])</td>
<td>(</td>
<td>r_3</td>
</tr>
</tbody>
</table>
The verification times of Verisig+Flow* for each subset are shown in Table 2. Most cases take less than 30 minutes to verify, which is acceptable for an offline computation. Note that this verification task is harder than MC not because of the larger dimension of the state space but because of the discrete DNN outputs. This means that Verisig+Flow* needs to enumerate and verify all possible paths from the initial set. This process is computationally expensive since the number of paths could grow exponentially with the length of the scenario (set to 30 steps in this case study). One approach to reduce the computation time would be to use the Markov property of dynamical systems and skip states that have been verified previously. We plan to explore this idea as part of future work.

In summary, this section shows that Verisig can verify both safety (avoiding obstacles) and bounded liveness (going up a hill) properties in different and challenging domains. The plant models can be nonlinear systems specified in either discrete or continuous time. The next section shows that Verisig+Flow* also scales well to larger DNNs and is competitive with other approaches for verification of DNN properties in isolation.

6 COMPARISON WITH OTHER DNN VERIFICATION TECHNIQUES

This section complements the Verisig evaluation in Section 5 by analyzing the scalability of the proposed approach. We train DNNs of increasing size on the MC problem and compare the verification times against the times produced by another suggested approach to the verification of sigmoid-based DNNs, namely one using a MILP formulation of the problem [7]. We verify properties about DNNs only (without considering the closed-loop system), since existing approaches cannot be used to argue about the closed-loop system.

As noted in the introduction, the two main classes of DNN verification techniques that have been developed so far are SMT- and MILP-based approaches to the verification of ReLU-based DNNs. Since both of these techniques were developed for piecewise-linear activation functions, neither of them can be directly applied to sigmoid-based DNNs. Yet, it is possible to extend them to sigmoid functions by bounding the sigmoid from above and below by piecewise-linear functions. In particular, we implement the MILP-based approach for comparison purposes since it can also be used to reason about the reachability of a DNN, similar to Verisig+Flow*.

The encoding of each sigmoid-based neuron into an MILP problem is described in detail in [7]. It uses the so called Big M method [33], where conservative upper and lower bounds are derived for each neuron using interval analysis. The encoding uses a binary variable for each linear piece of the approximating function such that when that variable is equal to 1, the inputs are within the bounds of that linear piece (all binary variables have to sum up to 1 in order to enforce that the inputs are within the bounds of exactly one linear piece). Thus, the MILP contains as many binary variables per neuron as there are linear pieces in the approximating function. Finally, one can use Gurobi to solve the MILP and compute a reachable set of the outputs given constraints on the inputs.

To compare the scalability of the two approaches, we trained multiple DNNs on the MC problem by varying the number of layers from two to ten and the number of neurons per layer from 16 to 128. A DNN is assumed to be “trained” if most tested episodes result in a reward of at least 90 – since this is a scalability comparison only, no closed-loop properties were verified. For each trained DNN, we record the time to compute the reachable set of control actions for input constraints $\rho_0 \in [-0.52,-0.5]$ and $\tau_0 = 0$ using both Verisig+Flow* and the MILP-based approach. For fair comparison, the two techniques were tuned to have similar approximation error; thus, we used roughly 100 linear pieces to approximate the sigmoid.

The comparison is shown in Figure 6. The MILP-based approach is faster for small networks and for large networks with few layers. As the number of layers is increased, however, the MILP-based approach’s runtimes increase exponentially due to the increasing number of binary variables in the MILP. Verisig+Flow*, on the other hand, scales linearly with the number of layers since the same computation is run for each layer (i.e., in each mode). This means that Verisig+Flow* can verify properties about fairly deep networks; this fact is noteworthy since deeper networks have been shown to learn more efficiently than shallow ones [31].

Another interesting aspect of the behavior of the MILP-based approach can be seen in Figure 6c. The verification time for the nine-layer DNN is much faster than for the eight-layer one, probably due to Gurobi exploiting a corner case in that specific MILP. This suggests that the fast verification times of the MILP-based approach should be treated with caution as it is not known which example can trigger a worst-case behavior. In conclusion, Verisig+Flow* scales linearly and predictably with the number of layers and can be used in a wide range of closed-loop systems with DNN controllers.
7 CONCLUSION AND FUTURE WORK

This paper presented Verisig, a hybrid system approach to verifying safety properties of closed-loop systems with DNN controllers. We showed that the verification problem is decidable for networks with one hidden layer and decidable for general DNNs if Schanuel’s conjecture is true. The proposed technique uses the fact that the sigmoid is a solution to a quadratic differential equation, which allows us to transform the DNN into an equivalent hybrid system. Given this transformation, we cast the DNN verification problem into a hybrid system verification problem, which can be solved by existing reachability tools such as Flow*. We evaluated both the applicability and scalability of Verisig+Flow* in two case studies.

For future work, it would be interesting to investigate whether one could use sigmoid-based DNNs to approximate DNNs with other activation functions (with analytically bounded error). This would enable us to verify properties about arbitrary DNNs and would greatly expand the application domain of Verisig.

A second direction for future work is to speed up the verification computation by exploiting the fact that the sigmoid dynamics are monotone and quadratic. Although the proposed technique is already scalable to a wide range of applications, it still makes use of a general-purpose hybrid system verification tool, i.e., Flow*. That is why, developing a specialized sigmoid verification tool might bring significant benefits in terms of scalability and precision.

ACKNOWLEDGMENTS

We thank Xin Chen (University of Dayton, Ohio) for his help with encoding the case studies in Flow*. We also thank Vicenc Rubies Royo (University of California, Berkeley) for sharing and explaining his code on approximating MIPs with DNNs. Last, but not least, we thank Luan Nguyen and Oleg Sokolsky (University of Pennsylvania) for fruitful discussions about the verification technique.

REFERENCES