NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems

Hoang-Dung Tran[1,2], Xiaodong Yang[1], Diego Manzanas Lopez[1], Patrick Musau[1], Luan Viet Nguyen[3], Weiming Xiang[5], Stanley Bak[4], and Taylor T. Johnson[1(✉)]

[1] University of Nebraska, Lincoln, USA
taylor.johnson@vanderbilt.edu
[2] Vanderbilt University, Nashville, USA
[3] University of Dayton, Dayton, USA
[4] Stony Brook University, Stony Brook, USA
[5] Augusta University, Augusta, USA

Abstract. This paper presents the Neural Network Verification (NNV) software tool, a set-based verification framework for deep neural networks (DNNs) and learning-enabled cyber-physical systems (CPS). The crux of NNV is a collection of reachability algorithms that make use of a variety of set representations, such as polyhedra, star sets, zonotopes, and abstract-domain representations. NNV supports both exact (sound and complete) and over-approximate (sound) reachability algorithms for verifying safety and robustness properties of feed-forward neural networks (FFNNs) with various activation functions. For learning-enabled CPS, such as closed-loop control systems incorporating neural networks, NNV provides exact and over-approximate reachability analysis schemes for linear plant models and FFNN controllers with piecewise-linear activation functions, such as ReLUs. For similar neural network control systems (NNCS) that instead have nonlinear plant models, NNV supports over-approximate analysis by combining the star set analysis used for FFNN controllers with zonotope-based analysis for nonlinear plant dynamics building on CORA. We evaluate NNV using two real-world case studies: the first is safety verification of ACAS Xu networks, and the second deals with the safety verification of a deep learning-based adaptive cruise control system.

The material presented in this paper is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) through contract number FA8750-18-C-0089, the National Science Foundation (NSF) under grant numbers SHF 1910017 and FMitF 1918450, and the Air Force Office of Scientific Research (AFOSR) through award numbers FA9550-18-1-0122 and FA9550-19-1-0288. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFOSR, DARPA, or NSF.

S. K. Lahiri and C. Wang (Eds.): CAV 2020, LNCS 12224, pp. 3–17, 2020.
https://doi.org/10.1007/978-3-030-53288-8_1

Keywords: Neural networks · Machine learning · Cyber-physical systems · Verification · Autonomy

1 Introduction

Deep neural networks (DNNs) have quickly become one of the most widely used tools for dealing with complex and challenging problems in numerous domains, such as image classification [10,16,25], function approximation, and natural language translation [11,18]. Recently, DNNs have been used in safety-critical cyber-physical systems (CPS), such as autonomous vehicles [8,9,52] and air traffic collision avoidance systems [21]. Although utilizing DNNs in safety-critical applications can demonstrate considerable performance benefits, assuring the safety and robustness of these systems is challenging because DNNs possess complex nonlinear characteristics. Moreover, it has been demonstrated that their behavior can be unpredictable due to slight perturbations in their inputs (i.e., adversarial perturbations) [36].

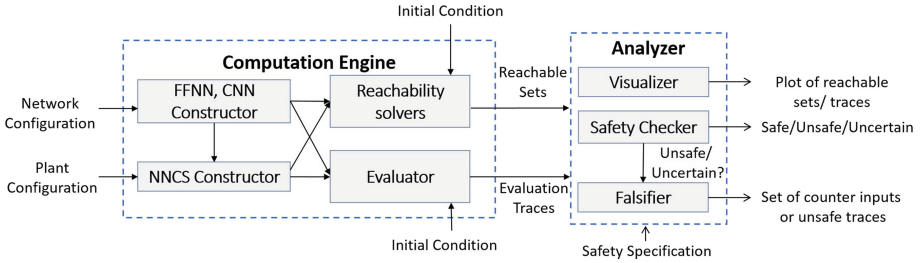

Fig. 1. An overview of NNV and its major modules and components.

In this paper, we introduce the NNV (**N**eural **N**etwork **V**erification) tool, which is a software framework that performs set-based verification for DNNs and learning-enabled CPS, known colloquially as neural network control systems (NNCS) as shown in Fig. 2[1]. NNV provides a set of reachability algorithms that can compute both the exact and over-approximate reachable sets of DNNs and NNCSs using a variety of set representations such as polyhedra [40,53–56], star sets [29,38,39,41], zonotopes [32], and abstract domain representations [33]. The reachable set obtained from NNV contains all possible states of a DNN from bounded input sets or of a NNCS from sets of initial states of a plant model. NNV declares a DNN or a NNCS to be safe if, and only if, their reachable sets do not violate safety properties (i.e., have a non-empty intersection with any state satisfying the negation of the safety property). If a safety property is violated,

[1] The source code for NNV is publicly available: https://github.com/verivital/nnv/. A CodeOcean capsule [43] is also available: https://doi.org/10.24433/CO.0221760.v1.

Table 1. Overview of major features available in NNV. Links refer to relevant files/-classes in the NNV codebase. BN refers to batch normalization layers, FC to fully-connected layers, AvgPool to average pooling layers, Conv to convolutional layers, and MaxPool to max pooling layers.

Feature	Exact analysis	Over-approximate analysis
Components	FFNN, CNN, NNCS	FFNN, CNN, NNCS
Plant dynamics (for NNCS)	Linear ODE	Linear ODE, Nonlinear ODE
Discrete/Continuous (for NNCS)	Discrete Time	Discrete Time, Continuous Time
Activation functions	ReLU, Satlin	ReLU, Satlin, Sigmoid, Tanh
CNN Layers	MaxPool, Conv, BN, AvgPool, FC	MaxPool, Conv, BN, AvgPool, FC
Reachability methods	Star, Polyhedron, ImageStar	Star, Zonotope, Abstract-domain, ImageStar
Reachable set/Flow-pipe Visualization	Yes	Yes
Parallel computing	Yes	Partially supported
Safety verification	Yes	Yes
Falsification	Yes	Yes
Robustness verification (for FFNN/CNN)	Yes	Yes
Counterexample generation	Yes	Yes

NNV can construct a complete set of counter-examples demonstrating the set of all possible unsafe initial inputs and states by using the star-based exact reachability algorithm [38,41]. To speed up computation, NNV uses parallel computing, as the majority of the reachability algorithms in NNV are more efficient when executed on multi-core platforms and clusters.

NNV has been successfully applied to safety verification and robustness analysis of several real-world DNNs, primarily feedforward neural networks (FFNNs) and convolutional neural networks (CNNs), as well as learning-enabled CPS. To highlight NNV's capabilities, we present brief experimental results from two case studies. The first compares methods for safety verification of the ACAS Xu networks [21], and the second presents safety verification of a learning-based adaptive cruise control (ACC) system.

2 Overview and Features

NNV is an object-oriented toolbox written in Matlab, which was chosen in part due to the prevalence of Matlab/Simulink in the design of CPS. NNV uses the MPT toolbox [26] for polytope-based reachability analysis and visualization [40], and makes use of CORA [3] for zonotope-based reachability analysis of nonlinear plant models [38]. NNV also utilizes the Neural Network Model Transformation Tool (NNMT) for transforming neural network models from Keras and Tensorflow into Matlab using the Open Neural Network Exchange (ONNX) format, and the Hybrid Systems Model Transformation and Translation tool (HyST) [5]

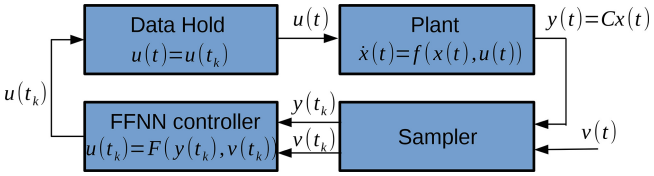

Fig. 2. Architecture of a typical neural network control system (NNCS).

for plant configuration. NNV makes use of YALMIP [27] for some optimization problems and MatConvNet [46] for some CNN operations.

The NNV toolbox contains two main modules: a *computation engine* and an *analyzer*, shown in Fig. 1. The computation engine module consists of four subcomponents: 1) the *FFNN constructor*, 2) the *NNCS constructor*, 3) *the reachability solvers*, and 4) *the evaluator*. The FFNN constructor takes a network configuration file as an input and generates a FFNN object. The NNCS constructor takes the FFNN object and the plant configuration, which describes the dynamics of a system, as inputs and then creates an NNCS object. Depending on the application, either the FFNN (or NNCS) object will be fed into a reachability solver to compute the reachable set of the FFNN (or NNCS) from a given initial set of states. Then, the obtained reachable set will be passed to the analyzer module. The analyzer module consists of three subcomponents: 1) a *visualizer*, 2) a *safety checker*, and 3) a *falsifier*. The visualizer can be called to plot the obtained reachable set. Given a safety specification, the safety checker can reason about the safety of the FFNN or NNCS with respect to the specification. When an exact (sound and complete) reachability solver is used, such as the star-based solver, the safety checker can return either "safe," or "unsafe" along with a set of counterexamples. When an over-approximate (sound) reachability solver is used, such as the zonotope-based scheme or the approximate star-based solvers, the safety checker can return either "safe" or "*uncertain*" (unknown). In this case, the falsifier automatically calls the evaluator to generate simulation traces to find a counterexample. If the falsifier can find a counterexample, then NNV returns unsafe. Otherwise, it returns unknown. Table 1 shows a summary of the major features of NNV.

3 Set Representations and Reachability Algorithms

NNV implements a set of reachability algorithms for *sequential* FFNNs and CNNs, as well as NNCS with FFNN controllers as shown in Fig. 2. The reachable set of a sequential FFNN is computed layer-by-layer. The output reachable set of a layer is the input set of the next layer in the network.

3.1 Polyhedron [40]

The polyhedron reachability algorithm computes the exact polyhedron reachable set of a FFNN with ReLU activation functions. The exact reachability

computation of layer L in a FFNN is done as follows. First, we construct the affine mapping $\bar{I}$ of the input polyhedron set I, using the weight matrix W and the bias vector b, i.e., $\bar{I} = W \times I + b$. Then, the exact reachable set of the layer R_L is constructed by executing a sequence of stepReLU operations, i.e., $R_L = stepReLU_n(stepReLU_{n-1}(\cdots(stepReLU_1(\bar{I}))))$. Since a *stepReLU* operation can split a polyhedron into two new polyhedra, the exact reachable set of a layer in a FFNN is usually a union of polyhedra. The polyhedron reachability algorithm is computationally expensive because computing affine mappings with polyhedra is costly. Additionally, when computing the reachable set, the polyhedron approach extensively uses the expensive conversion between the H-representation and the V-representation. These are the main drawbacks that limit the scalability of the polyhedron approach. Despite that, we extend the polyhedron reachability algorithm for NNCSs with FFNN controllers. However, the propagation of polyhedra in NNCS may lead to a large degree of conservativeness in the computed reachable set [38].

3.2 Star Set [38,41] (code)

The star set is an efficient set representation for simulation-based verification of large linear systems [6,7,42] where the superposition property of a linear system can be exploited in the analysis. It has been shown in [41] that the star set is also suitable for reachability analysis of FFNNs. In contrast to polyhedra, the affine mapping and intersection with a half space of a star set is more easily computed. NNV implements an enhanced version of the exact and over-approximate reachability algorithms for FFNNs proposed in [41] by minimizing the number of LP optimization problems that need to be solved in the computation. The exact algorithm that makes use of star sets is similar to the polyhedron method that makes use of *stepReLU* operations. However, it is much faster and more scalable than the polyhedron method because of the advantage that star sets have in affine mapping and intersection. The approximate algorithm obtains an over-approximation of the exact reachable set by approximating the exact reachable set after applying an activation function, e.g., ReLU, Tanh, Sigmoid. We refer readers to [41] for a detailed discussion of star-set reachability algorithms for FFNNs.

We note that NNV implements enhanced versions of earlier star-based reachability algorithms [41]. Particularly, we minimize the number of linear programming (LP) optimization problems that must be solved in order to construct the reachable set of a FFNN by quickly estimating the ranges of all of the states in the star set using only the ranges of the predicate variables. Additionally, the extensions of the star reachability algorithms to NNCS with linear plant models can eliminate the explosion of conservativeness in the polyhedron method [38,39]. The reason behind this is that in star sets, the relationship between the plant state variables and the control inputs is preserved in the computation since they are defined by a unique set of predicate variables. We refer readers to [38,39] for a detailed discussion of the extensions of the star-based reachability algorithms for NNCSs with linear/nonlinear plant models.

3.3 Zonotope [32] (code)

NNV implements the zonotope reachability algorithms proposed in [32] for FFNNs. Similar to the over-approximate algorithm using star sets, the zonotope algorithm computes an over-approximation of the exact reachable set of a FFNN. Although the zonotope reachability algorithm is very fast and scalable, it produces a very conservative reachable set in comparison to the star set method as shown in [41]. Consequently, zonotope-based reachability algorithms are usually only more efficient for very small input sets. As an example it can be more suitable for robustness certification.

3.4 Abstract Domain [33]

NNV implements the abstract domain reachability algorithm proposed in [33] for FFNNs. NNV's abstract domain reachability algorithm specifies an abstract domain as a star set and estimates the *over-approximate ranges* of the states based on the ranges of the new introduced predicate variables. We note that better ranges of the states can be computed by solving LP optimization. However, better ranges come with more computation time.

3.5 ImageStar Set [37] (code)

NNV recently introduced a new set representation called the ImageStar for use in the verification of deep convolutional neural networks (CNNs). Briefly, the ImageStar is a generalization of the star set where the anchor and generator vectors are replaced by multi-channel images. The ImageStar is efficient in the analysis of convolutional layers, average pooling layers, and fully connected layers, whereas max pooling layers and ReLU layers consume most of the computation time. NNV implements exact and over-approximate reachability algorithms using the ImageStar for serial CNNs. In short, using the ImageStar, we can analyze the robustness under adversarial attacks of the real-world VGG16 and VGG19 deep perception networks [31] that consist of >100 million parameters [37].

4 Evaluation

The experiments presented in this section were performed on a desktop with the following configuration: Intel Core i7-6700 CPU @ 3.4 GHz 8 core Processor, 64 GB Memory, and 64-bit Ubuntu 16.04.3 LTS OS.

4.1 Safety Verification of ACAS Xu Networks

We evaluate NNV in comparison to Reluplex [22], Marabou [23], and ReluVal [49], by considering the verification of safety property ϕ_3 and ϕ_4 of the ACAS Xu

neural networks [21] for all 45 networks.[2] All the experiments were done using 4 cores for computation. The results are summarized in Table 2 where (SAT) denotes the networks are safe, (UNSAT) is unsafe, and (UNK) is unknown. We note that (UNK) may occur due to the conservativeness of the reachability analysis scheme. Detailed verification results are presented in the appendix of the extended version of this paper [44]. For a fast comparison with other tools, we also tested a subset of the inputs for Property 1–4 on all the 45 networks. We note that the polyhedron method [40] achieves a timeout on most of networks, and therefore, we neglect this method in the comparison.

Verification Time. For property ϕ_3, NNV's exact-star method is about 20.7× faster than Reluplex, 14.2× faster than Marabou, 81.6× faster than Marabou-DnC (i.e., divide and conquer method). The approximate star method is 547× faster than Reluplex, 374× faster than Marabou, 2151× faster than Marabou-DnC, and 8× faster than ReluVal. For property ϕ_4, NNV's exact-star method is 25.3× faster than Reluplex, 18.0× faster than Marabou, 53.4× faster than Marabou-DnC, while the approximate star method is 625× faster than Reluplex, 445× faster than Marabou, 1321× faster than Marabou-DnC.

Table 2. Verification results of ACAS Xu networks.

ACAS XU ϕ_3	SAT	UNSAT	UNK	TIMEOUT			TIME(s)
				1 h	2 h	10 h	
Reluplex	3	42	0	2	0	0	28454
Marabou	3	42	0	1	0	0	19466
Marabou DnC	3	42	0	3	3	1	111880
ReluVal	3	42	0	0	0	0	416
Zonotope	0	2	43	0	0	0	3
Abstract Domain	0	0	45	0	0	0	8
NNV Exact Star	3	42	0	0	0	0	1371
NNV Appr. Star	0	29	16	0	0	0	52
ACAS XU ϕ_4							
Reluplex	3	42	0	0	0	0	11880
Marabou	3	42	0	0	0	0	8470
Marabou DnC	3	42	0	2	2	0	25110
ReluVal	3	42	0	0	0	0	27
Zonotope	0	1	44	0	0	0	5
Abstract Domain	0	0	45	0	0	0	7
NNV Exact Star	3	42	0	0	0	0	470
NNV Appr. Star	0	32	13	0	0	0	19

[2] We omit properties ϕ_1 and ϕ_2 for space and due to their long runtimes, but they can be reproduced in the artifact.

Conservativeness. The approximate star method is much less conservative than the zonotope and abstract domain methods. This is illustrated since it can verify more networks than the zonotope and abstract domain methods, and is because it obtains a tighter over-approximate reachable set. For property ϕ_3, the zonotope and abstract domain methods can prove safety of 2/45 networks, (4.44%) and 0/45 networks, (0%) respectively, while NNV's approximate star method can prove safety of 29/45 networks, (64.4%). For property ϕ_4, the zonotope and abstract domain method can prove safety of 1/45 networks, (2.22%) and 0/45 networks, (0.00%) respectively while the approximate star method can prove safety of 32/45, (71.11%).

4.2 Safety Verification of Adaptive Cruise Control System

To illustrate how NNV can be used to verify/falsify safety properties of learning-enabled CPS, we analyze a learning-based ACC system [1,38], in which the ego (following) vehicle has a radar sensor to measure the distance to the lead vehicle in the same lane, D_{rel}, as well as the relative velocity of the lead vehicle, V_{rel}. The ego vehicle has two control modes. In speed control mode, it travels at a driver-specified set speed $V_{set} = 30$, and in spacing control mode, it maintains a safe distance from the lead vehicle, D_{safe}. We train a neural network with 5 layers of 20 neurons per layer with ReLU activation functions to control the ego vehicle using a control period of 0.1 s.

We investigate safety of the learning-based ACC system with two types of plant dynamics: 1) a discrete linear plant, and 2) a nonlinear continuous plant governed by the following differential equations:

$$\dot{x}_{lead}(t) = v_{lead}(t),\ \dot{v}_{lead}(t) = \gamma_{lead}, \quad \dot{\gamma}_{lead}(t) = -2\gamma_{lead}(t) + 2a_{lead} - \mu v_{lead}^2(t),$$
$$\dot{x}_{ego}(t) = v_{ego}(t),\ \dot{v}_{ego}(t) = \gamma_{ego}, \qquad \dot{\gamma}_{ego}(t) = -2\gamma_{ego}(t) + 2a_{ego} - \mu v_{ego}^2(t),$$

where $x_{lead}(x_{ego})$, $v_{lead}(v_{ego})$ and $\gamma_{lead}(\gamma_{ego})$ are the position, velocity and acceleration of the lead (ego) vehicle respectively. $a_{lead}(a_{ego})$ is the acceleration control input applied to the lead (ego) vehicle, and $\mu = 0.0001$ is a friction parameter. To obtain a discrete linear model of the plant, we let $\mu = 0$ and discretize the corresponding linear continuous model using a zero-order hold on the inputs with a sample time of 0.1 s (i.e., the control period).

Verification Problem. The scenario we are interested in is when the two vehicles are operating at a safe distance between them and the ego vehicle is in speed control mode. In this state the lead vehicle driver suddenly decelerates with $a_{lead} = -5$ to reduce the speed. We want to verify if the neural network controller on the ego vehicle will decelerate to maintain a safe distance between the two vehicles. To guarantee safety, we require that $D_{rel} = x_{lead} - x_{ego} \geq D_{safe} = D_{default} + T_{gap} \times v_{ego}$ where $T_{gap} = 1.4$ s and $D_{default} = 10$. Our analysis investigates whether the safety requirement holds during the 5 s after the lead vehicle decelerates. We consider safety of the system under the following initial conditions: $x_{lead}(0) \in [90, 92]$, $v_{lead}(0) \in [20, 30]$, $\gamma_{lead}(0) = \gamma_{ego}(0) = 0$, $v_{ego}(0) \in [30, 30.5]$, and $x_{ego} \in [30, 31]$.

Table 3. Verification results for ACC system with different plant models, where VT is the verification time (in seconds).

v_lead(0)	Linear plant		Nonlinear plant	
	$Safety$	$VT(s)$	$Safety$	$VT(s)$
[29, 30]	SAFE	9.60	UNSAFE	346.62
[28, 29]	SAFE	9.45	UNSAFE	277.50
[27, 28]	SAFE	9.82	UNSAFE	289.70
[26, 27]	UNSAFE	17.80	UNSAFE	315.60
[25, 26]	UNSAFE	19.24	UNSAFE	305.56
[24, 25]	UNSAFE	18.12	UNSAFE	372.00

Verification Results. For linear dynamics, NNV can compute both the exact and over-approximate reachable sets of the ACC system in bounded time steps, while for nonlinear dynamics, NNV constructs an over-approximation of the reachable sets. The verification results for linear and nonlinear models using the over-approximate star method are presented in Table 3, which shows that safety of the ACC system depends on the initial velocity of the lead vehicle. When the initial velocity of the lead vehicle is smaller than 27 (m/s), the ACC system with the discrete plant model is unsafe. Using the exact star method, NNV can construct a *complete* set of counter-example inputs. When the over-approximate star method is used, if there is a potential safety violation, NNV simulates the system with 1000 random inputs from the input set to find counter examples. If a counterexample is found, the system is *UNSAFE*, otherwise, NNV returns a safety result of *UNKNOWN*. Figure 3 visualizes the reachable sets of the relative distance D_{rel} between two vehicles versus the required safe distance D_{safe} over time for two cases of initial velocities of the lead vehicle: $v_{lead}(0) \in [29, 30]$ and $v_{lead}(0) \in [24, 25]$. We can see that in the first case, $D_{ref} \geq D_{safe}$ for all 50 time steps stating that the system is safe. In the second case, $D_{ref} < D_{safe}$ in some control steps, so the system is unsafe. NNV supports a *reachLive* method to perform analysis and reachable set visualization on-the-fly to help the user observe the behavior of the system during verification.

The verification results for the ACC system with the nonlinear model are all *UNSAFE*, which is surprising. Since the neural network controller of the ACC system was trained with the linear model, it works quite well for the linear model. However, when a small friction term is added to the linear model to form a nonlinear model, the neural network controller's performance, in terms of safety, is significantly reduced. This problem raises an important issue in training neural network controllers using simulation data, and these schemes may not work in real systems since there is always a mismatch between the plant model in the simulation engine and the real system.

Verification Times. As shown in Table 3, the approximate analysis of the ACC system with discrete linear plant model is fast and can be done in 84 s. NNV

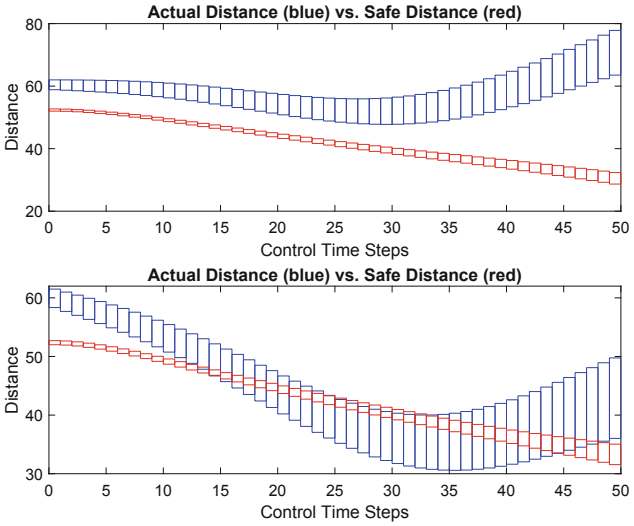

Fig. 3. Two scenarios of the ACC system. In the first (top) scenario ($v_{lead}(0) \in [29, 30]$ m/s), safety is guaranteed, $D_{rel} \geq D_{safe}$. In the second scenario (bottom) ($v_{lead}(0) \in [24,\ 25]$ m/s), safety is violated since $D_{ref} < D_{safe}$ in some control steps.

also supports exact analysis, but is computationally expensive as it constructs all reachable states. Because there are splits in the reachable sets of the neural network controller, the number of star sets in the reachable set of the plant increases quickly over time [38]. In contrast, the over-approximate method computes the interval hull of all reachable sets at each time step, and maintains a single reachable set of the plant throughout the computation. This makes the over-approximate method faster than the exact method. In terms of plant models, the nonlinear model requires more computation time than the linear one. As shown in Table 3, the verification for the linear model using the over-approximate method is 22.7× faster on average than of the nonlinear model.

5 Related Work

NNV was inspired by recent work in the emerging fields of neural network and machine learning verification. For the "open-loop" verification problem (verification of DNNs), many efficient techniques have been proposed, such as SMT-based methods [22,23,30], mixed-integer linear programming methods [14,24,28], set-based methods [4,17,32,33,48,50,53,57], and optimization methods [51,58]. For the "closed-loop" verification problem (NCCS verification), we note that the Verisig approach [20] is efficient for NNCS with nonlinear plants and with Sigmoid and Tanh activation functions. Additionally, the recent regressive polynomial rule inference approach [34] is efficient for safety verification of NNCS with nonlinear plant models and ReLU activation functions. The satisfiability modulo convex (SMC) approach [35] is also promising for NNCS with discrete linear

plants, as it provides both soundness and completeness guarantees. ReachNN [19] is a recent approach that can efficiently control the conservativeness in the reachability analysis of NNCS with nonlinear plants and ReLU, Sigmoid, and Tanh activation functions in the controller. In [54], a novel simulation-guided approach has been developed to reduce significantly the computation cost for verification of NNCS. In other learning-enabled systems, falsification and testing-based approaches [12,13,45] have shown a significant promise in enhancing the safety of systems where perception components and neural networks interact with the physical world. Finally, there is significant related work in the domain of safe reinforcement learning [2,15,47,59], and combining guarantees from NNV with those provided in these methods would be interesting to explore.

6 Conclusions

We presented NNV, a software tool for the verification of DNNs and learning-enabled CPS. NNV provides a collection of reachability algorithms that can be used to verify safety (and robustness) of real-world DNNs, as well as learning-enabled CPS, such as the ACC case study. For closed-loop systems, NNV can compute the exact and over-approximate reachable sets of a NNCS with linear plant models. For NNCS with nonlinear plants, NNV computes an over-approximate reachable set and uses it to verify safety, but can also automatically falsify the system to find counterexamples.

References

1. Model Predictive Control Toolbox. The MathWorks Inc., Natick, Massachusetts (2019). https://www.mathworks.com/help/mpc/ug/adaptive-cruise-control-using-model-predictive-controller.html
2. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
3. Althoff, M.: An introduction to cora 2015. In: Proceedings of the Workshop on Applied Verification for Continuous and Hybrid Systems (2015)
4. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, pp. 731–744. Association for Computing Machinery, New York (2019)
5. Bak, S., Bogomolov, S., Johnson, T.T.: Hyst: a source transformation and translation tool for hybrid automaton models. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, pp. 128–133. ACM (2015)
6. Bak, S., Duggirala, P.S.: Simulation-equivalent reachability of large linear systems with inputs. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 401–420. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_20

7. Bak, S., Tran, H.D., Johnson, T.T.: Numerical verification of affine systems with up to a billion dimensions. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 23–32. ACM (2019)
8. Bojarski, M., et al.: End to end learning for self-driving cars (2016). arXiv preprint arXiv:1604.07316
9. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: Deepdriving: Learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2722–2730 (2015)
10. Cireşan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification (2012). arXiv preprint arXiv:1202.2745
11. Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: Proceedings of the 25th International Conference on Machine Learning, pp. 160–167. ACM (2008)
12. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. In: NASA Formal Methods Symposium, pp. 357–372. Springer (2017)
13. 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
14. Dutta, S., Jha, S., Sanakaranarayanan, S., Tiwari, A.: Output range analysis for deep neural networks (2017). arXiv preprint arXiv:1709.09130
15. Fulton, N., Platzer, A.: Verifiably safe off-model reinforcement learning. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 413–430. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_28
16. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2414–2423 (2016)
17. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP) (2018)
18. Goldberg, Y.: A primer on neural network models for natural language processing. J. Artif. Intell. Res. **57**, 345–420 (2016)
19. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: Reachability analysis of neural-network controlled systems (2019). arXiv preprint arXiv:1906.10654
20. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Hybrid Systems: Computation and Control (HSCC) (2019)
21. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pp. 1–10. IEEE (2016)
22. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
23. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
24. Kouvaros, P., Lomuscio, A.: Formal verification of cnn-based perception systems (2018). arXiv preprint arXiv:1811.11373

25. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
26. Kvasnica, M., Grieder, P., Baotić, M., Morari, M.: Multi-parametric toolbox (MPT). In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 448–462. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24743-2_30
27. Löfberg, J.: Yalmip : A toolbox for modeling and optimization in MATLAB. In: Proceedings of the CACSD Conference,Taipei, Taiwan (2004). http://users.isy.liu.se/johanl/yalmip
28. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks (2017). arXiv preprint arXiv:1706.07351
29. Lopez, D.M., Musau, P., Tran, H.D., Johnson, T.T.: Verification of closed-loop systems with neural network controllers. In: Frehse, G., Althoff, M. (eds.) ARCH19, 6th International Workshop on Applied Verification of Continuous and Hybrid Systems, EPiC Series in Computing, vol. 61, pp. 201–210. EasyChair (2019)
30. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 243–257. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_24
31. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv preprint arXiv:1409.1556
32. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems, pp. 10825–10836 (2018)
33. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 1–30 (2019). Article 41
34. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Hybrid Systems: Computation and Control (HSCC) (2019)
35. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Hybrid Systems: Computation and Control (HSCC) (2019)
36. Szegedy, C., et al.: Intriguing properties of neural networks (2013). arXiv preprint arXiv:1312.6199
37. Tran, H.D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. In: 32nd International Conference on Computer-Aided Verification (CAV). Springer (2020)
38. Tran, H.D., Cei, F., Lopez, D.M., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control. In: ACM SIGBED International Conference on Embedded Software (EMSOFT 2019). ACM (2019)
39. Tran, H.D., Cei, F., Lopez, D.M., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control (July 2019)
40. Tran, H.D., et al.: Parallelizable reachability analysis algorithms for feed-forward neural networks. In: 7th International Conference on Formal Methods in Software Engineering (FormaliSE2019), Montreal, Canada (2019)
41. Tran, H.D., et al.: Star-based reachability analysis for deep neural networks. In: 23rd International Symposium on Formal Methods, FM 2019. Springer International Publishing (2019)

42. Tran, H.D., Nguyen, L.V., Hamilton, N., Xiang, W., Johnson, T.T.: Reachability analysis for high-index linear differential algebraic equations (daes). In: 17th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2019). Springer International Publishing (2019)
43. Tran, H.D., et al.: NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems (CodeOcean Capsule) (2020). https://doi.org/10.24433/CO.0221760.v1
44. Tran, H.D., et al.: NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems (2020). arXiv preprint arXiv:2004.05519
45. Tuncali, C.E., Fainekos, G., Ito, H., Kapinski, J.: Simulation-based adversarial test generation for autonomous vehicles with machine learning components (2018). arXiv preprint arXiv:1804.06760
46. Vedaldi, A., Lenc, K.: Matconvnet: Convolutional neural networks for matlab. In: Proceedings of the 23rd ACM International Conference on Multimedia, pp. 689–692. ACM (2015)
47. Verma, A., Murali, V., Singh, R., Kohli, P., Chaudhuri, S.: Programmatically interpretable reinforcement learning. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, Proceedings of Machine Learning Research, PMLR, 10–15 Jul 2018, vol. 80, pp. 5045–5054 (2018)
48. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in Neural Information Processing Systems, pp. 6369–6379 (2018)
49. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th USENIX Security Symposium (USENIX Security 18). USENIX Association, Baltimore (2018)
50. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals (2018). arXiv preprint arXiv:1804.10829
51. Weng, T.W., et al.: Towards fast computation of certified robustness for relu networks (2018). arXiv preprint arXiv:1804.09699
52. Wu, B., Iandola, F.N., Jin, P.H., Keutzer, K.: Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: CVPR Workshops, pp. 446–454 (2017)
53. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE Trans. Neural Netw. Learn. Syst. **29**(11), 5777–5783 (2018)
54. Xiang, W., Tran, H.D., Yang, X., Johnson, T.T.: Reachable set estimation for neural network control systems: A simulation-guided approach. IEEE Trans. Neural Netw. Learn. Syst. 1–10 (2020)
55. Xiang, W., Tran, H.D., Johnson, T.T.: Reachable set computation and safety verification for neural networks with relu activations (2017). arXiv preprint arXiv:1712.08163
56. Xiang, W., Tran, H.D., Johnson, T.T.: Specification-guided safety verification for feedforward neural networks. In: AAAI Spring Symposium on Verification of Neural Networks (2019)
57. Yang, X., Tran, H.D., Xiang, W., Johnson, T.: Reachability analysis for feedforward neural networks using face lattices (2020). arXiv preprint arXiv:2003.01226

58. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems, pp. 4944–4953 (2018)
59. Zhu, H., Xiong, Z., Magill, S., Jagannathan, S.: An inductive synthesis framework for verifiable reinforcement learning. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, pp. 686–701. Association for Computing Machinery, New York (2019)