Taming Reachability Analysis of DNN-Controlled Systems via Abstraction-Based Training

Jiaxu Tian[1], Dapeng Zhi[1], Si Liu[2], Peixin Wang[3], Guy Katz[4], and Min Zhang[1](✉)

[1] Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
zhangmin@sei.ecnu.edu.cn
[2] ETH Zurich, Zurich, Switzerland
[3] University of Oxford, Oxford, UK
peixin.wang@cs.ox.ac.uk
[4] The Hebrew University of Jerusalem, Jerusalem, Israel

Abstract. The intrinsic complexity of deep neural networks (DNNs) makes it challenging to verify not only the networks themselves but also the hosting DNN-controlled systems. Reachability analysis of these systems faces the same challenge. Existing approaches rely on over-approximating DNNs using simpler polynomial models. However, they suffer from low efficiency and large overestimation, and are restricted to specific types of DNNs. This paper presents a novel abstraction-based approach to bypass the crux of over-approximating DNNs in reachability analysis. Specifically, we extend conventional DNNs by inserting an additional abstraction layer, which abstracts a real number to an interval for training. The inserted abstraction layer ensures that the values represented by an interval are indistinguishable to the network for both training and decision-making. Leveraging this, we devise the first black-box reachability analysis approach for DNN-controlled systems, where trained DNNs are only queried as black-box oracles for the actions on abstract states. Our approach is sound, tight, efficient, and agnostic to any DNN type and size. The experimental results on a wide range of benchmarks show that the DNNs trained by using our approach exhibit comparable performance, while the reachability analysis of the corresponding systems becomes more amenable with significant tightness and efficiency improvement over the state-of-the-art white-box approaches.

1 Introduction

Deep neural networks (DNNs) have demonstrated their remarkable capability of driving systems to perform specific tasks intelligently in open environments.

R. Dimitrova et al. (Eds.): VMCAI 2024, LNCS 14500, pp. 73–97, 2024.
https://doi.org/10.1007/978-3-031-50521-8_4

They determine optimal actions during interactions between the hosting systems and their surroundings. Formally verifying DNNs can provide safety guarantees [27,48,55], which is, however, difficult in practice due to their black-box nature and lack of interpretability [10,61]. Furthermore, their hosting systems aggravate the difficulty since determining system actions requires computations over nonlinear system dynamics [19,54].

Reachability analysis, one of the powerful formal methods, has been widely applied to the verification of continuous and hybrid systems [7,11,15]. Its successful applications include invariant checking [24,29], robust control [37,49], fault detection [50,53], set-based predication [6,46], etc. The essence of reachability analysis is to compute all reachable system states from given initial state(s), which can be used in various verification tasks such as model checking [9]. As an emerging approach to verifying DNN-controlled systems, reachability analysis has already been shown to be promisingly effective [21,23,32].

The Problem. Compared to continuous hybrid systems, it is significantly more challenging to compute reachable states for DNN-controlled systems due to the embedded complex and inexplicable DNNs. In addition to over-approximating nonlinear system dynamics [14,25,38], one also has to over-approximate the embedded DNNs for computing overestimated action sets [30,32] of such systems. Specifically, given a set S_i of continuous system states at time step i,[1] one first overestimates a set $\tilde{A}_i$ of actions that will be applied to S_i by over-approximating the neural network on S_i, and then overestimates a set $\tilde{S}_{i+1}$ of successors by applying $\tilde{A}_i$ to S_i using over-approximated system dynamics. We consider such dual over-approximations as *white-box* approaches since all the information of DNNs, such as architectures, activation functions, and weights, shall be known before defining appropriate over-approximated models [21,23,32]. Consequently, these approaches are restricted to certain types of DNNs. For instance, Verisig 2.0 [32] does not support neural networks with the ReLU activation functions; Sherlock [21] is only applicable to ReLU-based networks; ReachNN* [23] is not scalable against the network size and introduces more overestimation; Polar [30] also suffers from the efficiency problem when dealing with networks with differentiable activation functions (e.g., Tanh). Moreover, dual over-approximations introduce large overestimation accumulatively, which results in a considerable number of unreachable states in the overestimated sets.

Our Approach. We present a novel abstraction-based approach for bypassing the over-approximation of DNNs in computing the reachable states of DNN-controlled systems. Our approach introduces an *abstraction layer* into the neural network before training, which abstracts concrete system states into abstract ones. This abstraction ensures that concrete states that are abstracted into the same state share the same action determined by the trained DNN. Leveraging this property, we can therefore compute the actions of a set of concrete states by mapping them to the corresponding abstract states and by feeding the abstract states into the trained DNNs to query for the output action. As DNNs are used

[1] Continuous time is uniformly discretized into time steps.

as *black-box* oracles during the entire process, it suffices to know how system states are abstracted and to query the trained DNNs with the abstract states for the actions. Hence, the over-approximation of DNNs for computing actions is decently bypassed. Consequently, the overestimation due to the embedded network is avoided and no assumption is made on a network including its size, weight, architecture, and activation function.

The abstraction-based training also allows us to avoid state explosion during the computation of reachable states. This is because adjacent abstract states, e.g., the two intervals $[0, 1]$ and $[1, 2]$, can be efficiently aggregated, e.g., to $[0, 2]$, which substantially restrains the exponential growth in the number of computed reachable states. Additionally, we propose a parallel optimization via initial-set partitioning, which further accelerates the process of computing reachable states.

We have implemented our proposed approach into a tool called BBReach and extensively evaluated over a wide range of benchmarks. The experimental results show that DNNs trained by using our abstraction-based approach achieve competitive performance in terms of system cumulative reward. Our approach provides a black-box alternative to the reachability analysis of DNN-controlled systems, which bypasses the crux of DNN over-approximation and significantly improves the state-of-the-art white-box counterparts with respect to the tightness and efficiency in reachable state computation.

Contributions. Overall, we provide:

1. a novel abstraction-based training approach of DNNs, which mitigates the limitation of DNN over-approximation in the reachability analysis of DNN-controlled systems, without sacrificing the performance of trained DNNs (Sect. 4);
2. the first, sound black-box approach for the reachability analysis of trained DNN-controlled systems, which not only enhances computational tightness and efficiency, but also are compatible with various DNNs (Sect. 5); and
3. a prototype BBReach and an extensive assessment, which shows that BBReach improves existing white-box tools with respect to both the precision of results and the computational efficiency (Sect. 6).

2 Preliminaries

2.1 DNN-Controlled Systems

A DNN-controlled system is typically a cyber-physical system where a DNN is planted and trained as a decision-making controller. It can be modeled as a 6-tuple $\mathcal{D} = \langle S, S^0, A, \pi, f, \delta \rangle$, where S is the set of n-dimensional system states on n continuous variables, $S^0 \subseteq S$ is the set of initial states, A is the set of system actions, $\pi : S \to A$ is a policy function realized by the DNN in the system, $f : S \times A \to \dot{S}$ is a non-linear continuous environment dynamics represented by an ordinary differential equation (ODE) [26] that maps the current state and control input (i.e., action) into the derivative of states with respect to time t_c, and δ is the time step size.

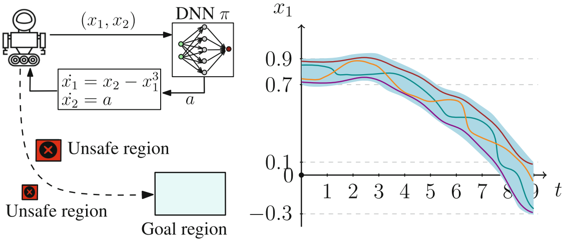
In a DNN-controlled system, an agent reacts to the environment over time. The time is usually discretized by a time scale δ called the time step size, assuming that actions during each time scale δ are constants [44]. At each time step $i \in \mathbb{N}$, the agent first observes a state s_i from the environment and feeds the state into the network to compute a constant action a_i. The agent then transits to the successor state s_{i+1} by performing a_i on s_i according to some environment dynamics f. During the training phase of DNN-controlled systems, the agent also receives a reward r_i which is determined by a reward function $r_i = R(s_i, a, s_{i+1})$ from the environment after each state transition. Once the task is finished, e.g., the agent reaches the goal region at time step T, we obtain the sequence of traversed system states from an initial state, called a *trace*, and the cumulative reward $\sum_{i=0}^{T} r_i$ which quantitatively measures the system performance.

Example 1. (A DNN-Controlled System). Figure 1(a) shows a DNN-controlled system where a two-dimensional agent moves from the region $x_1 \in [0.7, 0.9]$, $x_2 \in [0.7, 0.9]$ to the goal region $x_1' \in [-0.3, 0.1]$, $x_2' \in [-0.35, 0.05]$, trying to avoid the red unsafe regions. The environment dynamics f is defined by the following ODEs:

$$\dot{x_1} = x_2 - x_1^3 \qquad \dot{x_2} = a \tag{1}$$

The action $a = \pi(x_1, x_2)$ is computed by applying the DNN π to the values of x_1 and x_2. Based on a and f, the successor state s' can be computed for the agent to move. Figure 1(b) shows the traces (colored lines) of the system from some selected concrete initial states and an over-approximated set of reachable states (blue area) on the x_1 dimension from all the initial states.

The DNN planted in a system must be trained first so that it can determine optimal actions to complete a task. After making a decision, a loss is computed by a predefined loss function based on the reward that the agent receives for the decision. The parameters in the neural network are updated based on the loss by backpropagation [35]. The objective of the training phase is to maximize the cumulative reward. Once the training is completed, the network implements a state-action policy function that maps each system state to its optimal action. It drives the system to run and to interact with the environment.



(a) The system workflow. (b) System traces and reachable states.

Fig. 1. The workflow of the DNN-controlled system in Example 1, the execution traces (colored lines) and an over-estimated set of reachable states (blue region) with respect to the dimension of x_1. (Color figure online)

2.2 Reachability Problem of DNN-Controlled Systems

Given a DNN-controlled system $\mathcal{D}$, whether or not a state is reachable is known as the *reachability problem*. The verification of safety properties can be reduced to the reachability problem. For instance, one can verify whether a system never moves to unsafe states or not, such as those in Example 1. Unfortunately, the problem is *undecidable* even for conventional cyber-physical systems that are controlled by explicit programmable rules, let alone uninterpretable neural networks. This is because such systems are more expressive than two-counter state machines whose reachability problem is proved to be undecidable [41].

When the set of initial states is a singleton, it is straightforward to compute the reachable state at any given time t_c. Let δ be a time scale during which system actions can be considered constant. Given an initial state s_0, the state at time $t_c = k\delta + t'_c$ for some integer $k \geq 0$ and $0 \leq t'_c \leq \delta$ is defined as follows:

$$\varphi_f(s_0, \pi, t_c) = s_k + \int_0^{t'_c} f(s, \pi(s_k))dx,$$

where $s_{i+1} = s_i + \int_0^{\delta} f(s, \pi(s_i))dx$ for all $i \in \{0, \ldots, k-1\}$. Intuitively, we can compute the state s_{i+1} at the $(i+1)$-th time step based on the state s_i at its preceding time step i and the corresponding action $\pi(s_i)$. The state at t_c can be computed based on s_k, plus the offset caused by performing action $\pi(s_k)$ on state s_k with t'_c time scale.

Definition 1. (Reachable States of DNN-controlled Systems). Given a DNN-controlled system $\mathcal{D} = \langle S, S^0, A, \pi, f, \delta \rangle$, the sets of all the reachable states of the system at and during time t_c are denoted as $Reach_f^{t_c}(S_0)$ and $Reach_f^{[0,t_c]}(S_0)$, respectively. We have $Reach_f^{t_c}(S_0) = \{\varphi_f(s, \pi, t_c) | s \in S_0\}$ and $Reach_f^{[0,t_c]}(S_0) = \{\varphi_f(s, \pi, t) | s \in S_0, t \in [0, t_c]\}$.

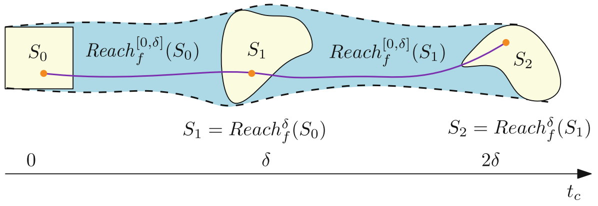

S_0
$Reach_f^{[0,\delta]}(S_0)$
S_1
$Reach_f^{[0,\delta]}(S_1)$
S_2
$S_1 = Reach_f^{\delta}(S_0)$
$S_2 = Reach_f^{\delta}(S_1)$
0
δ
2δ
t_c

Fig. 2. Reachable states of DNN-controlled systems.

Figure 2 depicts an example of the reachable states from S_0. For each time step i, we compute the set $Reach_f^{[0,\delta]}(S_i)$ of all the reachable states during the time period from i to $i+1$. The actions used for computing $Reach_f^{[0,\delta]}(S_i)$ are the constants determined by the DNN π on the states in S_i. In particular, we compute the set $S_{i+1} = Reach_f^{\delta}(S_i)$ of the reachable states at step $i+1$. Note that S_{i+1} is a subset of $Reach_f^{[0,\delta]}(S_i)$. We need to compute S_{i+1} independently from $Reach_f^{[0,\delta]}(S_i)$ because it is the basis of computing the reachable states in next step.

The procedure depicted in Fig. 2 indicates that the problem of computing $Reach_f^{[0,t_c]}(S_0)$ can be reduced to the problem of computing one-time-step reachable states, i.e., $Reach_f^{[0,\delta]}(S_0)$ and $Reach_f^{\delta}(S_0)$. However, the reduced problem is still intractable. This is because S_0 is usually an infinite set, meaning that it is impractical to enumerate each state in S_0, feed it into the DNN to compute the corresponding action, and then compute the state by Formula 2 for the set $Reach_f^{\delta}(S_0)$. Computing the states in $Reach_f^{[0,\delta]}(S_0)$ is even more challenging due to the continuous time in $[0, \delta]$.

3 Motivation

The combination of nonlinear dynamics and neural network controllers makes the calculation of $Reach_f^{[0,\delta]}(S_0)$ intractable. This is because the function φ_f (Formula 2) can not be expressed in a known closed form for most nonlinear dynamics f [13]. Additionally, a DNN π neither can be replaced by a known form equivalent function. A pragmatic solution is to compute tight over-approximation for φ_f and π. Most of the state-of-the-art approaches, such as Verisig [33], Polar [30], and ReachNN [31], adopt this strategy.

Without loss of generality, we show the process of over-approximating $Reach_f^{\delta}(S_0)$ in Example 1 using Polar. Given a set S_0 of states, Polar first over-approximates the neural network using a Taylor model (p, I_r) [39] on domain S_0 such that $\forall s \in S_0, \pi(s) \in p(s) + [-\epsilon, \epsilon]$, where p is a polynomial over the set of state variables $x_1, \ldots, x_n$ such as $p(x_1, x_2) = 0.5 + 0.1x_1 + 0.6x_1x_2 + 0.3x_1^2x_2$ and $I_r = [-\epsilon, \epsilon]$ is called the remainder interval. The range of $\pi(s)$ can be overestimated based on the Taylor model. Next, Polar over-approximates the solution of environment dynamics φ_f using another Taylor model over domains $s_0 \in S_0, \pi(s_0) \in p(s_0) + [-\epsilon, \epsilon], t_c \in [0, \delta]$ and obtains $x'_1 \in p_1(x_1, x_2, t_c) + [-\epsilon_1, \epsilon_1]$, $x'_2 \in p_2(x_1, x_2, t_c) + [-\epsilon_2, \epsilon_2]$. Finally, Polar produces an overestimated set of S_1 at time δ based on x'_1 and x'_2. A smaller range of I_r means less over-approximation error.

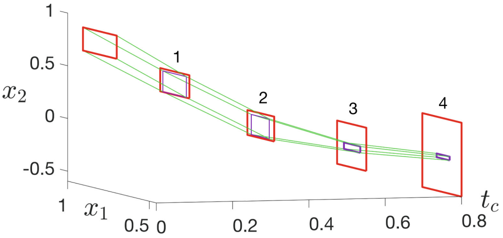


Fig. 3. An example of overestimation blowup of computed reachable states.

Suppose the initial region in Example 1 is $x_1 \in [0.7, 0.9], x_2 \in [0.7, 0.9]$. The overestimated reachable states can be calculated over 4 time steps according to the aforementioned method, which are depicted as red boxes (□) in Fig. 3. For comparison, Fig. 3 also shows the reachable states by simulation with 1000 samples, which are shown as the small violet boxes (□). We observe that the overestimation is amplified at the third and the fourth time step. At the third time step, the calculated remainder interval of the Taylor model for network is

$[-0.98, 0.98]$ while the one at the fourth time step is $[-4.47, 4.47]$. Correspondingly, the remainder intervals of the Taylor model for dynamics are $[-0.17, 0.17]$ and $[-0.46, 0.46]$ at the third and the fourth time step. The overestimation is accumulated and amplified step by step.

The above example shows that overestimation is mainly introduced by the over-approximation of the DNN. We further observe that if we could group the states in S_0 into several subsets such that all the states in the same subset have the same action according to π, we do not need to over-approximate π but, instead, replace $\pi(s)$ with its corresponding action. That is, if we know that all the states in a set S'_0 share the same action, e.g., a, according to π, the problem of computing $\mathit{Reach}_f^\delta(S'_0)$ can be simplified to solving the following problem:

$$\bigcup_{s'_0 \in S'_0} \{s'_0 + \int_0^\delta f(s'_0, a)dx\}. \tag{2}$$

Naturally, we only need to over-approximate φ_f to solve the above problem. Therefore, we identify a condition of bypassing over-approximating π: S_0 can be divided into a finite number of subsets such that the states in the same subset have the same action according to π. This will be elaborated in our following abstraction-based approach.

4 Abstraction-Based Training

Given a DNN π and a set S_0 of system states, it is almost intractable to group the states in S_0 that have the same action according to π. It becomes even worse when actions are continuous, where each state in S_0 may have a different action from others. Instead of calculating these states *ex post facto*, we propose an *ex ante* approach by abstraction-based training, in which system states are first grouped by abstraction before training, and a trained DNN provably yields a unique action for the states in the same group.

4.1 Approach Overview

The process of grouping a set of system states and making them indistinguishable to neural networks is called *abstraction.* A group is considered as an abstract state. The indistinguishability of the states in the same group guarantees that a DNN computes a unique action for those states. This idea is inspired by the abstraction approaches in formal methods, by which system states are abstracted to reduce state space and improve verification scalability without losing the soundness of verification results [17]. The same idea is also studied in the AI communities. State abstraction has been proved useful for conventional Reinforcement Learning (RL) [2,4,51] and recently applied to Deep RL for training DNN controllers [34]. Studies show that one can train nearly optimal system policies via approximate state abstraction, while the trained policies are more concise and amenable for reasoning and verification than those trained on concrete states [2,34].

To implement state abstraction into deep learning, we extend ordinary DNN architectures by introducing an *abstraction layer* between the input layer and the first hidden layer. This layer is used to map concrete system states in a group to the same abstract representation, which is propagated throughout the hidden layers for training. We call a neural network that contains such an abstraction layer an *abstract neural network* (ANN). Note that an ANN is a special model of DNN. In what follows, we call the systems with ANN controllers *ANN-controlled systems* to differ from those controlled by conventional DNNs.

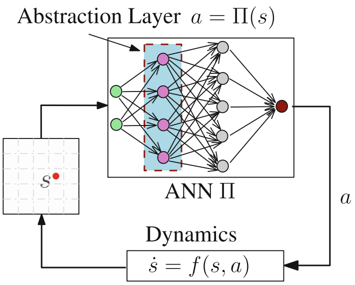


Fig. 4. Abstraction-based training.

The training of ANNs is almost the same as for conventional DNNs. Figure 4 shows the training workflows with ANNs. We can simply replace DNNs with ANNs in existing training algorithms, such as Deep Q-Network (DQN) [42] and Deep Deterministic Policy Gradient (DDPG) [36], as the inserted abstraction layers in ANNs are invisible to these algorithms.

Therefore, an algorithm that supports training DNN-controlled systems can be seamlessly adapted to train ANN-controlled systems. When applying these algorithms to ANNs, the only difference is that we need to freeze the parameters on the edges between the input layer and the abstraction layer because they are determined and fixed according to the way in which system states are abstracted. Parameter freezing is a common operation in deep learning and is supported by most of the training platforms such as TensorFlow [1] and PyTorch [45]. After encoding the abstraction layer and freezing the parameters, a network can be trained just like conventional DNNs by these training algorithms.

4.2 Interval-Based State Abstraction

We propose a general approach for encoding interval-based abstractions into equivalent abstraction layers. Interval-based state abstraction is a very primitive, yet effective abstraction approach. In the domain of abstract interpretation [17], it is known as *interval abstract domain* and has been well studied for system [3] and program verification [28], as well as neural network approximation [59]. By interval-based abstraction, the domain of each dimension is evenly divided into several intervals. The Cartesian product of the intervals in all the dimensions constitutes a finite and discrete set, with each element representing an infinite set of concrete states.

Definition 2. (Interval-Based State Abstraction). Given an n-dimensional continuous state space S and an abstract state space S_ϕ obtained by discretizing S based on an abstraction granularity γ, for every concrete state

$s = (x_1, \ldots, x_n) \in S$ and abstract state $s_\phi = (l_1, u_1, \ldots, l_n, u_n) \in S_\phi$, the interval-based abstraction function $\phi : S \to S_\phi$ is defined as $\phi(s) = s_\phi$ if and only if for each dimension $1 \leq i \leq n : l_i \leq x_i < u_i$.

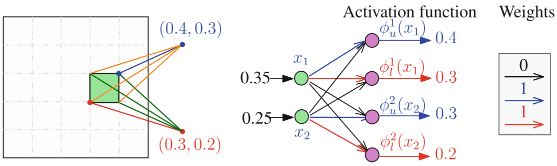


Fig. 5. An example of defining abstraction layers.

Specifically, the abstract state space S_ϕ is obtained by dividing each dimension in the original n-dimensional state space S into a set of intervals, which means that each abstract state can be represented as a $2n$-dimensional vector $(l_1, u_1, \ldots, l_n, u_n)$. We also call the $2n$-dimensional vector as interval box. In what follows, an interval box is used to represent a set of concrete states that fall into it. That is, for a $2n$-dimensional vector $(l_1, u_1, \ldots, l_n, u_n)$, we use it to represent the set of n-dimensional concrete states $\{(x_1, \ldots, x_n) \mid l_i \leq x_i < u_i, \forall 1 \leq i \leq n\}$. In this work, we divide the state space uniformly for better scalability so that we do not need extra data structure to store the mapping between S and S_ϕ. More specifically, let L_i and U_i be the lower and upper bounds for the i-th dimension of S. We define the abstraction granularity as an n-dimensional vector $\gamma = (d_1, d_2, \ldots, d_n)$, and then evenly divide the i-th dimension into $(U_i - L_i)/d_i$ intervals.

An interval-based abstraction can be naturally encoded as an abstraction layer. The layer consists of $2n$ neurons, each of which represents an element in the $2n$-dimensional vector $(l_1, u_1, \ldots, l_n, u_n)$. Each neuron has an activation function in either of the following two forms:

$$\phi_l^i(x_i) = L_i + \lfloor \frac{(x_i - L_i)}{d_i} \rfloor d_i, \quad \phi_u^i(x_i) = L_i + \lfloor \frac{(x_i - L_i + d_i)}{d_i} \rfloor d_i$$

for converting the value x_i in a concrete state to its lower and upper bounds, respectively. The sign $\lfloor \cdot \rfloor$ is the floor function. The weights of the edges connecting the i-th neuron in the input layer to the $(2i-1)$-th and $2i$-th neurons in the abstraction layer are assigned a value of 1, whereas the weights of all other edges are set with 0.

Example 2. Suppose that the ranges of both x_1 and x_2 in Example 1 are $[0, 0.5]$, and they are evenly partitioned into 5 intervals. The state space $[0, 0.5] \times [0, 0.5]$ is then uniformly partitioned into 25 interval boxes, as shown in Fig. 5. A concrete state such as $(0.35, 0.25)$ is mapped to an *interval box* represented by the corresponding lower bounds $(0.3, 0.2)$ of the first dimension and upper bounds $(0.4, 0.3)$ of the second dimension.

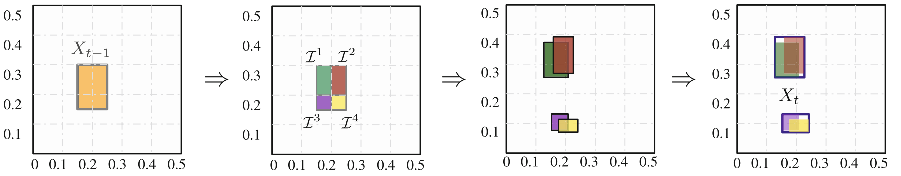
This abstraction can be realized by an abstraction layer, where there are four neurons and their activation functions are $\phi_u^1(x) = \phi_u^2(x) = \lfloor \frac{x+0.1}{0.1} \rfloor \times 0.1$ and $\phi_l^1(x) = \phi_l^2(x) = \lfloor \frac{x}{0.1} \rfloor \times 0.1$, respectively.

5 Abstraction-Based Reachability Analysis

5.1 Approach Overview

```
Algorithm 1: Overall process.
   Input  : Initial set $S_0$, ANN $\pi$, step size $\delta$, dynamics $f$, abstraction
            function $\phi$, maximal time step $T$
   Output: Over-approximation sets $\bigcup_{t=1}^{T} X_t$
1  Compute $I_0$ satisfying $S_0 \subseteq I_0$, $X_0 \leftarrow [I_0]$
2  foreach $t$ in $\{1, ..., T\}$ do
3      interval_arr $\leftarrow \{\}$
4      foreach $I$ in $X_{t-1}$ do
5          $\mathbf{B}_I \leftarrow segment(I, \phi)$
6          foreach $\mathcal{I}$ in $\mathbf{B}_I$ do
7              $a \leftarrow \pi(\hat{s})$ for some $\hat{s} \in \mathcal{I}$
8              $\mathcal{I}' \leftarrow post(\mathcal{I}, a, f)$
9              interval_arr $\leftarrow$ interval_arr $\cup \{\mathcal{I}'\}$
10     $X_t = aggregate$(interval_arr)
11 return $\bigcup_{t=1}^{T} X_t$
```

With the abstraction layer, we propose our abstraction-based black-box reachability analysis approach for ANN-controlled systems. Given an ANN-controlled system, a set S_0 of initial states and a maximal time step T, our task is to calculate a sequence of over-approximation sets consisting of interval boxes, denoted by $X_0, X_1, \ldots, X_T$, which are over-approximations of the actually reachable state sets $S_0, S_1, \ldots, S_T$ with $S_{t+1} = Reach_f^{\delta}(S_t)$, $0 \leq t < T$. The overall process is presented in Algorithm 1. It is an iterative process of calculating an over-approximated set X_t of states that are reachable from a set X_{t-1} of states after time δ. After we determine the range of $\pi(s)$ over $s \in X_{t-1}$, the reachable states during the time slot $(t\delta, (t+1)\delta]$ can be over-approximated as a continuous system without a neural network. In what follows, we focus on the computation of the over-approximation sets $X_0, X_1, ...X_T$.



(i) Original Interval (ii) Interval Segmentation (iii) Successor Intervals (iv) Adjacent Aggregation

Fig. 6. An example of over-approximating one-step reachable states.

Figure 6 depicts an example of one time-step iteration. Without loss of generality, we suppose that X_{t-1} is a singleton, e.g., $X_{t-1} = \{I\}$, where I is an interval box. We segment I into four smaller interval boxes (Fig. 6(ii) and Line 5 of Algorithm 1) based on the abstraction function ϕ that is used for training the network.

We then compute the action for the states in each segmented interval box by arbitrarily selecting a state $\hat{s}$ in the box and then feeding $\hat{s}$ into π to get the output (Line 7), e.g., a. Next, we compute a set $\mathcal{I}'$ of successor states of the states in $\mathcal{I}$ by over-approximating the environment dynamic f in Formula 2 (Fig. 6(iii) and Line 8 of Algorithm 1). Finally, we aggregate those adjacent successor interval boxes (Fig. 6(iv) and Line 10 of Algorithm 1) and obtain an over-estimated set X_t of reachable states at time step t.

5.2 Key Operations in Algorithm 1

We now describe in detail three key operations in Algorithm 1, namely *interval segmentation*, *post operation*, and *adjacent interval aggregation*. We fulfill the interval set propagation at each time step $t \in \mathbb{N}$ for the ANN-controlled systems based on these interval operations.

Interval Segmentation. Given an interval box I and an abstraction function ϕ, *segment*(I, ϕ) returns a set B_I of interval boxes which satisfy the following three *segmentation conditions*:

1. All the interval boxes constitute I;
2. Interval boxes do not overlap each other;
3. All the states in the same interval box have a unique action according to the trained ANN.

For conventional DNNs, one has to resort to brute-force interval splitting to find consistent regions that satisfy the above three conditions; this approach is only applicable to discrete action space [8]. We can easily partition I into such a set B_I, thanks to the specialized design of ANN. First, we determine the set of abstract states that intersect with I and denote the set by $\mathbf{S}_I = \{s_\phi \mid s_\phi \cap I \neq \emptyset\}$. We then calculate the intersection part between I and the abstract states in $\mathbf{S}_I$ individually. Each intersection part is a segmented interval box. In this way, we obtain a set of segmented interval boxes that satisfy the aforementioned three conditions and denote it by $\mathbf{B}_I = \{\mathcal{I} \mid \mathcal{I} = s_\phi \cap I \wedge s_\phi \in \mathbf{S}_I\}$. With the interval segmentation, through feeding an arbitrary state in the segmented interval box $\mathcal{I} \in B_I$ into ANN, we can obtain the corresponding unique action performed on $\mathcal{I}$. Since B_I is a finite set, the decisions of the network controller on I can be directly obtained without the layer-by-layer analysis process as in the white-box approaches [30,32]. This makes our reachability analysis approach a black-box one.

Recall the example in Fig. 6(i), where the black dotted lines denote the partition of the state space with abstraction granularity $\gamma = (0.1, 0.1)$. There exists an interval box $I = (0.15, 0.25, 0.15, 0.3)$ that intersects with four abstract states. The intersection of each abstract state with I is a segmented interval box. We have four interval boxes $B_I = \{\mathcal{I}^1, \mathcal{I}^2, \mathcal{I}^2, \mathcal{I}^4\}$. Apparently, the segmented interval boxes in B_I satisfy the three segmentation conditions.

Post Operation. Given an interval box $\mathcal{I}$, the action a applied to $\mathcal{I}$ and environment dynamics f, *post*$(\mathcal{I}, a, f)$ returns an interval box $\mathcal{I}'$, which is an over-approximation set of all the successor states by applying a to the states in $\mathcal{I}$ after δ.

We can solve $post(\mathcal{I}, a, f)$ as an ordinary continuous system without neural networks. Suppose that the environment dynamics is an ODE $\dot{s} = f(s, a)$. We use a Taylor model $p'(s, a, t_c) + I'_r$ to over-approximate the function $\varphi_f(s, a, t_c)$ over the domain $s \in \mathcal{I}, t_c \in [0, \delta]$. That is,

$$Reach_f^{[0,\delta]}(\mathcal{I}) = \bigcup_{s \in \mathcal{I}, t_c \in [0,\delta]} \{\varphi_f(s, a, t_c)\} \subseteq p'(s, a, t_c) + I'_r,$$

where I'_r is a remainder interval. The successor interval box $\mathcal{I}'$ can be calculated through evaluating the range of $p'(s, a, \delta) + I'_r$.

Let us consider an example for the segmented interval box $\mathcal{I}^1 = (0.15, 0.2, 0.2, 0.3)$ in Fig. 6(ii). The dynamics is defined as in Example 1. Suppose the action for the states in the interval box is $a = 0.5$ and the time scale $\delta = 0.1$. We can compute an over-approximated Taylor model for the solution of dynamics $f : \dot{x_1} = x_2 - x_1^3, \dot{x_2} = 0.5$ over $s \in \mathcal{I}^1, t_c \in [0, 0.1]$. The Taylor models for state variable x_1, x_2 are as follows:

$$\begin{aligned} x'_1 = {} & 1.75 \times 10^{-1} + 1.91 \times 10^{-8} x_2 + 2.5 \times 10^{-2} x_1 + 0.245 t_c \\ & - 1.25 \times 10^{-10} x_1^2 + 5 \times 10^{-2} x_2 t_c - 2.3 \times 10^{-3} x_1 t_c + 0.239 t_c^2 \\ & - 5 \times 10^{-10} x_1 x_2 t_c + \ldots + [-1.03 \times 10^{-4}, 8.94 \times 10^{-5}] \\ x'_2 = {} & x_2 + 0.5 t_c + [-0, 0] \end{aligned}$$

Using these two expressions, we can over-approximate the set of reachable states at every moment during $[0, 0.1]$. In particular, we have $(0.172, 0.232, 0.25, 0.35)$ when $t_c = 0.1$.

Adjacent Interval Aggregation. Interval segmentation may lead to the exponential blowup in the number of intervals as the number of time steps increases. As exemplified in Fig. 6(iii), four successor intervals are obtained after applying corresponding actions and environment dynamics to the states in $\mathcal{I}^1, \ldots, \mathcal{I}^4$.

To cope with the explosion of successor intervals, we provide a dual operation of segmentation called *adjacent interval aggregation*, which aggregates multiple intervals together at the price of introducing a little overestimation. This operation is based on the interval hull

Algorithm 2: Adjacent interval aggregation.

```
Input  : An interval array $IntArr$
Output: The aggregation results $Arr$
Initialize flag ← [False, False,...], $Arr$ ← []
Construct the adjacency matrix $M$
foreach $I_p$ in $IntArr$ do
    if not flag[$I_p$] then
        Initialize queue ← [$I_p$]
        flag[$I_p$] ← True
        while queue is not empty do
            $I$ ← queue.pop()
            $I_{adjs}$ ← getAdjacent($I$, $M$)
            foreach item in $I_{adjs}$ do
                $I_p$ ← aggInterval($I_p$, item)
                if not flag[item] then
                    queue.put(item)
                    flag[item] ← True
        $Arr$.add($I_p$)
return $Arr$
```

operation [43] except that we establish a criterion for determining which intervals can be aggregated into their interval hull. For instance, the green and brown intervals in Fig. 6(iii) can be aggregated, while the other small ones can be aggregated too. However, large overestimation would be introduced if the four interval boxes were aggregated to be one.

To balance the number of intervals and the overestimation introduced by aggregation, we define three cases for the adjacency relation between interval boxes, i.e., *inclusion*, *intersection*, and *separation*. Only the intervals in the three cases are aggregated. Given two interval boxes $A = (l_1, u_1, \ldots, l_n, u_n)$ and $B = (l'_1, u'_1, \ldots, l'_n, u'_n)$, as well as a preset distance threshold $h = (h_1, \ldots, h_n)$, the three cases are defined as follows:

1. **Inclusion:** An interval box is completely included in the other, i.e., $\forall i : (l_i \leq l'_i \wedge u_i \geq u'_i) \vee (l_i \geq l'_i \wedge u_i \leq u'_i)$.
2. **Intersection:** A and B have a partial overlap, i.e., $\exists! d : l'_d \leq l_d \leq u'_d \leq u_d \vee l_d \leq l'_d \leq u_d \leq u'_d$ and $\forall i, i \neq d : |l_i - l'_i| \leq h_i \wedge |u_i - u'_i| \leq h_i$.
3. **Separation:** A is isolated from B, i.e., $\exists! d : l_d - u'_d \leq h_d \vee l'_d - u_d \leq h_d$; and $\forall i, i \neq d : |l_i - l'_i| \leq h_i \wedge |u_i - u'_i| \leq h_i$.

To accelerate interval aggregation, we devise an efficient algorithm to aggregate three or more interval boxes each time if they constitute a sequence of adjacent intervals. Algorithm 2 shows the pseudo code. We first pre-construct an adjacency matrix (Line 2) to store the adjacent relations between the interval boxes in $IntArr$ firstly. Then, we implement this adjacent interval aggregation procedure using breadth-first search (Lines 5-14). Specifically, we consider each interval box in $IntArr$ as a node and each adjacent relation as an undirected edge. For each interval box I_p that is not traversed, all the interval boxes connected to I_p will be aggregated into their minimum bounding rectangle.

In Algorithm 2, the time complexity of building the adjacency matrix is $O(n^2)$. In the aggregation procedure, each interval box is traversed at most once, and the complexity of searching for the adjacent interval boxes for each interval box is $O(n)$. Therefore, Algorithm 2 is in $O(n^2)$.

Example 3. Let us revisit the system in Example 1 and suppose that $IntArr$ consists of 4 interval boxes, i.e., $\widehat{I_1} = (0.08, 0.16, 0.3, 0.4)$, $\widehat{I_2} = (0.17, 0.25, 0.32, 0.42)$, $\widehat{I_3} = (0.19, 0.27, 0.07, 0.2)$, $\widehat{I_4} = (0.2, 0.28, 0.1, 0.21)$, and the distance threshold is $h = (0.02, 0.02)$. According to the definition of adjacent relations, $\widehat{I_1}$ is adjacent to $\widehat{I_2}$ (Separation) and $\widehat{I_3}$ is adjacent to $\widehat{I_4}$ (Intersection). Hence, $\widehat{I_1}$ is aggregated with $\widehat{I_2}$, and $\widehat{I_3}$ is aggregated with $\widehat{I_4}$. Finally, we obtain $Arr = \{I_{1,2} = (0.08, 0.25, 0.3, 0.42), I_{3,4} = (0.19, 0.28, 0.07, 0.21)\}$.

5.3 The Soundness

We show a proof sketch for the soundness of Algorithm 1. The soundness means that any state that is reachable at time t_c from some initial state of an ANN-controlled system must be in the over-approximation set at t_c.

Theorem 1. (Soundness of Algorithm 1). *Given an ANN-controlled system with a set S_0 of initial states and an environment dynamic f, if a state s' is reached at time $t_c = k\delta + t'_c, k \in \mathbb{N}, t'_c \in [0, \delta)$ from some initial state $s_0 \in S_0$, then we must have $s' = Reach_f^{t_c}(s_0) \in Reach_f^{t'_c}(X_k)$.*

To prove Theorem 1, we first show the soundness of the *post* operation and interval aggregation. The soundness of the two operations is formulated by the following two lemmas, respectively.

Lemma 1. (Soundness of *post* Operation). *For each interval box $\mathcal{I} \in B_I$, there is $s_{t+1} \in post(\mathcal{I}, \pi(s_t), f)$ for all $s_t \in \mathcal{I}$ where $s_{t+1} = \varphi_f(s_t, \pi(s_t), \delta)$.*

Proof. After the segmentation process, we have $\forall s \in \mathcal{I} : \pi(s) = \pi(s_t) = a$ where a is a constant. With a constant action and the Lipschitz continuity of f, we can guarantee that there exists a unique solution of the ODE for a single initial state [40]. Then the solution of the ODE namely $\varphi_f(s, a, t_c)$ could be enclosed by a Taylor model [39] over $s(0) \in \mathcal{I}$ and $t_c \in [0, \delta]$. Thus, we could obtain the conservative result $s_{t+1} = \varphi_f(s_t, a, \delta) \in Reach_f^{\delta}(\mathcal{I}) \subset post(\mathcal{I}, a, f)$. □

Lemma 2. (Soundness of Interval Aggregation). *Suppose A is the aggregated set of successor intervals for a set X of interval boxes. For all $I \in X$, there exists $\widehat{I} \in A$ such that $I \subseteq \widehat{I}$.*

Proof. In Algorithm 2, every interval box in X needs to be traversed. For each interval box $I \in X$, there exist two cases: (i) I is not involved in the adjacent interval aggregation process. In this case, I will be directly added to A, thus $\exists \widehat{I} = I : I \subseteq \widehat{I}$. (ii) I is aggregated into another interval box I'. Since the *aggregate* operation produces the minimum bounding rectangle which encloses all interval boxes involved, we have $\exists \widehat{I} = I' : I \subseteq \widehat{I}$. Consequently, we conclude that $\forall I \in X,\ \exists \widehat{I} :\ I \subseteq \widehat{I} \wedge \widehat{I} \in A$. □

According to Algorithm 1, Theorem 1 can be proved by induction on the steps t_c based on Lemmas 1 and 2. The base case is straightforward when $t_c = 0$. In the induction case, we can prove that Theorem 1 holds on $[t\delta, (t+1)\delta]$ according to the two lemmas and the hypothesis that it holds on an arbitrary $t_c = t\delta$.

Proof. (Theorem 1). Starting from s_0, we can obtain the trajectory as $s_0, a_0, s_1, a_1, ...$ in which $a_t = \pi(s_t)$ and $s_{t+1} = \varphi(s_t, a_t, \delta)$. Then by induction on the time step t, the induction schema is as follows:

Base Case: $t_c = 0$. Since $s_0 \in S_0 \wedge S_0 \subseteq X_0$, we have $s_0 \in X_0 = Reach_f^0(X_0)$.

Induction Step: $t_c = t\delta$. Assume $s' = s_t \in X_t = Reach_f^0(X_t)$ holds. Since X_t consists of a set of interval boxes, there exists an interval box $I_{X_t}^{n_1}$ satisfying $s_t \in I_{X_t}^{n_1} \wedge I_{X_t}^{n_1} \in X_t$. Then, let us consider the segmentation process for $I_{X_t}^{n_1}$, we divide $I_{X_t}^{n_1}$ into a set of interval boxes $\mathbf{B}_{I_{X_t}^{n_1}} = \{\mathcal{I}_{X_t}^1, \mathcal{I}_{X_t}^2, \ldots, \mathcal{I}_{X_t}^{max}\}$ with $I_{X_t}^{n_1} = \bigcup_{n=1}^{max} \mathcal{I}_{X_t}^{n}$. Thus, there exists some $n_2 \in \mathbb{Z}^+$ such that $s_t \in \mathcal{I}_{X_t}^{n_2}$.

For $t_c \in [t\delta, (t+1)\delta)$, we have $s' = Reach_f^{t'_c}(s_t)$. Since $s_t \in \mathcal{I}_{X_t}^{n_2}$, we have $s' = Reach_f^{t'_c}(s_t) \in Reach_f^{t'_c}(\mathcal{I}_{X_t}^{n_2}) \subseteq Reach_f^{t'_c}(X_t)$.

For $t_c = (t+1)\delta$, we have $s' = s_{t+1}$, Based on Lemma 1, we have $s_{t+1} \in post(\mathcal{I}_{X_t}^{n_2}, \pi(s_t), f)$. After the adjacent interval aggregation process, X_{t+1} consists of the aggregation result. According to Lemma 2, we have $\exists \widehat{I} : post(\mathcal{I}_{X_t}^{n_2}, \pi(s_t), f) \subseteq \widehat{\mathcal{I}} \wedge \widehat{\mathcal{I}} \in X_{t+1}$. Therefore, we have $s_{t+1} \in \widehat{\mathcal{I}} \wedge \widehat{\mathcal{I}} \in X_{t+1}$ and we can conclude that $s' = s_{t+1} \in X_{t+1} = Reach_f^0(X_{t+1})$.

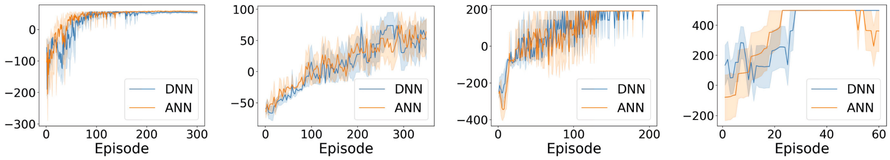
Theorem 1 is proved. □

6 Implementation and Experiments

We conduct a comprehensive assessment of our approach and compare it with the state-of-the-art white-box tools. Our goal is to demonstrate the advances of the proposed abstraction-based training and black-box reachability analysis approaches. These include (i) comparable performance of trained systems and negligible time overhead in the training (Sect. 6.2), (ii) tighter over-approximated sets of reachable states, as well as higher scalability and efficiency (Sect. 6.3), and (iii) the effectiveness of the adjacent interval aggregation algorithm in reducing state explosion (Sect. 6.4). We also explore how our approach performs under different abstraction granularity levels (Sect. 6.4).

6.1 Implementation and Benchmarks

Implementation. We implement our approach in a tool called BBReach in Python. We use Ariadne [16] to solve the reachability problems defined on segmented interval boxes (i.e., $post(\mathcal{I}, a, f), \mathcal{I} \in B_I$). Additionally, we employ the parallelized computing by initial-set partition [13], a standard approach used in the reachability analysis of hybrid systems to obtain tighter bounds of reachable states. With the initial set partitioned into k subsets, the k sub-problems can be solved in parallel, which accelerates our approach with multiple cores.



(a) B1 (b) B2 (c) B3 (d) B4

Fig. 7. Trend of cumulative rewards (y-axis) of the systems controlled by ANNs (orange) and DNNs (blue) trained by DDPG. (Color figure online)

Benchmarks. The benchmarks, as commonly adopted by most of the existing reachability analysis approaches such as Verisig 2.0 [32] and Polar [30], consist of seven reinforcement learning tasks with the dimensions ranging from 2 to 6. A reach-avoid property is defined for each task by specifying the goal region and unsafe region of the agent in the task. A trained DNN must guarantee that the reach-avoid property is satisfied when the agent is driven by the DNN.

For each task, we train four neural networks (two smaller networks chosen from [32] and two larger networks), thus 28 instances in total, with different activation functions and sizes of neurons. We also train the networks with different abstraction granularity levels to evaluate how abstraction granularity affects the efficiency. We use the well-known DDPG algorithm to train neural networks. Note that our approach makes no assumption on training algorithms and thus is applicable to other DRL algorithms. The detailed settings are provided in [56].

Experimental Setup. All experiments are conducted on a workstation equipped with a 32-core AMD Ryzen Threadripper CPU @ 3.6GHz and 256GB RAM, running Ubuntu 22.04.

6.2 Performance of Trained Neural Networks

We show that the extended abstract neural networks can be trained to achieve comparable performance against those conventional ones that have the same architectures and activation functions and are trained in the same approach. For each case, we train 5 times and record the cumulative reward during the training process with and without the abstraction layer. Figure 7 unfolds a comparison of the trend of cumulative rewards during training between these two training approaches in B1-B4 (the other three are given in [56].) The solid lines and the shadows indicate the average reward and 95% confidence interval, respectively. The results show that an extended abstract neural network can make near-optimal decisions even under the constraint that it must yield the same action on each partitioned interval box. Importantly, the abstraction-based training incurs little and negligible time overhead only in several seconds, as shown in Table 1.

Table 1. Training time (s).

Task	ANN	DNN
B1	13.7	11.0
B2	7.4	6.6
B3	6.4	5.1
B4	5.8	3.2
B5	57.4	49.8
Tora	47.2	44.3
ACC	23.4	21.6

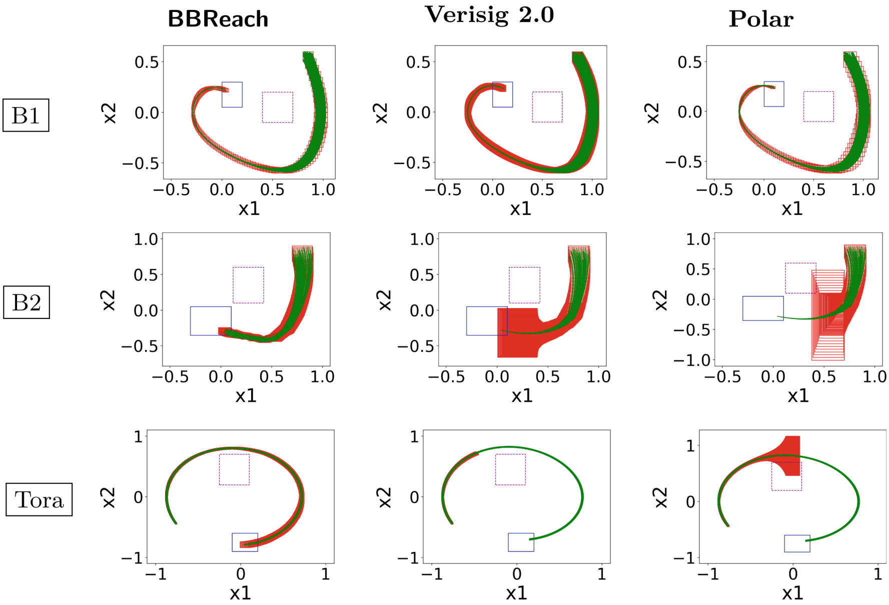


Fig. 8. Over-approximated reachable states (red box: over-approximated set; green lines: simulation trajectories; blue box: goal region; purple box: unsafe region). (Color figure online)

6.3 Tightness and Efficiency

We compare the tightness of the over-approximated reachable states by plotting the over-approximation sets computed by our approach and the state-of-the-art white-box tools including Polar [30] and Verisig 2.0 [32]. Because BBReach is designed for ANNs-controlled systems, while the white-box tools are for DNNs-controlled ones, the policy models for each task are different. To make the comparison as fair as possible, we use the same network architecture to train the ANN and DNN for the same task except that the ANN includes an additional abstraction layer. We also guarantee that all the trained systems can achieve the best cumulative reward for the same task. For instance, we initialize the neural networks with smaller weights as otherwise Verisig 2.0 would introduce larger over-approximation error (see Appendix B in [56]). In particular, we also simulated the trained systems and recorded trajectories as the baseline.

Figure 8 shows three representative cases. Verification succeeds if the system never enters the unsafe region (purple box) before reaching the goal region (blue box) which is also known as satisfying the reach-avoid property. All the four tools successfully verify the reach-avoid property in case B1, yet Verisig 2.0 is less tight than the other two. In case B2, BBReach outperforms the other two tools and succeeds in verifying the reach-avoid property. Both Verisig 2.0 and Polar terminate before reaching the goal region due to too large overestimation, and Polar outputs the over-approximation sets that intersect with the unsafe region.

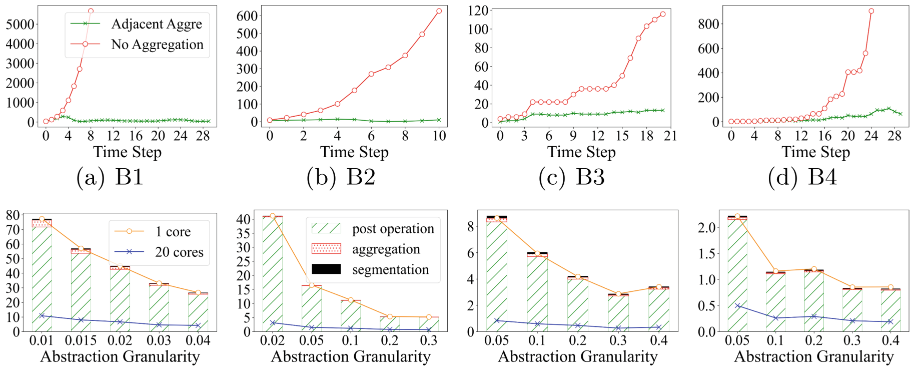
Table 2. The verification results of reach-avoid properties and the time cost (s).

Task	Dim	Network	BBReach			Verisig 2.0					Polar			
			1C	**20Cs**	**VR**	**1C**	**Impr.**	**20Cs**	**Impr.**	**VR**	**1C**	**Impr.**	**Impr.***	**VR**
B1	2	Tanh$_{2\times 20}$	45.7	6.88	✓	45	0.98×	38	5.52×	✓	17	0.37×	2.47×	✓
		Tanh$_{3\times 100}$	42.8	5.53	✓	413	9.65×	123	22.24×	✓	125	2.92×	22.60×	✓
		ReLU$_{2\times 20}$	42.9	6.44	✓	—	—	—	—	✗^{c}	3	0.07×	0.47×	✓
		ReLU$_{3\times 100}$	52.5	8.65	✓	—	—	—	—		—	—	—	✗^{b}
B2	2	Tanh$_{2\times 20}$	10.0	1.19	✓	5.2	0.52×	4.1	3.45×	✗^{a}	5	0.50×	4.20×	✓
		Tanh$_{3\times 100}$	10.8	1.36	✓	—	—	—	—	✗^{b}	—	—	—	✗^{b}
		ReLU$_{2\times 20}$	8.6	1.30	✓	—	—	—	—	✗^{c}	3	0.35×	2.31×	✓
		ReLU$_{3\times 100}$	12.4	1.42	✓	—	—	—	—		—	—	—	✗^{b}
B3	2	Tanh$_{2\times 20}$	4.2	0.47	✓	36	8.57×	28	59.57×	✓	18	4.29×	39.29×	✓
		Tanh$_{3\times 100}$	4.3	0.50	✓	357	83.02×	88	176.00×	✓	91	91.16×	182.00×	✓
		ReLU$_{2\times 20}$	4.1	0.47	✓	—	—	—	—	✗^{c}	8	1.95×	17.02×	✓
		ReLU$_{3\times 100}$	4.2	0.47	✓	—	—	—	—		14	3.33×	29.79×	✓
B4	3	Tanh$_{2\times 20}$	1.3	0.32	✓	7	5.38×	5.1	15.94×	✓	5	3.85×	15.63×	✓
		Tanh$_{3\times 100}$	1.0	0.24	✓	114	114.00×	31	129.17×	✓	27	27.00×	112.50×	✓
		ReLU$_{2\times 20}$	1.9	0.48	✓	—	—	—	—	✗^{c}	2	1.05×	4.17×	✓
		ReLU$_{3\times 100}$	1.8	0.43	✓	—	—	—	—		5	2.78×	11.63×	✓
B5	3	Tanh$_{3\times 100}$	13.3	2.48	✓	157	11.80×	44	17.74×	✓	38	2.86×	15.32×	✓
		Tanh$_{4\times 200}$	8.2	1.63	✓	1443	175.98×	191	117.18×	✓	157	19.15×	96.32×	✓
		ReLU$_{3\times 100}$	5.8	1.08	✓	—	—	—	—	✗^{c}	7	1.21×	6.48×	✓
		ReLU$_{4\times 200}$	13.5	2.50	✓	—	—	—	—		49	3.63×	19.60×	✓
Tora	4	Tanh$_{3\times 20}$	133.2	8.61	✓	69	0.52×	46	5.34×	✓	45	0.34×	5.23×	✓
		Tanh$_{4\times 100}$	112.3	9.78	✓	—	—	—	—	—	✗^{b}	—	—	✗^{b}
		ReLU$_{3\times 20}$	124.7	9.97	✓	—	—	—	—	✗^{c}	30	0.24×	3.01×	✓
		ReLU$_{4\times 100}$	128.1	7.54	✓	—	—	—	—		53	0.41×	7.03×	✓
ACC	6	Tanh$_{3\times 20}$	15.4	4.53	✓	113	7.34×	50	11.04×	✓	84	5.45×	18.54×	✓
		Tanh$_{4\times 100}$	15.2	4.51	✓	2617	172.17×	375	83.15×	✓	677	44.54×	150.11×	✓
		ReLU$_{3\times 20}$	15.2	4.45	✓	—	—	—	—	✗^{c}	26	1.71×	5.84×	✓
		ReLU$_{4\times 100}$	18.4	5.49	✓	—	—	—	—		58	3.15×	10.56×	✓

Remarks. Improvement: time speedup of BBReach compared to Verisig or Polar (Verisig or Polar/BBReach). * denotes the comparison between BBReach with 20 cores (Cs) and Polar. Tanh/ReLU$_{n\times k}$: a neural network with the activation function Tanh/ReLU, n hidden layers, and k neurons per hidden layer. VR: verification result. ✓: the reach-avoid problem is successfully verified. ✗^{type}: the reach-avoid problem cannot be verified due to *type*: (a) large over-approximation error, (b) the calculation did not finish, (c) not applicable. —: no data available due to ✗^{b} or ✗^{c}

Nevertheless, the simulation results show the trained DNN-controlled system should satisfy the reach-avoid requirements. For Tora, BBReach significantly surpasses other tools. None of the two white-box tools finishes before reaching the goal region because of the huge over-approximation error. For instance, the resulting bound of action, upon Verisig 2.0's termination, reaches 10^7 which is too large to proceed, although the increase of reachable states by simulation is approximately in linear. The comparison results for B3, B4, B5, and ACC are similar as for B1. We refer to our technical report [56] for more detailed results.

Table 2 shows the verification results of all the 28 instances in column **VR**. BBReach successfully verifies all the instances, while Verisig 2.0 succeeds in 11



(a) B1 (b) B2 (c) B3 (d) B4

(e) B1 (tanh) (f) B2 (tanh) (g) B3 (tanh) (h) B4 (tanh)

Fig. 9. Differential (Row 1) and decomposing (Row 2) analysis results. Y-axis in (a–d) indicates the number of interval boxes while in (e–h) the time overhead in seconds. Due to the space limitation, we use a scalar value g_1 to denote the n-dimensional abstraction granularity vector $\gamma = (g_1, ..., g_1)$.

instances and Polar in 24 instances. Verisig 2.0 reports 1 unknown case (marked by ✗^a, indicating that over-approximated sets get outside of the goal region). Additionally, Verisig 2.0 and Polar report 1 case and 4 cases of terminating before reaching the goal region, respectively, denoted by ✗^b. These also reflect that BBReach is tighter and introduces less overestimation than other tools.

Table 2 also shows the time cost. Note that Verisig 2.0 is not applicable to ReLU neural networks (marked by ✗^c). BBReach costs much less time than Verisig 2.0 (up to 176× speedup) with parallelization enabled. Even with a single core, BBReach incurs less overhead than Verisig 2.0 in most cases. Compared to Polar, BBReach consumes more time in dealing with small-sized neural networks in B1, B2, and Tora because the finer-grained abstraction granularity is chosen in the three cases, which affects the performance (see Sect. 6.4). Nevertheless, BBReach consumes less time in all the remaining cases than Polar. In addition, BBReach outperforms Polar with up to 182× speedup (the latest release of Polar does not support parallelization), thanks to the parallel acceleration.

The efficiency advantage of BBReach becomes more notable with larger networks such as $\text{Tanh}_{4\times200}$, thanks to the black-box feature of our approach. BBReach consumes almost the same time even for larger neural networks as for small neural networks (e.g., $\text{Tanh}_{2\times20}$). In contrast, the time cost of both the white-box approach almost always increases significantly with larger neural networks. Moreover, Polar incurs more overhead to process the neural networks with the Tanh activation function compared to ReLU, while BBReach consumes similar times for both activation functions. Consequently, it is fair to conclude that BBReach is more efficient and scalable to large-sized neural networks with any activation functions. It is also evident that, via a decent design of neural networks, the reachability analysis for DNN-controlled systems is achievable while the planted decision-making neural networks are treated as black-box oracles, with significant rightness and efficiency outperformance over the white-box approaches.

6.4 Differential and Decomposing Analysis

Differential Analysis. To demonstrate the significance of the adjacent interval aggregation in Algorithm 2, we measure the growth rate of the number of interval boxes with adjacent aggregation, as well as with no aggregation. Figure 9(a-d) shows the comparison results on B1-B4 (the results for the other six benchmarks are similar and given in [56]. We observe that the number of interval boxes grows rapidly with no aggregation, which implies a dramatically increased verification overhead. With the adjacent interval aggregation, the number of interval boxes is extremely small and stable.

Decomposing Analysis. We evaluate how different abstraction granularity levels affect the performance of BBReach and its components. Abstraction granularity is a crucial hyper-parameter used in both training and calculation of over-approximation sets. To better understand the impact of abstraction granularity, for each benchmark, in addition to the default abstraction granularity levels (details can be found in [56]), we choose two finer and two coarser levels, respectively, to evaluate the verification efficiency on both Tanh and ReLU neural networks. We also measure the time consumed by each of the three steps, i.e., interval segmentation, post operation, and adjacent interval aggregation.

We present in Fig. 9(e-h) the results with the Tanh neural network in B1-B4 (the remaining results are similar and given in [56]). With a single core, as the abstraction granularity becomes coarse-grained, the verification time decreases; however, a fairly fine-grained abstraction granularity, e.g., (0.01, 0.01), could result in much higher verification overhead. We also observe that the post operation takes most of the verification time, while the overhead of the other two steps is negligible. Finally, as expected, the parallelization (with 20 cores) can significantly accelerate BBReach.

7 Related Work

Our work is a sequel of recently emerged approaches for the reachability analysis of DNN-controlled systems such as Verisig 2.0 [32], Polar [30], ReachNN* [31]. Besides these states of the art, NNV [58] introduces the star set analysis technique [57] to deal with the neural network and combines with the tool called CORA [5] for the reachability analysis of non-linear systems. JuliaReach [47] integrates the over-approximation of environment dynamics and DNNs together using Taylor models and zonotope. All these approaches treat DNNs as white boxes by over-approximating them with efficiently computable models such as Taylor models [13]. Due to the intrinsic complexity of DNNs, these white-box approaches are applicable to only a limited type of DNNs on small scales.

Our abstraction-based training method follows those machine learning methodologies which advocate a similar idea of pre-processing training data using either abstraction [2,34], fuzzing [12] or granulation [52] for various purposes of reducing the size of models, capturing uncertainties in input data and extracting abstract knowledge. Recent studies show that, rather than training on concrete datasets, training on symbolic datasets is helpful to build verification-friendly neural networks [20] and network-controlled systems [34]. The approach in [2] is focused on the training of finite-state systems, while the one in [34] needs to extend existing training methods to admit abstract states. Our design of abstract neural networks is more decent than the approach in [34] because we only need to insert abstraction layers into neural networks and do not impose any other changes to training algorithms.

There are several black-box but unsound verification approaches for DNN-controlled systems. For instance, Fan et al. proposed a hybrid approach of combining black-box simulation and white-box transition graph for a probabilistic verification result [22]. Xue et al. proposed a black-box model-checking approach for continuous-time dynamical systems based on Probably Approximate Correctness (PAC) learning [60]. Dong et al. built a discrete-time Markov chain from extracted trajectories of a DRL system and verified safety properties by probabilistic model checking [18]. However, these approaches are not sound and can only compute error probability and confidence with probably approximate correctness guarantees. The fundamental reason for the unsoundness is that only partial behaviors of systems can be modeled when conventional neural networks are treated as black-box oracles, i.e., fixing concrete system states and feeding them into the networks to determine the state transitions.

8 Conclusion and Future Work

We have presented an efficient and tight approach for the reachability analysis of DNN-controlled systems by bypassing the time-consuming and imprecise over-approximation of the DNNs in systems via abstraction-based training. Our method demonstrates the possibility of achieving sound but black-box reachability analysis through a decent abstraction-based training approach, breaking conventional intuitions that black-box methods only offer approximate correctness guarantees [22,60] and that over-approximating DNNs is inevitable for sound verification [30–32]. Compared to white-box approaches, our black-box approach offers several benefits, including significant efficiency improvements, improved tightness of computed overestimation sets, applicability and scalability to a wider range of extended abstract DNNs, regardless of their architectures, activation functions, and neuron size. Nevertheless, the reachability analysis part may suffer from state explosion in the worst case when the number of reachable states increases exponentially, as faced by all the related white-box approaches [30–32]. One possible solution is to coarsen the abstraction to reduce the size of abstract states, and learn an easy-to-verify linear policy for each coarsened abstract state. Such an approach has been successfully applied to reinforcement learning [4] and requires further investigation in the DNN-based setting.

Our work sheds light on a promising direction for studying efficient and sound formal verification approaches for DNN-controlled systems by treating black-box-featured DNNs as black boxes. We believe that this first black-box reachability analysis approach for DNN-controlled systems would stimulate more future work, such as new abstraction methods, runtime verification and model-checking of more complex safety and liveness properties.

Acknowledgment. The work has been supported by the National Key Project (2020AAA0107800), NSFC Programs (62161146001, 62372176), Huawei Technologies Co., Ltd., the Shanghai International Joint Lab (22510750100), the Shanghai Trusted Industry Internet Software Collaborative Innovation Center, the Engineering and Physical Sciences Research Council (EP/T006579/1), the National Research Foundation (NRF-RSS2022-009), Singapore, and the Shanghai Jiao Tong University Postdoc Scholarship.

References

1. Abadi, M., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI, vol. 16, pp. 265–283. Savannah, GA, USA (2016)
2. Abel, D.: A theory of state abstraction for reinforcement learning. In: AAAI, vol. 33, pp. 9876–9877 (2019)
3. Afzal, M., et al.: Veriabs: verification by abstraction and test generation. In: ASE, pp. 1138–1141. IEEE (2019)
4. Akrour, R., Veiga, F., Peters, J., Neumann, G.: Regularizing reinforcement learning with state abstraction. In: IROS, pp. 534–539. IEEE (2018)
5. Althoff, M.: An introduction to CORA 2015. In: Cyber-Physical Systems Virtual Organization (CPS-VO 2015), pp. 120–151 (2015)
6. Althoff, M., Magdici, S.: Set-based prediction of traffic participants on arbitrary road networks. IEEE Trans. Intell. Veh. **1**(2), 187–202 (2016)
7. Alur, R., et al.: The algorithmic analysis of hybrid systems. Theoret. Comput. Sci. **138**(1), 3–34 (1995)
8. Bacci, E., Parker, D.: Probabilistic guarantees for safe deep reinforcement learning. In: Bertrand, N., Jansen, N. (eds.) FORMATS 2020. LNCS, vol. 12288, pp. 231–248. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57628-8_14
9. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
10. Baluta, T., Chua, Z.L., Meel, K.S., Saxena, P.: Scalable quantitative verification for deep neural networks. In: ICSE, pp. 312–323. IEEE (2021)
11. Bertsekas, D.P., Rhodes, I.B.: On the minimax reachability of target sets and target tubes. Automatica **7**(2), 233–247 (1971)
12. Campos Souza, P.V.: Fuzzy neural networks and neuro-fuzzy networks: a review the main techniques and applications used in the literature. Appl. Soft Comput. **92**, 106275 (2020)
13. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: RTSS, pp. 183–192. IEEE (2012)
14. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
15. Christakis, M., et al.: Automated safety verification of programs invoking neural networks. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 201–224. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_9

16. Collins, P., Bresolin, D., et al.: Computing the evolution of hybrid systems using rigorous function calculus. IFAC Proc. Vol. **45**(9), 284–290 (2012)
17. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252 (1977)
18. Dong, Y., Zhao, X., Huang, X.: Dependability analysis of deep reinforcement learning based robotics and autonomous systems through probabilistic model checking. In: IROS, pp. 5171–5178. IEEE (2022)
19. Dreossi, T., et al.: VerifAI: a toolkit for the formal design and analysis of artificial intelligence-based systems. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 432–442. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_25
20. Drews, S., Albarghouthi, A., D'Antoni, L.: Proving data-poisoning robustness in decision trees. In: PLDI, pp. 1083–1097 (2020)
21. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: HSCC, pp. 157–168 (2019)
22. Fan, C., Qi, B., Mitra, S., Viswanathan, M.: DryVR: data-driven verification and compositional reasoning for automotive systems. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 441–461. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_22
23. Fan, J., Huang, C., Chen, X., Li, W., Zhu, Q.: ReachNN*: a tool for reachability analysis of neural-network controlled systems. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 537–542. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_30
24. Fang, X., Calinescu, R., Gerasimou, S., Alhwikem, F.: Fast parametric model checking through model fragmentation. In: ICSE, pp. 835–846. IEEE (2021)
25. Frehse, G.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
26. Gallestey, E., Hokayem, P.: Lecture notes in nonlinear systems and control (2019)
27. Gomes, L.: When will Google's self-driving car really be ready? It depends on where you live and what you mean by "ready" [news]. IEEE Spectr. **53**(5), 13–14 (2016)
28. Heo, K., Oh, H., Yang, H.: Resource-aware program analysis via online abstraction coarsening. In: ICSE, pp. 94–104. IEEE (2019)
29. Hildebrandt, C., Elbaum, S., Bezzo, N.: Blending kinematic and software models for tighter reachability analysis. In: ICSE(NIER), pp. 33–36 (2020)
30. Huang, C., Fan, J., Chen, X., Li, W., Zhu, Q.: POLAR: a polynomial arithmetic framework for verifying neural-network controlled systems. In: Bouajjani, A., Holík, L., Wu, Z. (eds.) Automated Technology for Verification and Analysis. ATVA 2022. LNCS, vol. 13505, pp. 414–430. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19992-9_27
31. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: ReachNN: reachability analysis of neural-network controlled systems. ACM Trans. Embed. Comput. Syst. **18**(5s), 1–22 (2019)
32. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G., Lee, I.: Verisig 2.0: verification of neural network controllers using Taylor model preconditioning. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 249–262. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_11

33. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verifying the safety of autonomous systems with neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(1), 1–26 (2020)
34. Jin, P., Tian, J., Zhi, D., Wen, X., Zhang, M.: TRAINIFY: A CEGAR-driven training and verification framework for safe deep reinforcement learning. In: Shoham, S., Vizel, Y. (eds) Computer Aided Verification. CAV 2022. Lecture Notes in Computer Science, vol. 13371, pp. 193–218. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-13185-1_10
35. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
36. Lillicrap, T.P., Hunt, J.J., Pritzel, A., et al.: Continuous control with deep reinforcement learning. In: ICLR, OpenReview.net (2016)
37. Limon, D., Bravo, J., Alamo, T., Camacho, E.: Robust MPC of constrained nonlinear systems based on interval arithmetic. IEE Proc. Control Theory App. **152**(3), 325–332 (2005)
38. Lygeros, J., Tomlin, C., Sastry, S.: Controllers for reachability specifications for hybrid systems. Automatica **35**(3), 349–370 (1999)
39. Makino, K., Berz, M.: Taylor models and other validated functional inclusion methods. Int. J. Pure Appl. Math. **6**, 239–316 (2003)
40. Meiss, J.D.: Differential dynamical systems, Mathematical modeling and computation, vol. 14. SIAM (2007)
41. Minsky, M.L.: Computation. Prentice-Hall Englewood Cliffs, Hoboken (1967)
42. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
43. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to interval analysis. SIAM (2009)
44. Park, S., Kim, J., Kim, G.: Time discretization-invariant safe action repetition for policy gradient methods. In: NeurIPS 2021, vol. 34, pp. 267–279 (2021)
45. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. In: NeurIPS, vol. 32 (2019)
46. Pereira, A., Althoff, M.: Over approximative human arm occupancy prediction for collision avoidance. IEEE Trans. Autom. Sci. Eng. **15**(2), 818–831 (2017)
47. Schilling, C., Forets, M., Guadalupe, S.: Verification of neural-network control systems by integrating Taylor models and zonotopes. In: AAAI, vol. 36, pp. 8169–8177 (2022)
48. Schmidt, L., Kontes, G., Plinge, A., Mutschler, C.: Can you trust your autonomous car? Interpretable and verifiably safe reinforcement learning. In: IV, pp. 171–178. IEEE (2021)
49. Schürmann, B., Kochdumper, N., Althoff, M.: Reached model predictive control for disturbed nonlinear systems. In: CDC, pp. 3463–3470. IEEE (2018)
50. Scott, J., Raimondo, D., Marseglia, G., Braatz, R.: Constrained zonotopes: a new tool for set-based estimation and fault detection. Automatica **69**, 126–136 (2016)
51. Singh, S.P., Jaakkola, T., Jordan, M.I.: Reinforcement learning with soft state aggregation. NeurIPS **7**, 361–368 (1995)
52. Song, M., Jing, Y., Pedrycz, W.: Granular neural networks: a study of optimizing allocation of information granularity in input space. Appl. Soft Comput. **77**, 67–75 (2019)
53. Su, J., Chen, W.H.: Model-based fault diagnosis system verification using reachability analysis. IEEE Trans. Syst. Man Cybern. Syst. **49**(4), 742–751 (2017)
54. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: HSCC, pp. 147–156 (2019)

55. Szegedy, C., et al.: Intriguing properties of neural networks. In: ICLR (2014)
56. Tian, J., Zhi, D., Liu, S., Wang, P., Katz, G., Zhang, M.: Taming reachability analysis of DNN-controlled systems via abstraction-based training (2023)
57. Tran, H.-D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 18–42. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_2
58. Tran, H.-D., Yang, X., Manzanas Lopez, D., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 3–17. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_1
59. Wang, Z., Albarghouthi, A., Prakriya, G., Jha, S.: Interval universal approximation for neural networks. In: POPL, vol. 6, pp. 1–29. ACM (2022)
60. Xue, B., Zhang, M., Easwaran, A., Li, Q.: Pac model checking of black-box continuous-time dynamical systems. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(11), 3944–3955 (2020)
61. Zhang, Y., et al.: QVIP: an ILP-based formal verification approach for quantized neural networks. In: ASE, pp. 1–13. No. 80 (2022)