

VERIFICATION OF LEARNING-ENABLED CYPER-PHYSICAL SYSTEMS

By

Dung Hoang Tran

Dissertation

Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

August 7, 2020

Nashville, Tennessee

Approved:

Taylor T. Johnson, Ph.D.

Janos Sztipanovits, Ph.D.

Gabor Karsai, Ph.D.

Xenofon Koutsoukos, Ph.D.

Daniel Work, Ph.D.

Stanley Bak, Ph.D.

TABLE OF CONTENTS

	Page
LIST OF TABLES	ii
LIST OF FIGURES	iii
I Introduction	1
I.1 Motivation	1
I.2 Contribution	2
I.2.1 Star-Based Reachability for Verification of Feedforward Neural Networks	2
I.2.2 Star-Based Reachability for Verification of Neural Network Control Systems	3
I.2.3 ImageStar-Based Reachability for Verification of Convolutional Neural Networks	3
I.2.4 ImageStar-Based Reachability for Verification of Semantic Segmentation Networks	4
I.2.5 NNV: Neural Network Verification Tool	5
I.3 Thesis Outline	6
I.4 Copyright Acknowledgments	6
II State-of-Art Verification, Falsification, and Testing for Deep Learning Systems	8
II.1 Deep Neural Network Verification	8
II.1.1 Geometric and Reachability Methods	8
II.1.2 MILP Methods	9
II.1.3 Satisfiability and SMT Methods	10
II.1.4 Other Optimization-Based Methods	11
II.1.5 Other Methods	12
II.1.6 Summary	13
II.2 Neural Network Control System Verification	14
II.2.1 Extended Polyhedron Approach	14
II.2.2 Verisig Approach	15
II.2.3 Sherlock Approach	15
II.2.4 ReachNN Approach	16
II.2.5 SMC Approach	16
II.2.6 Summary	17
II.3 Testing and Falsification	17
III Star-Based Reachability for Verification of Feedforward Neural Networks	18
III.1 Preliminaries	18
III.1.1 Machine Learning Models and Symbolic Verification Problem	18
III.1.2 Generalized Star Sets	19
III.1.3 Zonotope	20
III.2 Reachability of FNNs with ReLU Activation Functions	20
III.2.1 Exact and complete analysis	20
III.2.2 Over-approximate analysis	23
III.2.3 Zonotope pre-filter	24
III.2.4 Reachability algorithms (code)	25
III.3 Dealing with Other Piecewise Activation Functions	25
III.3.1 Reachability of a satlin layer	27
III.3.2 Reachability of a satlins layer	27

III.3.3	Reachability of a leaky ReLU layer	27
III.4	Evaluation	33
III.4.1	Safety Verification for ACAS Xu DNNs	33
III.4.2	Robustness Certification of Image Classification DNNs under adversarial attacks	40
IV	Star-Based Reachability for Verification of Neural Network Control Systems	41
IV.1	System Model and Problem Formulation	41
IV.1.1	System model	41
IV.1.2	Problem formulation	41
IV.2	Reachability Analysis of Neural Network Control Systems	42
IV.2.1	Exact reachability analysis of the neural network controller	42
IV.2.2	Exact reachability analysis of the discrete linear plant	43
IV.2.3	Reachability algorithm for NNCS	43
IV.2.4	Extension to NNCS with nonlinear plants	43
IV.3	Verification of Neural Network Control Systems	44
IV.3.1	Safety verification	44
IV.3.2	Characterization of safe initial condition	45
IV.4	Evaluation	45
IV.4.1	Advanced Emergency Braking System	46
IV.4.1.1	Scenario of Interest	46
IV.4.1.2	Safety Specification	46
IV.4.1.3	RL-based Controller	47
IV.4.1.4	System Identification and Validation	49
IV.4.1.5	Safety Verification of the AEBS	51
IV.4.1.6	Safe Initial Conditions of the AEBS	55
IV.4.2	Adaptive Cruise Control System	56
IV.4.2.1	System Description	56
IV.4.2.2	Scenario of Interest	57
IV.4.2.3	Verification Results	59
IV.4.2.4	Timing Performance	60
V	ImageStar-Based Reachability for Verification of Convolutional Neural Networks	61
V.1	Problem formulation	61
V.2	ImageStar	61
V.3	Reachability of CNN using ImageStars	62
V.3.1	Reachability of a convolutional layer	62
V.3.2	Reachability of an average pooling layer	63
V.3.3	Reachability of a fully connected layer	64
V.3.4	Reachability of a batch normalization layer	64
V.3.5	Reachability of a max pooling layer	65
V.3.5.1	Exact reachability of a max pooling layer	65
V.3.5.2	Over-approximate reachability of a max pooling layer	66
V.3.6	Reachability of a ReLU layer	67
V.3.6.1	Exact reachability of a ReLU layer	67
V.3.6.2	Over-approximate reachability of a ReLU layer	69
V.4	Evaluation	70
V.4.1	Robustness Verification of MNIST Classification Networks	70
V.4.2	Robustness Verification of VGG16 and VGG19	72
V.4.3	Exact Analysis vs. Approximate Analysis	74
V.5	Discussion	77
V.5.1	Effect of Input Sizes on Verification Performance	77
V.5.2	Benefit of Parallel Computing	77

VI ImageStar-Based Reachability for Verification of Semantic Segmentation Neural Networks	79
VI.1 Preliminaries	79
VI.1.1 Semantic Segmentation Networks and Reachability	79
VI.1.2 Adversarial Attacks and Robustness	79
VI.2 Verification	80
VI.2.1 Reachability of a Transposed (Dilated) Convolutional Layer	81
VI.2.2 Reachability of a Pixel-classification Layer	81
VI.2.3 Verification Algorithm	82
VI.3 Evaluation	83
VI.3.1 Robustness and Sensitivity of Different Network Architectures	84
VI.3.2 Verification Performance	88
VI.4 Training robust and verification-friendly networks.	89
VII NNV: Neural Network Verification Tool	90
VII.1 Overview and Features	90
VII.2 Set Representations and Reachability Algorithms	91
VII.2.1 Polyhedron [83]	91
VII.2.2 Star Set [84, 81] (code)	92
VII.2.3 Zonotope [74] (code)	92
VII.2.4 Abstract Domain [75]	93
VII.2.5 ImageStar Set [80] (code)	93
VII.3 Evaluation	93
VII.3.1 Safety verification of ACAS Xu networks	93
VII.3.2 Safety Verification of Adaptive Cruise Control System	94
VIII Conclusion and Future Directions	98
VIII.1 Conclusion	98
VIII.2 Future Directions	98
VIII.2.1 Scalability vs. Conservativeness	98
VIII.2.2 Formal specifications and compositional verification	99
VIII.2.3 Verification for Recurrent Neural Networks	100
VIII.2.4 Run-time Verification for Neural Network Control Systems	100
VIII.2.5 Robust and Safe Learning	101
ACKNOWLEDGMENTS	102
BIBLIOGRAPHY	104

LIST OF TABLES

Table	Page
II.1	14
II.2	14
III.1	35
III.2	36
III.3	39
IV.1	49
IV.2	54
IV.3	56
IV.4	58
IV.5	60
V.1	73
V.2	73
V.3	73
V.4	74
V.5	75

VI.1	Semantic Segmentation Network Benchmarks. Notation: ‘I’: input, ‘C’: convolution, ‘TC’: transposed convolution, ‘DC’: dilated convolution, ‘R’: ReLU, ‘B’: batch normalization, ‘AP’: average-pooling, ‘MP’: max-pooling, ‘S’: softmax, ‘L’: label (pixel classification)	83
VII.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 (including dilated convolution), TConv to transposed convolutional layers and MaxPool to max pooling layers. . .	91
VII.2	Verification results of ACAS Xu networks.	94
VII.3	Verification results for ACC system with different plant models, where VT is the verification time (in seconds).	95

LIST OF FIGURES

Figure	Page
II.1	9
II.2	13
III.1	21
III.2	24
III.3	34
III.4	37
III.5	38
III.6	39
IV.1	41
IV.2	46
IV.3	46
IV.4	47
IV.5	49
IV.6	50
IV.7	51
IV.8	52
IV.9	53

IV.10	The Over-approximation errors of the polyhedron and the interval approaches are exploded quickly after only 2 time steps. These over-approximation errors are reduced significantly by the proposed star method.	54
IV.11	Number of stars in the reachable sets of the AEBS grows over time with the exact star-based method.	54
IV.12	The inverse TTC over time is smaller than the worst case inverse full braking time $\tau^{-1}(\max(v))$. The AEBS is safe (for 60 time steps) with the initial conditions $d_0 \in [97, 97.5]$, $v_0 \in [25.2, 25.5]$	55
IV.13	Safe region of the initial conditions of the AEBS with the RL controller.	56
IV.14	Neural network adaptive cruise control system.	56
IV.15	The safe distance does not intersect with the relative distance, the ACC system is safe. . .	58
IV.16	A falsification trace of the ACC system with the first controller (3×20) and $x_{lead}(0) \in [65, 70]$. . .	59
V.1	An example of an ImageStar.	62
V.2	The reachable set of a convolutional layer with an ImageStar input set is another ImageStar.	63
V.3	The reachable set of an average pooling layer with an ImageStar input set is another ImageStar.	64
V.4	Exact reachable set of a max pooling layer with an ImageStar input set is a union of ImageStars.	66
V.5	Over-approximate reachable set of a max pooling layer with an ImageStar input set is another ImageStar.	67
V.6	stepReLU operation on an ImageStar.	68
V.7	approxStepReLU operation on an ImageStar.	69
V.8	An example of output ranges of the small MNIST classification networks using different approaches.	72
V.9	Exact ranges of VGG19 shows that VGG19 correctly classifies the input image as a bell pepper.	75
V.10	A counter-example shows that VGG19 misclassifies the input image as a strawberry instead of a bell pepper.	76
V.11	Total reachability time of each type of layers in the VGG19 in which the max pooling and ReLU layers dominate the total reachability time of the network.	76
V.12	Number of ImageStars in exact analysis increases with input size.	78
V.13	The reachability time in the exact analysis reduces as the number of cores used in computation increases.	78

VI.1	MNIST networks: $\overline{RV}, \overline{RS}, VT$ vs. $\overline{N}_{attackedPixels}$ ($\Delta_{\epsilon} = 0.001$).	84
VI.2	MNIST networks: $\overline{RV}, \overline{RS}, VT$ vs. Δ_{ϵ} ($N_{max} = 20$).	85
VI.3	Example of $R_f(N_1), N_{unknown} = 6$ ($N_{max} = 50, \Delta_{\epsilon} = 0.003$).	86
VI.4	Example $R_f(N_2), N_{unknown} = 19$ ($N_{max} = 50, \Delta_{\epsilon} = 0.003$).	86
VI.5	M2NIST networks: $\overline{RV}, \overline{RS}, VT$ vs. $\overline{N}_{attackedPixels}$ ($\Delta_{\epsilon} = 10^{-5}$).	87
VI.6	Reach-Times of M2NIST networks ($N_{max} = 25, \Delta_{\epsilon} = 10^{-5}$).	87
VI.7	Example of $R_f(N_4), N_{unrobust} = 43$ ($N_{max} = 25, \Delta_{\epsilon} = 0.00001$).	88
VI.8	Example of $R_f(N_5), N_{unrobust} = 51$ ($N_{max} = 25, \Delta_{\epsilon} = 0.00001$).	88
VII.1	An overview of NNV.	90
VII.2	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.	96
VIII.1	Run-time verification example.	100

CHAPTER I

Introduction

I.1 Motivation

Over the last two decades, there has been a blossom in artificial intelligence (AI), with implications reaching everywhere from healthcare, marketing, banking, gaming to the automotive industry. It is not an exaggeration to claim that AI will significantly benefit and affect many aspects of human life now and in the future. Although AI is powerful and performs even better than humans on many complicated tasks, it has stimulated a longstanding debate for many years between researchers, tech companies, and lawmakers as to whether we can bet human lives on AI?

To be able to use AI in safety-critical applications, there is an urgent need for methods that can prove the safety of AI systems. Conventional methods for demonstrating the safety of AI systems using extensive simulation and rigorous testing are usually costly and incomplete. For example, to achieve the catastrophic failure rates of less than one per hour, autonomous vehicle systems need to perform billions of miles of test-driving [44]. More importantly, these driving tests do not cover all corner-cases that may arise in the field. Consequently, new approaches based on formal methods, safe planning and synthesis, and robust learning are urgently needed for not only proving but also enhancing the safety and reliability of AI systems. In principle, these new approaches can automatically explore all unforeseen scenarios when verifying or falsifying the safety of AI systems. They also can generate provably correct planning decisions, safe control actions, and improve the robustness of AI systems under uncertain scenarios and adversarial attacks. As an essential step to tackle these challenging problems, **this document focuses on developing computationally efficient and scalable formal techniques to prove the safety and robustness of safety-critical AI systems at an acceptable cost.**

Deep neural networks (DNNs) have become one of the most powerful techniques to deal with challenging and complex problems such as image processing [55] and natural language translation [35, 57] due to its learning ability on large data sets. Recently, the power of DNNs has inspired a new generation of intelligent autonomy, which makes use of DNNs-based learning enable components such as autonomous vehicles [14] and air traffic collision avoidance systems [39]. Although utilizing DNNs is a promising approach, assuring the safety of autonomous applications containing neural network components is difficult because DNNs usually have complex characteristics and behavior that are generally unpredictable. Notably, it has been proved that well-trained DNNs may not be robust and are easy to be fooled by a slight change in the

input [60]. Several recent incidents in autonomous driving (e.g., Tesla and Uber) raises an urgent need for techniques and tools that can formally verify the safety and robustness of DNNs before utilizing them in safety-critical applications. In this thesis, we propose a verification framework for deep neural networks and neural-network-based control systems using set-based reachability analysis. Particularly, our approach focuses on four major challenging problems: 1) safety verification of feedforward neural networks (FNNs), 2) safety verification of neural-network-based control systems (NNCSs), 3) robustness verification of image classification convolutional neural networks (CNNs) and 4) robustness analysis of semantic segmentation networks (SSNs).

I.2 Contribution

I.2.1 Star-Based Reachability for Verification of Feedforward Neural Networks

In this document, we propose a novel, fast and scalable approach [84] for the exact and over-approximate reachability analysis of FNNs with ReLU activation functions using the concept of star sets [9], or shortly “star”. Star fits perfectly for the reachability analysis of DNNs due to its following essential characteristics: 1) an efficient (exact) representation of large input sets; 2) fast and cheap affine mapping operations; 3) inexpensive intersections with half-spaces and checking empty. By utilizing star, we avoid the expensive affine mapping operation in a polyhedron-based approach [83] and thus, reduce the verification time significantly. Our approach performs reachability analysis for feedforward DNNs layer-by-layer. In the case of exact analysis, the output reachable set of each layer is a union of a set of stars. Based on this observation, the star-based exact reachability algorithm naturally can be designed for efficient execution on multi-core platforms where each layer can handle multiple input sets at the same time. In the case of over-approximate analysis, the output reachable set of each layer is a single star which can be constructed by doing point-wise over-approximation of the reachable set at all neurons of the layer.

We evaluate the proposed algorithms in comparison, Reluplex [41], zonotope [74] and abstract domain [75] approaches on safety verification of the ACAS Xu neural networks [39] and robust certification of image classification DNN. The experimental results show that our exact reachability algorithm can achieve $27\times$ - $30\times$ faster than Reluplex when running on a multi-core platform (only six cores were used for verification). Notably, our exact algorithm can visualize the precise behavior of the ACAS Xu networks and can construct the complete set of counterexample inputs in the case that a safety property is violated. Our over-approximate reachability algorithm can achieve $960\times$ - $1400\times$ faster than Reluplex. It successfully verifies many safety properties of ACAS Xu networks while the zonotope and abstract domain approaches fail due to their large over-approximation errors. Our over-approximate reachability algorithm also provides a better robustness certification for image classification DNN in comparison with the zonotope and abstract domain

approaches.

I.2.2 Star-Based Reachability for Verification of Neural Network Control Systems

Safety verification of neural network control systems (NNCS) is a challenging problem because the behaviors of the systems are difficult to estimate or characterize. To explicitly analyze the safety of NNCS, we need to calculate the exact or overapproximate reachable set containing all possible trajectories of the plant that takes the control set from the neural network controller as inputs. The output set of the plant is feedback to the controller to compute the control set for the next control step. Therefore, if the error in the reachable set computation is large, it quickly becomes larger and larger over time, which results in too conservative reachable sets that cannot be used for safety verification. In addition, the scalability and efficiency of the reachable set computation are crucial for safety verification of control systems with DNN controllers. It is required methods that can compute the reachable set of NNCS with large neural network controllers with a reasonable computation time and a small over-approximation error. However, calculating an exact or tight, overapproximate reachable set of a neural network quickly is fundamentally difficult due to the non-linearity of the network. This challenging problem has not addressed well in the existing literature.

In this document, we propose a new reachability analysis approach for safety verification of CPS with neural network controllers using star set [81]. We limit our reachability analysis approach to feed-forward neural network controllers with ReLU/Saturation activation functions. Our reachability algorithms can compute both exact and over-approximate reachable sets of linear NNCS. Exact reachable set computation is expensive since the number of the reachable sets increases over time steps. In contrast, the over-approximate reachability scheme is much cheaper as it produces a single reachable set at each time step. Importantly, by using star sets, our reachability analysis approach can eliminate or reduce significantly the over-approximation errors which is the main reason that makes the obtained reachable sets more and more conservative over time as shown in the polyhedron approach [100, 103, 83] (and maybe in some existing methods). Our approach successfully verifies the safety of the advanced emergency braking system (AEBS) and the learning-based adaptive cruise control systems (ACC). This demonstrates the promising applicability of our approach in verifying safety properties of neural network-based autonomous systems at design time. We note that the polyhedron and interval approaches fail to prove the safety property of the system due to its over-approximation errors explode quickly over time.

I.2.3 ImageStar-Based Reachability for Verification of Convolutional Neural Networks

The convolutional neural network (CNN) is a significant innovation in the field of deep learning that has been used in many practical applications such as face recognition [49], image classification [46] and document

analysis [51]. Recently, it has been shown that CNNs are vulnerable to adversarial attacks where a well-trained CNN can give wrong results if there is a tiny change in the input [33]. To apply CNNs in safety-critical applications such as autonomous driving, there is an urgent need for evaluating the robustness of a trained CNN.

Although rigorous results and tools have been proposed for neural network verification, only a few methods can deal with CNN [45, 75, 74, 42, 68]. Importantly, in those existing approaches, only the one proposed in [68] can verify the real-world deep CNN such as VGGNet [73] using the concept of L_0 distance between two images. This impressive optimization-based approach estimates a tight bound on the number of pixels in an image that may be changed without affecting the classification result of the network. It can also generate efficiently adversarial examples that can be used for training more robust networks. As a complementary approach for verifying the robustness of real-world deep CNNs, *we propose in this document a set-based analysis method based on the concept of ImageStar, a new set representation capturing a set of an infinite number of images caused by an adversarial attack on an image. Using ImageStar, we propose the exact and over-approximate reachability algorithms to construct the reachable sets containing all possible outputs of a CNN under adversarial attacks.* These reachable sets are then used to reason about the robustness of the network. When CNN violates its robustness property, our exact reachability scheme can construct a *set of adversarial examples*. Our approach is different from [68] in two aspects. First, our method does not provide the robustness guarantee of a network in terms of the number of pixels that are allowed to be changed, i.e., L_0 distance. Instead, we prove the robustness of the network on an image attacked by disturbances bounded by arbitrary linear constraints. Second, our approach relies on reachable set computing of a network corresponding to a bounded input set. To the best of our knowledge, our approach is the first approach that can compute the exact reachable set of the real-world VGG networks with small input set while the current set-based approaches [75, 74] provides only an over-approximation of the actual reachable sets, and more importantly, they cannot deal with VGG networks due to scalability issue.

I.2.4 ImageStar-Based Reachability for Verification of Semantic Segmentation Networks

Most state-of-the-art techniques for robustness verification of DNNs focus on image classification networks [45, 75, 74, 42, 68]. There is a lack of methods that can verify the robustness of semantic segmentation networks (SSN), which perform more complex tasks than image classification networks. Recently, a rigorous testing-based approach has been proposed to evaluate the robustness of real-world SSNs [7]. In this context, a wide range of SSN architectures have been evaluated. From thorough testing-based evaluation, the authors have presented an insightful discussion about how robust different SSN architectures are corresponding to different adversarial attacks. Such work provides a better understanding and potential defenses against ad-

versarial examples. However, while such testing-based methods are scalable, they cannot provide formal guarantees for SSN robustness.

In this document, *we propose the first formal verification approach for establishing the robustness of SSNs using reachability analysis*. The central idea of our approach is, provided an input image that is attacked with some bounded disturbance, construct a reachable output set that contains all possible pixel labels. From the reachable output set, we can formally guarantee SSN robustness at the pixel-level, i.e., a pixel is provably classified correctly (or not). We can also compute the percentage of the number of pixels in the image that are correctly classified, which we call the robustness value of the SSN corresponding to the attack. An average robustness value of the network can be obtained by analyzing sets of images.

Our reachability-based approach builds on ImageStars, which are efficient data structures for verifying convolutional neural networks (CNNs) [80] to construct the input set and compute the reachable set layer-by-layer throughout the SSN. Our approach focuses on two popular SSN architectures, including dilated CNNs and transposed CNNs. We evaluate the proposed method on a set of SSNs trained with different architectures on variants of the MNIST and M2NIST data sets [50, 51, 18, 2] using digits as segmentation masks on black backgrounds. From thorough experiments, we discuss the robustness of various architectures, such as what types of SSNs are amenable to verification.

I.2.5 NNV: Neural Network Verification Tool

As a part of this document, *we introduce NNV (Neural Network Verification), which is a tool that performs set-based verification for DNNs and learning-enabled CPS* [86]. NNV offers a collection of reachability algorithms that compute both the exact and over-approximate reachable sets of the states of DNNs and NNCSs using variety of set representations such as polyhedra [83, 102, 100, 103, 101], the star set [84, 81, 82, 59], zonotopes [74], and abstract domain representations[75]. The obtained reachable set from NNV contains all the possible states of a DNN or an NNCS corresponding to bounded input sets and initial states of the plant (only for the NNCS). NNV declares a DNN and an NNCS to be safe if and only if their reachable sets do not violate safety properties. That is, all states in the reachable sets satisfy the safety properties. In the case of a violation, 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 [84, 81]. To speed up the computation, NNV exploits the power of parallel computing. The majority of reachability algorithms in NNV are more efficient when executed on multi-core platforms and clusters. We note that NNV implements enhanced versions of the star-based reachability algorithms [84]. Particularly, we minimize the number of linear programming (LP) optimization problems that we must solve in order to construct the reachable set of a DNN by quickly estimating the ranges of all states in a star set using only the ranges of the predicate

variables.

NNV has been successfully applied in verifying the safety and robustness of many real-world DNNs and learning-enabled CPS. It is currently used in Northrup Grumman Company for the DARPA Assured Autonomy project. It is also used in VMWARE research and General Motor. NNV has received positive feedback for industrial users, e.g., the feedback from a researcher in General Motor is given at <https://github.com/verivital/nnv/issues/18>).

I.3 Thesis Outline

This thesis is organized as follows. Chapter III presents reachability algorithms for verification of feed-forward neural networks (FFNN) using star set. Chapter IV extends the star-set approach to neural network control systems (NNCS). Chapter V investigates the reachability analysis and robustness verification of image classification convolutional neural networks (CNNs) using a new concept named ImageStar, an efficient extension of star set data structure. Chapter VI studies the robustness verification of semantic segmentation networks under adversarial attacks using ImageStar-based reachability analysis. Chapter VII presents the implementation of all proposed techniques in our neural network verification (NNV) tool. Chapter VIII concludes the thesis and discusses some future research directions.

I.4 Copyright Acknowledgments

The required copyright statements for permission to reprint portions of [84, 81, 80, 86] are included in the following.

- For portions of [84] reproduced in this dissertation, we acknowledge the Springer copyright: © Springer Nature Switzerland AG 2019. Reprinted, with permission, from Hoang-Dung Tran and Taylor T. Johnson, “Star-Based Reachability Analysis of Deep Neural Networks,” in Proceedings of the 23rd Symposium on Formal Method. LNCS. Springer, Porto, Portugal, October 2019, vol 11800, pp. 670–686.
- For portions of [81] reproduced in this dissertation, we acknowledge the ACM SIGBED copyright: © ACM New York 2019. Reprinted, with permission, from Hoang-Dung Tran and Taylor T. Johnson, “Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control,” in the 2019 International Conference on Embedded Software (EMSOFT 2019) and ACM Transactions on Embedded Computing. ACM New York, New York, USA, October 2019, vol 18, issues. 5s, pp. 1-22.
- For portions of [80] reproduced in this dissertation, we acknowledge the Springer copyright: © Springer-Verlag 2020. Reprinted, with permission, from Hoang-Dung Tran and Taylor T. Johnson, “Verification

of Deep Convolutional Neural Networks Using ImageStars,” in The 32nd International Conference on Computer Aided Verification. LNCS. Springer, Los Angeles, California, USA, July 2020.

- For portions of [86] reproduced in this dissertation, we acknowledge the Springer copyright: © Springer-Verlag 2020. Reprinted, with permission, from Hoang-Dung Tran and Taylor T. Johnson, “NNV: A Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems,” in The 32nd International Conference on Computer Aided Verification. LNCS. Springer, Los Angeles, California, USA, July 2020.

CHAPTER II

State-of-Art Verification, Falsification, and Testing for Deep Learning Systems

II.1 Deep Neural Network Verification

Safety verification and robustness certification of DNNs have attracted a huge attention from different communities such as machine learning [58, 45, 3, 74, 4, 91, 93, 106], formal methods [64, 41, 102, 100, 37, 24, 101], and security [31, 91, 90], and a recent survey of the area is available [98]. Verifying neural networks is a difficult problem, and it has been demonstrated that validating even simple properties about their behavior is NP-complete [41]. The difficulties encountered in verification mainly arise from the presence of activation functions and the complex structure of neural networks. Moreover, neural networks are large-scale, nonlinear, non-convex, and often incomprehensible to humans. The action of a neuron depends on its activation function described as $y_i = f(\sum_{j=1}^n \omega_{ij}x_j + \theta_i)$, where x_j is the j th input of the i th neuron, ω_{ij} is the weight from the j th input to the i th neuron, θ_i is called the bias of the i th neuron, y_i is the output of the i th neuron, and $f(\cdot)$ is the activation function. Typically, the activation function is either the rectified linear unit, logistic sigmoid, hyperbolic tangent, the exponential linear unit, or another linear function. In general, existing methods for neural network verification can be categorized into geometric (reachability) methods, mix-integer linear programming (MILP) methods, satisfiability (SAT)-based and satisfiability modulo theory (SMT)-based methods, optimization-based methods, and others.

II.1.1 Geometric and Reachability Methods

To circumvent the difficulties brought by the nonlinearities present in the neural networks, the majority of recent results focus on activation functions of piecewise linear forms, $f(x) = \max(0, x)$, and in particular the Rectified Linear Unit (ReLU). Taking advantage of the piecewise linear feature of ReLU and considering the input as polyhedra or special classes of polyhedra such as zonotopes or hyper-rectangles, the verification process can be turned into a sequence of operations on polyhedra. For instance in [100, 83], the computation process involves standard polytope operations, such as intersection and projection, and all of these can be computed by employing sophisticated computational geometry tools, such as MPT3 [34]. The essence of the approach is to be able to obtain an exact output set with respect to the input set. However, the number of polytopes involved in the computation process increases exponentially with the number of neurons in its worst case performance which makes the method not scalable to neural networks with a large number of neurons. Remarkably, due to the parallelability of the approach, parallel computing techniques can be employed to speed up the computation to some extent. In the framework of zonotopes, a verification engine

for ReLU neural networks called AI² was proposed in [31]. In their approach, the authors abstract perturbed inputs and safety specifications as zonotopes, and reason about their behavior using operations for zonotopes. The framework AI² is capable of handling neural networks of realistic size, and, in particular, their approach has had success dealing with convolutional neural networks. Another special class of polyhedra which are called interval sets or hyper-rectangles is also considered for verification problems. Those interval-based methods perform reachability analysis as propagation of interval sets across hidden layers and eventually derive the output intervals. A specification-guided method is developed to provide an adaptive partitioning methods for input space [102]. By making use of the information of specification, unnecessary partition can be avoided so that the computational complexity can be reduced significantly. In [91], an interval symbolic method is developed to compute rigorous bounds for outputs of neural networks. Their approach is easily parallelizable and makes use of symbolic interval analysis in order to minimize overestimations. The authors implement their approach as part of ReluVal, a system for checking the security properties of ReLU-based neural networks.

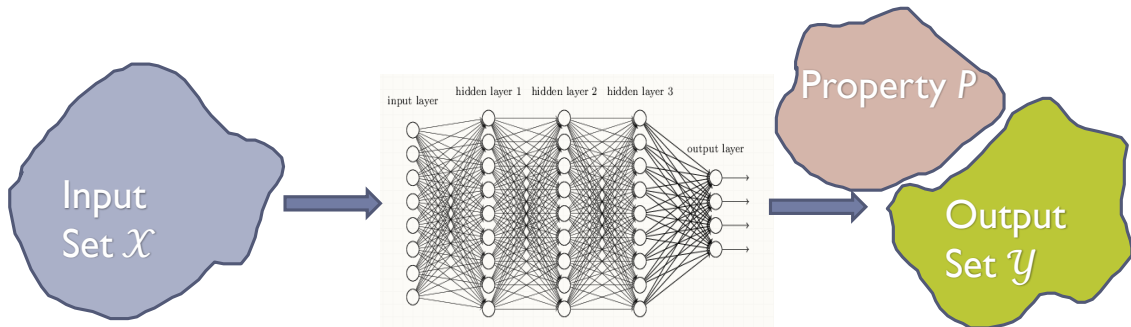


Figure II.1: Illustration of neural network reachability, where the output reachable set of a mathematical function F representing the neural network's behavior under a set of inputs I is defined and computed in an exact or overapproximative manner.

II.1.2 MILP Methods

The use of binary variables to encode piecewise linear functions is standard in optimization [88]. In [58], the constraints of ReLU functions are encoded as an MILP. Combining output specifications that are expressed in terms of linear programming (LP), the verification problem for output set eventually turns to the feasibility

problem of MILP. For layer i , the MILP encoding is given as

$$\begin{aligned}
 C_i = \{ & x_j^{[i]} \geq W_j^{[i]} x^{[i-1]} + \theta_j^i, \\
 & x_j^{[i]} \leq W_j^{[i]} x^{[i-1]} + \theta_j^i + M\delta_j^{[i]}, \\
 & x_j^{[i]} \geq 0, \\
 & x_j^{[i]} \leq M(1 - \delta_j^{[i]} \mid j = 1 \dots |L^{[i]}| \}
 \end{aligned} \tag{II.1}$$

where M is sufficiently large so that it is larger than the maximum possible output at any node. A similar MILP problem is formulated in [79], where the authors conduct a robustness analysis and search for adversarial examples in ReLU neural networks. It is well known that MILP is an NP-hard problem and, in [24, 26], the authors elucidate significant efforts for solving MILP problems efficiently to make the approach scalable. Their methods combine MILP solvers with a local search yielding a more efficient solver for range estimation problems of ReLU neural networks than several other approaches. Basically, a local search is conducted using a gradient search and then a global search is formulated as MILP. Instead of finding the global optimum directly, it performs the search seeking values greater/smaller than the upper/lower bound obtained in the preceding local search. This is the primary reason for the computational complexity reduction. This MILP-based approach is integrated in their tool called Sherlock [23]. In [61], an MILP encoding scheme is used for a class of neural networks whose input spaces are encoded as binaries. This MILP encoding has a similar flavor to the other encodings present in the research literature for non-binarized networks. In their framework, since all the inputs are integer values, the real valued variables can be rounded so that they can be safely removed, resulting in a reformulated integer linear programming (ILP) problem that is smaller in comparison to the original MILP encoding. With the ILP encoding, a SAT solver is utilized in order to reason about the behavior of a mid-size binarized neural network.

II.1.3 Satisfiability and SMT Methods

In [41], an SMT solver called Reluplex is developed. An algorithm, that stems from the Simplex algorithm for linear functions, for ReLU functions is proposed. Due to the piecewise linear feature of ReLU functions, each node is divided into two nodes. Thus, in their formulation, each node consists of a forward-facing and backward-facing node. If the ReLU semantics are not satisfied, two additional update functions are given to fix the mismatching pairs. Thus, the search process is similar to the Simplex algorithm that pivots and updates the basic and non-basic variables with the addition of a fixing process for ReLU activation pairs. This method is applied on a deep neural network implementation of a next-generation airborne collision avoidance system for unmanned aircrafts (ACAS-X), which has been used as a benchmark for a number of successive works.

In [69], they use bounded model checking (BMC) to create formulas that are solved using the SMT-solver iSAT3, which is able to deal with transcendental functions such as \exp and \cos that frequently appear in neural network controllers and plants. Although the verification framework is rigorously developed, the verification problem is hardly solved due to the curse of dimensionality and state-space explosion problems. An approach for finding adversarial inputs using SMT solvers that relies on a layer-by-layer analysis is presented in [37]. The work focuses on the robustness of a neural network where safety is defined in terms of classification invariance within a small neighborhood of one individual input. An exhaustive search of the region is conducted by employing discretization and propagating the analysis layer by layer. In a similar manner, a recent paper, proposed by Ruan et al. [68], generalizes the local robustness criterion into a global notion on a set of test examples. In another work, a software tool, called Planet, was developed based on the MILP verification approaches [28]. This LP-based framework combine SAT solving and linear over-approximation of piecewise linear functions in order to verify ReLU neural networks against convex specifications. Given the output of a ReLU denoted by d and the input $c \in [l, u]$, the relationship between c and d can be approximated by the linear constraints $d \geq 0$, $d \geq c$, and $d \geq u \frac{c-l}{u-l}$. Based on the LP problem formulation, additional heuristic algorithms were developed to detect infeasibility and imply phase inference faster. Pulina et al present an abstraction-refinement and SMT-based tool for verifying feed-forward neural networks. Their scheme is based on encoding the network into a boolean satisfaction problem over linear arithmetic constraints [65].

II.1.4 Other Optimization-Based Methods

As some of the earliest papers for neural network verification, in [64, 66], a piecewise-linearization of the nonlinear activation functions is used to reason about their behavior. In this framework, the authors replace the activation functions with piecewise constant approximations and use the bounded model checker hybrid satisfiability (HySAT) [29] to analyze various properties. The authors highlight the difficulty of scaling this technique and, currently, are only able to tackle small networks with at most 20 hidden nodes. In [101], a simulation-based approach was developed, which used a finite number of simulations/computations to estimate the reachable set of multi-layer neural networks in a general form. Despite this success, the approach lacks the ability to resolve the reachable set computation problem for neural networks that are large-scale, non-convex, and nonlinear. Still, simulation-based approaches, like the one developed in [101], present a plausibly practical and efficient way of reasoning about neural network behavior. The critical step in improving simulation-based approaches is bridging the gap between finitely many simulations and the essentially infinite number of inputs that exist in the continuity set. A critical concept that is introduced in the work is called maximal sensitivity, which measures of the maximal deviation of outputs for a set of inputs suffering disturbances in a bounded cell. The output set of the neural network can be over-approximated by the union

of a finite number of reachtubes computed using a union of individual cells that cover the input set. Thus, verification of a network can be done by checking the existence of intersections of the estimated reachable set and safety regions. This approach has been extended to allow for the reachable set estimation and verification of nonlinear autoregressive-moving average (NARMA) models in the form of neural networks [97] as well as closed-loop system verification with the help of the state-of-the-art reachability tool for hybrid systems dealing with the plant dynamics [96]. In particular, it is applicable to a variety of neural networks regardless of the specific form of the activation functions. Given a neural network, there is a trade-off between the precision of the reachable set estimation and the number of simulations used to execute the procedure. In addition, since the approach executes in a layer-by-layer manner, the approximation error will accumulate as the number of layers present in the network increases. In this case, more simulations are required at the expense of increasing the computational cost. A novel approach for neural network verification based on optimization duality has been developed [27]. The verification problem is posed as an optimization problem that tries to find the largest violation of a property related to the output of the network.

II.1.5 Other Methods

There exists a rich literature of other methods for neural network verification [98, 56], but we highlight a few. A comparison of the verification approaches mentioned above can be found in [15]. Additionally, the authors present a novel approach for neural network verification called Branch and Bound Optimization. This approach adds one more layer behind the output layer $cy - b$ to represent the linear property $cy > b$ that we wish to verify. If $cy - b > 0$, it means that the property is satisfied, otherwise it is unsatisfiable. Thus, the verification problem is converted into a computation of the minimum or maximum value of the output of the neural network. By treating the neural network as a nonlinear function, model-free optimization methods are utilized to find optimal solution. In order to have a global optimum, the input space is also discretized into sub-regions. This approach is not only applicable to ReLU neural networks, but the model-free method allows the approach to be applied to neural networks with more general activation functions. However, despite its generalization capabilities, in the model-free framework, there is no guarantee that the algorithm will converge to a solution.

Cheng et al. have studied the verification of Binarized neural networks (BNNs) [17]. The forward propagation of input signals is reduced to bit arithmetic. The authors argue that the verification of BNNs can be reduced to hardware verification and represents a more scalable problem than traditional neural network verification. A randomized approach for rigorously verifying neural networks in safety critical applications has been developed [105]. In an effort to mitigate challenges related to the curse of dimensionality, the authors make use of Monte Carlo methods to estimate the probability of neural network failure. However,

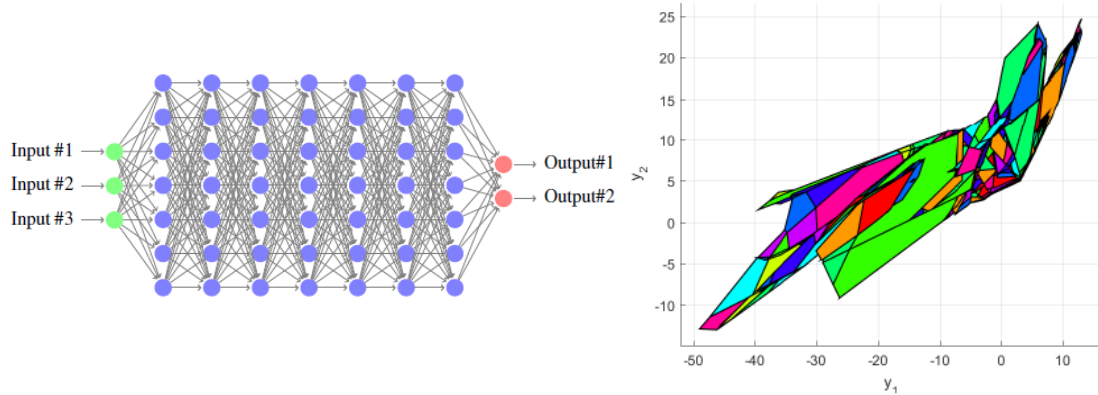


Figure II.2: Example output reachable set computation for a neural network with 3 inputs, 2 outputs, and 7 hidden layers with 7 neurons each, where all activation functions are ReLUs and all parameters of the network (weights, biases) are chosen randomly. The input set $I = \{x \mid \|x\|_\infty \leq 1, x \in \mathbb{R}^3\}$ is a cube and convex, while the output set shown is non-convex, represented as the union of the different colored polygons.

although Monte Carlo methods are more efficient than methods that deterministically search through hyper-rectangular input spaces, they are probabilistic in nature. The authors further demonstrate that although the number of samples needed to guarantee this may be large, it is not as prohibitive as other methods. Another fascinating area in neural network verification is falsification. Several ideas for integrating semantics into adversarial learning have been explored, including a semantic modification space and the use of more detailed information about the outputs produced by machine learning models [22]. In work by Tsui Weng et al. [94], an attack independent robustness metric against adversarial examples for neural networks is described. Their approach converts the robustness analysis into a local Lipschitz constant estimation problem and uses Extreme Value Theory for efficient solving. In [78], an automatic test case generator is presented that leverages real-world changes in driving conditions like rain, fog, lighting conditions, etc. The tool, called DeepTest, systematically explores different parts of the deep neural network logic by generating test inputs that maximize the number of activated neurons. An improved version of the tool, called DeepXplore, is proposed in [63], which is the first efficient whitebox testing framework for large-scale deep learning systems.

II.1.6 Summary

In summary, analyzing the behavior of a DNN can broadly be categorized into exact and over-approximate analyses. For the exact analysis, the SMT-based [41] and polyhedron-based approaches [83, 100] are notable representatives. For the over-approximate analysis, the mixed-integer linear program (MILP) [24], interval arithmetic- [91, 90], zonotope- [74], input partition- [102], linearization- [93], and abstract-domain- [75] based are fast and efficient approaches. While the over-approximate analysis is usually faster and more scal-

Name	Network Type	Approach	Activation Function	Size of Network	Completeness
Reluplex [41]	FFNN	SMT	ReLU	300 neurons	Yes
Marabou [42]	FFNN	SMT	ReLU	300 neurons	Yes
DeepZ [74]	FFNN, CNN	Reachability	ReLU, Sigmoid, Tanh	88000 neurons	No
DeepPoly [75]	FFNN, CNN	Reachability	ReLU, Sigmoid, Tanh	88000 neurons	No
Plannet [28]	FFNN, CNN	SAT, LP	ReLU	1341 neurons	Yes
Sherlock [76]	FFNN	MILP	ReLU	3822 neurons	No
ReluVAL [91]	FFNN	Interval Analysis	ReLU	300 neurons	Yes
Fast-Lin [93]	FFNN	Optimization	ReLU	7168 neurons	No
Fast-Lip [93]	FFNN	Optimization	ReLU	7168 neurons	No
NSVerify [58]	FFNN	MILP	ReLU	4746 neurons	Yes

Table II.1: Verification approaches for neural networks.

Name	Plant Dynamics	Discrete/Continuous	Activation Function	Size of Controller
Polyhedron-based [103]	Linear	Discrete	ReLU	≤ 100 neurons
Verisig [38]	Linear, Nonlinear	Discrete, Continuous	Sigmoid, Tanh	≤ 50 neurons
SMC-based [77]	Linear	Discrete	ReLU	≤ 200 neurons
Sherlock [76]	Linear, Nonlinear	Discrete, Continuous	ReLU	≤ 500 neurons
ReachNN [36]	Nonlinear	Discrete, Continuous	ReLU, Sigmoid, Tanh	≤ 100 neurons

Table II.2: Verification approaches for NNCS.

able than the exact analysis, it guarantees only the soundness of the result. In contrast, the exact analysis is usually more time-consuming and less scalable. However, it guarantees both the soundness and completeness of the result [41]. Although the over-approximate analysis is fast and scalable, it is unclear how good the over-approximation is in term of conservativeness since the exact result is not available for comparison. Importantly, if an over-approximation approach is too conservative for neural networks with small or medium sizes, it will potentially produce huge conservative results for DNNs with a large number of layers and thousands of neurons since the over-approximation error is accumulated quickly over layers. Therefore, a scalable, exact reachability analysis is crucial not only for formal verification of DNNs, but also for estimating the conservativeness of current and up-coming over-approximation approaches. Table II.1 summarizes representative approaches for neural network verification.

II.2 Neural Network Control System Verification

Verification of CPS with learning-enabled components have become an emerging research topic recently. Several methods have been proposed to verify the safety of feedback neural network control systems [25, 97, 38, 76, 103, 77].

II.2.1 Extended Polyhedron Approach

The early polyhedron-based approach for ReLU feedforward neural networks has been extended for safety verification of a neural network controlled systems in [103] where a ReLU network controller controls a discrete linear switching plant model. This approach computes the polyhedron-based reachable set of the neural network controller at every control step. The convex hull of all polyhedra in the reachable set is used as the control input to the plant model to compute the plant’s reachable set. The proposed method works for neural network controlled systems with a small number of neurons and low-dimensional plant models. However, taking the convex hull of all polyhedra in the reachable set as the control input to the plant makes this approach become conservative due to the over-approximation error. Importantly, the over-approximation error may be accumulated quickly over time steps when dealing with a large set of initial states.

Recently, the authors have proposed a new simulation-guided approach for safety verification of an NNCS [99]. The novel idea of this approach is the combination of interval arithmetic and simulation-guided input set partition to estimate the neural network’s output ranges, which are used as inputs for the plant model reachability. The experiments have shown that the proposed approach can efficiently verify the safety of a neural network-based adaptive cruise control system. Additionally, this approach is much less expensive than the maximum sensitivity based approach [101].

II.2.2 Verisig Approach

Verisig [38] proposes an approach that transforms a neural network controller with *sigmoid* activation function to an equivalent nonlinear hybrid system by exploiting the fact that sigmoid is the solution to a quadratic differential equation. Particularly, a neural network with L layers and N neurons per layer can be represented as a hybrid system with $L + 1$ modes and $2N$ states. The hybrid system representation of the neural network controller is combined with the plant model to form a single hybrid system representation of the neural network control system, which is then fed to hybrid systems verification tools such as Flow* [16] and dReach [43] for verifying the system safety properties. The Verisig approach has applied successfully on safety verification of the mountain car benchmark and DNN-based quadcopter application. The authors evaluated the scalability of their approach via investigating the reachable set computation times between the Verisig + Flow* and the prior mix-integer linear programming (MILP) approach for DNNs with increasing sizes. Interestingly, the Verisig + Flow* reachability time increases linearly with the size of the network, while the MILP approach shows an exponential jump in the reachability time.

II.2.3 Sherlock Approach

A common approach for the reachability of NNCS combines the flow-pipe construction of ODEs and range analysis for neural network controllers to construct the reachable set. However, this approach may suffer from large overestimation errors due to the wrapping effect [62], which motivated the proposal of a new abstraction method to abstract function computed by the network using a local polynomial approximation along with rigorous error bounds [76]. Formally, given a set of inputs, the authors obtain a polynomial function capturing the outputs using regression. The difference between the polynomial abstraction and the network is bounded by an error interval that is computable. The combination of the polynomial function and the error interval yields a local Taylor model overapproximation of the network, which can be integrated into the flow-pipe construction tool Flow*. The major challenge in this approach is the computation of the error interval, which was solved in three steps. First, the author obtained a piecewise linearization (PWL) of the polynomial with a given tolerance bound on their difference. Then, the maximum and minimum differences between the network and PWL are computed using MILP solver combining with the local gradient descent search method [24]. The experimental results have shown that this abstraction-based method is fast and scalable for NNCS verification, and more importantly, it can reduce overapproximation errors significantly in the reachable set computation process and can deal with relatively large input sets.

II.2.4 ReachNN Approach

Similar to the Sherlock approach, [36] proposes new reachability approach based on Bernstein polynomials abstraction to verify NNCS with more general activation functions such as Sigmoid and Tanh. To avoid wrapping effect in the reachability, tightly bound on the difference between Bernstein polynomial and the corresponding neural network is essential. The authors proposed two approaches to compute this bound, including a priori theoretical approach based on existing results on Bernstein polynomials and a posteriori approach based on adaptive sampling. Although the first approach is time-efficient, it usually obtains a coarse result that causes the accumulation in the over-approximation error in the reachability. In contrast, the posterior approach requires more computation time but can get very tightly bound. Theoretically, we can achieve arbitrary precision in computing the bound by increasing the number of sampling points of the input set of the network. However, higher precision requires larger computation time and resources. The proposed approach can speed up by parallelizing the calculation of the sampling-based error bound using GPU. The experimental results have shown that the ReachNN approach is less conservative than the Sherlock and Verisig approaches but requires much more computation time.

II.2.5 SMC Approach

The Satisfiability Modulo Convex (SMC) approach [77] is the first approach to dealing with LiDAR image inputs handled by a neural network controller. It synthesizes a set of safe initial states of a neural network control system working in a given workspace containing a set of polytopic obstacles. The SMC approach’s central idea is the construction of a finite-state abstraction of the system, which is then analyzed by a standard reachability method to compute the set of safe initial states. This approach first constructs a set of affine imaging-functions mapping the autonomous robot position to the LiDAR image by partitioning the workspace into smaller sets called “imaging-adapted sets” in polynomial-time. A finite-state abstraction of the closed-loop system is then computed by leveraging the partitioned workspace, the discrete-time linear dynamics of the robot, and a pre-trained ReLU network controller. Finally, an SMC formula is encoded to analyze the behavior of the network under the constraints on the robot dynamics and partitioned workspace. The encoded SMC formula is fed into an SMC solver [72], which combines a Boolean satisfiability solver and a convex programming solver, to iteratively reason about the behavior of the system.

II.2.6 Summary

The main challenges in safety verification of NNCS are the over-approximation error in the reachability analysis step and the computation time. So far, different methods have both advantages and disadvantages and can deal with a variety of activation functions and plant models. The polyhedron approach may face with the over-approximation error accumulation over time steps. The Verisig approach can avoid the over-approximation error accumulation due to the wrapping effect. However, it is only efficient for network controllers with Sigmoid/Tanh activation functions. Similarly, the Sherlock approach can only deal with the ReLU activation function. The ReachNN approach is the one that can deal with different types of activation functions while reducing the over-approximation error. However, the over-approximation error reduction comes with an expensive computational cost. The SMC approach currently can only deal with discrete linear plant models. A summary of recent verification methods is given in Table II.2.

II.3 Testing and Falsification

Besides verification methods, testing and falsification for NNCS are efficient approaches for enhancing the safety and reliability of learning-based autonomous systems. These approaches are especially scalable and efficient for systems with larger neural network components, e.g., perception component in autonomous driving cars, where the verification approaches are not applicable due to their limited scalability. In the testing context, a simulation-based test generation framework for autonomous vehicles with machine learning components has been proposed in [87] to enhance the reliability of autonomous driving systems. In the

falsification context, a compositional falsification framework for CPS with machine learning components has been proposed in [20]. In this framework, a temporal logic falsifier cooperates efficiently with a machine learning analyzer to find falsifying executions of the system. The effectiveness of the proposed framework was shown via Automatic Emergency Braking System (AEBS).

CHAPTER III

Star-Based Reachability for Verification of Feedforward Neural Networks

III.1 Preliminaries

III.1.1 Machine Learning Models and Symbolic Verification Problem

A feed-forward neural network (FNN) consists of an input layer, an output layer, and multiple hidden layers in which each layer comprises of neurons that are connected to the neurons of preceding layer labeled using weights. Given an input vector, the output of an FNN is determined by three components: the weight matrices W_k , representing the weighted connection between neurons of two consecutive layers $k - 1$ and k , the bias vectors b_k of each layer, and the activation function f applied at each layer. Mathematically, the output of a neuron i is defined by:

$$y_i = f(\sum_{j=1}^n \omega_{ij}x_j + b_i),$$

where x_j is the j^{th} input of the i^{th} neuron, ω_{ij} is the weight from the j^{th} input to the i^{th} neuron, b_i is the bias of the i^{th} neuron. In this paper, we are interested in FNN with ReLU activation functions defined by $ReLU(x) = \max(0, x)$.

Definition 1 (Reachable Set of FNN). *Given a bounded convex polyhedron input set defined as $\mathcal{I} \triangleq \{x \mid Ax \leq b, x \in \mathbb{R}^n\}$, and an k -layers feed-forward neural network $F \triangleq \{L_1, \dots, L_k\}$, the reachable set $F(\mathcal{I}) = \mathcal{R}_{L_k}$ of the neural network F corresponding to the input set \mathcal{I} is defined incrementally by:*

$$\begin{aligned} \mathcal{R}_{L_1} &\triangleq \{y_1 \mid y_1 = f_1(W_1x + b_1), x \in \mathcal{I}\}, \\ \mathcal{R}_{L_2} &\triangleq \{y_2 \mid y_2 = f_2(W_2y_1 + b_2), y_1 \in \mathcal{R}_{L_1}\}, \\ &\vdots \\ \mathcal{R}_{L_k} &\triangleq \{y_k \mid y_k = f_k(W_ky_{k-1} + b_k), y_{k-1} \in \mathcal{R}_{L_{k-1}}\}, \end{aligned}$$

where W_k , b_k and f_k are the weight matrix, bias vector and activation function of the k^{th} layer L_k , respectively. The reachable set \mathcal{R}_{L_k} contains all outputs of the neural network corresponding to all input vectors x in the input set \mathcal{I} .

Definition 2 (Safety Verification of FNN). *Given a k -layers feed-forward neural network F , and a safety specification \mathcal{S} defined as a set of linear constraints on the neural network outputs $\mathcal{S} \triangleq \{y_k \mid Cy_k \leq d\}$, the neural network F is called to be safe corresponding to the input set \mathcal{I} , we write $F(\mathcal{I}) \models \mathcal{S}$, if and only if $\mathcal{R}_{L_k} \cap \neg\mathcal{S} = \emptyset$, where \mathcal{R}_{L_k} is the reachable set of the neural network with the input set \mathcal{I} , and \neg is the*

symbol for logical negation. Otherwise, the neural network is called to be unsafe $F(\mathcal{S}) \neq S$.

III.1.2 Generalized Star Sets

Definition 3 (Generalized Star Set [9]). A generalized star set (or simply star) Θ is a tuple $\langle c, V, P \rangle$ where $c \in \mathbb{R}^n$ is the center, $V = \{v_1, v_2, \dots, v_m\}$ is a set of m vectors in \mathbb{R}^n called basis vectors, and $P: \mathbb{R}^m \rightarrow \{\top, \perp\}$ is a predicate. The basis vectors are arranged to form the star's $n \times m$ basis matrix. The set of states represented by the star is given as:

$$\llbracket \Theta \rrbracket = \{x \mid x = c + \sum_{i=1}^m (\alpha_i v_i) \text{ such that } P(\alpha_1, \dots, \alpha_m) = \top\}. \quad (\text{III.1})$$

Sometimes we will refer to both the tuple Θ and the set of states $\llbracket \Theta \rrbracket$ as Θ . In this work, we restrict the predicates to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq d$ where, for p linear constraints, $C \in \mathbb{R}^{p \times m}$, α is the vector of m -variables, i.e., $\alpha = [\alpha_1, \dots, \alpha_m]^T$, and $d \in \mathbb{R}^{p \times 1}$. A star is an empty set if and only if $P(\alpha)$ is empty.

Proposition III.1.1. Any bounded convex polyhedron $\mathcal{P} \triangleq \{x \mid Cx \leq d, x \in \mathbb{R}^n\}$ can be represented as a star.

Proof. The polyhedron \mathcal{P} is equivalent to the star set Θ with the center $c = [0, 0, \dots, 0]^T$, the basic vectors $V = \{e_1, e_2, \dots, e_n\}$ in which e_i is the i^{th} basic vector of \mathbb{R}^n , and the predicate $P(\alpha) \triangleq C\alpha \leq d$. \square

Proposition III.1.2. [Affine Mapping of a Star] Given a star set $\Theta = \langle c, V, P \rangle$, an affine mapping of the star Θ with the affine mapping matrix W and offset vector b defined by $\bar{\Theta} = \{y \mid y = Wx + b, x \in \Theta\}$ is another star with the following characteristics.

$$\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P} \rangle, \bar{c} = Wc + b, \bar{v} = \{Wv_1, Wv_2, \dots, Wv_m\}, \bar{P} \equiv P.$$

Proof. From the definition of a star, we have $\bar{\Theta} = \{y \mid y = Wc + b + \sum_{i=1}^m (\alpha_i Wv_i)\}$, such that $P(\alpha_1, \dots, \alpha_m) = \top$ which implies that $\bar{\Theta}$ is another star with the center $\bar{c} = Wc + b$, basic vectors $\bar{V} = \{Wv_1, Wv_2, \dots, Wv_m\}$ and the same predicate P as the original star Θ . \square

Proposition III.1.3 (Star and Half-space Intersection). The intersection of a star $\Theta \triangleq \langle c, V, P \rangle$ and a half-space $\mathcal{H} \triangleq \{x \mid Hx \leq g\}$ is another star with following characteristics.

$$\begin{aligned} \bar{\Theta} &= \Theta \cap \mathcal{H} = \langle \bar{c}, \bar{V}, \bar{P} \rangle, \bar{c} = c, \bar{V} = V, \bar{P} = P \wedge P', \\ P'(\alpha) &\triangleq (H \times V_m)\alpha \leq g - H \times c, V_m = [v_1 \ v_2 \ \dots \ v_m]. \end{aligned}$$

Proof. For any $x \in \Theta \cap \mathcal{H}$, we have $x = c + \sum_{i=1}^m (\alpha_i v_i) \wedge Hx \leq g$, or equivalently, $x = c + \sum_{i=1}^m (\alpha_i v_i) \wedge H \times V_m \times \alpha \leq g - H \times c$ which implies that the intersection is another star with the same center c and basic vectors V as Θ , and an updated predicate $\bar{P} = P \wedge P', P'(\alpha) \triangleq H \times V_m \times \alpha \leq g - H \times c$. \square

III.1.3 Zonotope

Definition 4 (Zonotope). A zonotope Z is a tuple $\langle l, G \rangle$ where $l \in \mathbb{R}^n$ is the center, $G = \{g_1, g_2, \dots, g_m\}$ is a set of m generators in \mathbb{R}^n . The set of states represented by a zonotope is given as:

$$Z = \{x \mid x = l + \sum_{i=1}^m (\alpha_i g_i) \text{ such that } -1 \leq \alpha_i \leq 1\}. \quad (\text{III.2})$$

A zonotope is basically a star set in which all predicate variables in the ranges of $[-1, 1]$. The affine mapping of a zonotope is another zonotope. However, the intersection of a zonotope and a half-space generally is not a zonotope. One advantage of a zonotope compared with a star set is that we can compute quickly the ranges of a state in a zonotope without solving LP optimization. For example, the range of the state $x(j)$ in a zonotope is:

$$l(j) - \sum_i^m |g_i(j)| \leq x(j) \leq l(j) + \sum_i^m |g_i(j)|.$$

III.2 Reachability of FNNs with ReLU Activation Functions

III.2.1 Exact and complete analysis

In this section, we investigate the exact and complete analysis of FNNs with ReLU activation functions. Since any bounded convex polyhedron can be represented as a star (Proposition III.1.1), we assume the input set \mathcal{S} of an FNN is a star set. From Definition 5, one can see that the reachable set of an FNN is derived layer-by-layer. Since the affine mapping of a star is also a star (Proposition III.1.2), the core step in computing the exact reachable set of a layer with a star input set is applying the ReLU activation function on the star input set, i.e., compute $ReLU(\Theta)$, $\Theta = \langle c, V, P \rangle$. For a layer L with n neurons, the reachable set of the layer can be computed by executing a sequence of n stepReLU operations as follows $\mathcal{R}_L = ReLU_n(ReLU_{n-1}(\dots ReLU_1(\Theta)))$.

The stepReLU operation on the i^{th} neuron, i.e., $ReLU_i(\cdot)$, works as follows. First, the input star set Θ is decomposed into two subsets $\Theta_1 = \Theta \wedge x_i \geq 0$ and $\Theta_2 = \Theta \wedge x_i < 0$. Note that from Proposition III.1.3, Θ_1 and Θ_2 are also stars. Let assume that $\Theta_1 = \langle c, V, P_1 \rangle$ and $\Theta_2 = \langle c, V, P_2 \rangle$. Since the later set has $x_i < 0$, applying the ReLU activation function on the element x_i of the vector $x = [x_1 \dots x_i \ x_{i+1} \dots x_n]^T \in \Theta_2$ will lead to the new vector $x' = [x_1 \ x_2 \ \dots \ 0 \ x_{i+1} \ \dots \ x_n]^T$. This procedure is equivalent to mapping Θ_2 by the mapping matrix $M = [e_1 \ e_2 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$. Also, applying the ReLU activation function on the element x_i of the vector $x \in \Theta_1$ does not change the set since we have $x_i \geq 0$. Consequently, the result of the stepReLU operation on

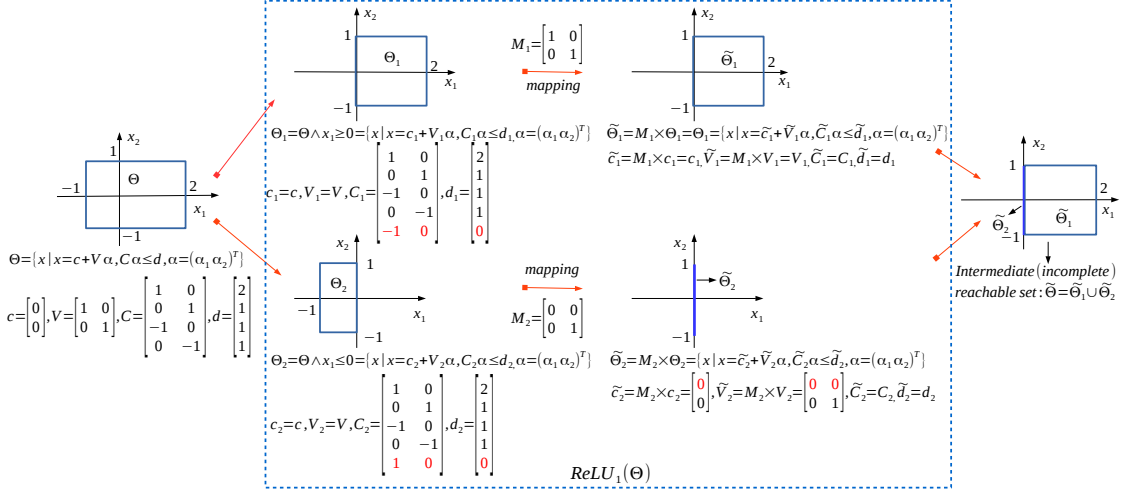


Figure III.1: An example of a stepReLU operation on a layer with two neurons.

input set Θ at the i^{th} neuron is a union of two star sets $ReLU_i(\Theta) = \langle c, V, P1 \rangle \cup \langle Mc, MV, P2 \rangle$. A concrete example of the first stepReLU operation on a layer with two neurons is depicted in Figure III.1.

The number of stepReLU operation can be reduced if we know beforehand the ranges of all states in the input set. For example, if we know that x_i is always larger than zero, then we have $ReLU_i(\Theta) = \Theta$, or in other words, we do not need to execute the stepReLU operation on the i^{th} neuron. Therefore, to minimize the number of stepReLU operations and the computation time, we first *determine the ranges of all states in the input set by solving n -linear programming problems*.

Lemma III.2.1. *The worst-case complexity of the number of stars in the reachable set of an N -neurons FNN is $\mathcal{O}(2^N)$.*

Proof. Given a star input set, each stepReLU operation produces at most two more stars which leads to the total number of stars in the worst case of one layer is 2^{n_L} where n_L is the number of neurons in the layer. For an FNN, the output reachable sets of one layer is the inputs of the next layer. Therefore, in the worst-case, the total number of stars in the reachable set of an k -layers and N -neurons FNN is $2^{n_{L_1}} \times \dots \times 2^{n_{L_k}} = 2^{n_{L_1} + \dots + n_{L_k}} = 2^N$. \square

Lemma III.2.2. *The worst-case complexity of the number of constraints of a star in the reachable set of an N -neuron FNN is $\mathcal{O}(N)$.*

Proof. From the stepReLU sub-procedure, we can see that given a star input set Θ , each stepReLU operation produces one or two stars that have at most one more constraint than the star input set. Therefore, with a layer of n neurons, at most n - stepReLU operations are executed which result star reachable sets in which each one

has at most n constraints more than the star input set. Consequently, the number of constraints in a star input set increases linearly over layers, and thus, the worst-case complexity of the number of constraints of a star in the reachable set of an N -neurons FNN is $\mathcal{O}(N)$. \square

Theorem III.2.3 (Verification complexity). *Let F be an N -neuron FNN, Θ be a star set with p linear constraints and m -variables in the predicate, \mathcal{S} be a safety specification with s linear constraints. In the worst case, the safety verification or falsification of the neural network $F(\Theta) \models \mathcal{S}$? is equivalent to solving 2^N feasibility problems in which each has $N + p + s$ linear constraints and m -variables.*

Proof. From Lemma III.2.1, there are at most 2^N stars in the reachable set of the neuron network. Also, from Lemma III.2.2, each star has at most $N + p$ constraints. To verify or falsify the safety of the neural network, we need to check if each star in the reachable set intersects with the safety specification. From Proposition III.1.3, this intersection creates a new star with at most $N + p + s$ constraints. Note that the number of variables m in the predicate of a star does not change over stepReLU operations or in the intersection operation with the half-space. Therefore, the new star has m -variables and at most $N + p + s$ linear constraints, and checking the intersection is equivalent to checking if the new star is an empty set which is a feasibility linear programming problem which can be solved efficiently in polynomial time. \square

Remark III.2.4. *Although in the worst-case, the number of stars in the reachable set of an FNN is 2^N , in practice, the actual number of stars is usually much smaller than the worst-case result which enhances the applicability of the star-based exact reachability analysis for practical DNNs.*

Theorem III.2.5 (Safety and complete counter input set). *Let F be an FNN, $\Theta = \langle c, V, P \rangle$ be a star input set, $F(\Theta) = \cup_{i=1}^k \Theta_i$, $\Theta_i = \langle c_i, V_i, P_i \rangle$ be the reachable set of the neural network, and \mathcal{S} be a safety specification. Denote $\bar{\Theta}_i = \Theta_i \cap \neg \mathcal{S} = \langle c_i, V_i, \bar{P}_i \rangle$, $i = 1, \dots, k$. The neural network is safe if and only if $\bar{P}_i = \emptyset$ for all i . If the neural network violates its safety property, then the complete counter input set containing all possible inputs in the input set that lead the neural network to unsafe states is $\mathcal{C}_\Theta = \cup_{i=1}^k \langle c, V, \bar{P}_i \rangle$, $\bar{P}_i \neq \emptyset$.*

Proof. Safety. The exact reachable set is a union of stars. It is trivial that the neural network is safe if and only if all stars in the reachable set do not intersect with the unsafe region, i.e., $\bar{\Theta}_i$ is an empty set for all i , or equivalently, the predicate \bar{P}_i is empty for all i (Definition 6).

Complete counter input set. Note that all star sets in computation process are defined on the same predicate variable $\alpha = [\alpha_1, \dots, \alpha_m]^T$ which is unchanged in the computation (only the number of constraints on α changes). Therefore, when $\bar{P}_i \neq \emptyset$, it contains values of α that makes the neural network unsafe. It is worth noticing that from the basic predicate P , new constraints are added over stepReLU operations, thus, \bar{P}_i

contains all constraints of the basic predicate P . Consequently, the complete counter input set containing all possible inputs that make the neural network unsafe is defined by $\mathcal{C}_\Theta = \cup_{i=1}^k \langle c, V, \bar{P}_i \rangle$, $\bar{P}_i \neq \emptyset$. \square

III.2.2 Over-approximate analysis

Although the exact and complete analysis can compute the exact reachable sets of a ReLU FNN, the number of stars grows exponentially with the number of layers and leads to an increase in computation cost that limits scalability. In this section, we investigate an over-approximation reachability algorithm for ReLU FNNs in which at each layer, only a single star is constructed by using the following approximation rule.

Lemma III.2.6. *For any input $x \in [l, u]$, the output set $Y = \{y \mid y = \text{ReLU}(x)\}$ satisfies:*

- *If $l \geq 0$, then $y = x$.*
- *If $u \leq 0$, then $y = 0$.*
- *If $l < 0$ and $u > 0$, then $Y \subset \bar{Y} = \{y \mid y \geq 0, y \leq \frac{u(x-l)}{u-l}, y \geq x\}$.*

The over-approximation rules for the ReLU activation function of different approaches are depicted in Figure III.2 which shows that our approximation rule is less conservative than the zonotope's [74] and new abstract domain's rules [75]. The zonotope-based approach [74] over-approximates the ReLU activation function by a minimal parallelogram while the abstract-domain approach [75] over-approximates the ReLU activation function by a triangle. Our star-based approach also over-approximates the ReLU activation function with a triangle as in the abstract-domain approach. However, the new abstract-domain approach only uses lower bound and upper bound constraints for the output $y_i = \text{ReLU}(x_i)$ to avoid the state space explosion [75], for example, in Figure III.2, these constraints are $y_i \geq 0$, $y_i \leq u_i(x_i - l_i)/(u_i - l_i)$. Notably, ***the zonotope and the new abstract domain approaches construct the reachable set based on estimated ranges which is usually very conservative.*** Consequently, these approaches obtain coarse a over-approximation of the actual reachable set which will be shown in the later section. ***To obtain a tighter over-approximation, our star set approach uses three constraints for the output y_i instead. Additionally, it constructs the reachable set using the ranges computed from solving LP problems.***

Similar to the exact approach, the over-approximate reachable set of a Layer with n neurons can be computed by executing a sequence of n *approximate-stepReLU* operations that work as follows. First, we compute the lower bound and upper bound of the input at the i^{th} neuron. If the lower bound is not negative, the approximate-stepReLU operation returns a new intermediate reachable set which is exactly the same as its input set. If the upper bound is not positive, the approximate-stepReLU operation returns a new intermediate reachable set which is the same as its input set except the i^{th} state variable is zero. If the lower bound is negative and the upper bound is positive, the approximate-stepReLU operation introduces a new variable α_{m+1}

to capture the over-approximation of ReLU function at the i^{th} neuron. As a result, the obtained intermediate reachable set has one more variable and three more linear constraints in the predicate in comparison with the corresponding input set. Therefore, in the worst case, the over-approximate reachability algorithm will obtain a reachable set with $N + m_0$ variables and $3N + n_0$ constraints in the predicate, where m_0, n_0 respectively are the number of variables and linear constraints of the predicate of the input set and N is the total number of neurons of the FNN.

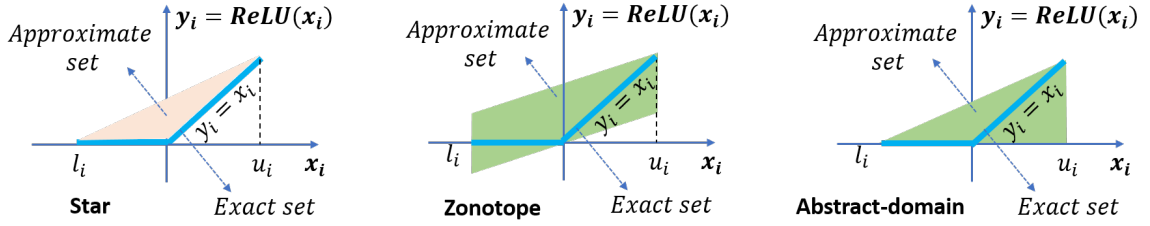


Figure III.2: The star set approach is less conservative than the zonotope [74] and new abstract-domain approaches [75].

III.2.3 Zonotope pre-filter

One can see that, computing the ranges of all states in a star set is an important step in our star set approach that requires solving a set of LP optimization problems. In the exact analysis, the number of LP problems increase exponentially since the number of star sets grows exponentially as proved in Lemma III.2.1. Therefore, to optimize the computation time and enhance the scalability of the star set approach, we need to minimize the number of LP problems solved in the analysis. Fortunately, *for exact analysis, we do not need to know the exact range of a state of the input to a specific neuron to compute the exact reachable set.* The only information we need to know is whether the range contains the zero point. If this is the case, then the star set is split into two new stars which can be constructed efficiently without using the range information. If the range does not contain the zero point, the new star set is constructed even more easily. If the zero point relies on the left hand side of the range, i.e., the input is larger than zero, the output star set at the neuron is equal to the input star set. If the zero point relies on the right hand side of the range, i.e., the input is smaller than zero, the output star set is a projection of the input star set in which the neuron output is projected to zero. *In the over-approximate analysis, the range information is needed to construct an over-approximate reachable set at a specific neuron if and only if it contains the zero point.*

Based on above important observation, *we propose a zonotope pre-filtering step [10] in which a star set is equipped with an outer-zonotope which is an over-approximation of the star set.* This outer-zonotope helps to estimate quickly the range of a state in a star set when doing reachability analysis as a zonotope is efficient for this task. Using this estimated range, we neglect all neurons in the layer that do not affect

the analysis. If the estimated range contains the zero point, we solve two LP problems to get the actual range of this specific input. We reexamine whether the actual range contain the zero point to perform an appropriate operation, i.e., splitting the input set in the exact analysis or constructing an over-approximation of the reachable set using the range information in the over-approximate analysis. We note that, *the new constructed star set inherits the outer-zonotope from its preceptor. Importantly, this outer-zonotope is also updated through the analysis.*

III.2.4 Reachability algorithms (code)

The improved star-based exact reachability algorithm using zonotope pre-filtering given in Algorithm 1 works as follows. The layer takes the star output sets of the preceding layer as input sets $I = [\Theta_1, \dots, \Theta_N]$. The main procedure in the algorithm is *layerReach* which processes the input sets I in parallel. On each input element $\Theta_i = \langle c_i, V_i, P_i, Z_i \rangle$, the main procedure maps the element with the layer weight matrix W and bias vector b which results a new star $I_1 = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$, where Z_i is the outer-zonotope of the star input set. The reachable set of the layer corresponding to the element Θ_i is computed by *reachReLU* sub-procedure which executes a *minimized sequence* of stepReLU/approxStepReLU operations on the new star I_1 , i.e., iteratively calls *stepReLU/approxStepReLU* sub-procedure. Note that that the *stepReLU* sub-procedure is designed to handle multiple star input sets since the number of star sets may increase after each stepReLU operation.

Remark III.2.7. *The star-based reachability analysis algorithm is much faster and more reliable than the polyhedron-based algorithm [83, 28] because the affine mapping step in reachable set computation can be done efficiently by matrix-vector multiplications while in the polyhedron-based approach, this step is very expensive especially for a layer with a large number of neurons since it may need to compute all vertices of the polyhedron input set [48].*

III.3 Dealing with Other Piecewise Activation Functions

The star set method can be extended to FNNs with other classes of piecewise activation functions such as satlin, satlins and leaky ReLU. In this section, we present the extension of the star set method for these type of activation functions. Similar to a ReLU layer, a reachable set of a layer with satlin, satlins, and leaky ReLU can be constructed by performing a sequence of step reachability operations. These step operations can produce an exact or over-approximate reachable set at specific neurons.

Algorithm 1 Improved star-based reachability with zonotope pre-filtering.

Input: $I = [\Theta_1 \cdots \Theta_N]$, W, b \triangleright star input sets, weight matrix, bias vector

Output: R \triangleright exact reachable set

- 1: **procedure** $R = \text{LAYERREACH}(I, W, b, \text{method})$
- 2: $R = \emptyset$
- 3: **parfor** $i = 1 : N$ **do** \triangleright parallel for loop
- 4: $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$
- 5: $R_1 = \text{reachReLU}(I_1, \text{method}), R = R \cup R_1$
- 6: **end parfor**
- 7: **procedure** $R_1 = \text{REACHReLU}(I_1, \text{method})$
- 8: $In = I_1$
- 9: $[lb, ub] = \text{In.Z.getRanges}$ \triangleright estimate ranges of all input variables
- 10: $\text{map} = \text{find}(ub < 0)$ \triangleright list of neglected neurons
- 11: $\text{In.c}(\text{map}, 1) = 0, \text{In.V}(\text{map}, :) = 0$ \triangleright update the input star set
- 12: $\text{In.Z.l}(\text{map}, 1) = 0, \text{In.Z.G}(\text{map}, :) = 0$ \triangleright update the outer-zonotope
- 13: $\text{map} = \text{find}(lb < 0 \ \& \ ub > 0)$ \triangleright construct computation map
- 14: $m = \text{length}(\text{map})$ \triangleright minimized number of step operations
- 15: **for** $i = 1 : m$ **do**
- 16: **if** $\text{method} = \text{exact}$ **then**
- 17: $In = \text{stepReLU}(In, \text{map}(i))$ \triangleright stepReLU operation
- 18: **else if** $\text{method} = \text{approx}$ **then**
- 19: $In = \text{approxStepReLU}(In, \text{map}(i))$ \triangleright approxStepReLU operation
- 20: $R_1 = In$
- 21: **procedure** $\tilde{R} = \text{STEPReLU}(\tilde{I}, i)$
- 22: $\tilde{R} = \emptyset, \tilde{I} = [\tilde{\Theta}_1 \cdots \tilde{\Theta}_k]$ \triangleright intermediate star input and output sets
- 23: **for** $j = 1 : k$ **do**
- 24: $[lb_j, ub_j] = \tilde{\Theta}_j.\text{getRange}(i)$ \triangleright get exact range of the j^{th} input
- 25: $\mathcal{R}_1 = \emptyset, M = [e_1 \ e_2 \ \cdots \ e_{i-1} \ 0 \ e_{i+1} \ \cdots \ e_n]$
- 26: **if** $lb_j \geq 0$ **then** $\mathcal{R}_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{Z}_j \rangle$
- 27: **if** $ub_j \leq 0$ **then** $R_1 = M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j, M\tilde{Z}_j \rangle$
- 28: **if** $lb_j < 0 \ \& \ ub_j > 0$ **then**
- 29: $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle,$
- 30: $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{Z}_j \rangle$
- 31: $\mathcal{R}_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
- 32: $\tilde{R} = \tilde{R} \cup R_1$
- 33: **procedure** $\tilde{R} = \text{APPROXSTEPReLU}(\tilde{I}, i)$
- 34: $\tilde{I} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$
- 35: $[l, u] = \tilde{\Theta}.\text{getRange}(i)$ \triangleright get actual range of the i^{th} input
- 36: $M = [e_1 \ e_2 \ \cdots \ e_{i-1} \ 0 \ e_{i+1} \ \cdots \ e_n]$
- 37: **if** $l \geq 0$ **then** $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$
- 38: **if** $u \leq 0$ **then** $\tilde{R} = M * \tilde{\Theta} = \langle M\tilde{c}, M\tilde{V}, \tilde{P}, M\tilde{Z} \rangle$
- 39: **if** $l < 0 \ \& \ u > 0$ **then**
- 40: $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$ \triangleright input set's predicate
- 41: $\alpha' = [\alpha_1, \dots, \alpha_m, \alpha_{m+1}]^T$ \triangleright new variable α_{m+1}
- 42: $C_1 = [0 \ 0 \ \cdots \ 0 \ -1], d_1 = 0$ $\triangleright \alpha_{m+1} \geq 0 \Leftrightarrow C_1 \alpha' \leq d_1$
- 43: $C_2 = [V(\tilde{i}, :) \ -1], d_2 = -\tilde{c}[i]$ $\triangleright \alpha_{m+1} \geq x[i] \Leftrightarrow C_2 \alpha' \leq d_2$
- 44: $C_3 = [\frac{-u}{u-1} \times V(\tilde{i}, :) \ 1], d_3 = \frac{ul}{u-1} \times (1 - \tilde{c}[i])$ $\triangleright \alpha_{m+1} \leq \frac{u(x[i]-l)}{u-1} \Leftrightarrow C_3 \alpha' \leq d_3$
- 45: $C_0 = [\tilde{C} \ 0_{m \times 1}], d_0 = \tilde{d}$
- 46: $C' = [C_0; C_1; C_2; C_3], d' = [d_0; d_1; d_2; d_3]$
- 47: $P'(\alpha') \triangleq C' \alpha' \leq d'$ \triangleright output set's predicate
- 48: $c' = M\tilde{c}, V' = M\tilde{V}, V' = [V' \ e_i]$ $\triangleright y[i] = \text{ReLU}(x[i]) = \alpha_{m+1}$
- 49: $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$

III.3.1 Reachability of a satlin layer

The satlin activation function is defined by:

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

The star-based reachability algorithm with zonotope pre-filtering for a satlin layer is given in Algorithm 2. Similar to a ReLU layer, we use a zonotope pre-filter to determine all neurons where splitting cannot happen. We update quickly the reachable set at these neurons, i.e., project the output to zero or one. Then, we consider the neurons where the splitting may occur. For those neurons, we solve LP optimization to determine their actual ranges to split the input set (in the exact analysis) or to construct an over-approximate reachable set (in the over approximate analysis).

III.3.2 Reachability of a satlins layer

The satlins activation function is defined by:

$$f(x) = \begin{cases} -1 & \text{if } x \leq -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

The reachability algorithms for a satlins layer is given in Algorithm 3.

III.3.3 Reachability of a leaky ReLU layer

The leaky ReLU activation function is defined by:

$$f(\gamma, x) = \begin{cases} \gamma x & \text{if } x \leq 0 \\ x & \text{if } -1 \leq x \leq 0 \end{cases}$$

The reachability algorithms for a leaky ReLU layer is given in Algorithm 4. These algorithms are similar to the ones for a ReLU layer except for the case that when the input to a specific neuron is smaller than zero, the output is proportional to the input with a coefficient γ instead of being set by zero.

Algorithm 2 Improved star-based reachability for a satlin layer.

Input: $I = [\Theta_1 \dots \Theta_N]$, W , b \triangleright star input sets, weight matrix, bias vector
Output: R \triangleright exact reachable set

- 1: **procedure** $R = \text{LAYERREACH}(I, W, b, \text{method})$
- 2: $R = \emptyset$
- 3: **parfor** $i = 1 : N$ **do** \triangleright parallel for loop
- 4: $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$
- 5: $R_1 = \text{reachSatlin}(I_1, \text{method})$, $R = R \cup R_1$
- 6: **end parfor**
- 7: **procedure** $R_1 = \text{REACHSATLIN}(I_1, \text{method})$
- 8: $In = I_1$
- 9: $[lb, ub] = In.Z.getRanges$ \triangleright estimate ranges of all input variables
- 10: $map = \text{find}(ub \leq 0)$ \triangleright list of neglected neurons
- 11: $In.c(map, 1) = 0$, $In.V(map, :) = 0$ \triangleright update the input star set
- 12: $In.Z.l(map, 1) = 0$, $In.Z.G(map, :) = 0$ \triangleright update the outer-zonotope
- 13: $map = \text{find}(lb \geq 1)$ \triangleright list of neglected neurons
- 14: $In.c(map, 1) = 1$, $In.V(map, :) = 0$ \triangleright update the input star set
- 15: $In.Z.l(map, 1) = 1$, $In.Z.G(map, :) = 0$ \triangleright update the outer-zonotope
- 16: $map = \text{find}(lb < 1 \ || \ ub > 0)$ \triangleright construct computation map
- 17: $m = \text{length}(map)$ \triangleright minimized number of step operations
- 18: **for** $i = 1 : m$ **do**
- 19: **if** $\text{method} = \text{exact}$ **then**
- 20: $In = \text{stepSatlin}(In, map(i))$ \triangleright stepSatlin operation
- 21: **else if** $\text{method} = \text{approx}$ **then**
- 22: $In = \text{approxStepSatlin}(In, map(i))$ \triangleright approxStepSatlin operation
- 23: $R_1 = In$
- 24: **procedure** $\tilde{R} = \text{STEPSATLIN}(\tilde{I}, i)$
- 25: $\tilde{R} = \emptyset$, $\tilde{I} = [\tilde{\Theta}_1 \dots \tilde{\Theta}_k]$ \triangleright intermediate star input and output sets
- 26: **for** $j = 1 : k$ **do**
- 27: $[lb_j, ub_j] = \tilde{\Theta}_j.getRange(i)$ \triangleright get exact range of the j^{th} input
- 28: $\mathcal{R}_1 = \emptyset$, $M = [e_1 \ e_2 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$
- 29: **if** $lb_j \geq 0$ & $ub_j \leq 1$ **then** $\mathcal{R}_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{Z}_j \rangle$
- 30: **if** $ub_j \leq 0$ **then** $R_1 = M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j, M\tilde{Z}_j \rangle$
- 31: **if** $lb_j < 0$ & $ub_j \leq 1$ **then**
- 32: $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$,
- 33: $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{Z}_j \rangle$
- 34: $\mathcal{R}_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
- 35: **if** $0 \leq lb_j \leq 1$ & $ub_j \geq 1$ **then**
- 36: $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \leq 1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$,
- 37: $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] \geq 1 = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j, \tilde{Z}''_j \rangle$
- 38: $\tilde{c}''_j = \tilde{c}_j$, $\tilde{c}''_j(i) = 1$, $\tilde{V}''_j = \tilde{V}_j$, $\tilde{V}''_j(i, :) = 0$,
- 39: $\tilde{Z}''_j = \tilde{Z}_j$, $\tilde{Z}''_j.l(i) = 1$, $\tilde{Z}''_j.G(i, :) = 0$
- 40: $\mathcal{R}_1 = \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$
- 41: **if** $lb_j \leq 0$ & $ub_j \geq 1$ **then**
- 42: $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge 0 \leq x[i] \leq 1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$
- 43: $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{Z}_j \rangle$
- 44: $\tilde{\Theta}'''_j = \tilde{\Theta}_j \wedge x[i] \geq 1 = \langle \tilde{c}'''_j, \tilde{V}'''_j, \tilde{P}'''_j, \tilde{Z}'''_j \rangle$
- 45: $\tilde{c}'''_j = \tilde{c}_j$, $\tilde{c}'''_j(i) = 1$, $\tilde{V}'''_j = \tilde{V}_j$, $\tilde{V}'''_j(i, :) = 0$,
- 46: $\tilde{Z}'''_j = \tilde{Z}_j$, $\tilde{Z}'''_j.l(i) = 1$, $\tilde{Z}'''_j.G(i, :) = 0$
- 47: $\mathcal{R}_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j \cup \tilde{\Theta}'''_j$
- 48: $\tilde{R} = \tilde{R} \cup R_1$

```

49: procedure  $\tilde{R} = \text{APPROXSTEP SATLIN}(\tilde{I}, i)$ 
50:    $\tilde{I} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$ 
51:    $[l, u] = \tilde{\Theta}.\text{getRange}(i)$   $\triangleright$  get actual range of the  $i^{\text{th}}$  input
52:    $M = [e_1 \ e_2 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$ 
53:   if  $l \geq 0$  &  $u \leq 1$  then  $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$ 
54:   if  $l \geq 1$  then
55:      $\tilde{c}' = \tilde{c}, \tilde{c}(i) = 1, \tilde{V}' = \tilde{V}, \tilde{V}(i, :) = 0, \tilde{Z}' = \tilde{Z}, \tilde{Z}.l(i) = 1, \tilde{Z}.G(i, :) = 0$ 
56:      $\tilde{R} = \tilde{\Theta}' = \langle \tilde{c}', \tilde{V}', \tilde{P}, \tilde{Z}' \rangle$ 
57:   if  $u \leq 0$  then  $\tilde{R} = M * \tilde{\Theta} = \langle M\tilde{c}, M\tilde{V}, \tilde{P}, M\tilde{Z} \rangle$ 
58:   if  $l < 0$  &  $0 < u \leq 1$  then
59:      $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$   $\triangleright$  input set's predicate
60:      $\alpha' = [\alpha_1, \dots, \alpha_m, \alpha_{m+1}]^T$   $\triangleright$  new variable  $\alpha_{m+1}$ 
61:      $C_1 = [0 \ 0 \ \dots \ 0 \ -1], d_1 = 0$   $\triangleright \alpha_{m+1} \geq 0 \Leftrightarrow C_1 \alpha' \leq d_1$ 
62:      $C_2 = [V(\tilde{i}, :) - 1], d_2 = -\tilde{c}[\tilde{i}]$   $\triangleright \alpha_{m+1} \geq x[\tilde{i}] \Leftrightarrow C_2 \alpha' \leq d_2$ 
63:      $C_3 = [\frac{-u}{u-1} \times V(\tilde{i}, :) \ 1], d_3 = \frac{ul}{u-1} \times (1 - \tilde{c}[\tilde{i}])$   $\triangleright \alpha_{m+1} \leq \frac{u(x[\tilde{i}] - l)}{u-1} \Leftrightarrow C_3 \alpha' \leq d_3$ 
64:      $C_0 = [\tilde{C} \ 0_{m \times 1}], d_0 = \tilde{d}$ 
65:      $C' = [C_0; C_1; C_2; C_3], d' = [d_0; d_1; d_2; d_3]$ 
66:      $P'(\alpha') \triangleq C' \alpha' \leq d'$   $\triangleright$  output set's predicate
67:      $c' = M\tilde{c}, V' = M\tilde{V}, V' = [V' \ e_i]$   $\triangleright y[\tilde{i}] = \text{satlin}(x[\tilde{i}]) = \alpha_{m+1}$ 
68:      $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$ 
69:   if  $0 \leq l < 1$  &  $u > 1$  then
70:      $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$   $\triangleright$  input set's predicate
71:      $\alpha' = [\alpha_1, \dots, \alpha_m, \alpha_{m+1}]^T$   $\triangleright$  new variable  $\alpha_{m+1}$ 
72:      $C_1 = [0 \ 0 \ \dots \ 0 \ 1], d_1 = 1$   $\triangleright \alpha_{m+1} \leq 1 \Leftrightarrow C_1 \alpha' \leq d_1$ 
73:      $C_2 = [-V(\tilde{i}, :) \ 1], d_2 = \tilde{c}[\tilde{i}]$   $\triangleright \alpha_{m+1} \leq x[\tilde{i}] \Leftrightarrow C_2 \alpha' \leq d_2$ 
74:      $C_3 = [\frac{1-l}{u-1} \times V(\tilde{i}, :) - 1], d_3 = \frac{l(1-l)}{u-1} \tilde{c}(\tilde{i}) - l$   $\triangleright \alpha_{m+1} \geq \frac{(1-l)(x[\tilde{i}] - l)}{u-1} + l \Leftrightarrow C_3 \alpha' \leq d_3$ 
75:      $C_0 = [\tilde{C} \ 0_{m \times 1}], d_0 = \tilde{d}$ 
76:      $C' = [C_0; C_1; C_2; C_3], d' = [d_0; d_1; d_2; d_3]$ 
77:      $P'(\alpha') \triangleq C' \alpha' \leq d'$   $\triangleright$  output set's predicate
78:      $c' = M\tilde{c}, V' = M\tilde{V}, V' = [V' \ e_i]$   $\triangleright y[\tilde{i}] = \text{satlin}(x[\tilde{i}]) = \alpha_{m+1}$ 
79:      $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$ 
80:   if  $l < 0$  &  $u > 1$  then
81:      $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$   $\triangleright$  input set's predicate
82:      $\alpha' = [\alpha_1, \dots, \alpha_m, \alpha_{m+1}]^T$   $\triangleright$  new variable  $\alpha_{m+1}$ 
83:      $C_1 = [0 \ 0 \ \dots \ 0 \ -1], d_1 = 0$   $\triangleright \alpha_{m+1} \geq 0 \Leftrightarrow C_1 \alpha' \leq d_1$ 
84:      $C_2 = [0 \ 0 \ \dots \ 0 \ 1], d_2 = 1$   $\triangleright \alpha_{m+1} \leq 1 \Leftrightarrow C_2 \alpha' \leq d_2$ 
85:      $C_3 = [-V(\tilde{i}, :) \ 1], d_3 = \frac{\tilde{c}[\tilde{i}]}{1-l} - \frac{l}{1-l}$   $\triangleright \alpha_{m+1} \leq \frac{x[\tilde{i}]}{1-l} - \frac{l}{1-l} \Leftrightarrow C_3 \alpha' \leq d_3$ 
86:      $C_4 = [\frac{1}{u} \times V(\tilde{i}, :) - 1], d_4 = -\frac{1}{u} \tilde{c}(\tilde{i})$   $\triangleright \alpha_{m+1} \geq \frac{x[\tilde{i}]}{u} \Leftrightarrow C_4 \alpha' \leq d_4$ 
87:      $C_0 = [\tilde{C} \ 0_{m \times 1}], d_0 = \tilde{d}$ 
88:      $C' = [C_0; C_1; C_2; C_3; C_4], d' = [d_0; d_1; d_2; d_3; d_4]$ 
89:      $P'(\alpha') \triangleq C' \alpha' \leq d'$   $\triangleright$  output set's predicate
90:      $c' = M\tilde{c}, V' = M\tilde{V}, V' = [V' \ e_i]$   $\triangleright y[\tilde{i}] = \text{satlin}(x[\tilde{i}]) = \alpha_{m+1}$ 
91:      $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$ 

```

Algorithm 3 Improved star-based reachability for a satlins layer.

Input: $I = [\Theta_1 \cdots \Theta_N]$, W , b \triangleright star input sets, weight matrix, bias vector

Output: R \triangleright exact reachable set

```

1: procedure  $R = \text{LAYERREACH}(I, W, b, \text{method})$ 
2:    $R = \emptyset$ 
3:   parfor  $i = 1 : N$  do  $\triangleright$  parallel for loop
4:      $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$ 
5:      $R_1 = \text{reachSatlin}(I_1, \text{method}), R = R \cup R_1$ 
6:   end parfor
7:   procedure  $R_1 = \text{REACHSATLINS}(I_1, \text{method})$ 
8:      $In = I_1$ 
9:      $[lb, ub] = In.Z.\text{getRanges}$   $\triangleright$  estimate ranges of all input variables
10:     $\text{map} = \text{find}(ub \leq -1)$   $\triangleright$  list of neglected neurons
11:     $In.c(\text{map}, 1) = -1, In.V(\text{map}, :) = 0$   $\triangleright$  update the input star set
12:     $In.Z.l(\text{map}, 1) = -1, In.Z.G(\text{map}, :) = 0$   $\triangleright$  update the outer-zonotope
13:     $\text{map} = \text{find}(lb \geq 1)$   $\triangleright$  list of neglected neurons
14:     $In.c(\text{map}, 1) = 1, In.V(\text{map}, :) = 0$   $\triangleright$  update the input star set
15:     $In.Z.l(\text{map}, 1) = 1, In.Z.G(\text{map}, :) = 0$   $\triangleright$  update the outer-zonotope
16:     $\text{map} = \text{find}(lb < 1 \parallel ub > -1)$   $\triangleright$  construct computation map
17:     $m = \text{length}(\text{map})$   $\triangleright$  minimized number of step operations
18:    for  $i = 1 : m$  do
19:      if  $\text{method} = \text{exact}$  then
20:         $In = \text{stepSatlins}(In, \text{map}(i))$   $\triangleright$  stepSatlins operation
21:      else if  $\text{method} = \text{approx}$  then
22:         $In = \text{approxStepSatlins}(In, \text{map}(i))$   $\triangleright$  approxStepSatlins operation
23:       $R_1 = In$ 
24:    procedure  $\tilde{R} = \text{STEPSATLINS}(\tilde{I}, i)$ 
25:     $\tilde{R} = \emptyset, \tilde{I} = [\tilde{\Theta}_1 \cdots \tilde{\Theta}_k]$   $\triangleright$  intermediate star input and output sets
26:    for  $j = 1 : k$  do
27:       $[lb_j, ub_j] = \tilde{\Theta}_j.\text{getRange}(i)$   $\triangleright$  get exact range of the  $j^{\text{th}}$  input
28:       $\mathcal{R}_1 = \emptyset, M = [e_1 \ e_2 \ \cdots \ e_{i-1} \ 0 \ e_{i+1} \ \cdots \ e_n]$ 
29:      if  $lb_j \geq -1 \ \& \ ub_j \leq 1$  then  $\tilde{\mathcal{R}}_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{Z}_j \rangle$ 
30:      if  $ub_j \leq -1$  then
31:         $\tilde{c}'_j = \tilde{c}_j, \tilde{c}'_j(i) = -1, \tilde{V}'_j = \tilde{V}_j, \tilde{V}'_j(i, :) = 0$ 
32:         $\tilde{Z}'_j = \tilde{Z}_j, \tilde{Z}'_j.l(i) = -1, \tilde{Z}'_j.G(i, :) = 0$ 
33:         $R_1 = \tilde{\Theta}'_j = \langle \tilde{c}'_j, \tilde{V}'_j, \tilde{P}_j, \tilde{Z}'_j \rangle$ 
34:      if  $lb_j < -1 \ \& \ ub_j \leq 1$  then
35:         $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq -1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle,$ 
36:         $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < -1 = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j, \tilde{Z}''_j \rangle$ 
37:         $\tilde{c}''_j = \tilde{c}_j, \tilde{c}''_j(i) = -1, \tilde{V}''_j = \tilde{V}_j, \tilde{V}''_j(i, :) = 0$ 
38:         $\tilde{Z}''_j = \tilde{Z}_j, \tilde{Z}''_j.l(i) = -1, \tilde{Z}''_j.G(i, :) = 0$ 
39:         $\mathcal{R}_1 = \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$ 
40:      if  $-1 \leq lb_j \leq 1 \ \& \ ub_j \geq 1$  then
41:         $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \leq 1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle,$ 
42:         $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] \geq 1 = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j, \tilde{Z}''_j \rangle$ 
43:         $\tilde{c}''_j = \tilde{c}_j, \tilde{c}''_j(i) = 1, \tilde{V}''_j = \tilde{V}_j, \tilde{V}''_j(i, :) = 0,$ 
44:         $\tilde{Z}''_j = \tilde{Z}_j, \tilde{Z}''_j.l(i) = 1, \tilde{Z}''_j.G(i, :) = 0$ 
45:         $\mathcal{R}_1 = \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$ 
46:      if  $lb_j < -1 \ \& \ ub_j > 1$  then
47:         $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge -1 \leq x[i] \leq 1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$ 
48:         $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < -1 = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j, \tilde{Z}''_j \rangle$ 
49:         $\tilde{c}''_j = \tilde{c}_j, \tilde{c}''_j(i) = -1, \tilde{V}''_j = \tilde{V}_j, \tilde{V}''_j(i, :) = 0,$ 
50:         $\tilde{Z}''_j = \tilde{Z}_j, \tilde{Z}''_j.l(i) = -1, \tilde{Z}''_j.G(i, :) = 0$ 
51:         $\tilde{\Theta}'''_j = \tilde{\Theta}_j \wedge x[i] \geq 1 = \langle \tilde{c}'''_j, \tilde{V}'''_j, \tilde{P}'''_j, \tilde{Z}'''_j \rangle$ 
52:         $\tilde{c}'''_j = \tilde{c}_j, \tilde{c}'''_j(i) = 1, \tilde{V}'''_j = \tilde{V}_j, \tilde{V}'''_j(i, :) = 0,$ 
53:         $\tilde{Z}'''_j = \tilde{Z}_j, \tilde{Z}'''_j.l(i) = 1, \tilde{Z}'''_j.G(i, :) = 0$ 
54:         $\mathcal{R}_1 = \tilde{\Theta}'_j \cup \tilde{\Theta}''_j \cup \tilde{\Theta}'''_j$ 
55:     $\tilde{R} = \tilde{R} \cup R_1$ 

```

```

56: procedure  $\tilde{R} = \text{APPROXSTEP SATLINS}(\tilde{I}, i)$ 
57:    $\tilde{I} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$ 
58:    $[l, u] = \tilde{\Theta}.\text{getRange}(i)$   $\triangleright$  get actual range of the  $i^{\text{th}}$  input
59:    $M = [e_1 \ e_2 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$ 
60:   if  $l \geq -1$  &  $u \leq 1$  then  $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$ 
61:   if  $l \geq 1$  then
62:      $\tilde{c}' = \tilde{c}, \tilde{c}(i) = 1, \tilde{V}' = \tilde{V}, \tilde{V}(i, :) = 0, \tilde{Z}' = \tilde{Z}, \tilde{Z}.l(i) = 1, \tilde{Z}.G(i, :) = 0$ 
63:      $\tilde{R} = \tilde{\Theta}' = \langle \tilde{c}', \tilde{V}', \tilde{P}, \tilde{Z}' \rangle$ 
64:   if  $u \leq -1$  then
65:      $\tilde{c}' = \tilde{c}, \tilde{c}(i) = -1, \tilde{V}' = \tilde{V}, \tilde{V}(i, :) = 0, \tilde{Z}' = \tilde{Z}, \tilde{Z}.l(i) = -1, \tilde{Z}.G(i, :) = 0$ 
66:      $\tilde{R} = \tilde{\Theta}' = \langle \tilde{c}', \tilde{V}', \tilde{P}, \tilde{Z}' \rangle$ 
67:   if  $l < -1$  &  $0 < u \leq 1$  then
68:      $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$   $\triangleright$  input set's predicate
69:      $\alpha' = [\alpha_1, \dots, \alpha_m, \alpha_{m+1}]^T$   $\triangleright$  new variable  $\alpha_{m+1}$ 
70:      $C_1 = [0 \ 0 \ \dots \ 0 \ -1], d_1 = 1$   $\triangleright \alpha_{m+1} \geq -1 \Leftrightarrow C_1 \alpha' \leq d_1$ 
71:      $C_2 = [\tilde{V}(i, :) \ -1], d_2 = -\tilde{c}(i)$   $\triangleright \alpha_{m+1} \geq x[i] \Leftrightarrow C_2 \alpha' \leq d_2$ 
72:      $\triangleright \alpha_{m+1} \leq \frac{(u+1)(x[i]-u)}{u-1} + u \Leftrightarrow C_3 \alpha' \leq d_3$ 
73:      $C_3 = [\frac{-u-1}{u-1} \times \tilde{V}(i, :) \ 1], d_3 = \frac{-u(u+1)}{u-1} \times \tilde{c}[i] + u$ 
74:      $C_0 = [\tilde{C} \ 0_{m \times 1}], d_0 = \tilde{d}$ 
75:      $C' = [C_0; C_1; C_2; C_3], d' = [d_0; d_1; d_2; d_3]$ 
76:      $P'(\alpha') \triangleq C' \alpha' \leq d'$   $\triangleright$  output set's predicate
77:      $c' = M\tilde{c}, V' = M\tilde{V}, V' = [V' \ e_i]$   $\triangleright y[i] = \text{satlins}(x[i]) = \alpha_{m+1}$ 
78:      $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$ 
79:   if  $-1 \leq l < 1$  &  $u > 1$  then
80:      $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$   $\triangleright$  input set's predicate
81:      $\alpha' = [\alpha_1, \dots, \alpha_m, \alpha_{m+1}]^T$   $\triangleright$  new variable  $\alpha_{m+1}$ 
82:      $C_1 = [0 \ 0 \ \dots \ 0 \ 1], d_1 = 1$   $\triangleright \alpha_{m+1} \leq 1 \Leftrightarrow C_1 \alpha' \leq d_1$ 
83:      $C_2 = [-\tilde{V}(i, :) \ 1], d_2 = \tilde{c}(i)$   $\triangleright \alpha_{m+1} \leq x[i] \Leftrightarrow C_2 \alpha' \leq d_2$ 
84:      $C_3 = [\frac{1-l}{u-l} \times \tilde{V}(i, :) \ -1], d_3 = \frac{l(1-l)}{u-l} \tilde{c}(i) - l$   $\triangleright \alpha_{m+1} \geq \frac{(1-l)(x[i]-l)}{u-l} + l \Leftrightarrow C_3 \alpha' \leq d_3$ 
85:      $C_0 = [\tilde{C} \ 0_{m \times 1}], d_0 = \tilde{d}$ 
86:      $C' = [C_0; C_1; C_2; C_3], d' = [d_0; d_1; d_2; d_3]$ 
87:      $P'(\alpha') \triangleq C' \alpha' \leq d'$   $\triangleright$  output set's predicate
88:      $c' = M\tilde{c}, V' = M\tilde{V}, V' = [V' \ e_i]$   $\triangleright y[i] = \text{satlins}(x[i]) = \alpha_{m+1}$ 
89:      $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$ 
90:   if  $l < -1$  &  $u > 1$  then
91:      $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$   $\triangleright$  input set's predicate
92:      $\alpha' = [\alpha_1, \dots, \alpha_m, \alpha_{m+1}]^T$   $\triangleright$  new variable  $\alpha_{m+1}$ 
93:      $C_1 = [0 \ 0 \ \dots \ 0 \ -1], d_1 = 1$   $\triangleright \alpha_{m+1} \geq -1 \Leftrightarrow C_1 \alpha' \leq d_1$ 
94:      $C_2 = [0 \ 0 \ \dots \ 0 \ 1], d_2 = 1$   $\triangleright \alpha_{m+1} \leq 1 \Leftrightarrow C_2 \alpha' \leq d_2$ 
95:      $C_3 = [-\frac{2}{1-l} \tilde{V}(i, :) \ 1], d_3 = \frac{2\tilde{c}[i]}{1-l} - 1$   $\triangleright \alpha_{m+1} \leq \frac{2x[i]}{1-l} - 1 \Leftrightarrow C_3 \alpha' \leq d_3$ 
96:      $C_4 = [\frac{2}{u+1} \times \tilde{V}(i, :) \ -1], d_4 = -\frac{2}{u+1} \tilde{c}(i) + 1$   $\triangleright \alpha_{m+1} \geq \frac{2(x[i]+1)}{u+1} - 1 \Leftrightarrow C_4 \alpha' \leq d_4$ 
97:      $C_0 = [\tilde{C} \ 0_{m \times 1}], d_0 = \tilde{d}$ 
98:      $C' = [C_0; C_1; C_2; C_3; C_4], d' = [d_0; d_1; d_2; d_3; d_4]$ 
99:      $P'(\alpha') \triangleq C' \alpha' \leq d'$   $\triangleright$  output set's predicate
100:     $c' = M\tilde{c}, V' = M\tilde{V}, V' = [V' \ e_i]$   $\triangleright y[i] = \text{satlins}(x[i]) = \alpha_{m+1}$ 
101:     $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$ 

```

Algorithm 4 Improved star-based reachability for a leaky ReLU layer.

Input: $I = [\Theta_1 \cdots \Theta_N], W, b$ \triangleright star input sets, weight matrix, bias vector

Output: R \triangleright exact reachable set

```

1: procedure  $R = \text{LAYERREACH}(I, W, b, \gamma, \text{method})$ 
2:    $R = \emptyset$ 
3:   parfor  $i = 1 : N$  do  $\triangleright$  parallel for loop
4:      $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$ 
5:      $R_1 = \text{reachLeakyReLU}(I_1, \gamma, \text{method}), R = R \cup R_1$ 
6:   end parfor
7:   procedure  $R_1 = \text{REACHLEAKYReLU}(I_1, \gamma, \text{method})$ 
8:      $In = I_1, M_i = [e_1 \ e_2 \ \cdots \ e_{i-1} \ \gamma \times e_i \ e_{i+1} \ \cdots \ e_n]$ 
9:      $[lb, ub] = In.Z.\text{getRanges}$   $\triangleright$  estimate ranges of all input variables
10:     $map = \text{find}(ub < 0)$   $\triangleright$  list of neglected neurons
11:     $In.c = M_{map}In.c, In.V = M_{map}In.V$   $\triangleright$  update the input star set
12:     $In.Z.l = M_{map}In.Z.l, In.Z.G = M_{map}In.Z.G$   $\triangleright$  update the outer-zonotope
13:     $map = \text{find}(lb < 0 \ \& \ ub > 0)$   $\triangleright$  construct computation map
14:     $m = \text{length}(map)$   $\triangleright$  minimized number of step operations
15:    for  $i = 1 : m$  do
16:      if  $\text{method} = \text{exact}$  then
17:         $In = \text{stepLeakyReLU}(In, map(i))$   $\triangleright$  stepLeakyReLU operation
18:      else if  $\text{method} = \text{approx}$  then
19:         $In = \text{approxStepLeakyReLU}(In, map(i))$   $\triangleright$  approxStepLeakyReLU operation
20:       $R_1 = In$ 
21:    procedure  $\tilde{R} = \text{STEPLEAKYReLU}(\tilde{I}, i, \gamma)$ 
22:       $\tilde{R} = \emptyset, \tilde{I} = [\tilde{\Theta}_1 \cdots \tilde{\Theta}_k]$   $\triangleright$  intermediate star input and output sets
23:      for  $j = 1 : k$  do
24:         $[lb_j, ub_j] = \tilde{\Theta}_j.\text{getRange}(i)$   $\triangleright$  get exact range of the  $j^{\text{th}}$  input
25:         $\mathcal{R}_1 = \emptyset, M = [e_1 \ e_2 \ \cdots \ e_{i-1} \ \gamma \times e_i \ e_{i+1} \ \cdots \ e_n]$ 
26:        if  $lb_j \geq 0$  then  $\mathcal{R}_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{Z}_j \rangle$ 
27:        if  $ub_j \leq 0$  then  $R_1 = M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j, M\tilde{Z}_j \rangle$ 
28:        if  $lb_j < 0 \ \& \ ub_j > 0$  then
29:           $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle,$ 
30:           $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{Z}_j \rangle$ 
31:           $\mathcal{R}_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$ 
32:         $\tilde{R} = \tilde{R} \cup R_1$ 
33:    procedure  $\tilde{R} = \text{APPROXSTEPLEAKYReLU}(\tilde{I}, i, \gamma)$ 
34:       $\tilde{I} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$ 
35:       $[l, u] = \tilde{\Theta}.\text{getRange}(i)$   $\triangleright$  get actual range of the  $i^{\text{th}}$  input
36:       $M = [e_1 \ e_2 \ \cdots \ e_{i-1} \ \gamma \times e_i \ e_{i+1} \ \cdots \ e_n]$ 
37:      if  $l \geq 0$  then  $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$ 
38:      if  $u \leq 0$  then  $\tilde{R} = M * \tilde{\Theta} = \langle M\tilde{c}, M\tilde{V}, \tilde{P}, M\tilde{Z} \rangle$ 
39:      if  $l < 0 \ \& \ u > 0$  then
40:         $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$   $\triangleright$  input set's predicate
41:         $\alpha' = [\alpha_1, \dots, \alpha_m, \alpha_{m+1}]^T$   $\triangleright$  new variable  $\alpha_{m+1}$ 
42:         $C_1 = [\tilde{V}(i, :) \ -1], d_1 = -\gamma\tilde{c}(i)$   $\triangleright \alpha_{m+1} \geq \gamma x[i] \Leftrightarrow C_1 \alpha' \leq d_1$ 
43:         $C_2 = [\tilde{V}(i, :) \ -1], d_2 = -\tilde{c}[i]$   $\triangleright \alpha_{m+1} \geq x[i] \Leftrightarrow C_2 \alpha' \leq d_2$ 
44:         $C_3 = [\frac{-u}{u-1} \times \tilde{V}(i, :) \ 1], d_3 = \frac{ul}{u-1} \times (1 - \tilde{c}[i])$   $\triangleright \alpha_{m+1} \leq \frac{u(x[i]-l)}{u-1} \Leftrightarrow C_3 \alpha' \leq d_3$ 
45:         $C_0 = [\tilde{C} \ 0_{m \times 1}], d_0 = \tilde{d}$ 
46:         $C' = [C_0; C_1; C_2; C_3], d' = [d_0; d_1; d_2; d_3]$ 
47:         $P'(\alpha') \triangleq C' \alpha' \leq d'$   $\triangleright$  output set's predicate
48:         $M = [e_1 \ e_2 \ \cdots \ e_{i-1} \ 0 \ e_{i+1} \ \cdots \ e_n]$ 
49:         $c' = M\tilde{c}, V' = M\tilde{V}, V' = [V' \ e_i]$   $\triangleright y[i] = \text{leakyReLU}(x[i]) = \alpha_{m+1}$ 
50:         $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$ 

```

III.4 Evaluation

In this section, we re-evaluate the improved star-based reachability algorithms in comparison to the newest version of Reluplex [41], the zonotopes [74], and the abstract domain [75] methods implemented in our NNV tool [86]. The implementation of the zonotope and new abstract domain methods allows us to visualize the over-approximate reachable set of these approaches to intuitively evaluate their conservativeness. All results presented in this section and their corresponding scripts are available online¹.

III.4.1 Safety Verification for ACAS Xu DNNs

The ACAS Xu networks are DNN-based advisory controllers that map the sensor measurements to advisories in the Airborne Collision Avoidance System X [39]. It is a series of 45 feedforward neural networks which map input variables to actions for horizontal maneuvers. The output means the order to the UAV to follow, and this can be: clear of conflict (COC), weak left, weak right, strong left or strong right. All the networks have 6 fully connected layers with a total of 300 neurons, 5 inputs and 5 outputs, with all ReLU activation functions. The inputs are:

- ρ : distance from ownship to intruder (feet)
- θ : angle to intruder relative to ownship heading direction (radians)
- ψ : heading angle of intruder relative to ownship heading direction (radians)
- v_{own} : speed of ownship (feet per second)
- v_{int} : speed of intruder (feet per second)

Two other variables, τ , time until loss of vertical separation (seconds), and a_{prev} , previous advisory, are discretized and used to generate the 45 neural networks mentioned.

The following safety properties are used to re-evaluate the performance of different methods.

- **Property ϕ_3 .**
 - If the intruder is directly ahead and is moving towards the ownship, the score for *COC* will not be minimal.
 - The desired output property is that the score for *COC* is not the minimal score.
 - It has 5 input constraints: $1500 \leq \rho \leq 1800$, $\theta \leq |0.06|$, $\psi \geq 3.10$, $v_{own} \geq 980$, $v_{int} \geq 960$.
- **Property ϕ_4 .**

¹https://github.com/verivital/nnv/tree/master/code/nnv/examples/Submission/FM2019_Journal

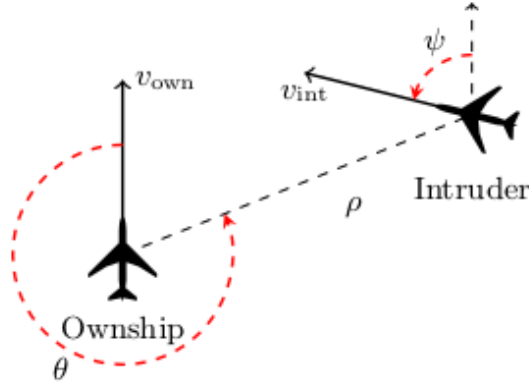


Figure III.3: Vertical view of a generic example of the ACAS Xu benchmark set.[41]

- If the intruder is directly ahead and is moving away from the ownship but at a lower speed than that of the ownship, the score for *COC* will not be minimal.
- The desired output property is that the score for *COC* is not the minimal score.
- It has 5 input constraints: $1500 \leq \rho \leq 1800$, $\theta \leq |0.06|$, $\psi = 0$, $v_{own} \geq 1000$, $700 \leq v_{int} \leq 800$.

Our experiments are done on a laptop with the following configuration: Intel Core i7-8850H CPU @ 2.6GHz 8 core Processor, 32 GB Memory, and 64-bit Windows 10 OS.² The verification results are presented in Tables III.1 and III.2. We used 6 cores for the exact reachability analysis of the ACAS Xu networks using the polyhedron- and star- based approaches, and only 1 core for the over-approximate reachability analysis approaches.

Verification results and timing performance. Safety verification using star-based reachability algorithms consists of two major steps. The first step constructs the whole reachable set of the networks. The second step checks the intersection of the constructed reachable set with the unsafe region. The verification time (VT) in our approach is the sum of the reachable set computation time (RT) and the safety checking time (ST). The reachable set computation time dominates (averagely 95% of) the verification time in all cases and the verification time varies for different properties.

Star set approach. The experimental results show that the improved exact star method is on average $27\times$ and $29.8\times$ times faster than Reluplex on property P_3 and P_4 respectively when we use parallel computing. Impressively, the approximate star method can achieve on average $1408\times$ (for property P_3) and $961\times$ (for property P_4) faster than Reluplex. This notable improvement illustrates the efficiency of star set in the reachability analysis and verification of piecewise linear DNNs where the affine mapping and half-space in-

²In [84], this experiment was done using Amazon Web Services Elastic Computing Cloud (EC2), on a powerful `m5a.24xlarge` instance with 96 cores and 384 GB of memory.

ID	Reluplex		Exact-Star			Approx-Star			Zonotope			Abstract Domain		
	Res.	VT	Res.	VT	Imp.	Res.	VT	Imp.	Res.	VT	Imp.	Res.	VT	Imp.
N_{11}	UNSAT	6156.00	UNSAT	383.68	16×	UNK	1.16	5326×	UNK	0.04	140157×	UNK	0.10	64073×
N_{12}	UNSAT	4942.00	UNSAT	244.63	20×	UNK	0.86	5760×	UNK	0.02	266171×	UNK	0.06	78243×
N_{13}	UNSAT	1134.00	UNSAT	55.96	20×	UNK	0.95	1199×	UNK	0.02	57670×	UNK	0.07	16221×
N_{14}	UNSAT	528.00	UNSAT	13.68	39×	UNSAT	0.17	3077×	UNK	0.02	24553×	UNK	0.08	6917×
N_{15}	UNSAT	317.00	UNSAT	18.25	17×	UNSAT	0.24	1348×	UNK	0.02	14100×	UNK	0.06	5091×
N_{16}	UNSAT	64.00	UNSAT	4.60	14×	UNSAT	0.09	701×	UNK	0.02	3673×	UNK	0.07	885×
N_{17}	SAT	1.00	SAT	2.17	0×	UNK	0.07	15×	UNK	0.02	62×	UNK	0.05	18×
N_{18}	SAT	3.00	SAT	1.73	2×	UNK	0.05	57×	UNK	0.02	187×	UNK	0.04	70×
N_{19}	SAT	2.00	SAT	1.55	1×	UNK	0.03	77×	UNK	0.01	137×	UNK	0.05	37×
N_{21}	UNSAT	1208.00	UNSAT	67.63	18×	UNK	0.55	2212×	UNK	0.02	65048×	UNK	0.06	20498×
N_{22}	UNSAT	653.00	UNSAT	22.24	29×	UNK	0.36	1833×	UNK	0.02	37376×	UNK	0.05	14029×
N_{23}	UNSAT	1043.00	UNSAT	38.68	27×	UNK	0.66	1586×	UNK	0.02	47091×	UNK	0.05	19669×
N_{24}	UNSAT	41.00	UNSAT	1.71	24×	UNSAT	0.05	819×	UNK	0.02	1856×	UNK	0.05	907×
N_{25}	UNSAT	235.00	UNSAT	8.45	28×	UNSAT	0.26	900×	UNK	0.02	13639×	UNK	0.06	3913×
N_{26}	UNSAT	79.00	UNSAT	1.64	48×	UNSAT	0.10	806×	UNK	0.02	4271×	UNK	0.06	1346×
N_{27}	UNSAT	116.00	UNSAT	4.16	28×	UNSAT	0.12	983×	UNK	0.02	7341×	UNK	0.07	1637×
N_{28}	UNSAT	83.00	UNSAT	1.74	48×	UNSAT	0.06	1498×	UNK	0.02	4568×	UNK	0.05	1552×
N_{29}	UNSAT	27.00	UNSAT	1.21	22×	UNSAT	0.02	1106×	UNSAT	0.01	1838×	UNK	0.03	776×
N_{31}	UNSAT	177.00	UNSAT	19.99	9×	UNSAT	0.22	810×	UNK	0.02	7332×	UNK	0.05	3380×
N_{32}	UNSAT	1561.00	UNSAT	210.03	7×	UNK	0.86	1819×	UNK	0.02	82581×	UNK	0.07	23248×
N_{33}	UNSAT	1105.00	UNSAT	35.06	32×	UNSAT	0.62	1784×	UNK	0.02	63345×	UNK	0.05	20432×
N_{34}	UNSAT	209.00	UNSAT	8.64	24×	UNK	0.78	268×	UNK	0.02	10255×	UNK	0.07	2906×
N_{35}	UNSAT	83.00	UNSAT	3.96	21×	UNSAT	0.13	629×	UNK	0.02	5085×	UNK	0.05	1594×
N_{36}	UNSAT	255.00	UNSAT	7.90	32×	UNK	0.34	749×	UNK	0.02	13679×	UNK	0.07	3597×
N_{37}	UNSAT	35.00	UNSAT	1.11	31×	UNSAT	0.04	922×	UNK	0.02	2225×	UNK	0.06	605×
N_{38}	UNSAT	179.00	UNSAT	2.87	62×	UNSAT	0.17	1082×	UNK	0.02	10501×	UNK	0.06	3228×
N_{39}	UNSAT	118.00	UNSAT	4.83	24×	UNSAT	0.11	1102×	UNK	0.02	7119×	UNK	0.05	2462×
N_{41}	UNSAT	201.00	UNSAT	8.77	23×	UNK	0.22	909×	UNK	0.02	11469×	UNK	0.04	4832×
N_{42}	UNSAT	2882.00	UNSAT	83.21	35×	UNK	0.52	5541×	UNK	0.02	160403×	UNK	0.06	50234×
N_{43}	UNSAT	1767.00	UNSAT	121.59	15×	UNK	0.82	2149×	UNK	0.02	90979×	UNK	0.06	31555×
N_{44}	UNSAT	86.00	UNSAT	2.19	39×	UNSAT	0.07	1280×	UNK	0.02	5094×	UNK	0.06	1367×
N_{45}	UNSAT	35.00	UNSAT	1.40	25×	UNSAT	0.06	561×	UNK	0.02	2045×	UNK	0.04	833×
N_{46}	UNSAT	300.00	UNSAT	11.27	27×	UNSAT	1.01	297×	UNK	0.02	14910×	UNK	0.07	4140×
N_{47}	UNSAT	126.00	UNSAT	3.50	36×	UNSAT	0.09	1476×	UNK	0.02	8054×	UNK	0.06	2152×
N_{48}	UNSAT	142.00	UNSAT	2.66	53×	UNSAT	0.07	2121×	UNK	0.02	8927×	UNK	0.06	2406×
N_{49}	UNSAT	143.00	UNSAT	4.03	35×	UNSAT	0.19	763×	UNK	0.02	7372×	UNK	0.06	2287×
N_{51}	UNSAT	1131.00	UNSAT	34.87	32×	UNK	0.47	2395×	UNK	0.02	62184×	UNK	0.05	24665×
N_{52}	UNSAT	151.00	UNSAT	7.28	21×	UNSAT	0.22	701×	UNK	0.02	9363×	UNK	0.03	4530×
N_{53}	UNSAT	341.00	UNSAT	8.53	40×	UNSAT	0.39	886×	UNK	0.02	20814×	UNK	0.05	6427×
N_{54}	UNSAT	62.00	UNSAT	3.02	21×	UNSAT	0.14	447×	UNK	0.02	3322×	UNK	0.06	1043×
N_{55}	UNSAT	86.00	UNSAT	4.36	20×	UNSAT	0.24	365×	UNK	0.02	4614×	UNK	0.06	1507×
N_{56}	UNSAT	283.00	UNSAT	4.75	60×	UNSAT	0.24	1184×	UNK	0.02	14773×	UNK	0.06	4483×
N_{57}	UNSAT	35.00	UNSAT	0.89	39×	UNSAT	0.02	1730×	UNSAT	0.02	2311×	UNK	0.04	831×
N_{58}	UNSAT	310.00	UNSAT	9.14	34×	UNSAT	0.18	1681×	UNK	0.02	16909×	UNK	0.06	4933×
N_{59}	UNSAT	19.00	UNSAT	1.05	18×	UNSAT	0.05	396×	UNK	0.01	1306×	UNK	0.02	799×
VT		28454.00		1480.60			14.01			0.84			2.56	
UNSAT	42/45		42/45			29/45			2/45			0/45		
Imp.				27×				1408×				29705×		9919×

Table III.1: Verification results for property P_3 on 45 ACAS Xu networks in which VT is the verification time in seconds. The exact-star method is on average $27\times$ faster than Reluplex. The approx-star is on average $1408\times$ faster than Reluplex. It can verify $29/45$ networks while the zonotope can verify only $2/45$ networks, and the new abstract domain approaches cannot verify any networks.

tersection operations can be done quickly. The improvement is also come from the utilization of the zonotope pre-filtering step. In comparison with the original star set method, the improved exact method reduces the total verification time by $5\times$ for property P_3 and by $2.6\times$ for property P_4 (the original exact method verifies 45 networks with 7457 seconds for P_3 and 1157 seconds for P_4). Similarly, the improved approximate method reduces the total verification time by $3.7\times$ for property P_3 and by $2.5\times$ for property P_4 (the original approx-

ID	Reluplex		Exact-Star			Approx-Star			Zonotope			Abstract Domain		
	Res.	VT	Res.	VT	Imp.	Res.	VT	Imp.	Res.	VT	Imp.	Res.	VT	Imp.
N_{11}	UNSAT	1291.00	UNSAT	76.95	17×	UNK	0.42	3107×	UNK	0.02	74511×	UNK	0.04	31883×
N_{12}	UNSAT	1267.00	UNSAT	47.54	27×	UNK	0.54	2336×	UNK	0.02	75568×	UNK	0.05	23811×
N_{13}	UNSAT	1150.00	UNSAT	36.29	32×	UNK	0.61	1890×	UNK	0.02	57693×	UNK	0.06	20327×
N_{14}	UNSAT	107.00	UNSAT	3.51	30×	UNK	0.14	774×	UNK	0.02	6034×	UNK	0.06	1842×
N_{15}	UNSAT	352.00	UNSAT	26.58	13×	UNK	0.21	1688×	UNK	0.02	21426×	UNK	0.03	10327×
N_{16}	UNSAT	219.00	UNSAT	12.80	17×	UNSAT	0.24	926×	UNK	0.02	13294×	UNK	0.05	4320×
N_{17}	SAT	1.00	SAT	2.03	0×	UNK	0.05	18×	UNK	0.01	70×	UNK	0.04	22×
N_{18}	SAT	3.00	SAT	2.10	1×	UNK	0.06	47×	UNK	0.01	200×	UNK	0.04	74×
N_{19}	SAT	3.00	SAT	1.94	2×	UNK	0.04	69×	UNK	0.02	195×	UNK	0.03	116×
N_{21}	UNSAT	330.00	UNSAT	15.69	21×	UNK	0.42	782×	UNK	0.02	20119×	UNK	0.05	6262×
N_{22}	UNSAT	415.00	UNSAT	14.79	28×	UNK	0.40	1044×	UNK	0.02	24470×	UNK	0.06	7016×
N_{23}	UNSAT	243.00	UNSAT	3.23	75×	UNSAT	0.12	2101×	UNK	0.02	14842×	UNK	0.03	7480×
N_{24}	UNSAT	86.00	UNSAT	3.55	24×	UNSAT	0.22	397×	UNK	0.02	4907×	UNK	0.03	2625×
N_{25}	UNSAT	151.00	UNSAT	11.95	13×	UNSAT	0.42	359×	UNK	0.02	8584×	UNK	0.05	2924×
N_{26}	UNSAT	118.00	UNSAT	5.65	21×	UNSAT	0.37	320×	UNK	0.02	6231×	UNK	0.07	1812×
N_{27}	UNSAT	34.00	UNSAT	2.47	14×	UNSAT	0.12	283×	UNK	0.02	1921×	UNK	0.07	512×
N_{28}	UNSAT	549.00	UNSAT	8.41	65×	UNK	1.24	442×	UNK	0.02	27415×	UNK	0.07	8003×
N_{29}	UNSAT	52.00	UNSAT	1.29	40×	UNSAT	0.04	1384×	UNK	0.01	3721×	UNK	0.03	1860×
N_{31}	UNSAT	478.00	UNSAT	14.23	34×	UNSAT	0.32	1490×	UNK	0.02	27161×	UNK	0.05	9575×
N_{32}	UNSAT	107.00	UNSAT	27.10	4×	UNSAT	0.21	504×	UNK	0.02	6648×	UNK	0.03	3313×
N_{33}	UNSAT	116.00	UNSAT	3.70	31×	UNSAT	0.12	992×	UNK	0.02	7316×	UNK	0.03	3956×
N_{34}	UNSAT	75.00	UNSAT	4.18	18×	UNSAT	0.13	582×	UNK	0.02	4733×	UNK	0.03	2245×
N_{35}	UNSAT	206.00	UNSAT	16.68	12×	UNSAT	0.71	292×	UNK	0.02	10914×	UNK	0.05	4387×
N_{36}	UNSAT	141.00	UNSAT	5.57	25×	UNSAT	0.31	461×	UNK	0.02	7652×	UNK	0.07	1979×
N_{37}	UNSAT	304.00	UNSAT	4.07	75×	UNSAT	0.11	2707×	UNK	0.02	14354×	UNK	0.07	4671×
N_{38}	UNSAT	131.00	UNSAT	2.98	44×	UNK	0.28	475×	UNK	0.02	5796×	UNK	0.05	2899×
N_{39}	UNSAT	621.00	UNSAT	13.04	48×	UNSAT	0.44	1424×	UNK	0.02	29817×	UNK	0.06	9948×
N_{41}	UNSAT	49.00	UNSAT	2.73	18×	UNSAT	0.07	661×	UNSAT	0.02	3158×	UNK	0.03	1931×
N_{42}	UNSAT	244.00	UNSAT	4.96	49×	UNSAT	0.32	761×	UNK	0.02	14887×	UNK	0.05	5170×
N_{43}	UNSAT	243.00	UNSAT	9.79	25×	UNSAT	0.32	750×	UNK	0.02	14721×	UNK	0.05	5205×
N_{44}	UNSAT	206.00	UNSAT	4.94	42×	UNK	0.57	362×	UNK	0.02	11038×	UNK	0.06	3643×
N_{45}	UNSAT	217.00	UNSAT	3.91	56×	UNSAT	0.22	992×	UNK	0.02	12636×	UNK	0.07	3225×
N_{46}	UNSAT	177.00	UNSAT	7.96	22×	UNSAT	0.30	582×	UNK	0.02	9665×	UNK	0.06	3215×
N_{47}	UNSAT	47.00	UNSAT	1.26	37×	UNSAT	0.09	496×	UNK	0.02	2843×	UNK	0.06	845×
N_{48}	UNSAT	195.00	UNSAT	6.10	32×	UNSAT	0.18	1094×	UNK	0.02	10859×	UNK	0.06	3280×
N_{49}	UNSAT	422.00	UNSAT	8.74	48×	UNSAT	0.17	2443×	UNK	0.02	26394×	UNK	0.05	8591×
N_{51}	UNSAT	513.00	UNSAT	19.88	26×	UNSAT	0.22	2305×	UNK	0.02	28381×	UNK	0.05	11007×
N_{52}	UNSAT	210.00	UNSAT	13.37	16×	UNSAT	0.15	1390×	UNK	0.02	13414×	UNK	0.04	5715×
N_{53}	UNSAT	124.00	UNSAT	5.16	24×	UNSAT	0.27	458×	UNK	0.02	7316×	UNK	0.05	2617×
N_{54}	UNSAT	144.00	UNSAT	3.61	40×	UNSAT	0.19	754×	UNK	0.02	8361×	UNK	0.04	3492×
N_{55}	UNSAT	114.00	UNSAT	5.29	22×	UNSAT	0.16	693×	UNK	0.02	6813×	UNK	0.06	2060×
N_{56}	UNSAT	160.00	UNSAT	3.01	53×	UNSAT	0.19	856×	UNK	0.02	9600×	UNK	0.04	3954×
N_{57}	UNSAT	38.00	UNSAT	1.13	34×	UNSAT	0.07	541×	UNK	0.02	2272×	UNK	0.05	772×
N_{58}	UNSAT	111.00	UNSAT	3.09	36×	UNSAT	0.28	399×	UNK	0.02	6110×	UNK	0.06	1715×
N_{59}	UNSAT	116.00	UNSAT	3.77	31×	UNSAT	0.14	821×	UNK	0.02	6647×	UNK	0.05	2205×
VT		11880.00		477.01			12.20			0.78			2.19	
UNSAT	42/45		42/45			32/45			1/45			0/45		
Imp.					29.8×			961.1×			14904.7×			5396.2×

Table III.2: Verification results for property P_4 on 45 ACAS Xu networks in which VT is the verification time in seconds. The exact-star method is on average $29.8\times$ faster than Reluplex. The approx-star is on average $961\times$ faster than Reluplex. It can verify $32/45$ networks while the zonotope can verify only $1/45$ networks, and the new abstract domain approaches cannot verify any networks.

imate method verifies 45 networks with 52 seconds for P_3 and 30 seconds for P_4). We note that due to these essential characteristics of a star set, the exact star-based method is also much more efficient and scalable than the polyhedron-based approach [83] whose the verification results are not presented in this paper.

Figure III.4 describes the benefits of parallel computing which can be exploited naturally with the star set approach. The figure shows that when a single core is used for verifying property P_4 on N_{13} , our approach

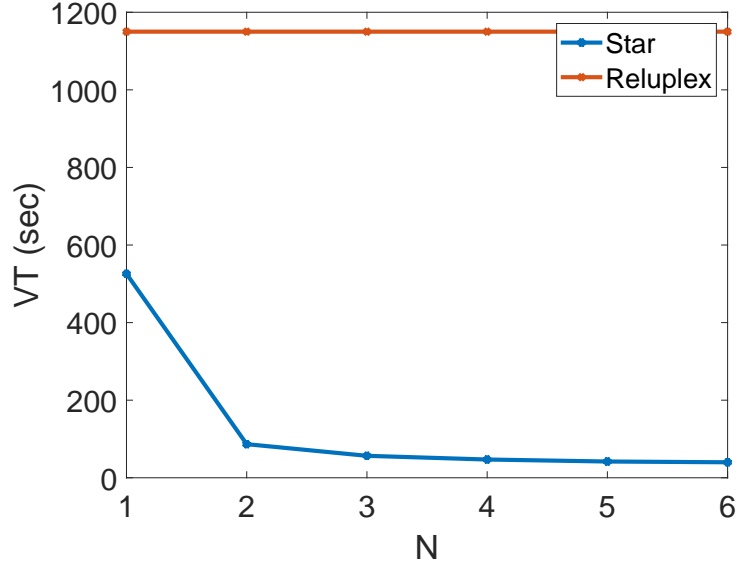


Figure III.4: Verification times for property ϕ_4 on N_{13} network with different number of cores.

takes 525.8 seconds which is $2.2\times$ faster than Reluplex (with 1150 seconds). With only 2 cores, our verification time drops quickly to 86.7 seconds which is $13.3\times$ faster than Reluplex. When 4 cores are used, the verification time decreases to 42 seconds which is $27.4\times$ faster than Reluplex. The figure shows that fact that when we use more cores for verification, the verification time may not be improved much. This is because the communication overhead between different cores becomes larger which affects directly to the verification time.

Zonotope-based method [74]. The experimental results show that the over-approximate, zonotope-based method is significantly faster than the exact methods. In some cases, it can verify the safety of the networks with a tiny verification time, for example, the zonotope-based method successfully verifies property ϕ_4 on N_{41} network in 0.02 seconds. Although the zonotope-based method is very time-efficient, it is unable to verify the safety of most of networks due to its huge over-approximation error. The zonotope approach can verify only $2/45$ ($\approx 4.44\%$) networks for property P_3 and $1/45$ ($\approx 2.22\%$) networks for property P_4 . In comparison with our approximate star method, we can verify $29/45$ ($\approx 64.44\%$) networks for property P_3 and $32/45$ ($\approx 71.11\%$) networks for property P_4 . This shows the fact that our method is significantly less conservative than the zonotope approach.

Abstract-domain based method [75] Similar to zonotope method, the abstract-domain is very time-efficient. However, it is the most conservative approach since it cannot verify any networks for both properties. We note that in [84], we implemented an improved version of the new abstract domain method in which we still solve LP optimization problems to find the lower and upper bounds of an input to a specific neuron

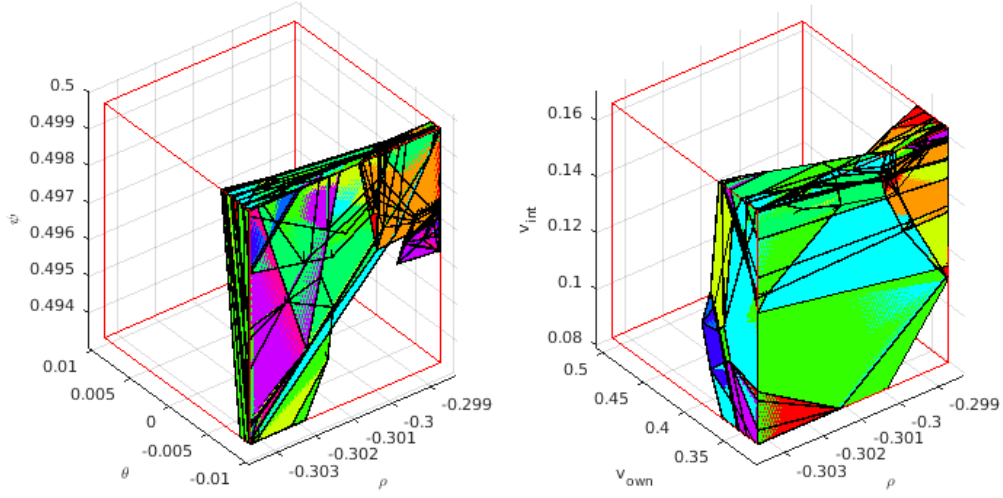


Figure III.5: The (normalized) complete counter input set for property ϕ'_4 on $N_{2.8}$ network is a part of the normalized input set (red boxes).

and use these bounds to construct the reachable set. In this paper, to have a fair comparison, we re-implement the original abstract domain method in which the lower and upper bounds are found using only ranges of new predicate variables. We have experienced that the abstract-domain method utilizing only the estimated ranges of new predicate variables to construct the reachable set is even more conservative than the zonotope approach.

Benefits of computing the reachable set. The computed reachable sets are useful for intuitively observe the complex behavior of the network. For example, Figure III.6 describes the behaviors of N_{51} network corresponding to property ϕ_4 requiring that the output *COC* is not the minimal score. From the figure, one can see that the *COC* $>$ *StrongRight* and thus, property ϕ_4 holds on N_{41} network. Importantly, as shown in the figure, via visualization, one can intuitively observe the conservativeness of different over-approximation approaches in comparison to the exact ones which is impossible if we use *ERAN*, a *C-Python* implementation of the zonotope and new abstract domain methods. We note that the reachable set obtained by the new abstract domain method is neglected for visualization because it is too large. Last but not least, the reachable set is useful in the case that we need to verify a set of safety properties corresponding to the same input set. In this case, once the reachable set is obtained, it can be re-used to check different safety properties without rerunning the whole verification procedure as Reluplex does, and thus helps saving a significant amount of time.

Complete counter example input set construction. Another strong advantage of our approach in comparison with other existing approaches is, in the case that a neural network violates its safety specification,

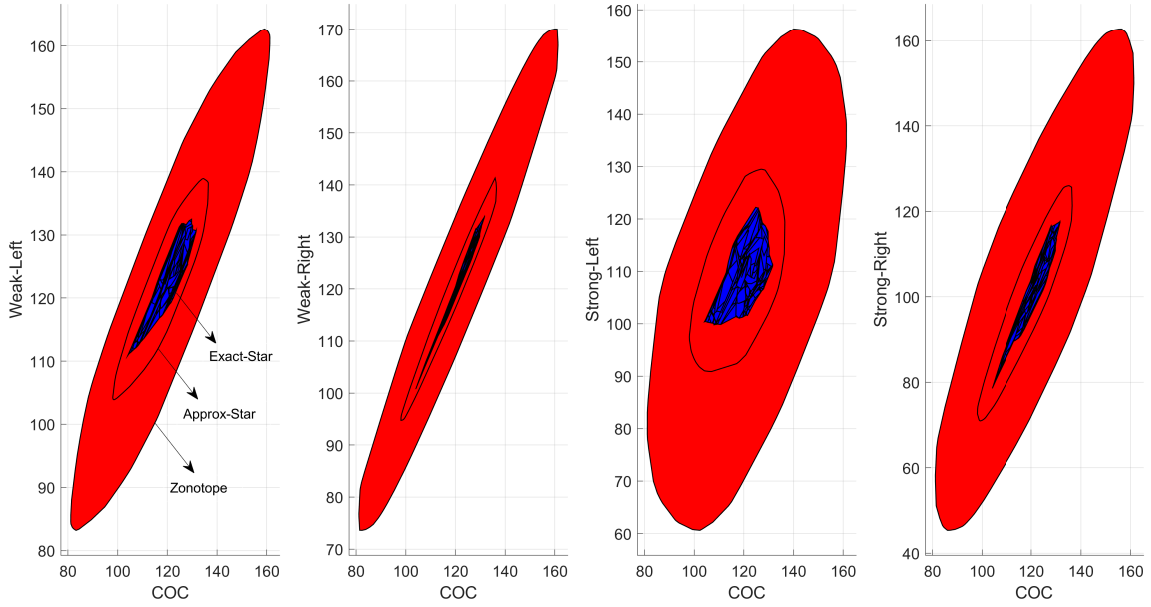


Figure III.6: Reachable sets of N_{41} network w.r.t property ϕ_4 with different methods.

our exact star method can construct a *complete counter input set* that leads the neural network to the unsafe region. The complete counter input set can be used as an adversarial input generator [33, 13] for robust training of the network. We note that finding a single counter input falsifying a safety property of a neural network can be done efficiently using only random simulations. However, constructing a complete counter input set that contains all counter inputs is very challenging because of the non-linearity of a neural network. To the best of our knowledge, our exact star-based approach is the only approach that can solve this problem. For example, assume that we want to check the following property $\phi'_4 \triangleq \neg(COC \geq 15.8 \wedge StrongRight \leq 15.09)$ on $N_{2,8}$ network with the same input constraints as in property ϕ_4 . Using the available reachable set of $N_{2,8}$ network, we can verify that the above property ϕ'_4 is violated in which 60 stars in 421 stars of the reachable set reach the unsafe region. Using Theorem III.2.5, we can construct a complete counter input set which is a union of 60 stars in 0.9893 seconds. This counter input set depicted in Figure III.5 is a part of the input set that contains all counter inputs that make the neural network unsafe.

III.4.2 Robustness Certification of Image Classification DNNs under adversarial attacks

Robustness certification of DNNs becomes more and more important as many safety-critical applications using image classification DNNs can be fooled easily by slightly perturbing a correctly classified input. A network is said to be δ -locally-robust at input point x if for every x' such that $\|x - x'\|_\infty \leq \delta$, the network assigns the same label to x and x' . In this case study, instead of proving the robustness of a network cor-

Net	Parameters	Tol	δ_{max}			
			Zonotope	Approximate-Star	Abstract-Domain	Exact-Star
N_1	k=5, N = 140	0.0001	0.0046	0.0048	0.0025	≥ 0.0058
N_2	k=5, N = 250	0.0001	0.0087	0.0101	0.0042	TimeOut
N_3	k=2, N = 1000	0.0001	0.0072	0.0089	0.0052	TimeOut
N_4	k = 1, N = 2000	0.0001	0.0027	0.0027	0.0027	TimeOut
N_5	k = 1, N = 4000	0.0001	0.0035	0.0035	0.0035	TimeOut

Table III.3: Maximum robustness values (δ_{max}) of image classification networks with different methods in which k is the number of hidden layers of the network, N is the total number of neurons, Tol is the tolerance error in searching.

responding to a given robustness certification δ , we focus on finding the maximum robustness certification value δ_{max} that a verification method can provide a robustness guarantee for the network. We investigate this interesting problem on a set of image classification DNN with different architectures trained (with an accuracy of 98%) using the well-known MNIST data set consisting of 60000 images of handwritten digits with a resolution of 28×28 pixels [50]. The trained networks have 784 inputs and a single output with expected value from 0 to 9. We find the maximum robustness verification value δ_{max} for the networks on an image of digit one with the assumption that there is a δ_{max} -bounded disturbance modifying the (normalized) values of the input vector x at all pixels of the image, i.e., $|x[i] - x'[i]| \leq \delta_{max}$. The result are presented in Table III.3. We note that the polyhedron and Reluplex approaches are not applicable for these networks because they cannot deal with a high-dimensional input space. The table shows that our approximate star approach produces larger upper bounds of the robustness values of the networks with many layers. For single layer networks, our approach gives the same results as the zonotope [74] and the abstract domain [75] methods. The exact-star method can prove that the network N_1 is robust with the bounded disturbance $\delta = 0.0058$. When $\delta > 0.0058$, we ran into the “out of memory” issue in parallel computation since the number of the reachable sets becomes too large. The exact star method reaches timeout (set as 1 hour) when finding the maximum robustness value for the other networks.

CHAPTER IV

Star-Based Reachability for Verification of Neural Network Control Systems

IV.1 System Model and Problem Formulation

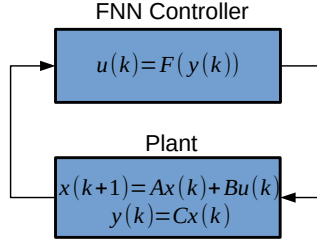


Figure IV.1: Neural network control system (NNCS).

IV.1.1 System model

In this chapter, we are interested in safety verification of CPS with neural network controllers as depicted in Figure IV.1 in which $x(k)$ and $y(k)$ are the state and the output of the plant at the time step k . The controller is a feedforward neural network (FNN) with the ReLU activation functions.

IV.1.2 Problem formulation

Problem IV.1.1 (Safety Verification of NNCS). *Given a CPS with an FNN controller F , and a discrete, linear plant P with the initial states $x(0)$ in an initial set X_0 , verify whether or not the state of the plant satisfies a safety property in a bounded time steps k_{max} . Formally, we want to verify if $\forall x(0) \in X_0 \rightarrow g(x(k)) \models S(g(x(k))), \forall 0 \leq k \leq k_{max}$ in which g is a nonlinear transformation function, S is a linear predicate over the transformed state variables $g(x(k))$ defining the safety requirements of the system.*

The core challenges in problem 1 are: 1) given the initial set of states of the plant, how can we efficiently compute the reachable set of the plant over time steps which depends on the control input produced by the FNN controller with nonlinear activation functions, 2) how can we transform the computed reachable set with a nonlinear transformation function to verify the safety property of the system. It is worth to emphasize that ***a small over-approximation error and timing efficiency in reachable set computation are two crucial metrics that determine the applicability of reachability analysis methods in safety verification of practical NNCS.*** Therefore, safety verification of NNCS requires computationally efficient methods that can compute the exact or tight over-approximate reachable sets of NNCS in a reasonable time. However, computing the exact or tight over-approximate reachable sets of an FNN is difficult and usually time-consuming. In addition, simple utilization of the control set from the controller to compute the reachable set of the plant may produce

a very coarse reachable set which is useless in safety verification. Overcome the challenges in problem 1 is a fundamental step to tackle the following important problem.

Problem IV.1.2 (Safety-critical initial condition of NNCS). *Given a CPS in problem 1 with the initial states $x(0) \in X_0$, determine the initial condition of the i^{th} state $x^i(0)$ that “may” make the system unsafe while keeping the initial conditions of other states unchanged. We call this initial condition is a “safety-critical initial condition” of the system and assume that the initial conditions of all states are independent.*

Problem 2 is even harder than problem 1 since it is almost impossible to perform backward analysis of CPS with neural network controllers to determine an unsafe initial condition (backward analysis is generally intractable in this case). In the following, we first present our core reachability algorithm for neural network control systems (NNCS). Then, we discuss handling the nonlinear transformation on the computed reachable set for checking the safety of the system, i.e., Problem 1 as well as searching safety-critical initial condition, i.e., Problem 2.

IV.2 Reachability Analysis of Neural Network Control Systems

The reachability analysis of a NNCS depicted in Figure IV.1 is done as follows. First, from the initial set of states X_0 of the plant P , the controller F takes the output set of the plant Y_0 as an input to compute the control set $U = F(Y_0)$. Note that Y_0 is an affine mapping of the initial set X_0 with the output matrix C , i.e., $Y_0 = CX_0$. The control set U is then applied to the plant to compute the set of the next state $X_1 = AX_0 + BU$. This routine is performed iteratively to obtain the reachable set of the plant X_0, X_1, \dots, X_k , $0 \leq k \leq k_{max}$. To obtain tight reachable sets of the NNCS, we compute the exact control set U given the output set Y . Also, we compute the exact reachable set of state X_k given its initial set X_{k-1} and the corresponding control set U_{k-1} .

IV.2.1 Exact reachability analysis of the neural network controller

The first step in our reachability analysis is to compute the exact control set $U_k = F(CX_k)$ using star-set approach [84]. Although the star set based method is similar to the polyhedron-based approach [83], it is much more efficient and scalable because star set is very fast in affine mapping which is the most expensive step in the polyhedron-based approach, especially for a high dimensional set. More importantly, the computed output set and the input set of the FNN are defined based on the same set of predicate variables, i.e., $\alpha = [\alpha_1, \dots, \alpha_m]^T$. This property is crucial in eliminating the over-approximation error in computing the reachable set for the plant as addressed in the following.

IV.2.2 Exact reachability analysis of the discrete linear plant

As shown in previous subsection, the exact control set $U_k = F(CX_k)$ is a union of stars, $U_k = \cup_{j=1}^L \tilde{\Theta}_j$. Therefore, the exact reachable set of the plant for the next step is also a union of stars, $X_{k+1} = AX_k + BU_k$. Interestingly, the state set $X_k = \langle c, V, P \rangle$ and the control set U_k are defined based on a unique predicate variable vector α and for any star in the control set, its predicate contains all linear constraints of the state set X_k as can be seen in Figure III.1. This leads to an important fact that, only a subset of X_k can lead to an individual control set $\tilde{\Theta}_j \in U$ and the predicate of this subset is exactly the predicate of the individual control set. Therefore, the next state set corresponding to the individual control set $\tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j \rangle$ is $X_{k+1}^j = \langle Ac + B\tilde{c}_j, AV + B\tilde{V}_j, \tilde{P}_j \rangle$. Consequently, the exact next state set of the plant is $X_{k+1} = \cup_{j=1}^L X_{k+1}^j$.

IV.2.3 Reachability algorithm for NNCS

As shown previously, we can compute the exact reachable set of NNCS depicted in Figure IV.1 by computing the exact control set and the exact state set of the plant. For a single initial state set, after one time step, it may produce many other state sets. Therefore, the number of state sets increases quickly over time which makes the exact analysis time-consuming even using parallel computing. To handle this state sets explosion, we can obtain a single convex hull of the state sets after every step and use it for the next step computation. Computing the convex hull for a set of stars is essentially computing the convex hull of a set of convex polyhedrons which is computationally expensive. To overcome this challenge, we instead compute the interval hull of a set of stars for the next step computation which can be done efficiently by solving a set of linear programming optimization problems. The experimental results show that, by using only the interval hull of the star state sets, we still can obtain a tight over-approximation of the exact reachable set for the NNCS and more importantly, the over-approximation error does not explode over time. The reachability algorithm for a NNCS is summarized in Algorithm 5 in which the user can choose to compute the exact or the over-approximate reachable sets of the NNCS.

Lemma IV.2.1. *The exact scheme in Algorithm 5 produces the exact reachable sets of the NNCS depicted in Figure IV.1.*

Proof. The proof can be derived inductively based on the exact computation of the reachable set of the plant and the neural network controller in every step. □

IV.2.4 Extension to NNCS with nonlinear plants

It is interesting to emphasize that the proposed star-based reachability algorithm can be extended to deal with neural network control systems with nonlinear plants. The core idea of the extension is that we can use

Algorithm 5 Reachability Algorithm for NNCS

```
1: %  $F$ : neural network controller
2: %  $A, B, C$ : plant's matrices  $x_{k+1} = Ax + Bu, y_k = Cx_k$ 
3: %  $I$ : initial set of states of the plant
4: %  $k_{max}$ : number of steps
5: %  $scheme$ : reachability analysis scheme, "exact" or "approx"
6: %  $R$ : reachable set
7: procedure  $R = \text{REACH}(F, A, B, C, I, k_{max}, scheme)$ 
8:    $R = \text{cell}(1, k_{max} + 1)$ 
9:    $R\{1, 1\} = I$ 
10:  for  $k = 1 : k_{max}$  do
11:     $X_k = R\{1, k\}, M = \text{length}(X_k)$ 
12:    for  $i = 1 : M$  do
13:       $X_k^i = X_k(i) = \langle c, V, P \rangle$ 
14:       $U_k = F(CX_k^i) = \cup_{j=1}^L \tilde{\Theta}_j = \cup_{j=1}^L \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j \rangle$ 
15:       $X_{k+1} = []$ 
16:      for  $j = 1 : L$  do
17:         $X_{k+1}^j = \langle Ac + B\tilde{c}_j, AV + B\tilde{V}_j, \tilde{P}_j \rangle$ 
18:         $X_{k+1} = [X_{k+1} \ X_{k+1}^j]$ 
19:      if  $scheme == exact$  then  $R\{1, k+1\} = X_{k+1}$ 
20:      else  $R\{1, k+1\} = \text{IntervalHull}(X_{k+1})$ 
```

existing hybrid systems reachability methods, such as the zonotope-based reachability algorithm in CORA [5] that we chose to use, to compute the reachable set of a nonlinear plant between two time steps t_k and t_{k+1} . This algorithm first further divides the time between t_k and t_{k+1} into N_p smaller time steps, and then performs a sound linearization-based reachable set computation for the plant along with N_p time steps to obtain a reachable set with N_p stars (we note that a zonotope is also a star). We refer readers to [6] for the technical details of this hybrid systems reachability approach. The last star in the union is the initial set of states of the plant for the next time interval $[t_{k+1}, t_{k+2}]$. This star is also feedback to the neural network controller. Then, the exact star-based reachability algorithm is invoked to compute the control input U for the next control step.

IV.3 Verification of Neural Network Control Systems

IV.3.1 Safety verification

Although safety properties in CPS are often represented as a linear predicate over the system's states x_k , there are many cases where the safety property is defined as a linear predicate over a variable z_k that is a *nonlinear transformation* of the system's states, i.e., $z_k = g(x_k)$, where g is a nonlinear function. Let $\mathcal{U}(z_k) \triangleq Hz_k \leq h$ be the unsafe region of a NNCS, then safety verification of the NNCS, i.e., Problem 1, is equivalent to checking $Z_k \cap \mathcal{U}(z_k) = \emptyset? \forall 0 \leq k \leq k_{max}$, where $Z_k = \{z_k | z_k = g(x_k), x_k \in X_k\}$ is the transformed reachable set of the system by applying $g(\cdot)$ to it. Since computing the exact transformed reachable set is computationally

Algorithm 6 Safety Verification for NNCS

Input: R, g, U : Reachable set of the NNCS, transformation function, unsafe region

Output: $safe = true$ or $safe = uncertain$

```
1: procedure  $safe = \text{VERIFY}(R, g, U)$ 
2:    $k_{max} = \text{length}(R)$ 
3:   for  $k = 1 : k_{max}$  do
4:      $X_k = R\{1, k\}$ 
5:      $z_k = \min(g(x_k)), \bar{z}_k = \max(g(x_k)), x_k \in X_k$ 
6:      $\tilde{Z}_k = [z_k, \bar{z}_k]$ 
7:     if  $\tilde{Z}_k \cap U = \emptyset$  then  $safe = true$ 
8:     else  $safe = uncertain$ , break
```

expensive and may be even infeasible, we compute an over-approximation of the exact transformed reachable set \tilde{Z}_k and use it for safety verification. The system is safe if $\tilde{Z}_k \cap \mathcal{U}(z_k) = \emptyset, \forall 0 \leq k \leq k_{max}$. Particularly, we compute the tightest interval bounding the exact transformed reachable set by solving the following nonlinear optimization problem:

$$\tilde{Z}_k = [z_k, \bar{z}_k], z_k = \min(g(x_k)), \bar{z}_k = \max(g(x_k)), x_k \in X_k.$$

Safety verification of the NNCS is summarized in Algorithm 6, which solves the above nonlinear optimization problem to obtain the tightest interval of the transformed reachable set and uses it to verify safety of the system at each time step.

IV.3.2 Characterization of safe initial condition

Safety verification of a NNCS can reason about the safety of the system w.r.t a specific initial condition. In some cases, we are interested in the upper bound of a particular state $x^i(0)$ in the initial condition where the safety of the system is still guaranteed. For example, if a car detects an obstacle and applies the brake to stop, it is important to know what is the maximum velocity of the vehicle such that the braking action can guarantee the safety of the car. To search for that maximum velocity, we start from the initial condition that the system is safe, then we increase the upper bound of the speed by some δ , i.e., $x^i(0) = x^i(0) + \delta$, and check the safety of the system with the new initial condition. We continue to increase the upper bound until the safety is uncertain. We can obtain the maximum allowable velocity with the error of $[-\delta, \delta]$.

IV.4 Evaluation

Our approach is implemented in *NNV* [83, 84], a Matlab toolbox for safety verification of DNNs and learning-enabled CPS. The proposed approach is evaluated on a practical automatic emergency braking system (AEBS) and an adaptive cruise control system (ACC) for an autonomous car. The experiment is done

on a computer with following configurations: Intel Core i7-8859H CPU @ 2.6GHz × 4 Processor, 32 GiB Memory, Window 10 Pro OS.¹

IV.4.1 Advanced Emergency Braking System

The architecture of the AEBS is described in Figure IV.2 in which the car is equipped with a perception component to detect automatically the obstacle on the road and a reinforcement learning (RL) based controller to control the brake of the car.

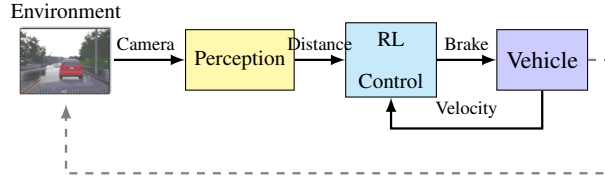


Figure IV.2: Emergency Braking System Architecture

IV.4.1.1 Scenario of Interest

In our system, we consider the scenario that the host car automatically detects another static vehicle and applies a brake to decelerate and stop to avoid the potential collision as shown in Figure IV.3.

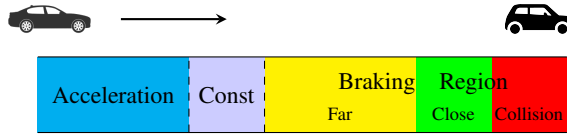


Figure IV.3: Illustration of Emergency Braking System

The host car starts from rest and accelerates to a random initial velocity v_0 , which introduces the uncertainty to the system. Then, the car keeps this velocity v_0 till an obstacle is detected at distance d_0 from the perception module and switches to the reinforcement learning braking controller. The goal of the controller is to stop the car to avoid the collision and also not too far from the obstacle, which means the car should stop within the safety and close region.

IV.4.1.2 Safety Specification

The safety property of the AEBS is defined based on the concept of time-to-collision (TTC) [52, 47]. TTC measures the time it would take to collide if the vehicle continues traveling based on the current acceleration of $a_k = u_k$ and velocity v_k . Smaller TTC means a higher collision risk. The safety specification of the AEBS can be written by

$$(\text{TTC}_k(d_k, v_k, a_k) > \tau(v_k)) \mathcal{U} \quad (k = k_{max})$$

¹All results presented in this paper and their corresponding scripts are available online at <https://github.com/verivital/nmv/releases/tag/emsoft2019>.

where $\tau(v_k)$ is the time to stop when applying the full brake for velocity v_k , shown in Figure IV.4, d_k is the current distance from the car to the obstacle, k_{max} is the maximum number of steps we want to verify the safety of the system, and \mathcal{U} is the until operator. Generally, the safety specification means that the car is safe if it still has enough time for a full braking action, i.e., full braking action can successfully stop the car before a collision occurs.

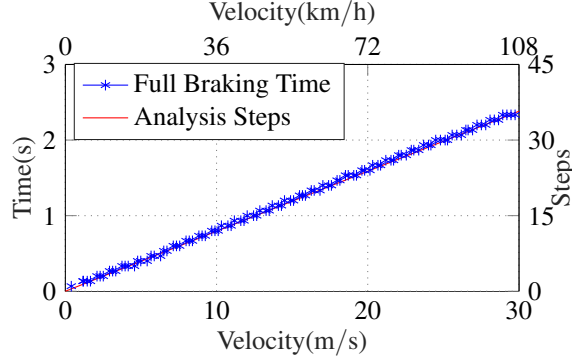


Figure IV.4: Obtaining the required number of reachability analysis steps from the full-braking characteristic of the car at different speeds.

Because of the discontinuity caused by the denominator when velocity or acceleration equals zero, it is more efficient to evaluate the collision risk using the inverse TTC introduced in [12]. The inverse TTC is proportional to the collision risk: the higher it is, the higher the collision risk is. The safety specification using the inverse TTC is given below,

$$(\text{TTC}_k^{-1}(d_k, v_k, a_k) < \tau^{-1}(v_k)) \mathcal{U} (k = k_{max}),$$

where, the inverse TTC is defined by:

$$\text{TTC}_k^{-1}(d_k, v_k, a_k) = \begin{cases} \frac{v_k}{d_k} & \text{for } a_k = 0 \\ \frac{-a_k}{v_k - \sqrt{v_k^2 + 2a_k d_k}} & \text{for } v_k^2 + 2a_k d_k \geq 0 \wedge a_k \neq 0 \\ 0 & \text{for } v_k^2 + 2a_k d_k < 0 \wedge a_k \neq 0. \end{cases}$$

IV.4.1.3 RL-based Controller

We train the RL-based controller for the host car using Deep Deterministic Policy Gradient (DDPG)[53], which is a popular reinforcement learning method that combines the value-based and the policy-based method. There are two parts in this approach including actor and critic. Critic uses the off-policy data to learn the Q-function, which evaluates how good the action a taken is in given state s . The actor can learn the continuous

action policy by using the Q-function. In practice, it is difficult to obtain the exact Q-function and policy function. Therefore, two neural networks are introduced to solve this problem, which is critic network $Q(s, a | \theta^Q)$ and actor network $\mu(s | \theta^\mu)$ with weights θ^Q and θ^μ . Coming back to our braking system, the reinforcement learning controller consumes the state s , consisting of distance to the leader vehicle d and host car's velocity v , and computes the action – brake T .

For a reinforcement learning system, the reward function should be appropriately designed to achieve the goal. In our case, the task is to stop the car in a safe and close region. Thus, we define the reward function as

$$r = -\alpha \times I \times \mathbf{1}(\text{collision}) - [(d_t - B) \times \beta + \lambda] \times \mathbf{1}(v_t = 0 \wedge d_t > B)$$

where d_t and v_t indicates the distance to the leader car and velocity at time step t , α , β and λ are coefficients greater than zero, $\mathbf{1}(\cdot)$ returns a value of 1 if the statement inside is true and 0 otherwise.

The term of the reward function, $-\alpha \times I \times \mathbf{1}(\text{collision})$ penalizes a collision event based on the collision impulse I . The other term $-[(d_t - B) \times \beta + \lambda] \times \mathbf{1}(v_t = 0 \wedge d_t > B)$ penalizes a too early stop based on B , the final distance to the boundary line between close and far region. During the braking process (before the car comes to a stop), there is no penalty or reward. Intuitively, this reward function will guide the car to stop within the close region.

We use CARLA [19] to generate the scenario and to train the reinforcement learning controller. The time step used in the simulation is $\Delta t = 1/15$ s. In the simulations, the vehicle firstly accelerates to a velocity of v_0 , and keeps the speed until it detects an obstacle at a distance d_0 . The d_0 and v_0 are the initial states of the braking system. To simulate a more realistic scenario, we introduce some uncertainty to the initial states of the system. The initial velocity of the vehicle is uniformly sampled between 90 km/h and 100 km/h, and the initial distance depends on the range of the perception module, which is approximately 100 m. After initial state, the car switches to the reinforcement learning controller which consists of two neural networks trained with DDPG algorithm with the hyper-parameters in Table IV.1 is presented below:

- Actor NN architecture²:

$$2(\text{State}) \times 50(\text{ReLU}) \times 30(\text{ReLU}) \times 1(\text{SatReLU}, \text{Action})$$

²SatReLU is the ReLU function with max value 1.

Table IV.1: Hyper-parameters for DDPG algorithm.

	Actor	Critic
Optimizer	Adam	Adam
Learning rate	10^{-4}	10^{-3}
Target update rate	0.9	0.9
Reply buffer size		10^5
Reply batch size		32
Discount factor		0.99
Reward function	$\alpha = 0.01, B = 5, \beta = 1.6, \lambda = 20$	

- Critic NN architecture³:

$$\left. \begin{array}{l} 2(\text{State}) \times 50(\text{ReLU}) \times 30 \\ 1(\text{Action}) \times 30 \end{array} \right\} \times 30(\text{ReLU}) \times 1(\text{Q Value})$$

We trained the reinforcement learning for 1000 episodes, and the neural network converges, showing an attractive performance. Also, one of the experiment trajectories is plotted in Figure IV.6. At the beginning of involving the reinforcement learning controller, the distance is 97.3m, and the velocity is 91.98km/h (= 25.55 m/s). After 128 steps, about 8.53s, the ego vehicle stops at about 1.88m far from the obstacle vehicle.

IV.4.1.4 System Identification and Validation

We transfer the braking system from CARLA to MATLAB & Simulink to perform reachability analysis and safety verification for the system. The diagram of the Simulink model of the AEBS is shown in Figure IV.5. For simulation and verification, only the actor is needed. The plant of the braking system is described by

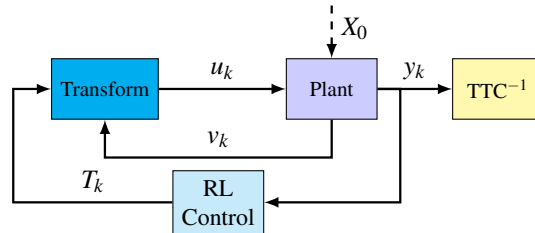


Figure IV.5: Emergency Braking System Simulink Diagram

³The empty activation function means no activation is applied.

following discrete state-space equation

$$\begin{cases} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \end{cases}$$

where $x_k = [d_k \ v_k]^T$ is the state vector including the distance d_k and the velocity v_k of the car at step k , u_k is the input, which is the acceleration applied to the plant, y_k is the output, and A, B, C, D are the coefficient matrices given below,

$$A = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ \Delta t \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

where $\Delta t = 1/15$ is simulation time step.

It is important to emphasize that the input of the plant u_k does not match with the output of the reinforcement learning controller T_k . The u_k is the acceleration applied to the car, but the T_k is the braking force. Thus, a neural network transformation with 80 neurons is trained to bridge this gap between u_k and T_k .

To validate the Simulink model of AEBS, we run experiments in Simulink and CARLA with the same initial states and compare them as shown in Figure IV.6. From the plot, we can see that the Simulink model captures very well the behaviors of the (actual) AEBS in CARLA.

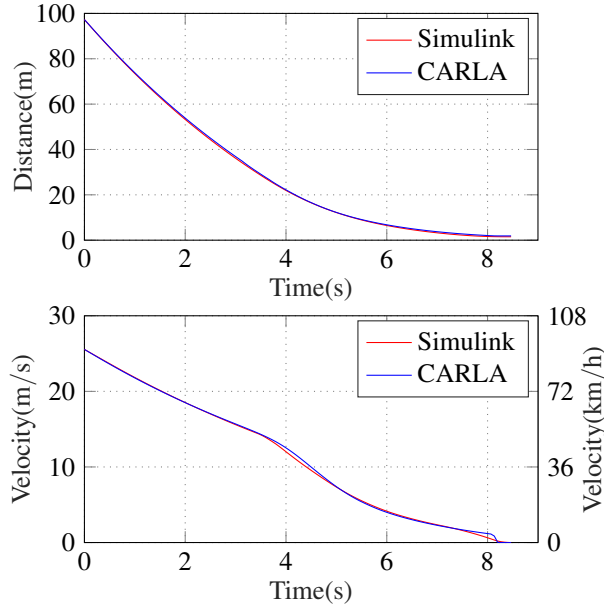


Figure IV.6: Validation of the Simulink model of AEBS. The Simulink model captures well the behaviors of the actual AEBS in CARLA.

IV.4.1.5 Safety Verification of the AEBS

Physical constraints for safety verification. To verify the safety of the AEBS, we need to take into account some essential physical constraints of the system. First, the AEBS system uses a perception component to detect the obstacle. The operating range of the perception component is from 0 to 100 meters. Therefore, we are going to verify the safety of the AEBS for the distance (between the car to the obstacle) from 10 to 100 meters (we assume that the car is at least 10 meters far away from the obstacle). Secondly, we limit the maximum allowable velocity of the car is 35 m/s, i.e., ≈ 80 miles per hour which is a usual upper limit of the speed on highways.

Thirdly, we need to know what is a reasonable constraint between initial conditions of the car's velocity and its distance to the obstacle such that if we apply a full braking action, the car is safe. This information is important that we should know before verifying the safety of AEBS because there are cases when even if we apply the full braking action, the collision still occurs. For example, the car is too close to the obstacle and is travelling at a high speed. From Figure IV.4, we approximate an analytical formula for the full braking time that is $\tau(v) \approx v/12.5$. When the full braking action occurs, the car goes a distance $d_s = 0.5a\tau^2 + v\tau$ before stopping, where a is the average acceleration of the car which is equal to $a = \Delta v/\Delta t = (0 - v)/\tau$. Therefore, the average travel distance of the car after applying a full brake is: $d_s = 0.5v\tau = 0.5v^2/12.5 = v^2/25$. To guarantee the safety, the initial distance of the car d_0 should be larger than this travel distance, i.e., $d_0 > d_s$. Combining the above limitation on the distance $d_{max} = 100$ m and the maximum allowable velocity $v_{max} = 35$ (m/s), a *safe initial condition region for full braking action* is depicted in Figure IV.7. By partitioning the safe initial condition region of the full braking action, we can derive *the reasonable initial conditions that need to be verified for the safety of the AEBS with the RL controller* as shown in Figure IV.8. This is because, under the safety aspect, the RL controller cannot overcome the full-braking action.

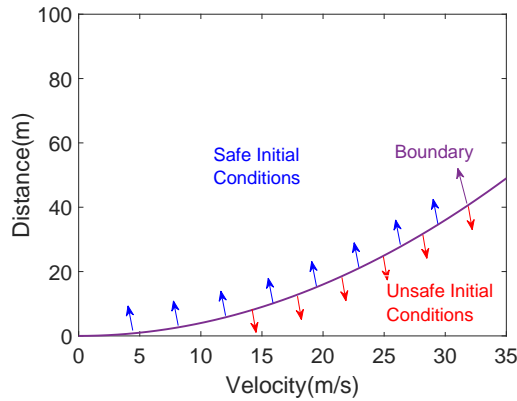


Figure IV.7: Safe initial conditions for full braking action.

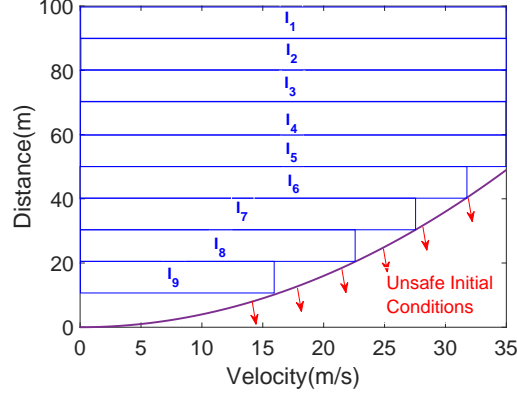


Figure IV.8: Set of initial conditions that needs to be verified for the AEBS with the RL controller.

Finally, we need to find out what the minimum number of steps that we should at least give a guarantee about the safety of the system is. We should prove the safety of the system at least $\tau(v)$ seconds in the future where $\tau(v)$ is the full braking time w.r.t the velocity v . Therefore, **the minimum number of steps that needs to prove the safety** is: $\min(k_{max}) = \tau(v)/\Delta t$. For example, if $v = 25$ m/s, we should at least prove the safety of the system until $k = k_{max} = 2/(1/15) = 30$ time steps.

Challenges and drawbacks of the polyhedron [83, 103] and interval [25] approaches. A main challenge in safety verification of AEBS is how to compute a tight reachable set of the AEBS model depicted in Figure IV.5. One can see that the control set $U = \{u_t\}$ applied to the plant is derived from the transformation component that takes the output set $T = \{T_t\}$ from the RL controller and the velocity $V = \{v_t\}$ as the input set. Therefore, to compute the control set U , we need to compute the output set T of the RL controller and then combine with the velocity set $V = \{v_t\}$ of the plant to form the input set for the transformation neural network. The problem is how to efficiently combine these sets to form the exact input set for the transformation neural network. This problem is unsolvable if we use the polyhedron-based [83, 103] or the interval [25] methods since the relationship between the output set T of the RL controller and the velocity set V of the plant cannot be preserved in the computation. This leads to a coarse combination which returns a coarse input set for the transformation neural network. Consequently, the over-approximation error is exploded quickly after only 2 time steps as shown in Figure IV.9 which makes the obtained reachable sets become too conservative and cannot be used for safety verification.

Minimizing overapproximation while maintaining scalability with star sets. As shown in Figure IV.9, our star-based approach is an efficient technique to overcome the main challenges discussed above. We compute the reachable set for the AEBS system in 50 steps. One can see that our star-based approach eliminates (in the exact method) or reduces significantly (in the over-approximation method) the over-approximation errors caused by the polyhedron-based and the interval approaches. The reachable sets computed from the

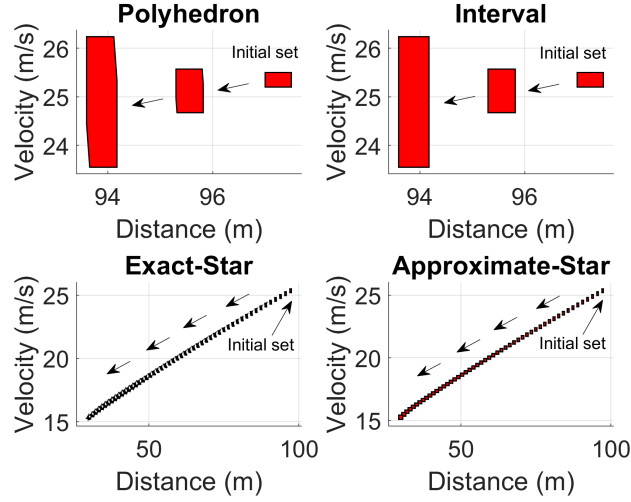


Figure IV.9: Reachable sets of the AEBS computed by the polyhedron and the interval approaches become too conservative are quickly after only 2 time steps while the proposed star methods obtains the exact or tight over-approximate reachable sets for many time steps. The initial conditions are $d_0 \in [97, 97.5]$, $v_0 \in [25.2, 25.5]$.

star-based approach are tight and useful for safety verification of the AEBS.

Error analysis. Since we can compute the exact reachable sets of the system, we can analyze the over-approximation errors of different approaches. These overapproximation errors are measured using the following metrics $OverApprox - Error = \max(|lb - lb_{exact}|, |ub - ub_{exact}|)$ which measures that largest distance between the lower- and upper- bounds of an output computed from the overapproximation methods and the exact lower- and upper-bounds computed by the exact star method. The overapproximation errors of the polyhedron, interval, and the proposed over-approximate star methods are depicted in Figure IV.10. One can see that the overapproximation errors of the polyhedron and the interval methods are increased quickly after only two steps while these errors are almost zeros for the first 45 steps and very small in the last five steps in the over-approximate star method.

Timing performance. The reachability analysis times of two proposed methods are presented in Table IV.2. The Table shows that the over-approximation method is faster than the exact method while still produces tight reachable sets for the system. From the figures, one can see that the reachable sets computed by the two methods are almost the same. The time improvement of using the over-approximation method increases as the number of time steps grows. The reason that makes the over-approximation method faster is, it produces only a single reachable set at every time step while in the exact method, the number of reachable sets may grow over time as depicted in Figure IV.11.

Checking safety using the computed reachable sets. To verify the safety of the AEBS, we consider the

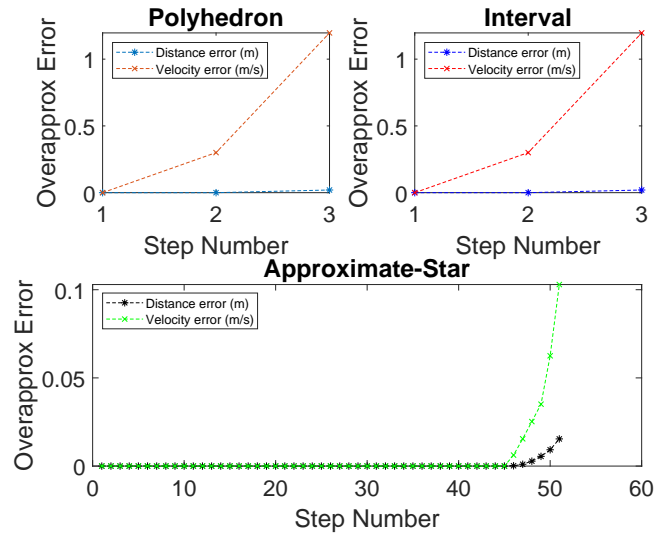


Figure IV.10: The Over-approximation errors of the polyhedron and the interval approaches are exploded quickly after only 2 time steps. These over-approximation errors are reduced significantly by the proposed star method.

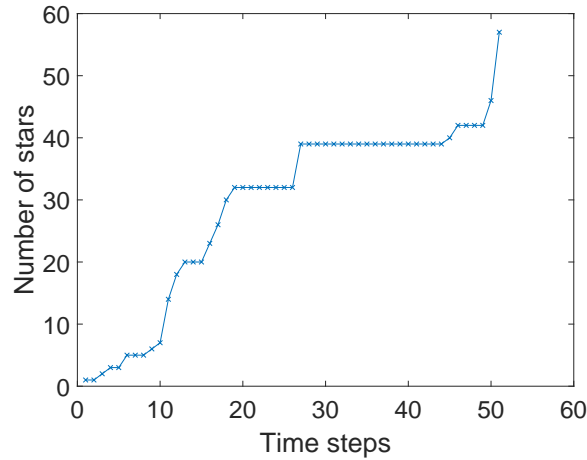


Figure IV.11: Number of stars in the reachable sets of the AEBS grows over time with the exact star-based method.

Method	N = 5	N = 10	N = 20	N = 30	N = 40	N = 50
Exact star	12.47	32.24	162.95	400.13	532.1	831
Over-approximate star	10.07	21.09	42.86	63.98	83.3	104.44
Time improvement	1.24x	1.53x	3.8x	6.25x	6.39x	7.96x

Table IV.2: Reachability analysis times (measured in seconds) of the exact and over-approximate star methods in which N is the number of time steps

worst case, i.e., we want to verify if the following constraint is satisfied in a bounded time, $\max(TTC^{-1}(d, a, v)) < \tau^{-1}(\max(v))$. To do that, we estimate the ranges of TTC^{-1} in 60 time steps (two times larger than the minimum requirement $k_{max} = 30$) using the ranges of the distance, velocity, and acceleration of the car from the computed reachable sets and check if it satisfies the requirement or not. The result is illustrated in Figure IV.12 which shows that the inverse TTC is smaller than the worst case inverse full braking time $\tau^{-1}(\max(v))$. Therefore, the AEBS is safe for 60 time steps in the future.

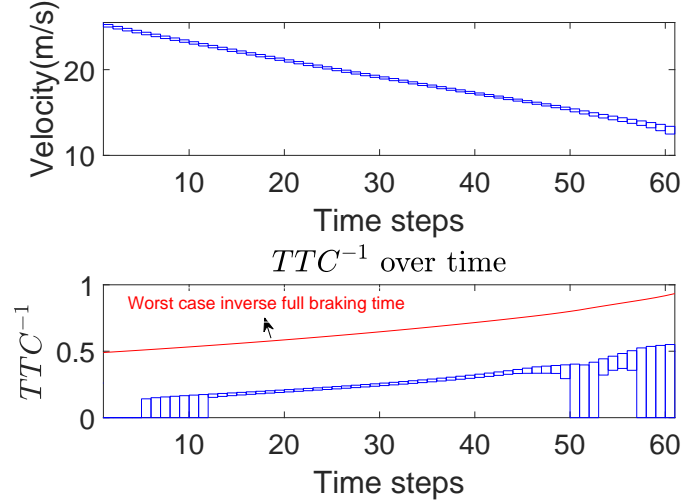


Figure IV.12: The inverse TTC over time is smaller than the worst case inverse full braking time $\tau^{-1}(\max(v))$. The AEBS is safe (for 60 time steps) with the initial conditions $d_0 \in [97, 97.5]$, $v_0 \in [25.2, 25.5]$.

IV.4.1.6 Safe Initial Conditions of the AEBS

From the physical constraints of the car, we have derived the set of initial conditions that need to be verified for the AEBS with RL controller as depicted in Figure IV.8. It is important to determine in these initial conditions, which regions are safe for the AEBS with RL controller and which ones are risks. We perform our safety verification methods on each partition $I_i, i = 1, 2, \dots, 9$ of the initial conditions to find the safe regions. We perform the search as follows. We partition the distance range $[10, 100]$ into 9 smaller ranges with the same width of 10, i.e., $d^i = [10i, 10(i+1)], 1 \leq i \leq 9$. For the i^{th} individual distance range, we search for the maximum velocity v_{max}^i such that the RL controller can guarantee the safety of the system in $k_{max} = 50$ time steps for the initial condition of $[d^i, v_{max}^i]$. The results of v_{max} are presented in Table IV.3. From the information of v_{max} , we visualize the safe region of the initial conditions for the AEBS as depicted in Figure IV.13.

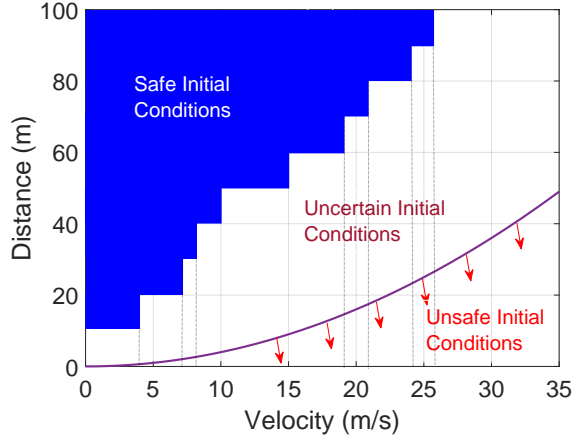


Figure IV.13: Safe region of the initial conditions of the AEBS with the RL controller.

$d(m)$	[0, 10]	[10, 20]	[20, 30]	[30, 40]	[40, 50]	[50, 60]	[60, 70]	[70, 80]	[80, 90]	[90, 100]
$v_{max}(m/s)$	—	4	7	8	10	15	19	21	24	26

Table IV.3: Safe initial conditions for the AEBS with RL controller in which d is the distance range and v_{max} is the maximum allowable velocity such that the system is still safe.

IV.4.2 Adaptive Cruise Control System

IV.4.2.1 System Description

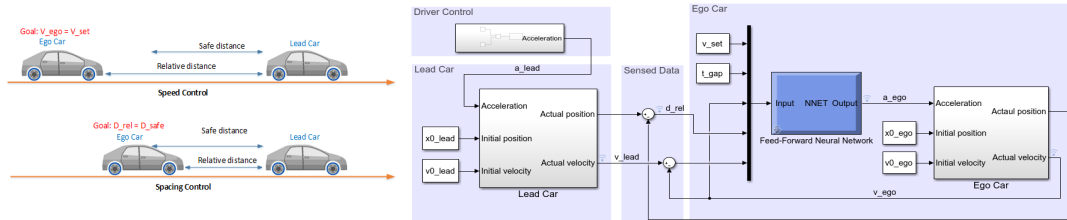


Figure IV.14: Neural network adaptive cruise control system.

The extension of the proposed star-based reachability algorithm is evaluated on the safety verification of neural network-based adaptive cruise control systems (ACC). The ACC system depicted in Figure IV.14 consists of two cars in which the ego car equipped with adaptive cruise control has a radar sensor to measure the distance to the lead car in the same lane, D_{rel} , as well as the relative velocity of the lead car, V_{rel} . In speed control mode, the ego car travels at a driver-set speed $V_{set} = 30$ while in spacing control mode, the ego car maintains a safe distance from the lead car, D_{safe} . Neural network adaptive cruise controllers with different

sizes are trained to replace the existing MPC controller. The control period is selected as 0.1 seconds. The car's dynamics are as follows.

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

where $x_{lead}(x_{ego})$, $v_{lead}(v_{ego})$ and $\gamma_{lead}(\gamma_{ego})$ are the position, velocity and acceleration of the lead (ego) car respectively, $a_{lead}(a_{ego})$ is the acceleration control input applied to the lead (ego) car, and $\mu = 0.0001$ is the friction parameter.

IV.4.2.2 Scenario of Interest

Safety-related scenario. The safety verification scenario of interest is that when the ego is in the speed control mode and the two cars are running with a safe distance between them, the lead car driver suddenly de-accelerate with $a_{lead} = -2$ to reduce the speed. We expect that the neural network controllers will also de-accelerate the ego car to remain a safe distance between two cars. Formally, the safety specification of the system is $D_{rel} = x_{lead} - x_{ego} \geq D_{safe} = D_{default} + T_{gap} \times v_{ego}$, where $T_{gap} = 1.4$ seconds and $D_{default} = 10$. We want to check if there is a collision in the next 5 seconds after the lead car de-accelerate. The initial conditions of the system are: $x_{lead}(0) \in [90, 110]$, $v_{lead}(0) \in [32, 32.2]$, $\gamma_{lead}(0) = \gamma_{ego}(0) = 0$, $v_{ego}(0) \in [30, 30.2]$, $x_{ego} \in [10, 11]$.

$x_{\text{lead}}(0)$	Controller 1 (3x20)		Controller 2 (5x20)		Controller 3 (7x20)		Controller 4(10x20)	
	Result	VT (sec)	Result	VT (sec)	Result	VT (sec)	Result	VT (sec)
[108, 110]	safe	211.84	safe	292.67	safe	398.41	safe	1762
[106, 108]	safe	210.16	safe	288.83	safe	393.35	safe	2270.3
[104, 106]	safe	211.54	safe	302.31	safe	412.81	safe	2674.5
[102, 104]	safe	215.21	safe	292.94	safe	446.47	safe	2863.8
[100, 102]	safe	222.87	safe	294.81	safe	440.94	safe	2606
[98, 100]	safe	233.02	safe	302.74	safe	491.29	uncertain	2855
[96, 98]	safe	237.12	safe	289.87	safe	515.43	uncertain	3249.9
[94, 96]	safe	238.46	safe	301.99	uncertain	571.75	uncertain	3851.5
[92, 94]	safe	259.46	safe	325.51	uncertain	598.22	uncertain	3220.2
[90, 92]	uncertain	265.29	safe	359.92	uncertain	558.65	uncertain	2336.9

Table IV.4: Verification results for the ACC system in which VT is the verification time and controller $k \times n$ means the controller has k hidden layer and n neuron per layer.

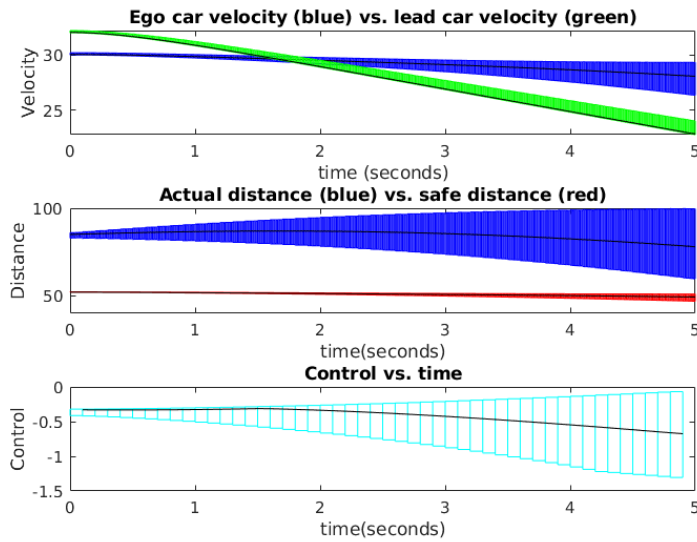


Figure IV.15: The safe distance does not intersect with the relative distance, the ACC system is safe.

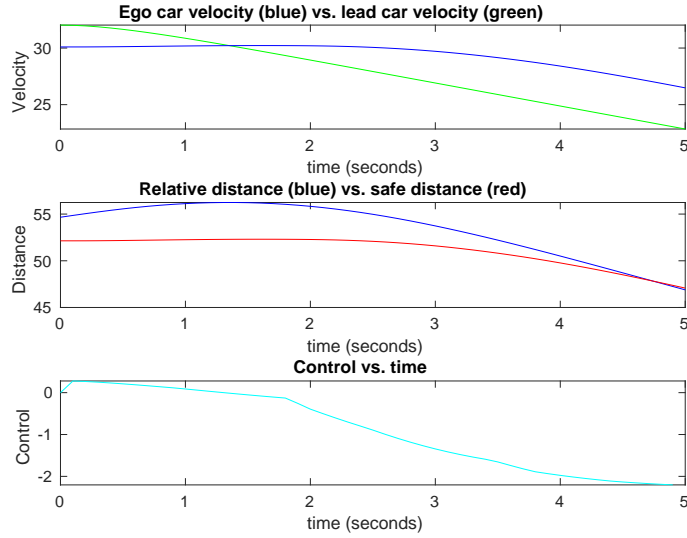


Figure IV.16: A falsification trace of the ACC system with the first controller (3x20) and $x_{lead}(0) \in [65, 70]$.

IV.4.2.3 Verification Results

Verification results. The verification results are presented in Table IV.4 which shows that the second controller is the safest controller since it guarantees the safety of the ACC system for the whole range of the lead car’s initial position. The safety of the system can be observed intuitively via Figure VII.2 which shows that the relative distance is larger than the safe distance. Interestingly, the controllers with a large number of neurons, e.g., the third and the fourth controllers, are not necessarily the good candidates for keeping the system safe. In many cases, the verification results for these controllers are uncertain which imply that these controllers may or may not control the system safely. In these cases, the relative distance reachable set intersects with the safe distance reachable set. However, we do not know this intersection is due to the over-approximation error of the related reachable sets or the relative distance is actually smaller than the required safe distance. Since the safety of the system may be violated in these cases, we further randomly generate simulation traces of the system to find counter example inputs that make the system unsafe. If counter example inputs are found, we can conclude that the system is actually unsafe. Otherwise, we can conclude nothing about the safety of the system. The falsification results for the NNACC system using 1000 random simulations are presented in Table IV.5. Interestingly, we cannot find counter examples for the system whenever $x_{lead} \in [70, 110]$. However, when $x_{lead} \in [65, 70]$, we can find counter examples of the system for all controllers. Figure IV.16 describes a counter example to prove that the NNACC system is unsafe as $x_{lead} \in [65, 70]$ in which the relative distance between two cars is smaller than the required safe distance. Note that in this case, even the controllers de-accelerates the ego car. It still can not guarantee the safety for

$\mathbf{x}_{\text{lead}}(\mathbf{0})$	Controller 1 (3 x 20)		Controller 2 (5 x 20)		Controller 3 (7 x 20)		Controller 4 (10 x 20)	
	N_c	FT (sec)	N_c	FT (sec)	N_c	FT (sec)	N_c	FT (sec)
[65, 70]	100	43.63	303	45.25	486	46.91	134	49.13
[70, 110]	0	44.14	0	45.35	0	46.82	0	49.07

Table IV.5: Falsification results for NNACC system with different neural network controllers using 1000 random simulations in which N_c is the number of counter examples and FT is the falsification time.

the system.

IV.4.2.4 Timing Performance

Timing performance. As depicted in Table IV.4, the verification time depends on the size of the controller. A controller with a large number of neurons causes a substantial verification time. Our approach can prove the safety of the NNACC system with the fourth controller having totally 200 neurons in some cases with reasonable verification times (less than 1 hour). A brief comparison with recent approaches [38, 76, 77] is given as follows. Verisig takes averagely 1690 seconds (on their personal computer) to verify a single safety property (corresponding to a single input set) in 30 time steps of the quadrotor system with 12 state variables and a neural network controller having 40 neurons while our approach spends averagely 275.68 seconds to verify a single safety property of the ACC system with 6 state variables and a neural network controller having 100 neurons in 50 time steps. The SMC-based approach can falsify safety property of neural network control system with fairly large number of neurons in the controller (22 to 182 neurons). However, in the case that there is no counter example exist, the SMC-based approach usually reaches timeout (= 1 hour). Its experimental results show that only three controllers (in 17 controllers) with 22, 32 and 82 neurons are successfully verified. An important factor making our approach potentially faster and more scalable than the Verisig and SMC-based approaches is, our approach can efficiently compute the exact reachable set of DNNs on multi-core platforms. Therefore, our verification time for neural network control system can be reduced significantly by exploiting the power of parallel computing. The new abstraction-based approach proposed recently in [76] is promisingly the fastest and the most scalable approach for safety verification of NNCS since it can compute the reachable set of NNCS with neural network controller of 500 neurons in 50 time steps with just 1081 seconds.

CHAPTER V

ImageStar-Based Reachability for Verification of Convolutional Neural Networks

V.1 Problem formulation

Reachability of CNN is the problem that given a trained CNN and some perturbed input set, we want to construct an output set containing all possible outputs of the network. In this paper, we consider the reachability of a CNN \mathcal{N} consisting of a series of layers L including the convolutional layer, fully connected layer, max-pooling layer, average pooling layer, and relu layer. Mathematically, we define a CNN with n layers as $\mathcal{N} = \{L_i\}, i = 1, 2, \dots, n$. The reachability of the CNN \mathcal{N} is defined based on the concept of *reachable sets*.

Definition 5 (Reachable set of a CNN). *An (output) reachable set $\mathcal{R}_{\mathcal{N}}$ of a CNN $\mathcal{N} = \{L_i\}, i = 1, 2, \dots, n$ corresponding to a linear input set \mathcal{I} is defined incrementally as*

$$\begin{aligned}\mathcal{R}_{L_1} &\triangleq \{y_1 \mid y_1 = L_1(x), x \in \mathcal{I}\}, \\ \mathcal{R}_{L_2} &\triangleq \{y_2 \mid y_2 = L_2(y_1), y_1 \in \mathcal{R}_{L_1}\}, \\ &\vdots \\ \mathcal{R}_{\mathcal{N}} = \mathcal{R}_{L_n} &\triangleq \{y_n \mid y_n = L_n(y_{n-1}), y_{n-1} \in \mathcal{R}_{L_{n-1}}\},\end{aligned}$$

where $L_i(\cdot)$ is a function presenting the operation corresponding to the i^{th} layer.

The definition shows that the reachable set of the CNN \mathcal{N} can be constructed *layer-by-layer*. The core computation is constructing the reachable set of each layer L_i embedded with a specific operation, i.e., convolution, affine mapping, max pooling, average pooling, ReLU.

V.2 ImageStar

Definition 6. *An ImageStar Θ is a tuple $\langle c, V, P \rangle$ where $c \in \mathbb{R}^{h \times w \times nc}$ is the anchor image, $V = \{v_1, v_2, \dots, v_m\}$ is a set of m images in $\mathbb{R}^{h \times w \times nc}$ called generator images, $P: \mathbb{R}^m \rightarrow \{\top, \perp\}$ is a predicate, and h, w, nc are the height, width and number of channels of the images respectively. The generator images are arranged to form the ImageStar's $h \times w \times nc \times m$ basis array. The set of images represented by the ImageStar is given as:*

$$\llbracket \Theta \rrbracket = \{x \mid x = c + \sum_{i=1}^m (\alpha_i v_i) \text{ such that } P(\alpha_1, \dots, \alpha_m) = \top\}.$$

$$\Theta = c + \alpha v = \begin{array}{|c|c|c|c|} \hline 0 & 4 & 1 & 2 \\ \hline 2 & 3 & 2 & 3 \\ \hline 1 & 3 & 1 & 2 \\ \hline 2 & 1 & 3 & 2 \\ \hline \end{array} + \alpha \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$c \in \mathbb{R}^{4 \times 4 \times 1}$ $v \in \mathbb{R}^{4 \times 4 \times 1}$

Figure V.1: An example of an ImageStar.

Sometimes we will refer to both the tuple Θ and the set of states $[\Theta]$ as Θ . In this work, we restrict the predicates to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq d$ where, for p linear constraints, $C \in \mathbb{R}^{p \times m}$, α is the vector of m -variables, i.e., $\alpha = [\alpha_1, \dots, \alpha_m]^T$, and $d \in \mathbb{R}^{p \times 1}$. A ImageStar is an empty set if and only if $P(\alpha)$ is empty.

Example 1 (ImageStar). A $4 \times 4 \times 1$ gray image with a bounded disturbance $b \in [-2, 2]$ applied on the pixel of the position $(1, 2, 1)$ can be described as an ImageStar depicted in Figure V.1.

Remark V.2.1. An ImageStar is an extension of the generalized star set defined in [9, 11, 85] recently. In a generalized star set, the anchor and the generators are vectors while in an ImageStar, the anchor and generators are images with multiple channels. We will show later that, an ImageStar is very efficient for reachability analysis of convolutional layers, fully connected layers and average pooling layers.

Proposition V.2.2 (Affine mapping of an ImageStar). An affine mapping of an ImageStar $\Theta = \langle c, V, P \rangle$ with a scale factor γ and an offset image β is another ImageStar $\Theta' = \langle c', V', P' \rangle$ in which the new anchor, generators and predicate are as follows.

$$c' = \gamma \times c + \beta, \quad V' = \gamma \times V, \quad P' \equiv P.$$

Note that, the scale factor γ can be a scalar or a vector containing scalar scale factors in which each factor is used to scale one channel in the ImageStar.

V.3 Reachability of CNN using ImageStars

In this section, we focus on the reachable set computation of the convolutional layer, fully connected layer, average pooling layer, max pooling layer, and ReLU layer with the input set is an ImageStar. The proofs of all lemmas in this section can be found in the appendix.

V.3.1 Reachability of a convolutional layer

We consider a two-dimensional convolutional layer with following parameters: the weights $W_{Conv2d} \in \mathbb{R}^{h_f \times w_f \times nc \times nf}$, the bias $b_{Conv2d} \in \mathbb{R}^{1 \times 1 \times nf}$, the padding size P , the stride S and the dilation factor D where h_f, w_f, nc are the

$$\begin{aligned}
\Theta = c + \alpha v &= \begin{array}{|c|c|c|c|} \hline 0 & 4 & 1 & 2 \\ \hline \times 1 & \times 1 & & \\ \hline 2 & 3 & 2 & 3 \\ \hline \times -1 & \times 0 & & \\ \hline 1 & 3 & 1 & 2 \\ \hline 2 & 1 & 3 & 2 \\ \hline \end{array} + \alpha \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 0 \\ \hline \times 1 & \times -1 & & \\ \hline 0 & 0 & 0 & 0 \\ \hline \times -1 & \times 0 & & \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix} \\
\Theta' = c' + \alpha v' &= \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 1 & -1 \\ \hline \end{array} + \alpha \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 0 \\ \hline \end{array}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix} \\
c' \in \mathbb{R}^{2 \times 2 \times 1} & \quad v' \in \mathbb{R}^{2 \times 2 \times 1}
\end{aligned}$$

Figure V.2: The reachable set of a convolutional layer with an ImageStar input set is another ImageStar.

height, width, and the number of channels of the filters in the layer respectively, and nf is the number of filters. The reachability of a convolutional layer is given in the following lemma.

Lemma V.3.1. *The reachable set of a convolutional layer with an ImageStar input set $\mathcal{S} = \langle c, V, P \rangle$ is another ImageStar $\mathcal{S}' = \langle c', V', P \rangle$ where $c' = \text{Convol}(c)$ is the convolution operation applied to the anchor image, $V' = \{v'_1, \dots, v'_m\}$, $v'_i = \text{ConvolZeroBias}(v_i)$ is the convolution operation with zero bias applied to the generator images, i.e., only using the weights of the layer.*

Example 2 (Reachable set of a convolutional layer). *The reachable set of a convolutional layer with single 2×2 filter and the ImageStar input set in Example 1 is described in Figure V.2 in which the weights and*

the bias of the filter are $W = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}$, $b = -1$ respectively, the stride is $S = [2 \ 2]$, the padding size is $P = [0 \ 0 \ 0 \ 0]$ and the dillation factor is $D = [1 \ 1]$.

V.3.2 Reachability of an average pooling layer

The reachability of an average pooling layer with the pooling size PS , the padding size P , the stride S is given below.

Lemma V.3.2. *The reachable set of a average layer with an ImageStar input set $\mathcal{S} = \langle c, V, P \rangle$ is another ImageStar $\mathcal{S}' = \langle c', V', P \rangle$ where $c' = \text{average}(c)$, $V' = \{v'_1, \dots, v'_m\}$, $v'_i = \text{average}(v_i)$, $\text{average}(\cdot)$ is the average operation applied to the anchor and generator images.*

Example 3 (Reachable set of an average pooling layer). *The reachable set of an 2×2 average pooling layer with the padding size $P = [0 \ 0 \ 0 \ 0]$, the stride $S = [2 \ 2]$ and the ImageStar input set in Example 1 is described in Figure V.3.*

$$\begin{aligned}
\Theta = c + \alpha v &= \begin{array}{|c|c|c|c|} \hline 0 & 4 & 1 & 2 \\ \hline 2 & 3 & 2 & 3 \\ \hline 1 & 3 & 1 & 2 \\ \hline 2 & 1 & 3 & 2 \\ \hline \end{array} + \alpha \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix} \\
\Theta' = c' + \alpha v' &= \begin{array}{|c|c|} \hline 2.25 & 2 \\ \hline 1.75 & 2 \\ \hline \end{array} + \alpha \begin{array}{|c|c|} \hline 0.25 & 0 \\ \hline 0 & 0 \\ \hline \end{array}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix} \\
& c' \in \mathbb{R}^{2 \times 2 \times 1} \quad v' \in \mathbb{R}^{2 \times 2 \times 1}
\end{aligned}$$

Figure V.3: The reachable set of an average pooling layer with an ImageStar input set is another ImageStar.

V.3.3 Reachability of a fully connected layer

The reachability of a fully connected layer is stated in the following lemma.

Lemma V.3.3. *Given a two-dimensional fully connected layer with the weight $W_{fc} \in \mathbb{R}^{n_{fc} \times m_{fc}}$ and the bias $b_{fc} \in \mathbb{R}^{n_{fc}}$, and an ImageStar input set $\mathcal{I} = \langle c, V, P \rangle$, the reachable set of the layer is another ImageStar $\mathcal{I}' = \langle c', V', P \rangle$ where $c' = W * \bar{c} + b$, $V' = \{v'_1, \dots, v'_m\}$, $v'_i = W_{fc} * \bar{v}_i$, $\bar{c}(\bar{v}_i) = \text{reshape}(c(v_i), [m_{fc}, 1])$. Note that it is required the consistency between the ImageStar and the weight matrix that is $m_{fc} = h \times w \times nc$, where h, w, nc are the height, width and number of channels of the ImageStar.*

V.3.4 Reachability of a batch normalization layer

In the prediction phase, a batch normalization layer normalizes each input channel x_i using the mean μ and variance σ^2 over the full training set. Then the batch normalization layer further shifts and scales the activations using the offset β and the scale factor γ that are learnable parameters. The formula for normalization is as follows.

$$\bar{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}, \quad y_i = \gamma \bar{x}_i + \beta.$$

where ε is a used to prevent division by zero. The batch normalization layer can be described as a tuple $\mathcal{B} = \langle \mu, \sigma^2, \varepsilon, \gamma, \beta \rangle$. The reachability of a batch normalization layer with an ImageStar input set is given in the following lemma.

Lemma V.3.4. *The reachable set of a batch normalization layer $\mathcal{B} = \langle \mu, \sigma^2, \varepsilon, \gamma, \beta \rangle$ with an ImageStar input set $\mathcal{I} = \langle c, V, P \rangle$ is another ImageStar $\mathcal{I}' = \langle c', V', P' \rangle$ where:*

$$c' = \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}} c + \beta - \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}} \mu, \quad V' = \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}} V, \quad P' \equiv P.$$

Proof. The reachable set of a batch normalization layer can be obtained in a straightforward fashion using

two affine mappings of the ImageStar input set. □

V.3.5 Reachability of a max pooling layer

Reachability of max pooling layer with an ImageStar input set is challenging because the value of each pixel of an image in the ImageStar depends on the predicate variables α_i . Therefore, the local max point when applying max-pooling operation may change with the values of the predicate variables. In this section, we investigate the exact reachability and over-approximate reachability of max pooling layer with an ImageStar input set. The former one obtains the exact reachable set while the later constructs an over-approximate reachable set.

V.3.5.1 Exact reachability of a max pooling layer

The central idea of the exact analysis of the max-pooling layer is finding a set of *local max point candidates* when we apply the max pooling operation on the image. We consider the max pooling operation on the ImageStar in Example 1 with the pool size of 2×2 , the padding size of $P = [0 \ 0 \ 0 \ 0]$, and the stride $S = [2 \ 2]$ to clarify the exact analysis step-by-step. First, the max-pooling operation is applied on 4 local regions I, II, III, IV , as shown in Figure V.4. The local regions II, III, IV have only one *max point candidate*, i.e., the pixel that has the maximum value in the region. Interestingly, the region I has two max point candidates of the positions $(1, 2, 1)$ and $(2, 2, 1)$ and these candidates corresponding to different conditions of the predicate variable α . For example, the pixel at the position $(1, 2, 1)$ is the max point if and only if $4 + \alpha \times 1 \geq 3 + \alpha \times 0$. Note that with $-2 \leq \alpha \leq 2$, we always have $4 + \alpha * 1 \geq 2 + \alpha \times 0 \geq 0 + \alpha \times 0$. Since the local region I has two max point candidates, and other regions have only one, the exact reachable set of the max-pooling layer is the union of two new ImageStars Θ_1 and Θ_2 . In the first reachable set Θ_1 , the max point of the region I is $(1, 2, 1)$ with an additional constraint on the predicate variable $\alpha \geq -1$. For the second reachable set Θ_2 , the max point of the region I is $(2, 2, 1)$ with an additional constraint on the predicate variable $\alpha \leq -1$. One can see that from a single ImageStar input set, the output reachable set of the max-pooling layer is split into two new ImageStars. Therefore, the number of ImageStars in the reachable set of the max-pooling layer may grow quickly if each local region has more than one max point candidates. The worst-case complexity of the number of ImageStars in the exact reachable set of the max-pooling layer is given below.

Lemma V.3.5. *The worst-case complexity of the number of ImageStars in the exact reachability of the max pooling layer is $\mathcal{O}(((p_1 \times p_2)^{h \times w})^{nc})$ where $[h, w, nc]$ is the size of the ImageStar output sets, and $[p_1, p_2]$ is the size of the max-pooling layer.*

Finding a set of local max point candidates is the core computation in the exact reachability of max-pooling layer. To optimize this computation, we divide the search for the local max point candidates into two

$$\Theta = c + \alpha v = \begin{array}{|c|c|c|c|} \hline 0 & 4 & 1 & 2 \\ \hline 2 & 3 & 2 & 3 \\ \hline 1 & 3 & 1 & 2 \\ \hline 2 & 1 & 3 & 2 \\ \hline \end{array} + \alpha \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

Region	Max point position	Max point value	Condition
II	(2, 4, 1)	$3 + \alpha * 0$	$-2 \leq \alpha \leq 2$
III	(3, 2, 1)	$3 + \alpha * 0$	$-2 \leq \alpha \leq 2$
IV	(4, 3, 1)	$3 + \alpha * 0$	$-2 \leq \alpha \leq 2$
I	(1, 2, 1)	$4 + \alpha * 1$	$-1 \leq \alpha \leq 2$
	(2, 2, 1)	$3 + \alpha * 0$	$-2 \leq \alpha \leq -1$

$$\Theta_1 = c_1 + \alpha v_1 = \begin{array}{|c|c|} \hline 4 & 3 \\ \hline 3 & 3 \\ \hline \end{array} + \alpha \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 0 \\ \hline \end{array}, P_1 \equiv \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$$

$$\Theta_2 = c_2 + \alpha v_2 = \begin{array}{|c|c|} \hline 3 & 3 \\ \hline 3 & 3 \\ \hline \end{array} + \alpha \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array}, P_2 \equiv \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}$$

Figure V.4: Exact reachable set of a max pooling layer with an ImageStar input set is a union of ImageStars.

steps. The first one is to estimate the ranges of all pixels in the ImageStar input set. We can solve $h_I \times w_I \times nc$ linear programming optimizations to find the exact ranges of these pixels, where $[h_I, w_I, nc]$ is the size of the input set. However, this is, unfortunately, a time-consuming computation. For example, **if a single linear optimization can be done in 0.01 seconds, for an ImageStar of the size $224 \times 224 \times 32$, we need about 10 hours to find the ranges of all pixels.** To overcome this bottleneck, we quickly estimate the ranges using only the ranges of the predicate variables to get rid of a vast amount of non-max-point candidates. In the second step, we solve a much smaller number of LP optimizations to determine the exact set of the local max point candidates and then construct the ImageStar output set based on these candidates.

Lemma V.3.5 shows that the number of ImageStars in the exact reachability of max-pooling layer may grow exponentially. To overcome this problem, we propose in the following an over-approximate reachability method for the max-pooling layer.

V.3.5.2 Over-approximate reachability of a max pooling layer

The central idea of the over-approximate analysis of the max-pooling layer is that if a local region has more than one max point candidates, we introduce a *new predicate variable* standing for the max point of that region. We revisit the example in the exact analysis to clarify this idea. Since the first local region *I* has two max point candidates, we introduce new predicate variable β to represent the max point of this region by adding three new constraints: 1) $\beta \geq 4 + \alpha * 1$, i.e., β must be equal or larger than the value of the

$$\Theta = c + \alpha v = \begin{bmatrix} 0 & 4 & 1 & 2 \\ 2 & 3 & 2 & 3 \\ 1 & 3 & 1 & 2 \\ 2 & 1 & 3 & 2 \end{bmatrix} + \alpha \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

Region	Max point position	Max point value	Condition
II	(2, 4, 1)	$3 + \alpha * 0$	$-2 \leq \alpha \leq 2$
III	(3, 2, 1)	$3 + \alpha * 0$	$-2 \leq \alpha \leq 2$
IV	(4, 3, 1)	$3 + \alpha * 0$	$-2 \leq \alpha \leq 2$
I	(1, 2, 1)	$4 + \alpha * 1$	$-1 \leq \alpha \leq 2$
	(2, 2, 1)	$3 + \alpha * 0$	$-2 \leq \alpha \leq -1$

$$\Theta' = c' + \alpha v_1' + \beta v_2' = \begin{bmatrix} 0 & 3 \\ 3 & 3 \end{bmatrix} + \alpha \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \beta \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

New predicate variable β

New constraints $\beta \geq 4 + \alpha * 1$
 $\beta \geq 3 + \alpha * 0$
 $\beta \leq 6$

$$\rightarrow P' \equiv \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 1 & -1 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \leq \begin{pmatrix} 2 \\ 2 \\ 1 \\ -4 \\ -3 \\ 6 \end{pmatrix}$$

Figure V.5: Over-approximate reachable set of a max pooling layer with an ImageStar input set is another ImageStar.

first candidate ; 2) $\beta \geq 3 + \alpha * 0$, i.e., β must be equal or larger than the value of the second candidate; 3) $\beta \leq 6$, i.e., β must be equal or smaller than the upper bound of the pixels values in the region. With the new predicate variable, a single over-approximate reachable set Θ' can be constructed in Figure V.5.

Lemma V.3.6. *The worst-case complexity of the new predicate variables introduced in the over-approximate analysis is $\mathcal{O}(h \times w \times nc)$ where $[h, w, nc]$ is the size of the ImageStar output set.*

V.3.6 Reachability of a ReLU layer

Similar to max-pooling layer, reachability of ReLU layer is also challenging because the value of each pixel in an ImageStar may be smaller than zero or larger than zero depending on the values of the predicate variables (with reminding that $ReLU(x) = \max(0, x)$). In this section, we investigate the exact and over-approximate reachability algorithms for a ReLU layer with an ImageStar input set. The techniques we use in this section is adapted from in [84].

V.3.6.1 Exact reachability of a ReLU layer

The central idea of the exact analysis of a ReLU layer with an ImageStar input set is performing a sequence of *stepReLU operations* over all pixels of the ImageStar input set. Mathematically, the exact reachable set of

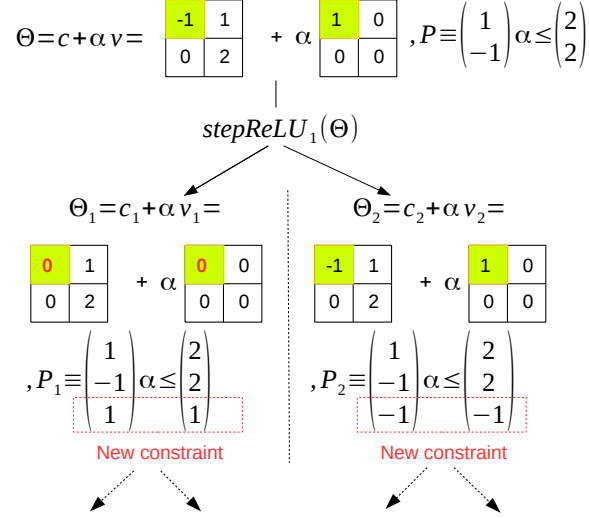


Figure V.6: stepReLU operation on an ImageStar.

a ReLU layer L can be computed as follows.

$$\mathcal{R}_L = \text{stepReLU}_N(\text{stepReLU}_{N-1}(\dots(\text{stepReLU}_1(\mathcal{I}))))),$$

where N is the total number of pixels in the ImageStar input set \mathcal{I} . The stepReLU_i operation determines whether or not a split occurs at the i^{th} pixel. If the pixel value is larger than zero, then the output value of that pixel remains the same. If the pixel value is smaller than zero then the output value of that pixel is reset to be zero. The challenge is that the pixel value depends on the predicate variables. Therefore, there is the case that the pixel value may be negative or positive with *an extra condition* on the predicate variables. In this case, we split the input set into two *intermediate* ImageStar reachable sets and apply the ReLU law on each intermediate reach set. An example of the stepReLU operation on an ImageStar is illustrated in Figure V.6. The value of the first pixel value $-1 + \alpha$ would be larger than zero if $\alpha \leq 1$, and in this case we have $\text{ReLU}(-1 + \alpha) = -1 + \alpha$. If $\alpha \leq 1$, then $\text{ReLU}(-1 + \alpha) = 0 + \alpha \times 0$. Therefore, the first stepReLU operation produces two intermediate reachable sets Θ_1 and Θ_2 , as shown in the figure. The number of ImageStars in the exact reachable set of a ReLU layer increases quickly along with the number of splits in the analysis, as stated in the following lemma.

Lemma V.3.7. *The worst-case complexity of the number of ImageStars in the exact analysis of a ReLU layer is $\mathcal{O}(2^N)$, where N is the number of pixels in the ImageStar input set.*

Similar to [84], to control the explosion in the number of ImageStars in the exact reachable set of a ReLU layer, we propose an over-approximate reachability algorithm in the following.

$$\Theta = c + \alpha v = \begin{bmatrix} -1 & 1 \\ 0 & 2 \end{bmatrix} + \alpha \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$\text{approxStepReLU}_1(\Theta)$

$$\Theta' = c' + \alpha v_1' + \beta v_2' = \begin{bmatrix} 0 & 1 \\ 0 & 2 \end{bmatrix} + \alpha \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \beta \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

New predicate variable β
New constraints: $\beta \geq 0$
 $\beta \geq x_1 \Leftrightarrow \beta \geq -1 + \alpha$
 $\beta \leq u_1(x_1 - l_1)/(u_1 - l_1)$
 $\Leftrightarrow \beta \leq (x_1 + 3)/4 = (2 + \alpha)/4$

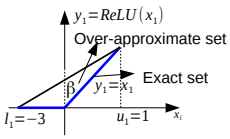
$$P' \equiv \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 1 & -1 \\ 0 & -1 \\ -0.25 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \leq \begin{pmatrix} 2 \\ 2 \\ -1 \\ 1 \\ 0 \\ 0.5 \end{pmatrix}$$


Figure V.7: approxStepReLU operation on an ImageStar.

V.3.6.2 Over-approximate reachability of a ReLU layer

The central idea of the over-approximate reachability of ReLU layer is replacing the stepReLU operation at each pixel in the ImageStar input set by an *approxStepReLU* operation. To do that, at each pixel where a split occurs, we introduce a new predicate variable to over-approximate the result of the stepReLU operation at that pixel. An example of the overStepReLU operation on an ImageStar is depicted in Figure V.7 in which the first pixel of the input set has the ranges of $[l_1 = -3, u_1 = 1]$ indicating that a split occurs at this pixel. To avoid this split, we introduce a new predicate variable β to over-approximate the exact intermediate reachable set (i.e., two blue segments in the figure) by a triangle. This triangle is determined by three constraints: 1) $\beta \geq 0$ (the $\text{ReLU}(x) \geq 0$ for any x); 2) $\beta \geq -1 + \alpha$ ($\text{ReLU}(x) \geq x$ for any x); 3) $\beta \leq 0.5 + 0.25\alpha$ (upper bound of the new predicate variable). Using this over-approximation, a single intermediate reachable set Θ' is produced as shown in the figure. After performing a sequence of approxStepReLU operations, we obtain a single over-approximate ImageStar reachable set for the ReLU layer. However, the number of predicate variables and the number of constraints in the obtained reachable set increase.

Lemma V.3.8. *In the worst case, the number of predicates variables and the number of constraints increases in the over-approximate reachability of a ReLU layer are $\mathcal{O}(N)$ and $\mathcal{O}(3 \times N)$ respectively, where N is the number of pixels in the ImageStar input set.*

One can see that determining where splits occur is crucial in the exact and over-approximate analysis of a ReLU layer. To do this, we need to know the ranges of all pixels in the ImageStar input set. However, as mentioned earlier, the computation of the exact range is expensive. To reduce the computation cost, we first use the estimated ranges of all pixels to get rid of a vast amount of non-splitting pixels. Then we compute the

exact ranges for the pixels where splits may occur to compute the exact or over-approximate reachable set of the layer.

V.4 Evaluation

The proposed reachability algorithms are implemented in NNV [86], a tool for verification of deep neural networks and learning-enabled autonomous CPS. NNV utilizes core functions in MatConvNet [89] for the analysis of the convolutional and average pooling layers. The evaluation of our approach consists of two parts. First, we evaluate our approach in comparison with the zonotope [74] and polytope methods [75] re-implemented in NNV via robustness verification of deep neural networks. Second, we evaluate the scalability of our approach and the DeepPoly polytope method using real-world image classifiers, VGG16, and VGG19 [73]. The experiments are done on a computer with following configurations: Intel Core i7-6700 CPU @ 3.4GHz \times 8 Processor, 62.8 GiB Memory, Ubuntu 18.04.1 LTS OS.¹

V.4.1 Robustness Verification of MNIST Classification Networks

We compare our approach with the zonotope and polytope methods in two aspects including verification time and conservativeness of the results. To do that, we train 3 CNNs a small, a medium, and a large CNN with 98%, 99.7% and 99.9% accuracy respectively using the MNIST data set consisting of 60000 images of handwritten digits with a resolution of 28×28 pixels [50]. The network architectures are given in the Appendix of the long version of this paper. The networks classify images into ten classes: $0, 1, \dots, 9$. The classified output is the index of the dimension that has maximum value, i.e., the argmax across the 10 outputs. We evaluate the robustness of the network under the well-known brightening attack used in [32]. The idea of a brightening attack is that we can change the value of some pixels independently in the image to make it brighter or darker to fool the network, to misclassify the image. In this case study, we darken a pixel of an image if its value x_i (between 0 and 255) is larger than a threshold d , i.e., $x_i \geq d$. Mathematically, we reduce the value of that pixel x_i to the new value x'_i such that $0 \leq x'_i \leq \delta \times x_i$.

The robustness verification is done as follows. We select 100 images that are correctly classified by the networks and perform the brightening attack on these, which are then used to evaluate the robustness of the networks. A network is robust to an input set if, for any *attacked* image, this is correctly classified by the network. We note that the input set contains an infinite number of images. Therefore, to prove the robustness of the network to the input set, we first compute the output set containing all possible output vectors of the network using reachability analysis. Then, we prove that in the output set, the correctly classified output always has the maximum value compared with other outputs. Note that we can neglect the *softmax* and

¹Codes are available online at https://github.com/verivital/nnv/tree/master/code/nnv/examples/Submission/CAV2020_ImageStar.

classoutput layers of the networks in the analysis since we only need to know the maximum output in the output set of the last fully connected layer in the networks to prove the robustness of the network.

We are interested in the percentage of the number of input sets that a network is provably robust and the verification times of different approaches under different values of d and θ . When d is small, the number of pixels in the image that are attacked is large and vice versa. For example, the average number of pixels attacked (computed on 100 cases) corresponding to $d = 250, 245$ and 240 are 15, 21 and 25 respectively. The value of δ dictates the size of the input set that can be created by a specific attack. Stated differently it dictates the range in which the value of a pixel can be changed. For example, if $d = 250$ and $\delta = 0.01$, the value of an attacked pixel may range from 0 to 2.55.

The experiments show that using the zonotope method, we cannot prove the robustness of any network. The reason is that the zonotope method obtains very conservative reachable sets. Figure V.8 illustrates the ranges of the outputs computed by our ImageStar (approximate scheme), the zonotope and polytope approaches when we attack a digit 0 image with brightening attack in which $d = 250$ and $\delta = 0.05$. One can see that, using ImageStar and polytope method, we can prove that the output corresponding to the digit 0 is the one that has a maximum value, which means that the network is robust in this case. However, the zonotope method produces very large output ranges that cannot be used to prove the robustness of the network. The figure also shows that our ImageStar method produces tighter ranges than the polytope method, which means our result is less conservative than the one obtained by the polytope method. We note that the zonotope method is very time-consuming. It needs 93 seconds to compute the reachable set of the network in this case, while the polytope method only needs 0.3 seconds, and our approximate ImageStar method needs 0.74 seconds. The main reason is that the zonotope method introduces many new variables when constructing the reachable set of the network, which results in the increase in both computation time and conservativeness.

The comparison of the polytope and our ImageStar method is given in Tables V.1, V.2, and V.3. The tables show that in all networks, our method is less conservative than the polytope approach since the number of cases that our approach can prove the robustness of the network is larger than the one proved by the polytope method. For example, for the small network, for $d = 240$ and $\delta = 0.015$, we can prove 71 cases while the polytope method can prove 65 cases. Importantly, the number of cases proved by DeepPoly reduces quickly when the network becomes larger. For example, for the case that $d = 240$ and $\delta = 0.015$, the polytope method is able to prove the robustness of the medium network for 38 cases while our approach can prove 88 cases. This is because the polytope method becomes more and more conservative when the network or the input set is large. The tables show that the polytope method is faster than our ImageStar method on the small network. However, it is slower than the ImageStar method on any larger networks in all cases. Notably, for the large network, the ImageStar approach is significantly faster than the polytope approach, 16.65 times

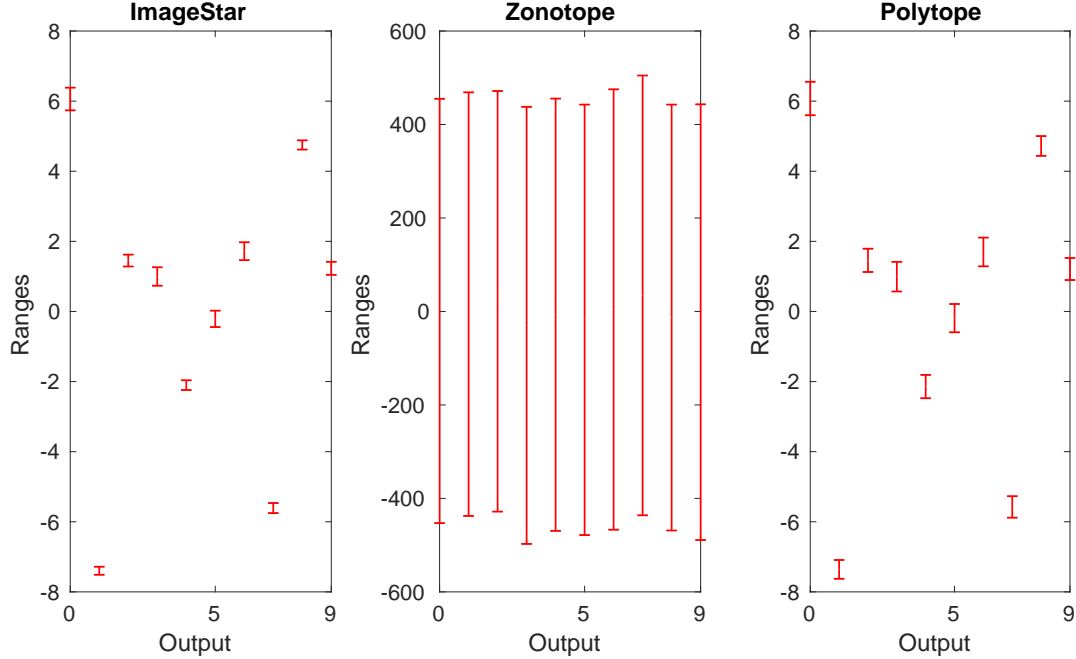


Figure V.8: An example of output ranges of the small MNIST classification networks using different approaches.

faster in average. The results also show that the polytope approach may run into memory problem for some large input sets.

V.4.2 Robustness Verification of VGG16 and VGG19

In this section, we evaluate the polytope and ImageStar methods on real-world CNNs, the VGG16 and VGG19 classification networks [73]. We use Foolbox [67] to generate the well-known DeepFool adversarial attacks [60] on a set of 20 bell pepper images. From an original image ori_im , Foolbox generates an adversarial image adv_im that can fool the network. The difference between two images is defined by $diff_im = adv_im - ori_im$. We want to verify if we apply $(l + \delta)$ percent of the attack on the original image, whether or not the network classifies the disturbed images correctly. The set of disturbed images can be represented as an ImageStar as follows $disb_im = ori_im + (l + \delta) \times diff_im$, where l is the percentage of the attack at which we want to verify the robustness of the network, and δ is a small perturbation around l , i.e., $0 \leq \delta \leq \delta_{max}$. Intuitively, l describes how close we are to the attack, and the perturbation δ represents the size of the input set.

Table V.4 shows the verification results of VGG16 and VGG19 with different levels of the DeepFool attack. The networks are robust if they classify correctly the set of disturbed images $disb_im$ as bell peppers. To guarantee the robustness of the networks, the output corresponding to the bell pepper label (index 946)

Robustness Results (in Percent)						
	$\delta = 0.005$		$\delta = 0.01$		$\delta = 0.015$	
	<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>
$d = 250$	86.00	87.00	84.00	87.00	83.00	87.00
$d = 245$	77.00	78.00	72.00	78.00	70.00	77.00
$d = 240$	72.00	73.00	67.00	72.00	65.00	71.00
Verification Times (in Seconds)						
$d = 250$	11.24	16.28	18.26	28.19	26.42	53.43
$d = 245$	14.84	19.44	24.96	40.76	38.94	85.97
$d = 240$	18.29	25.77	33.59	64.10	54.23	118.58

Table V.1: Verification results of the small MNIST CNN.

Robustness Results (in Percent)						
	$\delta = 0.005$		$\delta = 0.01$		$\delta = 0.015$	
	<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>
$d = 250$	86.00	99.00	73.00	99.00	65.00	99.00
$d = 245$	74.00	95.00	58.00	95.00	46.00	95.00
$d = 240$	69.00	90.00	49.00	89.00	38.00	88.00
Verification Times (in Seconds)						
$d = 250$	213.86	52.09	627.14	257.12	1215.86	749.41
$d = 245$	232.81	68.98	931.28	295.54	2061.98	1168.31
$d = 240$	301.58	102.61	1451.39	705.03	3148.16	2461.89

Table V.2: Verification results of the medium MNIST CNN.

Robustness Results (in Percent)						
	$\delta = 0.005$		$\delta = 0.01$		$\delta = 0.015$	
	<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>
$d = 250$	90.00	99.00	83.00	99.00	<i>MemErr</i>	99.00
$d = 245$	91.00	100.00	75.00	100.00	<i>MemErr</i>	100.00
$d = 240$	81.00	99.00	<i>MemErr</i>	99.00	<i>MemErr</i>	99.00
Verification Times (in Seconds)						
$d = 250$	917.23	67.45	5221.39	231.67	<i>MemErr</i>	488.69
$d = 245$	1420.58	104.71	6491.00	353.02	<i>MemErr</i>	1052.87
$d = 240$	1872.16	123.37	<i>MemErr</i>	476.67	<i>MemErr</i>	1522.50

Table V.3: Verification results of the large MNIST CNN.

needs to be the maximum output compared with others. The table shows that with a small input set, small δ , the polytope and ImageStar can prove the robustness of VGG16 and VGG19 with a reasonable amount of time. Notably, the verification times as well as the robustness results of the polytope and ImageStar methods are similar when they deal with small input sets except for two cases where ImageStar is faster than the polytope method. It is interesting to note that according to the verification results for the VGG and MNIST networks, deep networks seem to be more robust than shallow networks.

Robustness Results (in percentage)								
VGG16				VGG19				
$\delta = 10^{-7}$		$\delta = 2 \times 10^{-7}$		$\delta = 10^{-7}$		$\delta = 2 \times 10^{-7}$		
<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>	<i>Polytope</i>	<i>ImageStar</i>	<i>ImageStar</i>
$l = 0.96$	85.00	85.00	85.00	85.00	100.00	100.00	100.00	100.00
$l = 0.97$	85.00	85.00	85.00	85.00	100.00	100.00	100.00	100.00
$l = 0.98$	85.00	85.00	85.00	85.00	95.00	95.00	95.00	95.00
Verification Times (in Seconds)								
$l = 0.96$	319.04	318.60	327.61	319.93	320.91	314.14	885.07	339.30
$l = 0.97$	324.93	323.41	317.27	324.90	315.84	315.27	319.67	314.58
$l = 0.98$	315.54	315.26	468.59	332.92	320.53	320.44	325.92	317.95

Table V.4: Verification results of VGG networks.

V.4.3 Exact Analysis vs. Approximate Analysis

We have compared our ImageStar approximate scheme with the zonotope and polytope approximation methods. It is interesting to investigate the performance of ImageStar exact scheme in comparison with the approximate one. To illustrate the advantages and disadvantages of the exact scheme and approximate scheme, we consider the robustness verification of VGG16 and VGG19 on a single ImageStar input set created by an adversarial attack on a bell pepper image. The verification results are presented in Table V.5. The table shows that for a small perturbation δ , the exact and over-approximate analysis can prove the robustness of the VGG16 around some specific levels of attack in approximately one minute. We can intuitively verify the robustness of the VGG networks via visualization of their output ranges. An example of the output ranges of VGG19 for the case of $l = 0.95\%$, $\delta_{max} = 2 \times 10^{-7}$ is depicted in Figure V.9. One can see from the figure that the output of the index 946 corresponding to the bell pepper label is always the maximum one compared with others, which proves that VGG19 is robust in this case. From the table, it is interesting that VGG19 is not robust if we apply $\geq 98\%$ of the attack. Notably, the exact analysis can give us correct answers with a counter-example set in this case. However, the over-approximate analysis cannot prove that VGG19 is not robust since its obtained reachable set is an over-approximation of the exact one. Therefore, it may be the case that the over-approximate reachable set violates the robustness property because of its conservativeness. A counter-example generated by the exact analysis method is depicted in Figure V.10 in which the disturbed image is classified as strawberry instead of bell pepper since the strawberry output is larger than the bell

pepper output in this case.

I	δ_{\max}	VGG16				VGG19			
		Exact		Approximate		Exact		Approximate	
		Robust	VT	Robust	VT	Robust	VT	Robust	VT
50%	10^{-7}	Yes	64.56226	Yes	60.10607	Yes	234.11977	Yes	72.08723
	2×10^{-7}	Yes	63.88826	Yes	59.48936	Yes	1769.69313	Yes	196.93728
80%	10^{-7}	Yes	64.92889	Yes	60.31394	Yes	67.11730	Yes	63.33389
	2×10^{-7}	Yes	64.20910	Yes	59.77254	Yes	174.55983	Yes	200.89500
95%	10^{-7}	Yes	67.64783	Yes	59.89077	Yes	73.13642	Yes	67.56389
	2×10^{-7}	Yes	63.83538	Yes	59.23282	Yes	146.16172	Yes	121.91447
97%	10^{-7}	Yes	64.30362	Yes	59.79876	Yes	77.25398	Yes	64.43168
	2×10^{-7}	Yes	64.06285	Yes	61.23296	Yes	121.70296	Yes	107.17331
98%	10^{-7}	Yes	64.06183	Yes	59.89959	No	67.68139	Unkown	64.47035
	2×10^{-7}	Yes	64.01997	Yes	59.77469	No	205.00939	Unknown	107.42679
98.999%	10^{-7}	Yes	64.24773	Yes	60.22833	No	71.90568	Unknown	68.25916
	2×10^{-7}	Yes	63.67108	Yes	59.69298	No	106.84492	Unknown	101.04668

Table V.5: Verification results of the VGG16 and VGG19 in which *VT* is the verification time (in seconds) using the ImageStar exact and approximate schemes.

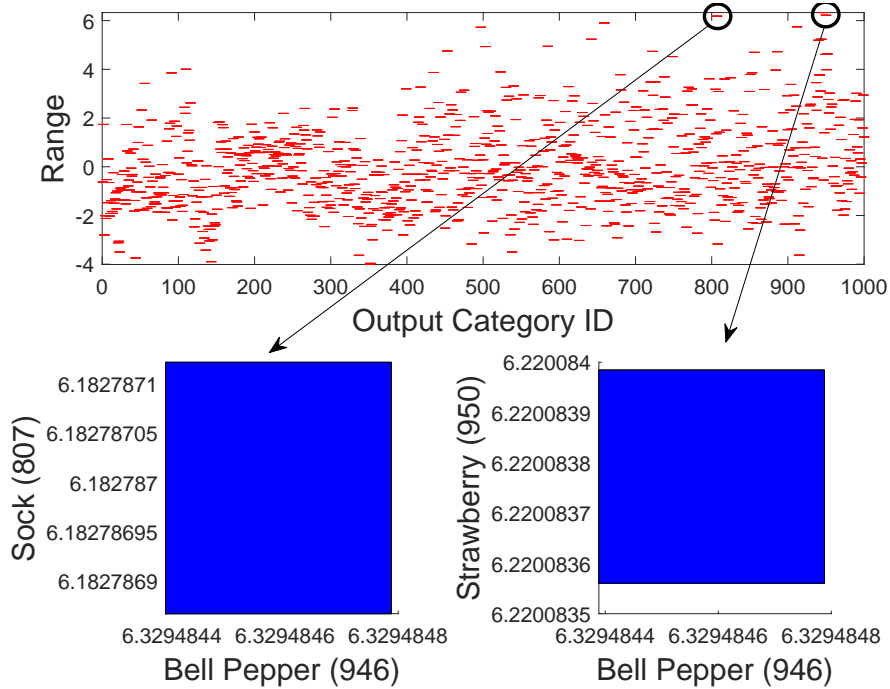


Figure V.9: Exact ranges of VGG19 shows that VGG19 correctly classifies the input image as a bell pepper.

To optimize the verification time, it is important to know the times consumed by each type of layers in the reachability analysis step. Figure V.11 described the total reachability times of the convolutional layers, fully connected layers, max pooling layers and ReLU layers in the VGG19 with 50% attack and 10^{-7} perturbation. As shown in the figure, the reachable set computation in the convolutional layers and fully connected layers

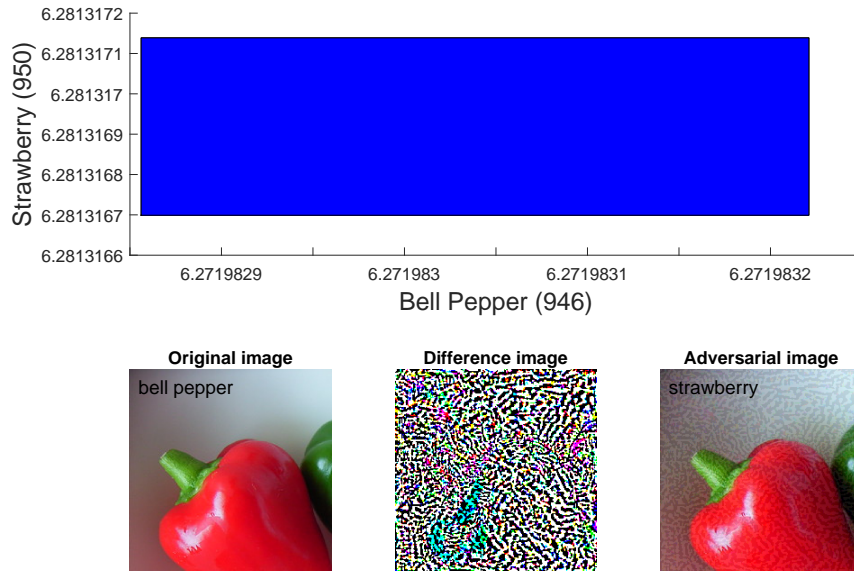


Figure V.10: A counter-example shows that VGG19 misclassifies the input image as a strawberry instead of a bell pepper.

can be done very quickly, which shows the advantages of the ImageStar data structure. Notably, the total reachability time is dominated by the time of computing the reachable set for 5 max pooling layers and 18 ReLU layers. This is because the computation in these layers concerns solving a large number of linear programming (LP) optimization problems such as finding lower bound and upper bound, and checking max point candidates. Therefore, to optimize the computation time, we need to minimize the number of LP problems in the future.

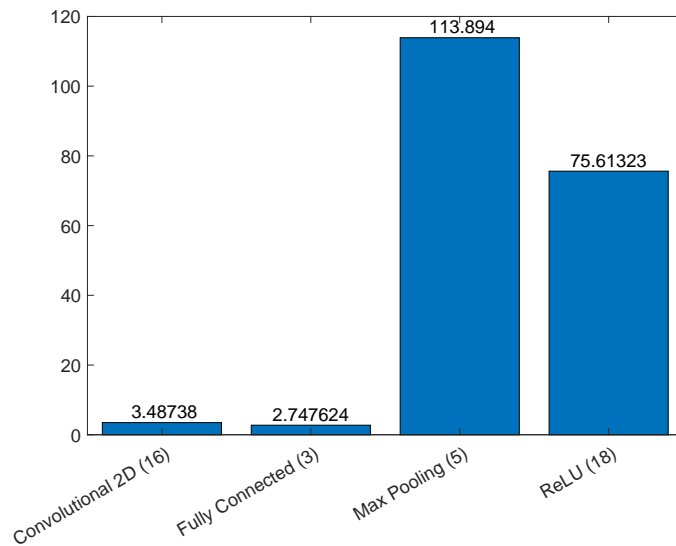


Figure V.11: Total reachability time of each type of layers in the VGG19 in which the max pooling and ReLU layers dominate the total reachability time of the network.

V.5 Discussion

V.5.1 Effect of Input Sizes on Verification Performance

When we apply our approach on real-world networks, it has been shown that **the size of the input set is the most important factor that affects the performance of verification approaches**. However, this important issue has not been emphasized in the existing literature. Most of the existing approaches focus on the size of the network that they can analyze. We believe that **all methods (including the method we proposed in this paper) are scalable for large networks only for small input sets**. When the input set is large, it causes three major problems in the analysis, which are the explosions in 1) computation time; 2) memory usage; and 3) conservativeness. In the exact analysis method, a large input set causes more splits in the max-pooling layer and the ReLU layer. A single ImageStar may split into many new ImageStars after these layers, which leads to the explosion in the number of ImageStars in the reachable set as shown in Figure V.12. Therefore, it requires more memory to handling the new ImageStars and more time for the computation. One may think that the over-approximate method can overcome this challenge since it obtains only one ImageStar at each layer and the cost we need to pay is only the conservativeness of the result. The fact is, an over-approximate method usually helps reduce the computation time, as shown in the experimental results. However, it is not necessarily efficient in terms of memory consumption. The reason is, if there is a split, it introduces a new predicate variable and new generator. If the number of generators and the dimensions of the ImageStar are large, it requires a massive amount of memory to store the over-approximate reachable set. For instance, **if there are 100 splits happened in the first ReLU layer of the VGG19, the second convolutional layer will receive an ImageStar of size $224 \times 224 \times 64$ with 100 generators. To store this ImageStar with double precision, we need approximately 2.4GB of memory**. In practice, the dimensions of the ImageStars obtained in the first several convolutional layers are usually large. Therefore, if splitting happens in these layers, we may need to deal with “out of memory” problem. We see that all existing approaches such as the zonotope [74] and polytope [75], all face the same challenges. Additionally, the conservativeness of an over-approximate reachable set is a crucial factor in evaluating an over-approximation approach. Therefore, the exact analysis still plays an essential role in the analysis of neural networks since it helps to evaluate the conservativeness of the over-approximation approaches.

V.5.2 Benefit of Parallel Computing

We can speed up the exact analysis using parallel computing. When there is more than one ImageStar in the reachable set, we can handle them independently on multiple cores machine. **Using parallel computing, the total reachability time can be reduced significantly** as shown in Figure V.13 for the VGG19 with $l = 50\%$, $\delta_{max} = 1.8 \times 10^{-7}$. To compute the reachable set of the networks with a large input set, we can

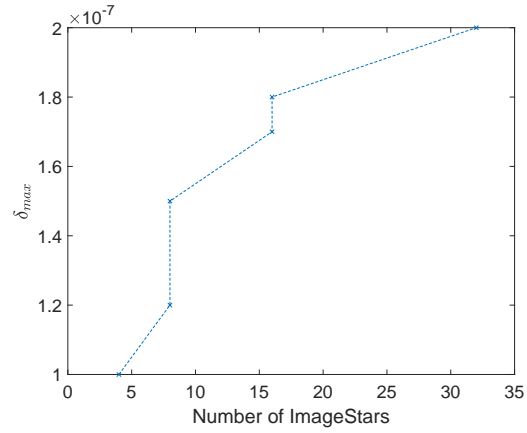


Figure V.12: Number of ImageStars in exact analysis increases with input size.

decompose the input set into a number of smaller input sets and perform the analysis in parallel on a multi-cores platform or on distributed systems.

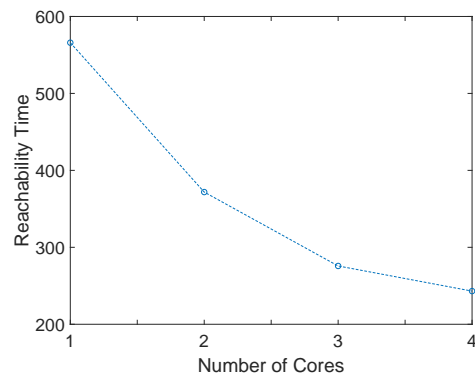


Figure V.13: The reachability time in the exact analysis reduces as the number of cores used in computation increases.

CHAPTER VI

ImageStar-Based Reachability for Verification of Semantic Segmentation Neural Networks

VI.1 Preliminaries

VI.1.1 Semantic Segmentation Networks and Reachability

Definition 7. A semantic segmentation network (SSN) f is a nonlinear function that maps each pixel x_{ij} in a multichannel input image x to a class y_{ij} from a set of classes $\mathcal{L} = \{1, 2, \dots, L\}$.

$$f : x \in \mathbb{R}^{h \times w \times c} \rightarrow y \in \mathbb{R}^{h \times w \times L} \quad (\text{VI.1})$$

where h , w , and c are the height, width, and number of channels of x (y), respectively. This definition has each output pixel y_{ij} prior to taking the softmax to yield the class of each pixel $\mathcal{L}^{h \times w}$.

Definition 8. Reachability analysis (or shortly, Reach) of a semantic segmentation network f on an ImageStar input set I is the process of computing all possible classes corresponding to every pixel in all input images x in the input set I .

$$\begin{aligned} \text{Reach}(f, I) : I &\rightarrow \mathcal{R}_f \\ x &\rightarrow y = f(x) \end{aligned} \quad (\text{VI.2})$$

We call \mathcal{R}_f the “pixel-class reachable set” of the network corresponding to the input set I in which each pixel-class $pc_{ij} \in \mathcal{R}_f$ may contain more than one class, i.e., $pc_{ij} = \{l_1, \dots, l_m\} \subseteq \mathcal{L}, m \geq 1$.

VI.1.2 Adversarial Attacks and Robustness

Definition 9. An adversarial attack adds noise x_{noise} with a small coefficient ε to the input image x to change the classification result of a network. Mathematically, an adversarial attack in this work is a linear parametrized function $g_{\varepsilon, x^{noise}}(\cdot)$ that takes an image as an input and produces the corresponding adversarial image.

$$x^{adv} = g_{\varepsilon, x^{noise}}(x) = x + \varepsilon \times x^{noise} \quad (\text{VI.3})$$

Developing new adversarial attack methods (i.e., obtaining x^{noise} and ε coefficient) has been an active research topic in machine learning for several years. Many powerful techniques have been proposed recently, such as the fast gradient sign method (FGSM) [33], DeepFool [60], etc. At the same time, defending against adversarial attacks is an active research area [54, 95]. In this paper, we focus on the robustness analysis of

semantic segmentation networks under adversarial attacks. We refer readers to [104] for a survey of state-of-the-art approaches for attacks and defences.

Definition 10. An unknown, bounded adversarial attack (UBAA) is an adversarial attack in which the value of the coefficient ε is unknown but bounded in a range $[\underline{\varepsilon}, \bar{\varepsilon}]$. An UBAA can be defined formally as a tuple $\mathcal{A} = \langle \underline{\varepsilon}, \bar{\varepsilon}, x^{noise} \rangle$.

Proposition VI.1.1 (UBAA as an ImageStar). Applying an UBAA $\mathcal{A} = \langle \underline{\varepsilon}, \bar{\varepsilon}, x^{noise} \rangle$ on an image x creates a set of images, which can be represented as an ImageStar $I = \langle c \equiv x, V \equiv x^{noise}, P(\alpha) \equiv P(\varepsilon) \equiv \underline{\varepsilon} \leq \varepsilon \leq \bar{\varepsilon} \rangle$.

Definition 11. Given an SSN f and an input image x , a pixel $x_{ij} \in x$ is called robust to an UBAA \mathcal{A} (or robust in short) if and only if: $\forall g_{\varepsilon, x^{noise}} \in \mathcal{A}, f(x_{ij}^{adv}) = f(x_{ij})$, where $x_{ij}^{adv} \in x^{adv} = g_{\varepsilon, x^{noise}}(x)$. If $\exists g_{\varepsilon, x^{noise}} \in \mathcal{A}$ such that $f(x_{ij}^{adv}) \neq f(x_{ij})$, the pixel x_{ij} is called unrobust.

We note that a pixel x_{ij} is attacked by the UBAA if the corresponding noise to the pixel is not zero, i.e., $x_{ij}^{noise} \neq 0$.

Definition 12. The robustness value (RV) of an SSN corresponding to an UBAA applied to an input image is defined as

$$RV = \frac{N_{robust}}{N_{pixels}} \times 100\%. \quad (\text{VI.4})$$

where N_{robust} is the total number of robust pixels under the attack, and N_{pixels} is the total number of pixels of the input image.

Definition 13. The robustness sensitivity (RS) of an SSN corresponding to an UBAA applied to an input image is defined as

$$RS = \frac{N_{unrobust} + N_{unknown}}{N_{attacked\ pixels}}. \quad (\text{VI.5})$$

where $N_{unrobust}$ is the total number of unrobust pixels under the attack, $N_{unknown}$ is the total number of pixels whose the robustness are unknown (may or may not robust), and $N_{attacked\ pixels}$ is the total number of attacked pixels of the input image.

While the robustness value quantifies the robustness of the network under an unknown bounded adversarial attack, the robustness sensitivity illustrates how significant the network output may change under the attack, i.e., if one pixel in the input image is attacked, how many pixels (on average) at the segmentation output image are influenced (unrobust or unknown).

VI.2 Verification

We consider two robustness verification problems.

Problem VI.2.1. *Given an SSN f , an image x and an UBAA \mathcal{A} , prove for every pixel $x_{ij} \in x$ that x_{ij} is robust or unrobust to the attack \mathcal{A} .*

Problem VI.2.2. *Given an SSN f , a set of N test images $\{x_1, \dots, x_N\}$ and an UBAA \mathcal{A} , compute the average robustness value \overline{RV} and the average robustness sensitivity \overline{RS} of the network (corresponding to \mathcal{A}).*

In this section, we investigate a reachability-based algorithm to prove the robustness of an SSN f under an UBAA \mathcal{A} at the pixel-level, i.e., Problem VI.2.1. The core of our approach is the computation of the pixel-class reachable set $\mathcal{R}_f = \text{Reach}(f, I)$ that contains all possible classes of every pixel in the input set I constructed by applying the attack \mathcal{A} on an image x (Proposition VI.1.1). The pixel-class reachable set is computed by propagating the ImageStar input set through the layers of the network.

We analyze the robustness of an SSN that is composed of the following layers: convolution, max-pooling, average-pooling, batch normalization, ReLU, transposed convolution, dilated convolution, softmax, and pixel-classification. Reachability of the convolution, max-pooling, average-pooling, batch normalization, and ReLU layers has been developed previously [80]. In this section, we develop reachability techniques for the up-sampling layers, including transposed convolution, dilated convolution and pixel-classification. We note that the softmax layer can be neglected in the analysis [80].

VI.2.1 Reachability of a Transposed (Dilated) Convolutional Layer

Lemma VI.2.3. *The reachable set of a transposed (dilated) convolutional layer with an ImageStar input set $\mathcal{I} = \langle c, V, P \rangle$ is another ImageStar $\mathcal{I}' = \langle c', V', P \rangle$ where $c' = \text{Trans}(\text{Dil})\text{Conv}(c)$ is the transposed (dilated) convolution operation applied to the anchor image, $V' = \{v'_1, \dots, v'_m\}$, $v'_i = \text{Trans}(\text{Dil})\text{ConvZeroBias}(v_i)$ is the transposed (dilated) convolution operation with zero bias applied to the generator images, i.e., only using the weights of the layer.*

Proof. Similar to the reachability of a convolutional layer [80], the transposed (dilated) convolution operation applied to an ImageStar is the combination of 1) the transposed (dilated) convolution operation with the bias on the anchor image, and 2) the transposed (dilated) convolution operation with zero bias on the basis images. This combination results in a new ImageStar output set. The convenience in computing the output set of a transposed convolutional layer comes from the linearity of the input set and operation itself. \square

VI.2.2 Reachability of a Pixel-classification Layer

The last layer in an SSN f is a pixel-classification layer, which assigns a specific class (label) to each pixel of an input image. Given an $h \times w \times nc$ input image, the size of the input x to the pixel-classification layer is $h \times w \times N$, where N is the number of classes (labels) of the network (we can neglect softmax layer in the

analysis). To assign a specific class l , $1 \leq l \leq N$ to a pixel $p_{ij} \in x$, $1 \leq i \leq h$, $1 \leq j \leq w$, the value of the pixel p_{ij} at channel l , i.e., $x(i, j, l)$, needs to be the maximum one among N channels. When the input to the network is an ImageStar set instead of a single image, the input to the pixel-classification layer is a $h \times w \times N$ ImageStar set. Depending on the value of the predicate variables in the input set, a pixel p_{ij} in the set may be assigned to more than one classes. For example, if l_1, \dots, l_m are the cross-channel max-point candidates of the pixel p_{ij} in N channels, the pixel-class reachable set of the layer at the considered pixel is $pc_{ij} = \{l_1, \dots, l_m\}$. By determining all cross-channel max-point candidates of all pixels in the input set, we can obtain the pixel-class reachable set of the layer, which is also the reachable set of the network $\mathcal{R}_f = [pc_{ij}]_{h \times w}$.

Similar to the max-pooling layer [80], determining all cross-channel max-point candidates of all pixels in the input set can be done via solving linear programming (LP) optimization problems, which is time-consuming due to the number of LPs required (or equivalently the size of the LP). To reduce computation time, we estimate the lower and upper bounds of the ImageStar input to the layer using only the ranges of the predicate variables. These bounds are then used to predict all possible cross-channel max-point candidates of all pixels .

VI.2.3 Verification Algorithm

Our reachability-based verification algorithm for an SSN is presented in Algorithm 7. The algorithm takes a network f , an input image x , an UBAA \mathcal{A} , and a reachability method (exact or approximate) as inputs and returns the pixel-class reachable set \mathcal{R}_f , the robustness value RV , and sensitivity RS of the network. The algorithm works as follows. First, it constructs the input set corresponding to the attack using Proposition VI.1.1 (line 2). Then, it computes the pixel-class reachable set of the network using reachability analysis layer-by-layer (line 3). Using the pixel-class reachable set, it verifies the robustness of each pixel in the reachable set by comparing its classes with the non-attacked output segmentation image, i.e., $y = f(x)$. If $\mathcal{R}_f(i, j) = y(i, j)$, the pixel $p_{i,j}$ is robust under the attack (line 10). If $\mathcal{R}_f(i, j) \neq y(i, j) \wedge y(i, j) \notin \mathcal{R}_f(i, j)$, the pixel $p_{i,j}$ is unrobust under the attack (line 12). Otherwise, the robustness of the pixel $p_{i,j}$ is *unknown* (may be robust or unrobust), due to overapproximation. Beyond verifying the robustness of each pixel in the reachable set, it also counts the numbers of 1) robust pixels N_{robust} (line 10), 2) unrobust pixels $N_{unrobust}$ (line 12), and 3) pixels with unknown robustness $N_{unknown}$ (line 13). Finally, it computes the robustness value and sensitivity of the network (line 12 and 13).

Average Robustness Value and Sensitivity. The robustness of a network under an UBAA should be evaluated on a set of test images (Problem VI.2.2). Suppose we have a test set of N images $X = \{x_1, \dots, x_N\}$ that we want to estimate the average robustness value \overline{RV} and sensitivity \overline{RS} over for the network. This can be done by computing the robustness value RV_k and sensitivity RS_k on each image x_k using Algorithm 7 and

Algorithm 7 Robustness verification of a semantic segmentation network.

Input: $f, x, \mathcal{A}, method$ \triangleright network, input image, attack, reachability method
Output: \mathcal{R}_f, RV, RS \triangleright pixel-class reachable set, robustness value, robustness sensitivity

```

1: procedure  $[ \mathcal{R}_f, RV, RS ] = \text{VERIFY}(f, x, \mathcal{A}, method)$ 
2:    $I = \text{constructInputSet}(x, \mathcal{A})$   $\triangleright$  construct an ImageStar input set
3:    $\mathcal{R}_f = \text{Reach}(f, I, method)$   $\triangleright$  compute the pixel-class reachable set
4:    $y = f(x)$   $\triangleright$  compute non-attacked output segmentation image
5:    $H = x.Height, W = x.Width$ 
6:    $N_{robust} = 0, N_{unrobust} = 0, N_{unknown} = 0, N_{attackedpixels} = 0$ 
7:   for  $i = 1 : H$  do
8:     for  $i = 1 : W$  do
9:       if  $\mathcal{A}.x^{noise}(i, j) \neq 0$  then  $N_{attackedpixels} = N_{attackedpixels} + 1$ 
10:      if  $\mathcal{R}_f(i, j) = y(i, j)$  then  $N_{robust} = N_{robust} + 1$   $\triangleright$  the pixel  $x_{ij}$  is robust under the attack
11:      else
12:        if  $y(i, j) \notin \mathcal{R}_f(i, j)$  then  $N_{unrobust} = N_{unrobust} + 1$   $\triangleright$  the pixel  $x_{ij}$  is unrobust
13:        else  $N_{unknown} = N_{unknown} + 1$   $\triangleright$  the pixel  $x_{ij}$  robustness is unknown
14:    $RV = (N_{robust} / (H \times W)) \times 100\%$   $\triangleright$  robustness value
15:    $RS = (N_{unrobust} + N_{unknown}) / N_{attackedpixels}$   $\triangleright$  robustness sensitivity

```

then taking the average of these values.

VI.3 Evaluation

Experimental Setup.

The approach is implemented in NNV, a software tool for verification of deep neural networks and learning-enable cyber-physical systems [86] by Tran et al. We evaluate our approach by verifying the robustness of a set of deep semantic segmentation networks trained on the MNIST and M2NIST datasets shown in Table VI.1. All networks are trained using Matlab Deep Learning Toolbox. The experiments are performed on a personal computer with the following configuration: Intel Core i7-8850H CPU @ 2.6GHz 8 core Processor, 32 GB Memory, and 64-bit Windows 10 OS. The overapproximate reachability method and 6 cores are used for computing pixel-class reachable set of all networks.

We randomly selected 100 MNIST images (of the size 28×28) and 100 M2NIST images to evaluate

ID	Name	Accuracy(IoU)	Down-sampling	Up-sampling	Input size	Layers
N_1	<i>mnist_ap_tc</i>	0.87	C+AP	TC	28×28	21 (1I, 7C, 3R, 4B, 2AP , 2TC , 1S, 1L)
N_2	<i>mnist_mp_tc</i>	0.85	C+MP	TC	28×28	21 (1I, 7C, 3R, 4B, 2MP , 2TC , 1S, 1L)
N_3	<i>mnist_dc</i>	0.83	C	DC	28×28	21 (1I, 3C, 3R, 3B, 9DC , 1S, 1L)
N_4	<i>m2nist_ap_dc</i>	0.62	C+AP	DC	64×84	16 (1I, 4C, 3R, 3AP , 3DC , 1S, 1L)
N_5	<i>m2nist_mp_dc</i>	0.75	C+AP	TC	64×84	22 (1I, 7C, 8R, 2AP , 2TC , 1S, 1L)
N_6	<i>m2nist_dc</i>	0.72	C	DC	64×84	24 (1I, 1C, 5R, 5B, 10DC , 1S, 1L)

Table VI.1: Semantic Segmentation Network Benchmarks. Notation: ‘I’: input, ‘C’: convolution, ‘TC’: transposed convolution, ‘DC’: dilated convolution, ‘R’: ReLU, ‘B’: batch normalization, ‘AP’: average-pooling, ‘MP’: max-pooling, ‘S’: softmax, ‘L’: label (pixel classification)

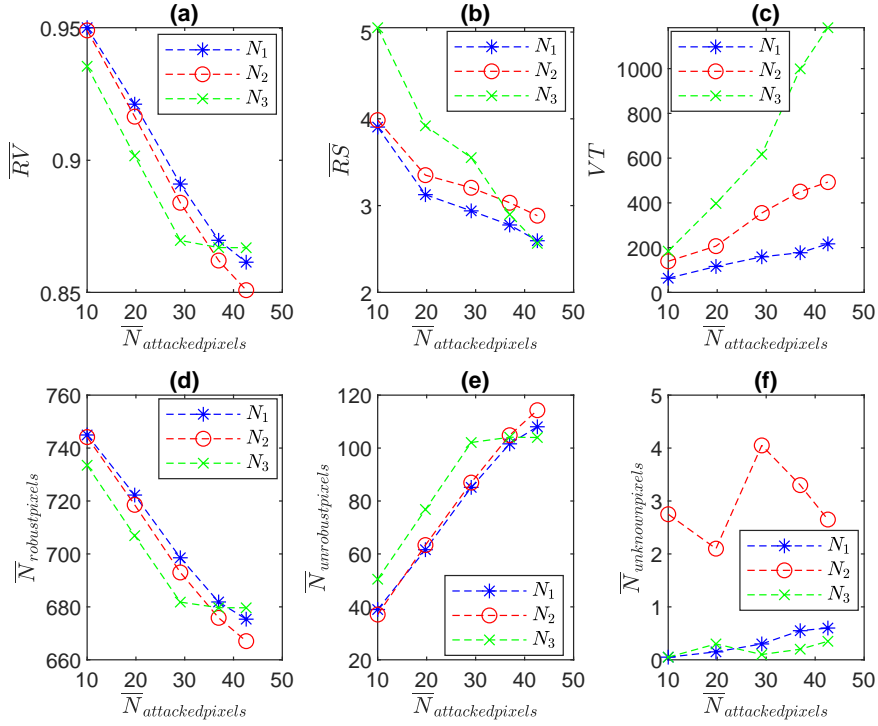


Figure VI.1: MNIST networks: $\overline{RV}, \overline{RS}, VT$ vs. $\overline{N}_{attackedPixels}$ ($\Delta_\epsilon = 0.001$).

the robustness of the trained networks. We attack each image x in two test sets using brightening attack [80]. Particularly, we darken a pixel $x(i, j)$ in the image if its value is larger than a threshold d , i.e. if $x(i, j) > d \rightarrow x^{adv}(i, j) = a \ll d$. Mathematically, the adversarial darkening attack on the image x can be described as:

$$x^{adv} = x + \epsilon \times x^{noise}, \quad 1 - \Delta_\epsilon \leq \epsilon \leq 1,$$

$$x^{noise}(i, j) = -x(i, j), \text{ if } x(i, j) > d, \text{ otherwise } x^{noise}(i, j) = 0.$$

One can see that for $\epsilon = 1$, we completely darken all the pixels whose values are larger than d ($= 150$ in our experiments), i.e., $x^{adv}(i, j) = 0$. The size of the input set caused by the attack is defined by Δ_ϵ . Generally, we have a large input set when Δ_ϵ is large. To evaluate the average robustness values (\overline{RV}) and sensitivities (\overline{RS}) of the networks (on the test sets) in the connection with the number of attacked pixels, we future restrict the maximum allowable number of attacked pixels by N_{max} .

VI.3.1 Robustness and Sensitivity of Different Network Architectures

Max-pooling vs. average-pooling. Max-pooling is a preferred choice for training deep neural networks compared with average-pooling because of its nonlinear characteristics. We investigate whether max-pooling

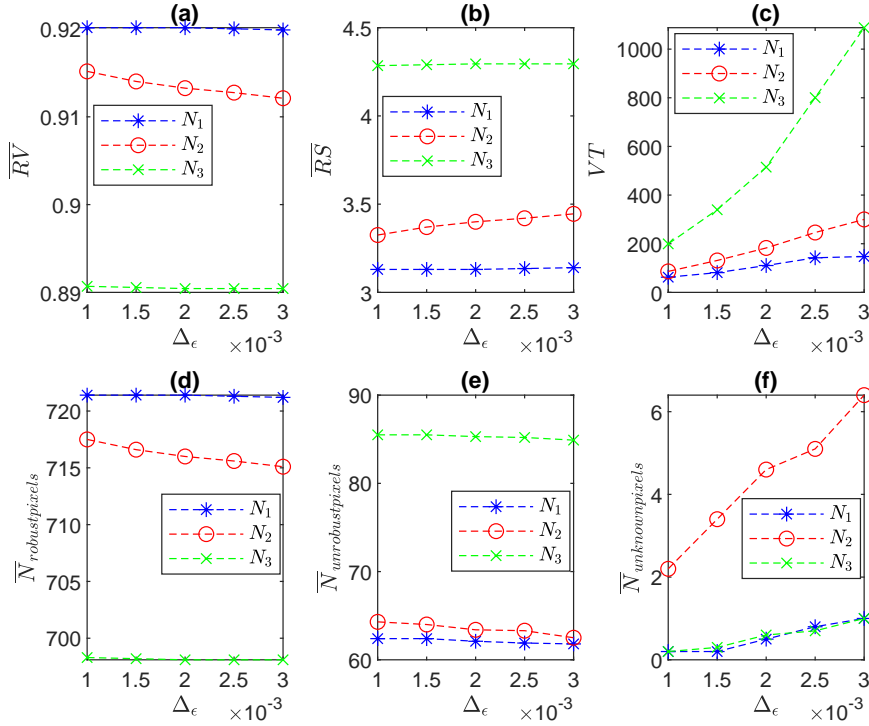


Figure VI.2: MNIST networks: \overline{RV} , \overline{RS} , VT vs. Δ_ϵ ($N_{max} = 20$).

is actually better than average-pooling in terms of accuracy and robustness of deep SSN. Figures VI.1, VI.2 illustrate the average robustness and sensitivities of MNIST networks under different number of attacked pixels (Figure VI.1, 20 images are used) and input sizes (Figure VI.2, 10 images are used). We focus on the first two networks, i.e. N_1 and N_2 . These networks have the same architectures (with 21 layers). The only difference is N_1 uses average-pooling for down-sampling while N_2 uses max-pooling for the same task (both networks use two transposed convolutional layers for up-sampling). With training, we experienced that N_1 is more accurate than N_2 , (0.87 IoU vs. 0.85 IoU, see Table VI.1). Interestingly, N_1 is also more robust than N_2 since it has a larger average robustness value (Figure VI.1 -a) and more robust pixels (Figure VI.1 -d). One can also see that the average-pooling-based network is less sensitive to the attack than the max-pooling-based network (Figure VI.1 -(b, e, f)). Notably, when more pixels are attacked or larger input sizes are used, the max-pooling-based network (i.e., N_2) produces more pixels with unknown robustness (Figures VI.1 -f, VI.2 -f, VI.3, VI.4). Lastly, when the input size increases, the robustness of the max-pooling-based network drops more quickly than the average-pooling-based networks (Figure VI.2 (a,d)) and its sensitivity increases faster (Figure VI.2 -b). We believe that the main reason causing the max-pooling-based network more sensitive to the attack is its high nonlinearity due to using max-pooling layers.

Accuracy vs. robustness; deeper network and ReLU layer robustness. Accuracy is one of the most



Figure VI.3: Example of $R_f(N_1), N_{unknown} = 6$ ($N_{max} = 50, \Delta_\epsilon = 0.003$).

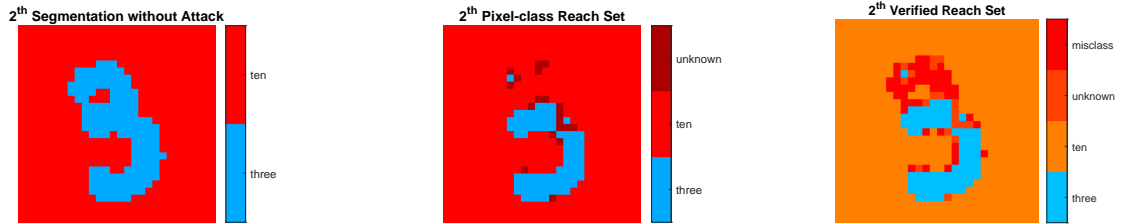


Figure VI.4: Example $R_f(N_2), N_{unknown} = 19$ ($N_{max} = 50, \Delta_\epsilon = 0.003$).

important factors for evaluating deep neural networks. We investigate whether more accurate and deeper networks are more robust compared to other architectures. To determine this, we analyze the robustness of two networks with different architectures and accuracy trained on M2NIST data set. The first network N_4 is based on dilated convolution with 16 layers and 0.62 (IoU) accuracy (Table VI.1). The second network N_5 is based on transposed convolution with 22 layers and 0.75 (IoU) accuracy. Here, the second network is deeper and more accurate than the first network. We run the robustness analysis on these two networks on a set of 20 M2NIST images. The results are depicted in Figure VI.5. In terms of robustness, the more accurate and deeper network N_5 is worse than the less accurate one N_4 (Figures VI.5 -(a,d), VI.7, and VI.8) when the number of attacked pixels is increases. Additionally, N_5 is also more sensitive to the attack than N_4 (Figure VI.5 -(b,e)). The main reason for this result is, the more accurate network contains many ReLU layers (8 ReLU layers) compared with the less accurate one (3 ReLU layers). Similar to the max-pooling layer, using many ReLU layers increases the nonlinearity of the network to capture complex features of images. Unfortunately, it also makes the network more sensitive to the attack.

Dilated convolution vs. Transposed convolution. Dilated convolution and transposed convolution are typical choices for semantic segmentation tasks. We compare these techniques in terms of accuracy and robustness. On MNIST networks, although the transposed-convolution networks N_1, N_2 and the dilated-convolution network N_3 have the same number of layers (21 layers with 3 ReLU), N_3 is less accurate than N_1 and N_2 (0.83 vs. 0.87 and 0.85 IoU, see Table VI.1). In terms of robustness, N_3 is less robust and more sensitive to the attack than N_1 and N_2 when the number of attacked pixels is smaller than 40 (Figure VI.1-(a,b,d,e)). When the input size increases, the dilated-convolution network is less robust and more sensitive to

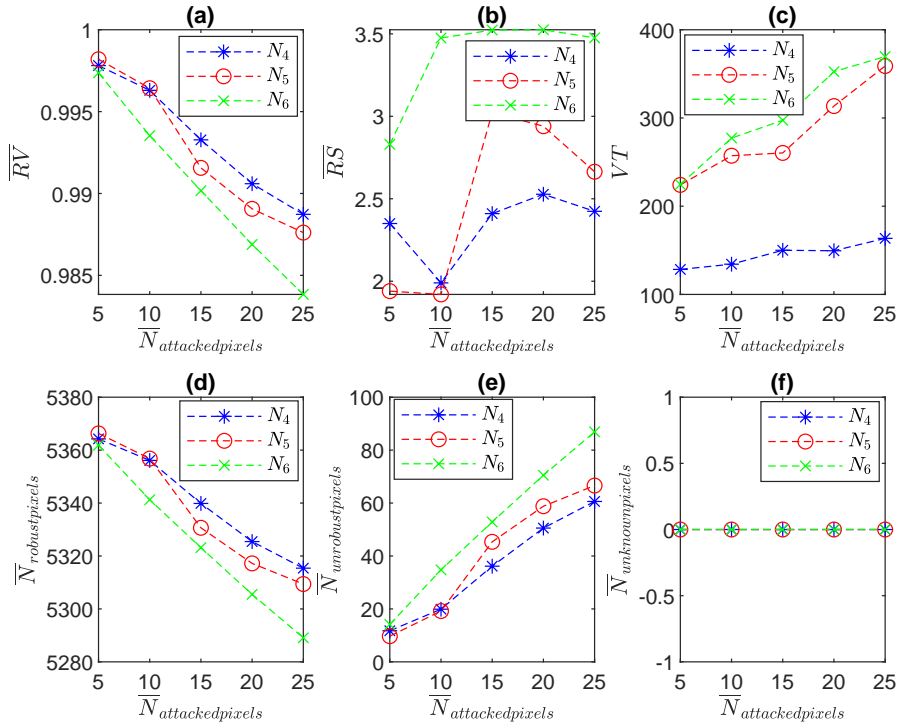


Figure VI.5: M2NIST networks: $\overline{RV}, \overline{RS}, VT$ vs. $\bar{N}_{attackedPixels}$ ($\Delta_{\epsilon} = 10^{-5}$).

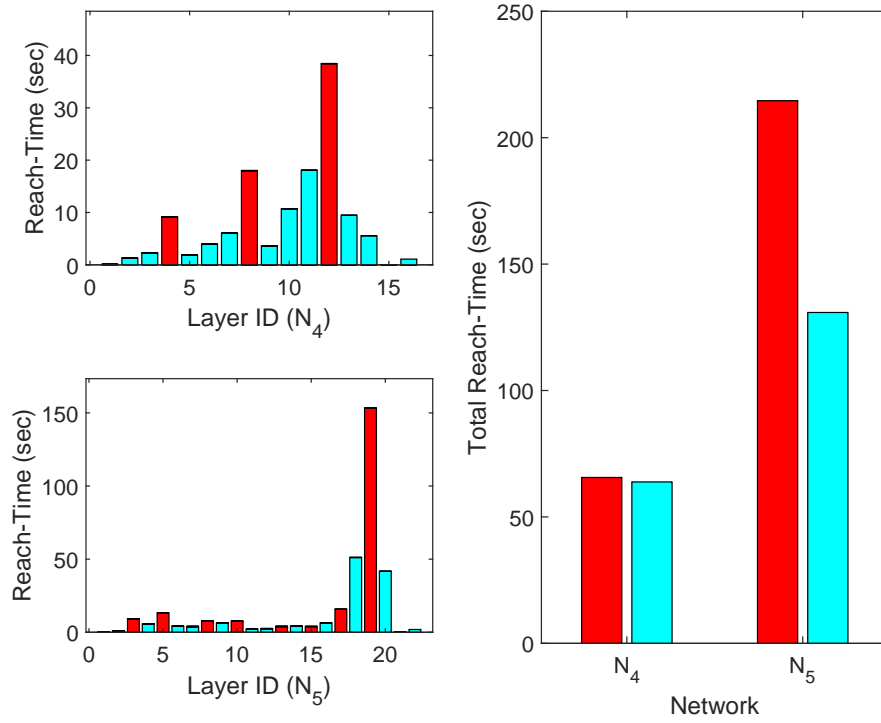


Figure VI.6: Reach-Times of M2NIST networks ($N_{max} = 25, \Delta_{\epsilon} = 10^{-5}$).

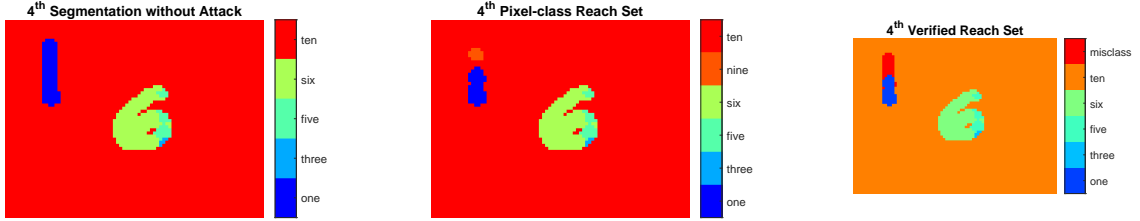


Figure VI.7: Example of $R_f(N_4), N_{unrobust} = 43$ ($N_{max} = 25, \Delta_\epsilon = 0.00001$).

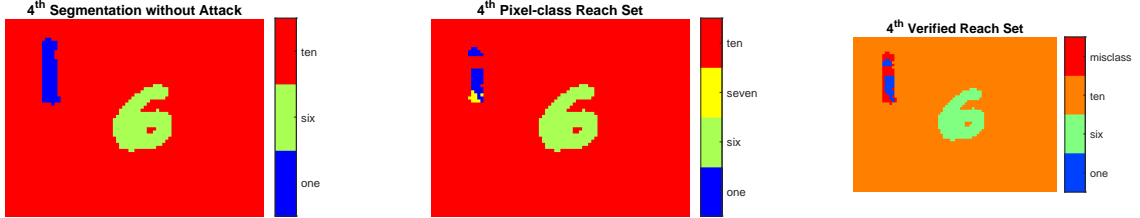


Figure VI.8: Example of $R_f(N_5), N_{unrobust} = 51$ ($N_{max} = 25, \Delta_\epsilon = 0.00001$).

the attack than the transposed-convolution networks (Figure VI.2-(a,b,d,e)). On M2NIST networks, by considering 21-layer (8 ReLU) transposed-convolution network N_5 and 24-layer (4 ReLU) dilated-convolution network N_6 , one can see that even using more layers, N_6 is less accurate than N_5 (0.72 vs. 0.75 IoU, see Table VI.1). In terms of robustness, N_6 is also less robust and more sensitive to the attack than N_5 (Figure VI.5-(a,b,d,e)).

VI.3.2 Verification Performance

More max-pooling and ReLU layers, more number of attacked pixels and large input size lead to greater verification time and memory consumption. Using max-pooling layer for down-sampling not only decreases the robustness of an SSN but also causes a dramatically increase in time and memory consumption in verification. Figures VI.1-c and VI.2-c show that the verification time (in seconds) of the max-pooling-based network N_2 grows significantly compared with the average-pooling-based network N_1 when increasing the number of attacked pixels $N_{attacked\ pixels}$ or the input size Δ_ϵ . When dealing with more number of attacked pixels or larger input size, the max-pooling layer introduces more predicate variables to over-approximate the reachable set which causes the increase both in computation time and memory usage [80]. Similar to max-pooling layer, ReLU layer is also a main source of robustness degradation. Additionally, it also dominates the reachability time of a network. Figure VI.6 shows that 3 ReLU layers constitutes 50.68% the total reachability time of 16-layer network N_4 and 8 ReLU layers constitutes 62.12% the total reachability time of 22-layer network N_5 .

Dilated convolution vs. transposed convolution. Figures VI.1-c, VI.2-c and VI.5-c consistently show

that the dilated-convolution-based networks requires more verification time than the ones using transposed convolution even in the case that they have the same or less number of ReLU layers (i.e., N_3 vs. N_1, N_2 , or N_6 vs. N_5).

VI.4 Training robust and verification-friendly networks.

Robust deep learning is an active area of research. Our analysis on robustness and sensitivity of SSNs can benefit robust deep learning methods by carefully choosing appropriate layers and architectures prior to training. As shown in this paper for semantic segmentation tasks, average-pooling seems to be better than max-pooling in both accuracy and robustness. In addition, it is also easier to verify. Importantly, since ReLU layers are crucial in training, users can minimize the number of ReLU layers used in training to enhance the robustness of the network, as well as easing the verification task. From our analysis, SSNs using average-pooling for down-sampling and transposed convolution for up-sampling seem to be better than dilated-convolution-based SSNs in both accuracy and robustness. Training accurate and robust deep neural networks requires knowledge about different layers and architecture characteristics. Generally, increasing nonlinearity of a network, e.g. by adding max-pooling and ReLU layers, may allow it to learn complex features with high accuracy, but may simultaneously result in a less robust network vulnerable to adversarial attacks. Therefore, optimizing the nonlinearity and the depth of a network in training is crucial to achieve simultaneously high accuracy and robust network that is amenable to verification.

CHAPTER VII

NNV: Neural Network Verification Tool

VII.1 Overview and Features

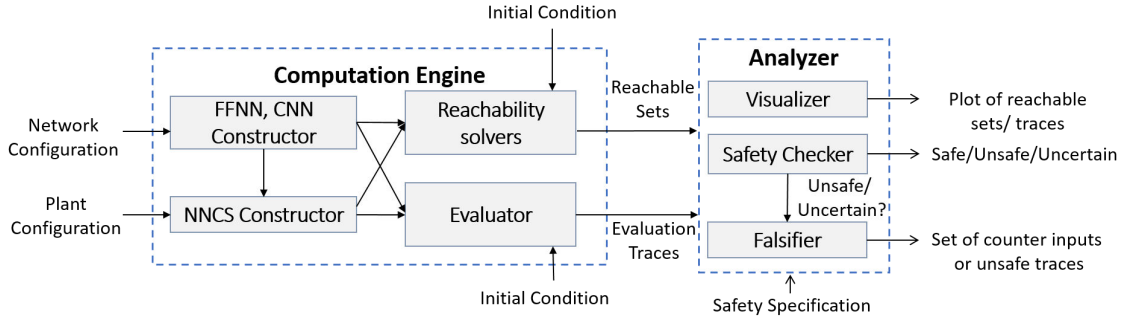


Figure VII.1: An overview of NNV.

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. It uses the MPT toolbox [48] for polytope-based reachability analysis and visualization [83], and makes use of CORA [5] for zonotope-based reachability analysis of nonlinear plant models [81]. 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) [8] for plant configuration.

The NNV toolbox contains two main modules: a *computation engine* and an *analyzer*, shown in Figure VII.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.

Feature	Exact Analysis	Over-approximate Analysis
Components	FFNN, CNN, SEGNET, NNCS	FFNN, CNN, SEGNET, 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, TConv, MaxPool, BN, AvgPool, FC	MaxPool, Conv, TConv, 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

Table VII.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 (including dilated convolution), TConv to transposed convolutional layers and MaxPool to max pooling layers.

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. A summary of NNV's major features is given in Table VII.1.

VII.2 Set Representations and Reachability Algorithms

NNV implements a set of reachability algorithms for *sequential* FFNNs and CNNs, as well as NNCS with FFNN controllers. 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.

VII.2.1 Polyhedron [83]

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., $R_L = stepReLU_n(stepReLU_{n-1}(\dots(stepReLU_1(\bar{I}))))$. 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 [81].

VII.2.2 Star Set [84, 81] (code)

The star set is an efficient set representation for simulation-based verification of large linear systems [9, 11, 85] where the superposition property of a linear system can be exploited in the analysis. It has been shown in [84] 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 [84] 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 [84] for a detailed discussion of star-set reachability algorithms for FFNNs.

We note that NNV implements enhanced versions of earlier star-based reachability algorithms [84]. 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 [81, 82]. 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 [81, 82] for a detailed discussion of the extensions of the star-based reachability algorithms for NNCSs with linear/nonlinear plant models.

VII.2.3 Zonotope [74] (code)

NNV implements the zonotope reachability algorithms proposed in [74] 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 [84]. 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.

VII.2.4 Abstract Domain [75]

NNV implements the abstract domain reachability algorithm proposed in [75] 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.

VII.2.5 ImageStar Set [80] (code)

NNV recently introduced a new set representation called the ImageStar for use in the verification of deep convolutional neural networks (CNNs) and semantic segmentation networks (SSNs). 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, transposed 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 [73] that consist of > 100 million parameters [80].

VII.3 Evaluation

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

VII.3.1 Safety verification of ACAS Xu networks

We evaluate NNV in comparison to Reluplex [41], Marabou [42], and ReluVal [92], by considering the verification of safety property ϕ_3 and ϕ_4 of the ACAS Xu neural networks [40] for all 45 networks.¹ All the experiments were done using 4 cores for computation. The results are summarized in Table VII.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. 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 [83] achieves a timeout on most of networks, and therefore, we neglect this method in the comparison.

¹We omit properties ϕ_1 and ϕ_2 for space and due to their long runtimes, but they can be reproduced in the artifact.

ACAS XU ϕ_3	SAT	UNSAT	UNK	TIMEOUT			TIME(s)
				1h	2h	10h	
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	10	35	0	0	0	72
NNV Exact Star	3	42	0	0	0	0	1371
NNV Appr. Star	0	29	16	0	0	0	52
<hr/>							
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

Table VII.2: Verification results of ACAS Xu networks.

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

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%).

VII.3.2 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, 81], 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

$\mathbf{v_lead(0)}$	Linear Plant		Nonlinear Plant	
	<i>Safety</i>	<i>VT(s)</i>	<i>Safety</i>	<i>VT(s)</i>
[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

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

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 seconds.

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:

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

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 seconds (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 $T_{gap} = 1.4$ seconds and $D_{default} = 10$. Our analysis investigates whether the safety requirement holds during the 5 seconds 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 $x_{ego} \in [30, 31]$.

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 VII.3, which shows that safety of the ACC system depends on

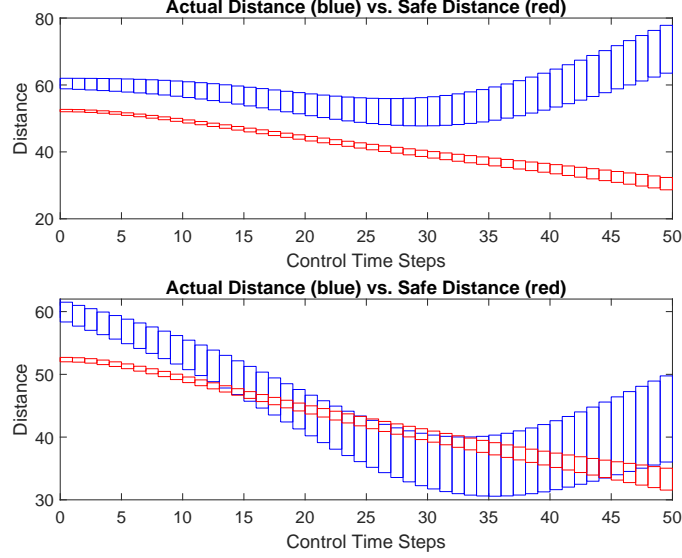


Figure VII.2: Two scenarios of the ACC system. In the first (top) scenario ($v_{lead}(0) \in [29, 30]m/s$), safety is guaranteed, $D_{rel} \geq D_{safe}$. In the second scenario (bottom) ($v_{lead}(0) \in [24, 25]m/s$), safety is violated since $D_{ref} < D_{safe}$ in some control steps.

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 VII.2 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 VII.3, the approximate analysis of the ACC system with discrete linear plant model is fast and can be done in 84 seconds. NNV 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 [81]. 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 VII.3, the verification for the linear model using the over-approximate method is $22.7\times$ faster on average than of the nonlinear model.

CHAPTER VIII

Conclusion and Future Directions

VIII.1 Conclusion

We have proposed a general framework for the verification of deep learning systems using set-based reachability analysis. Our framework uses the exact and approximate reachability schemes to verify the safety and robustness of deep neural networks as well as neural-network-based control systems. All reachability schemes implemented in our NNV tool work on a variety of set representations, including polyhedron, star, ImageStar, zonotope, and relaxed polytope abstract domain.

For safety and robustness verification of DNNs, we have focused mainly on feedforward neural networks, image classification convolutional neural networks, and semantic segmentation networks. We have shown that exact-reachability-based verification for networks with piecewise linear activation functions is feasible but costly in both computation time and memory consumption. In contrast, approximate-reachability-based verification is generally faster and less expensive than the exact approach. However, the approximate method produces an inevitable over-approximation error in the analysis. We have also shown that the exact reachable set of a neural-network-based control system with a linear plant model and ReLU network controller is computable. Notably, the approximate scheme works well in this case, with a much less computational cost and acceptable over-approximation error. Importantly, it can be extended to deal with NNCS with a nonlinear plant model and network controller having Sigmoid or Tanh activation function.

VIII.2 Future Directions

VIII.2.1 Scalability vs. Conservativeness

Scalability is still a major challenge for most existing verification techniques. It has been shown in [84] that the verification time using exact analysis increases exponentially. Particularly, besides the size of the network, the input set is an important factor affecting the verification time of the exact analysis method. Generally, a large network or a large input set requires more verification time. To improve scalability, a large body of research in neural network verification relies on over-approximation methods. Some recent approaches [75, 74] are optimistic about the scalability of their methods. However, it is shown in [84] that these methods only can deal with a very small input set due to the explosion of over-approximation error in the analysis, which leads to a conservative and frequently useless reachable set. In the future, we believe that new hybrid techniques that can combine the advantages of exact and over-approximate analyses are still needed to improve both scalability and conservativeness in neural network verification.

Chapter III has shown the fact that the reachability method using Zonotope is very fast but too conservative that cannot be used for verification of DNNs. To overcome this challenge, we are developing a new approach that combines zonotope reachability with input refinement using a maximum sensitive guided method. The high-level idea is if the reachable set produced by the zonotope reachability method is too conservative and cannot be used for verifying the safety property of the network. We split the input set into two new smaller sets, and then perform the verification task on these two new sets. We keep splitting the input set until we can prove the safety of the network for all new input sets, or we can find a counterexample. However, to reduce the number of splits significantly, we can use the maximum sensitive guided method in which we choose to split the input that maximizes the reduction of the conservativeness of the reachable set. We call this input is the max sensitive input. By determining the max sensitive input at each splitting, we can obtain an optimized splitting scheme where the number of splits is smallest.

VIII.2.2 Formal specifications and compositional verification

While a large body of research focuses on verifying neural networks and NNCS, fewer works investigate specification formalization for such systems [71, 30, 21]. For neural network verification, most current methods investigate safety and robustness properties, which can be specified as input to output relations of neural networks [71, 21]. For NNCS verification, existing approaches deal with safety specifications defined as predicates over the states of the plant model. In the real-world, learning-enabled CPS are complex in which several LECs, such as perception components and neural network controllers, interact with each other and the physical world, such as between a physical plant and its environment.

Defining meaningful system-level specifications for the whole system is relatively straightforward (such as collision avoidance), but the implications and constraints such system-level specifications place on LECs, especially those for perception, is non-trivial and needs to be investigated deeply. New specification languages for learning-enabled CPS are crucial to formally define the behavior of the systems and their subcomponents, and equally important, is defining libraries of specifications for meaningful perception problems, such as classification, semantic segmentation, and object detection/localization. A further challenge, particularly related to perception, is not only in defining specifications, but in evaluating specifications with respect to meaningful environmental scenarios and data. This challenge is fundamentally different than the typical approach for verification of closed-loop systems, where a plant model generates new inputs for a controller, and instead requires verification with respect to pre-recorded environmental data (such as images/video) or generation thereof. This is partly because it is unreasonable to expect formal models for the environment in which an NNCS operates, and at best, generative models such as GANs and realistic simulators may exist, beyond pre-recorded real-world data. Altogether, it is unclear under what circumstances compositional spec-

ification and verification for learning-enabled CPS are achievable, such as by verifying individual LECs and attempting to compose guarantees of individual components into system-level guarantees [70].

VIII.2.3 Verification for Recurrent Neural Networks

Recurrent neural networks (RNNs) are powerful machine learning models that are used in a wide range of applications such as machine translation, speech recognition, face detection, etc. Unlike feedforward neural networks, RNNs have internal states that help process variable-length sequences of inputs, making the reachability of RNNs more complex to analyze. Although a rigorous number of techniques and tools have been proposed for verifying properties of DNNs, there is very little work on RNNs verification. In [4], an unrolling-based approach has been proposed to verify an RNN in which the RNN is transformed into an equivalent feedforward neural network by the unrolling process, which is then used for verification. This approach has limited scalability because the transformation may create a large FFNN, which is expensive to verify. In the future, scalable set-based reachability methods for verification of RNNs are needed. The new methods should work for different types of RNNs and directly compute the reachable set of RNNs without performing any transformation or abstracting the networks.

VIII.2.4 Run-time Verification for Neural Network Control Systems

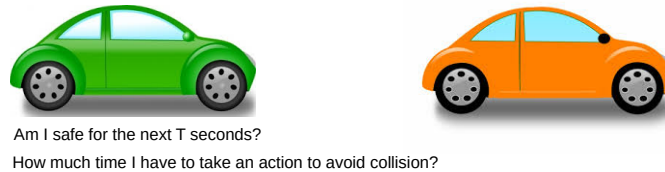


Figure VIII.1: Run-time verification example.

Most of the current verification techniques for learning-enabled CPS works in design time. In the real situation, it is useful for an autonomous system to have the ability for safety prediction at run-time. For example, if an autonomous system depicted in Figure VIII.1 knows that there is a potential collision in the next few seconds, it can take a smart action such as re-performing the planning algorithm to find a new safe path to avoid the collision. Therefore, it is essential to perform verification for learning-enabled CPS at run-time. The main challenge is how we can compute the reachable set of the system and verify its safety property for the next few seconds in every control period? To be able to do that, we need to have a very fast reachability algorithm that can quickly estimate an over-approximation of the exact behavior of the systems. In addition, the over-approximation of the reachable set cannot be too conservative to be used for safety verification. We are working on developing a light version of our star set approach to make it works in a real-time manner. We hope the new method can help to verify the safety of the advanced emergency braking system, learning-based adaptive cruise control system, and the learning-based UUV in real-time. The high-level idea of the

light version of our star set approach is, we neglect to solve linear programming problems when constructing the reachable set. Instead, we quickly estimate the output ranges of a star set using only the ranges of the predicate variables. The cost we need to pay is that the reachable set we construct is more conservative than the exact reachable set computed by the exact reachability scheme. However, since we can construct this conservative reachable set quickly, we can reason about the safety of the system at run-time. The central question we want to answer in this problem is that, how long will the system be safe from the current time to the future? If this “safe time” is smaller than a user-defined threshold, the system should take action to avoid the potential crash in the future. In this dangerous situation, our verification method can estimate the amount of time the system has to take a smart action to prevent the crash.

VIII.2.5 Robust and Safe Learning

The ultimate goal of the research in safe AI is to build safe, secure, and reliable learning-based autonomous systems. To do that, new learning methods that integrate verification techniques in the training process to enhance the robustness of a network or the safety of an NNCS are essential for the future of assured autonomy.

ACKNOWLEDGMENTS

First, I would like to express my profound gratitude to my supervising professor, Dr. Taylor T. Johnson, for his continuous support, dedicated guidance and tremendous encouragement that helps me to achieve valuable milestones in my research career. Dr. Taylor T. Johnson has given me full freedom to explore new and meaningful research directions. His profound knowledge, great vision, and helpful advice are essential for me to overcome numerous obstacles and challenges during my doctoral study.

I would also like to thank Dr. Janos Sztipanovits, Dr. Gabor Karsai, Dr. Xenofon Koutsoukos, Dr. Daniel Work, and Dr. Stanley Bak for their interest in my research, taking time to serve on my dissertation committee and providing insightful comments and suggestions on my studies and this proposal. Especially, I greatly appreciate Dr. Stanley Bak for the necessary support he has given me during the last three years in my Ph.D. period.

I would like to extend my thankfulness to my lab members and my colleagues, Dr. Weiming Xiang, Dr. Luan Viet Nguyen, Xiaodong Yang, Patrick Musau, Diego Manzanar Lopez, and Neelanjana Pal for their valuable discussions, hands-on assistance, and insightful comments on my works.

I owe a debt of gratitude to all of my friends for their help and advice during my studies. In particular, I truly appreciate Dr. Minh Quang Nguyen for helping me when I first started my doctoral program. My special thanks go to Dr. Cuong Manh Nguyen, Dr. Phuong Xuan Pham, Dr. Loan Bui, Dr. Son Nguyen and Dr. Trang Thuy Thai for their unlimited inspiration and encouragement during my doctoral study.

From the bottom of my heart, I would like to thank my parents Mr. Minh Hoang Tran and Mrs. Yen Thi Pham, my parents-in-law Mr. Vu Thanh Phan and Mrs. Tuyet Thi Pham for their immeasurable love and tremendous support.

Lastly but most importantly, I would like to take this opportunity to show my great appreciation to my wife, Mrs. Quynh Phan, for all her support, understanding, and patience during my journey to the doctoral degree. She is the most treasured gift that God has given me. She has been walking beside me through many difficult periods and enlightening my soul with her endless love. I am also thankful for the prestigious present my wife has given me, my daughter Sophia Tran. Looking at Sophia's smile, waiting for me every time I come home after work is one of the happiest times in my life.

The material presented in this dissertation 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 copy-

right 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.

BIBLIOGRAPHY

- [1] M. 2019b. *Model Predictive Control Toolbox*. The MathWorks Inc., Natick, Massachusetts, 2019.
- [2] F. Ahmad. Multidigit MNIST (M2NIST), 2018. <https://www.kaggle.com/farhanhubble/multimnistm2nist>.
- [3] M. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano. Reachability analysis for neural agent-environment systems. In *Sixteenth International Conference on Principles of Knowledge Representation and Reasoning*, 2018.
- [4] M. E. Akintunde, A. Kevochian, A. Lomuscio, and E. Pirovano. Verification of rnn-based neural agent-environment systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6006–6013, 2019.
- [5] M. Althoff. An introduction to cora 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.
- [6] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *2008 47th IEEE Conference on Decision and Control*, pages 4042–4048. IEEE, 2008.
- [7] A. Arnab, O. Miksik, and P. H. Torr. On the robustness of semantic segmentation models to adversarial attacks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 888–897, 2018.
- [8] S. Bak, S. Bogomolov, and T. T. Johnson. Hyst: a source transformation and translation tool for hybrid automaton models. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 128–133. ACM, 2015.
- [9] S. Bak and P. S. Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.
- [10] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson. Improved geometric path enumeration for verifying ReLU neural networks. In *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.
- [11] S. Bak, H.-D. Tran, and T. T. Johnson. 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*, pages 23–32. ACM, 2019.
- [12] V. E. Balas and M. M. Balas. Driver assisting by inverse time to collision. In *2006 World Automation Congress*, pages 1–6. IEEE, 2006.
- [13] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*, pages 2613–2621, 2016.
- [14] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
- [15] R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar. Piecewise linear neural network verification: A comparative study. *arXiv preprint arXiv:1711.00455*, 2017.
- [16] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.
- [17] C. Cheng, G. Nührenberg, and H. Ruess. Verification of Binarized Neural Networks. *CoRR*, abs/1710.03107, 2017.

- [18] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
- [19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.
- [20] T. Dreossi, A. Donzé, and S. A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. In *NASA Formal Methods Symposium*, pages 357–372. Springer, 2017.
- [21] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, and S. A. Seshia. A formalization of robustness for deep neural networks. *arXiv preprint arXiv:1903.10033*, 2019.
- [22] T. Dreossi, S. Jha, and S. A. Seshia. Semantic Adversarial Deep Learning. *ArXiv e-prints*, Apr. 2018.
- [23] S. Dutta, X. Chen, S. Jha, S. Sankaranarayanan, and A. Tiwari. Sherlock-a tool for verification of neural network feedback systems: demo abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 262–263. ACM, 2019.
- [24] S. Dutta, S. Jha, S. Sanakaranarayanan, and A. Tiwari. Output range analysis for deep neural networks. *arXiv preprint arXiv:1709.09130*, 2017.
- [25] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine*, 51(16):151–156, 2018.
- [26] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output Range Analysis for Deep Feedforward Neural Networks. In A. Dutle, C. Muñoz, and A. Narkawicz, editors, *NASA Formal Methods*, pages 121–138, Cham, 2018. Springer International Publishing.
- [27] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. In *UAI*, pages 550–559, 2018.
- [28] R. Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. *CoRR*, abs/1705.01320, 2017.
- [29] M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, Jun 2007.
- [30] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 63–78, 2019.
- [31] T. Gehr, M. Mirman, D. Drachler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.
- [32] T. Gehr, M. Mirman, D. Drachler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.
- [33] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [34] M. Herceg, M. Kvasnica, C. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proceedings of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013. <http://control.ee.ethz.ch/~mpt>.
- [35] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

- [36] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.
- [37] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.
- [38] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Hybrid Systems: Computation and Control (HSCC)*, 2019.
- [39] K. D. Julian, M. J. Kochenderfer, and M. P. Owen. Deep neural network compression for aircraft collision avoidance systems. *arXiv preprint arXiv:1810.04240*, 2018.
- [40] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th*, pages 1–10. IEEE, 2016.
- [41] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.
- [42] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.
- [43] S. Kong, S. Gao, W. Chen, and E. Clarke. dreach: δ -reachability analysis for hybrid systems. pages 200–205, 2015.
- [44] P. Koopman and M. Wagner. Autonomous vehicle safety: An interdisciplinary challenge. *IEEE Intelligent Transportation Systems Magazine*, 9(1):90–96, 2017.
- [45] P. Kouvaros and A. Lomuscio. Formal verification of cnn-based perception systems. *arXiv preprint arXiv:1811.11373*, 2018.
- [46] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [47] K. D. Kusano and H. Gabler. Method for estimating time to collision at braking in real-world, lead vehicle stopped rear-end crashes for use in pre-crash system design. *SAE International Journal of Passenger Cars-Mechanical Systems*, 4(2011-01-0576):435–443, 2011.
- [48] M. Kvasnica, P. Grieder, M. Baotić, and M. Morari. Multi-parametric toolbox (mpt). In *International Workshop on Hybrid Systems: Computation and Control*, pages 448–462. Springer, 2004.
- [49] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.
- [50] Y. LeCun. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998.
- [51] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [52] D. N. Lee. A theory of visual control of braking based on information about time-to-collision. *Perception*, 5(4):437–459, 1976.
- [53] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [54] X. Lin, H. Zhu, R. Samanta, and S. Jagannathan. Art: abstraction refinement-guided training for provably correct neural networks. *arXiv preprint arXiv:1907.10662*, 2019.

- [55] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. Van Ginneken, and C. I. Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.
- [56] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks. *CoRR*, abs/1903.06758, 2019.
- [57] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.
- [58] A. Lomuscio and L. Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.
- [59] D. M. Lopez, P. Musau, H.-D. Tran, and T. T. Johnson. Verification of closed-loop systems with neural network controllers. In G. Frehse and M. Althoff, editors, *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61 of *EPiC Series in Computing*, pages 201–210. EasyChair, April 2019.
- [60] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.
- [61] N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. Verifying Properties of Binarized Deep Neural Networks. *arXiv preprint arXiv:1709.06662*, 2017.
- [62] A. Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. In *Validation numerics*, pages 175–190. Springer, 1993.
- [63] K. Pei, Y. Cao, J. Yang, and S. Jana. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *CoRR*, abs/1705.06640, 2017.
- [64] L. Pulina and A. Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.
- [65] L. Pulina and A. Tacchella. NeVer: a tool for artificial neural networks verification. *Annals of Mathematics and Artificial Intelligence*, 62(3):403–425, Jul 2011.
- [66] L. Pulina and A. Tacchella. Challenging SMT solvers to verify neural networks. *AI Communications*, 25(2):117–135, 2012.
- [67] J. Rauber, W. Brendel, and M. Bethge. Foolbox v0. 8.0: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 5, 2017.
- [68] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for the l_0 norm. *arXiv preprint arXiv:1804.05805*, 2018.
- [69] K. Scheibler, L. Winterer, R. Wimmer, and B. Becker. Towards Verification of Artificial Neural Networks. In *MBMV*, pages 30–40, 2015.
- [70] S. A. Seshia. Compositional verification without compositional specification for learning-based systems. *University of California at Berkeley*, pages 1–8, 2017.
- [71] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue. Formal specification for deep neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 20–34. Springer, 2018.
- [72] Y. Shoukry, P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. Smc: Satisfiability modulo convex optimization. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 19–28, 2017.

- [73] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [74] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pages 10825–10836, 2018.
- [75] G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41, 2019.
- [76] S. S. Souradeep Dutta, Xin Chen. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Hybrid Systems: Computation and Control (HSCC)*, 2019.
- [77] X. Sun, H. Khedr, and Y. Shoukry. Formal verification of neural network controlled autonomous systems. In *Hybrid Systems: Computation and Control (HSCC)*, 2019.
- [78] Y. Tian, K. Pei, S. Jana, and B. Ray. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. *CoRR*, abs/1708.08559, 2017.
- [79] V. Tjeng and R. Tedrake. Verifying Neural Networks with Mixed Integer Programming. *arXiv preprint arXiv:1711.07356*, 2017.
- [80] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson. Verification of deep convolutional neural networks using imagestars. In *32nd International Conference on Computer-Aided Verification (CAV)*. Springer, July 2020.
- [81] H.-D. Tran, F. Cei, D. M. Lopez, T. T. Johnson, and X. Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. In *ACM SIGBED International Conference on Embedded Software (EMSOFT’19)*. ACM, October 2019.
- [82] H.-D. Tran, F. Cei, D. M. Lopez, T. T. Johnson, and X. Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. July 2019.
- [83] H.-D. Tran, P. Musau, D. M. Lopez, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson. Parallelizable reachability analysis algorithms for feed-forward neural networks. In *7th International Conference on Formal Methods in Software Engineering (FormaliSE2019), Montreal, Canada*, 2019.
- [84] H.-D. Tran, P. Musau, D. M. Lopez, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson. Star-based reachability analysis for deep neural networks. In *23rd International Symposium on Formal Methods (FM’19)*. Springer International Publishing, October 2019.
- [85] H.-D. Tran, L. V. Nguyen, N. Hamilton, W. Xiang, and T. T. Johnson. Reachability analysis for high-index linear differential algebraic equations (daes). In *17th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS’19)*. Springer International Publishing, August 2019.
- [86] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.
- [87] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. *arXiv preprint arXiv:1804.06760*, 2018.
- [88] R. J. Vanderbei. *Linear Programming: Foundations & Extensions (Second Edition)*. Springer, 2001.
- [89] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692. ACM, 2015.
- [90] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pages 6369–6379, 2018.

- [91] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1599–1614, 2018.
- [92] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, 2018. USENIX Association.
- [93] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.
- [94] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. *ArXiv e-prints*, Jan. 2018.
- [95] E. Wong and J. Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.
- [96] W. Xiang and T. T. Johnson. Reachability analysis and safety verification for neural network control systems. *arXiv preprint arXiv:1805.09944*, 2018.
- [97] W. Xiang, D. M. Lopez, P. Musau, and T. T. Johnson. Reachable set estimation and verification for neural network models of nonlinear dynamic systems. In *Safe, Autonomous and Intelligent Vehicles*, pages 123–144. Springer, 2019.
- [98] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. A. Rosenfeld, and T. T. Johnson. Verification for machine learning, autonomy, and neural networks survey. *CoRR*, abs/1810.01989, 2018.
- [99] W. Xiang, H. Tran, X. Yang, and T. T. Johnson. Reachable set estimation for neural network control systems: A simulation-guided approach. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–10, 2020.
- [100] W. Xiang, H.-D. Tran, and T. T. Johnson. Reachable set computation and safety verification for neural networks with relu activations. *arXiv preprint arXiv:1712.08163*, 2017.
- [101] W. Xiang, H.-D. Tran, and T. T. Johnson. Output reachable set estimation and verification for multi-layer neural networks. *IEEE transactions on neural networks and learning systems*, (99):1–7, 2018.
- [102] W. Xiang, H.-D. Tran, and T. T. Johnson. Specification-guided safety verification for feedforward neural networks. *AAAI Spring Symposium on Verification of Neural Networks*, 2019.
- [103] W. Xiang, H.-D. Tran, J. A. Rosenfeld, and T. T. Johnson. Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. *arXiv preprint arXiv:1802.06981*, 2018.
- [104] X. Yuan, P. He, Q. Zhu, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.
- [105] R. R. Zakrzewski. Randomized Approach to Verification of Neural Networks. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 4, pages 2819–2824 vol.4, July 2004.
- [106] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, pages 4944–4953, 2018.