Verification of Piecewise Deep Neural Networks: A Star Set Approach with Zonotope Pre-Filter

Hoang-Dung Tran[1], Neelanjana Pal[1], Diego Manzanas Lopez[1], Patrick Musau[1], Xiaodong Yang[1], Luan Viet Nguyen[2], Weiming Xiang[3][0000−0001−9065−8428], Stanley Bak[4] and Taylor T. Johnson[1][0000−0001−8021−9923]

[1] Institute for Software Integrated Systems, Vanderbilt University, TN, USA
[2] Dept. of Computer & Information Science, University of Pennsylvania, PA, USA
[3] School of Computer and Cyber Sciences & Augusta University
[4] Safe Sky Analytics

Abstract. Verification has emerged as a means to provide formal guarantees on neural network learning-based systems before using them in safety-critical applications. This paper proposes a new verification approach for deep neural networks (DNNs) with piecewise linear activation functions using reachability analysis. The core of our approach is a collection of reachability algorithms using star sets (or shortly, stars), an effective symbolic representation of high-dimensional polytopes. The star-based reachability algorithms compute the output reachable sets of a network with a given input set before using them for verification. For a neural network with piecewise linear activation functions, our approach can construct both the exact and over-approximate reachable sets of the network. To enhance the scalability of our approach, a star set is equipped with an outer-zonotope (a zonotope over-approximation of the star set) to quickly estimate the lower and upper bounds of an input set at a specific neuron to determine if splitting occurs at that neuron. This zonotope pre-filtering step reduces significantly the number of linear programming (LP) optimizations in the analysis and leads to a reduction in computation time, which enhances the scalability of the star set approach. Our reachability algorithms are implemented in a software prototype called the Neural Network Verification (NNV) tool, and can be applied to problems analyzing the robustness of machine learning methods, such as safety and robustness verification of DNNs. Our experiments show that our approach can achieve runtimes twenty to 1400 times faster than Reluplex, a satisfiability modulo theory (SMT)-based approach. It is also less conservative than the zonotope and abstract domain approaches proposed recently.

1 Introduction

Learning-based systems using deep neural networks (DNNs) have become popular choices for solving many complicated problems in practice such as image processing, natural language translation, market prediction [12,20,21]. Recently, these systems have appeared in safety critical applications such as autonomous

vehicles [6] and air traffic collision avoidance systems [14]. Utilizing NNs is promising but may put the system at a high risk which is unacceptable for safety critical applications. Notably, it has been proved that well-trained NNs are vulnerable to adversarial attacks where a slight change in the input can lead to a huge change at the output [23]. Recent incidents in autonomous driving (e.g., Tesla [11] and Uber [24]) raises an urgent need for techniques and tools that can formally verify the safety and robustness of NNs before utilizing them in safety-critical applications.

Safety verification and robustness certification of DNNs have attracted a huge attention from different communities such as machine learning [1, 2, 17, 22, 26, 33, 34, 39], formal methods [7, 13, 15, 25, 29, 36–38], and security [9, 32, 33], and a recent survey of the area is available [35]. Analyzing the behavior of a DNN can broadly be categorized into exact and over-approximate analyses. For the exact analysis, the SMT-based [15] and polyhedron-based approaches [29,36] are notable representatives. For the over-approximate analysis, the mixed-integer linear program (MILP) [7], interval arithmetic- [32, 33], zonotope- [26], input partition- [38], linearization- [34], and abstract-domain- [27] based are fast and efficient approaches. While the over-approximate analysis is usually faster and more scalable 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 [15]. 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.

Recently, a novel approach for verification of DNNs using star set has been proposed in [30]. 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 using star sets, we avoid the expensive affine mapping operation in polyhedron-based approach [29]* and thus, reduce the verification time significantly. The star set approach performs reachability analysis for feedforward DNNs layer-by-layer. For DNNs with piecewise linear activation functions, we can perform both exact analysis and over-approximate analysis. 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. Notably, *the star set approach is the first approach that can visualize the exact behavior of a DNN with piecewise linear activation functions.* Additionally, by exploiting the power of parallel computing, *the star set approach is applicable for real-world applications.*

In this paper, we enrich the original star set method [30] in following directions. Firstly, we improve the computation time and scalability of the star set approach using zonotope pre-filtering. Secondly, we extend the star set approach to new classes of piecewise activation functions such as saturating linear activation function (or shortly, *satlin*), symmetric saturating linear activation functions (or shortly, *satlins*), and leaky ReLUs. Lastly, we re-evaluate the improved star set approach on safety verification of all 45 ACAS XU networks [14] and robustness verification of a collection of image classification DNNs.

The improvement is based on a zonotope pre-filtering method [4] in which a star set is equipped with an *outer-zonotope* to estimate quickly the lower and upper bounds of the star input set at specific neuron to determine if splitting occurs at that neuron without needing to solve LP optimizations. In the original star set method, these bounds are determined by solving two LP problems. Thus, if the number of neurons is large, the number of LP problems increases quickly. Moreover, in the exact analysis, the number of LP problems increases exponentially as the number of stars in the reachable set grows exponentially [30]. Therefore, minimizing the number of LP problems is crucial for scalability of the star-set approach.

We re-evaluate the improved reachability algorithms in comparison with the original star set approach [30], Reluplex [15], zonotope [26], and abstract domain [27] approaches. The experimental results show that the improved algorithms is several times faster the original algorithms. It can achieve 20× to 1408× faster than Reluplex while can visualize the precise behavior of the ACAS Xu networks and can construct the complete set of counter example inputs in the case that a safety property is violated. Our over-approximate reachability algorithm is also much less conservative than the zonotope and new abstract domain approaches. 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.

In summary, the main contributions of this paper are: 1) propose novel, fast and scalable methods for the exact and over-approximate reachability analysis of DNNs with popular classes of piecewise linear activation functions; 2) implement the proposed methods in NNV toolbox that is available online for evaluation and comparison; 3) provide a thorough evaluation of the new methods via real-world case studies.

2 Preliminaries

2.1 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:

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

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 $F \triangleq \{L_1, \cdots, L_k\}$, the reachable set $F(\mathcal{I}) = \mathcal{R}_{L_k}$ of the neural network F corresponding to the input set I is defined incrementally by:*

$$\begin{aligned}
&\mathcal{R}_{L_1} \triangleq \{y_1 \mid y_1 = f_1(W_1 x + b_1),\ x \in \mathcal{I}\},\\
&\mathcal{R}_{L_2} \triangleq \{y_2 \mid y_2 = f_2(W_2 y_1 + b_2),\ y_1 \in \mathcal{R}_{L_1}\},\\
&\quad\vdots\\
&\mathcal{R}_{L_k} \triangleq \{y_k \mid y_k = f_k(W_k y_{k-1} + b_k)\ \ y_{k-1} \in \mathcal{R}_{L_{k-1}}\},
\end{aligned}$$

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 $F(\mathcal{I}) \vDash S$, 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 $F(\mathcal{I}) \nvDash S$.*

2.2 Generalized Star Sets

Definition 3 (Generalized Star Set [3]). 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, $V = \{v_1, v_2, \cdots, v_m\}$ is a set of m vectors in $\mathbb{R}^n$ called basis vectors, and $P : \mathbb{R}^m \to \{\top, \bot\}$ 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:

$$[\![\Theta]\!] = \{x \mid x = c + \Sigma_{i=1}^{m}(\alpha_i v_i) \text{ such that } P(\alpha_1, \cdots, \alpha_m) = \top\}. \quad (1)$$

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., $\alpha = [\alpha_1, \cdots, \alpha_m]^T$, and $d \in \mathbb{R}^{p \times 1}$. A star is an empty set, i.e., $\Theta = \emptyset$ if and only if the predicate $P(\alpha)$ is infeasible. In other words, we can say the predicate polyhedron $P(\alpha)$ is an empty set, i.e., $P(\alpha) = \emptyset$.

Proposition 1. *Any (bounded/unbounded) 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 $c = [0, 0, \cdots, 0]^T$, the basic vectors $V = \{e_1, e_2, \cdots, e_n\}$ in which e_i is the i^{th} basic vector of $\mathbb{R}^n$, and the predicate $P(\alpha) \triangleq C\alpha \leq d$.

Proposition 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.*

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

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

Proposition 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.*

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

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

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

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

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:

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

3 Reachability of FNNs with ReLU Activiation Functions

3.1 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 1), we assume the input set $\mathcal{I}$ of an FNN is a star set. From Definition 1, 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 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 $\mathcal{R}_L = ReLU_n(ReLU_{n-1}(\cdots ReLU_1(\Theta)))$, where $ReLU_i(\cdot)$ denotes the stepReLU opeartion at i^{th} neuron.

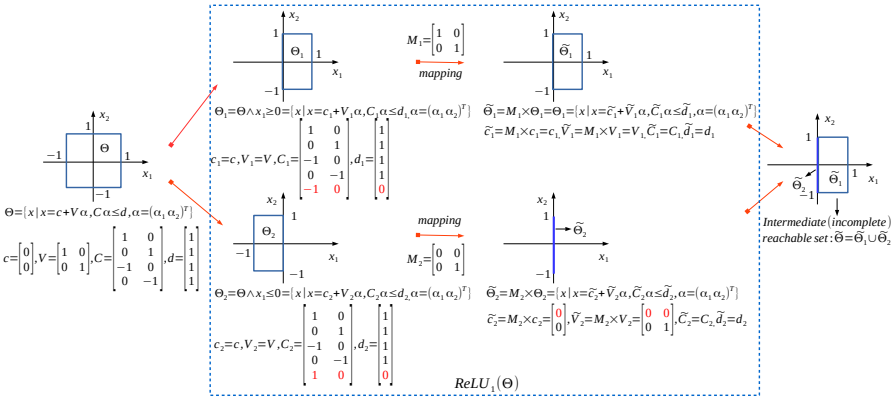

Fig. 1: An example of a stepReLU operation on a layer with two neurons.

The stepReLU operation, 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 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 $x = [x_1 \cdots x_i\ x_{i+1} \cdots x_n]^T \in \Theta_2$ will lead to the new vector $x' = [x_1\ x_2 \cdots 0\ x_{i+1} \cdots x_n]^T$. This procedure is equivalent to mapping Θ_2 by the mapping matrix $M = [e_1\ e_2 \cdots e_{i-1}\ 0\ e_{i+1} \cdots e_n]$. 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 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 1. We note that in our approach, we do not explicitly define the union operation of two stars. In reachability analysis, if a star set is split into two new stars, we say the reachable set is the union of two stars, which are simply stored in an array. If the reachable set is the union of N star sets, it is simply an array containing N star sets. By doing that, we can explore the power of parallel computing for reachability analysis of a neural network.

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 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 $2^{n_{L_1}} \times \cdots \times 2^{n_{L_k}} = 2^{n_{L_1}+\cdots+n_{L_k}} = 2^N$.

Lemma 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 $\mathcal{O}(N)$.

Theorem 1 (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 $F(\Theta) \models S$? is equivalent to solving 2^N feasibility problems in which each has $N+p+s$ linear constraints and m-variables.

Proof. From Lemma 1, there are at most 2^N stars in the reachable set of the neuron network. Also, from Lemma 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 negation of the safety specification, i.e., the unsafe region. We assume that the unsafe region can be represented in a half-space form or as a union of half-spaces. From Proposition 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.

Remark 1. 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. For example, in the ACAS Xu case study used in the evaluation section, each network has 300 neurons. Therefore, in the worst case, the number of stars in the reachable set is 2^{300}. However, depending on the size of the input set, the number of stars may vary from tens to millions instead of 2^{300}.

Theorem 2 (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 $\bar{\Theta}_i = \Theta_i \cap \neg S = \langle c_i, V_i, \bar{P}_i\rangle,\ i = 1, \cdots, k$. 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 3).

Complete counter input set. Note that all star sets in computation process are defined on the same predicate variable $\alpha = [\alpha_1, \cdots, \alpha_m]^T$ 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 $\mathcal{C}_\Theta = \cup_{i=1}^{k}\langle c, V, \bar{P}_i\rangle,\ \bar{P}_i \neq \emptyset$.

3.2 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 3. *For any input* $x \in [l,\ u]$*, the output set* $Y = \{y|\ y = ReLU(x)\}$ *satisfies:*

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

The over-approximation rules for the ReLU activation function of different approaches are depicted in Figure 2 which shows that our approximation rule is less conservative than the zonotope's [26] and new abstract domain's rules [27]. The zonotope-based approach [26] over-approximates the ReLU activation function by a minimal parallelogram while the abstract-domain approach [27] 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 $y_i = ReLU(x_i)$ to avoid the state space explosion [27], for example, in Figure 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. We remind that m is the number of predicate variables of the current star input set. 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.

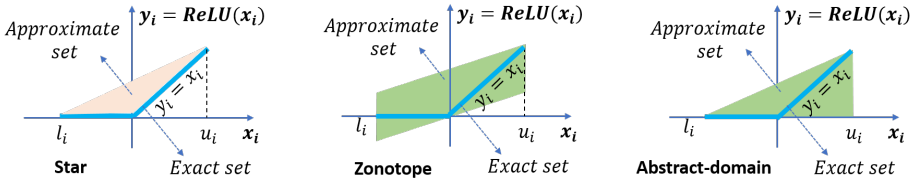

Fig. 2: The star set approach is less conservative than the zonotope [26] and new abstract-domain approaches [27].

Similar to the exact method, to verify the network's safety, we check the intersection between the obtained over-approximate reachable set and the unsafe region (specified in the half-space form). If the over-approximate reachable set does not intersect with the unsafe region, the network is safe. If it intersects with the unsafe region, the safety of the network is unknown due to the over-approximation error. One can see that, in the over-approximate method, only soundness is guaranteed.

3.3 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 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 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 [4] 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***. We note that the zono-tope pre-filtering technique has been introduced in [4] for only the exact reachability analysis. In this paper, we extend this technique to the approximate reachability analysis.

3.4 Reachability algorithms (code)

The improved star-based exact reachability algorithm using zonotope pre-filtering given in Algorithm 3.1 works as follows. The layer takes the star output sets of the preceding layer as input sets $I = [\Theta_1, \cdots, \Theta_N]$. 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 2. The star-based reachability analysis algorithm is much faster and more reliable than the polyhedron-based algorithm [8, 29] 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 [18].

4 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

Algorithm 3.1 Improved star-based reachability with zonotope pre-filtering.

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

procedure R = LayerReach$(I, W, b, method)$
  $R = \emptyset$
  parfor $i = 1 : N$ **do** ▷ parallel for loop
    $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$
    $R_1 = reachReLU(I_1,\ method)$, $R = R \cup R_1$
  end parfor
procedure R_1 = reachReLU$(I_1,\ method)$
  $In = I_1$
  $[lb, ub] = In.Z.getRanges$ ▷ estimate ranges of all input variables
  $map = find(ub < 0)$ ▷ list of neglected neurons
  $In.c(map, 1) = 0,\ In.V(map, :) = 0$ ▷ update the input star set
  $In.Z.l(map, 1) = 0,\ In.Z.G(map, :) = 0$ ▷ update the outer-zonotope
  $map = find(lb < 0\ \&\ ub > 0)$ ▷ construct computation map
  $m = length(map)$ ▷ minimized number of step operations
  for $i = 1 : m$ **do**
    if $method = exact$ **then**
      $In = stepReLU(In, map(i))$ ▷ stepReLU operation
    else if $method = approx$ **then**
      $In = approxStepReLU(In, map(i))$ ▷ approxStepReLU operation
  $R_1 = In$
procedure $\tilde{R}$ = stepReLU$(\tilde{I}, i)$
  $\tilde{R} = \emptyset$, $\tilde{I} = [\tilde{\Theta}_1 \ \cdots \ \tilde{\Theta}_k]$ ▷ intermediate star input and output sets
  for $j = 1 : k$ **do**
    $[lb_i,\ ub_i] = \tilde{\Theta}_j.getRange(i)$ ▷ get exact range of the j^{th} input
    $R_1 = \emptyset$, $M = [e_1\ e_2\ \cdots e_{i-1}\ 0\ e_{i+1}\ \cdots\ e_n]$
    if $lb_i \geq 0$ **then** $R_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{Z}_j \rangle$
    if $ub_i \leq 0$ **then** $R_1 = M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j, M\tilde{Z}_j \rangle$
    if $lb_i < 0\ \&\ ub_i > 0$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$,
      $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{Z}_j \rangle$
      $R_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
    $\tilde{R} = \tilde{R} \cup R_1$
procedure $\tilde{R}$ = approxStepReLU$(\tilde{I}, i)$
  $\tilde{I} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$
  $[l, u] = \tilde{\Theta}.getRange(i)$ ▷ get actual range of the i^{th} input
  $M = [e_1\ e_2\ \cdots e_{i-1}\ 0\ e_{i+1}\ \cdots\ e_n]$
  if $l \geq 0$ **then** $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$
  if $u \leq 0$ **then** $\tilde{R} = M * \tilde{\Theta} = \langle M\tilde{c}, M\tilde{V}, \tilde{P}, M\tilde{Z} \rangle$
  if $l < 0\ \&\ u > 0$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d},\ \alpha = [\alpha_1, \alpha_2, \cdots, \alpha_m]^T$ ▷ input set's predicate
    $\alpha' = [\alpha_1, \cdots, \alpha_m, \alpha_{m+1}]^T$ ▷ new variable α_{m+1}
    $C_1 = [0\ 0\ \cdots\ 0\ \text{-}1],\ d_1 = 0$ ▷ $\alpha_{m+1} \geq 0 \Leftrightarrow C_1\alpha' \leq d_1$
    $C_2 = [V(\tilde{i}, :)\ \text{-}1],\ d_2 = -\tilde{c}[i]$ ▷ $\alpha_{m+1} \geq x[i] \Leftrightarrow C_2\alpha' \leq d_2$
    $C_3 = [\frac{-u}{u-l} \times V(\tilde{i}, :)\ \ 1],\ d_3 = \frac{ul}{u-l} \times (1 - \tilde{c}[i])$ ▷ $\alpha_{m+1} \leq \frac{u(x[i]-l)}{u-l} \Leftrightarrow C_3\alpha' \leq d_3$
    $C_0 = [\tilde{C}\ 0_{m\times 1}],\ d_0 = \tilde{d}$
    $C' = [C_0; C_1; C_2; C_3],\ d' = [d_0; d_1; d_2; d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$ ▷ output set's predicate
    $c' = M\tilde{c},\ V' = M\tilde{V},\ V' = [V'\ e_i]$ ▷ $y[i] = ReLU(x[i]) = \alpha_{m+1}$
    $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$

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.

4.1 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 4.2. 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).

4.2 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 4.3.

4.3 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 4.4. 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 4.2 Improved star-based reachability for a satlin layer.

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

procedure R = LayerReach$(I, W, b, method)$
  $R = \emptyset$
  parfor $i = 1 : N$ **do** ▷ parallel for loop
    $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$
    $R_1 = reachSatlin(I_1,\ method)$, $R = R \cup R_1$
  end parfor
procedure R_1 = reachSatlin$(I_1,\ method)$
  $In = I_1$
  $[lb, ub] = In.Z.getRanges$ ▷ estimate ranges of all input variables
  $map = find(ub \leq 0)$ ▷ list of neglected neurons
  $In.c(map, 1) = 0,\ In.V(map, :) = 0$ ▷ update the input star set
  $In.Z.l(map, 1) = 0,\ In.Z.G(map, :) = 0$ ▷ update the outer-zonotope
  $map = find(lb \geq 1)$ ▷ list of neglected neurons
  $In.c(map, 1) = 1,\ In.V(map, :) = 0$ ▷ update the input star set
  $In.Z.l(map, 1) = 1,\ In.Z.G(map, :) = 0$ ▷ update the outer-zonotope
  $map = find(lb < 1\ ||\ ub > 0)$ ▷ construct computation map
  $m = length(map)$ ▷ minimized number of step operations
  for $i = 1 : m$ **do**
    if $method = exact$ **then**
      $In = stepSatlin(In, map(i))$ ▷ stepSatlin operation
    else if $method = approx$ **then**
      $In = approxStepSatlin(In, map(i))$ ▷ approxStepSatlin operation
  $R_1 = In$
procedure $\tilde{R}$ = stepSatlin$(\tilde{I}, i)$
  $\tilde{R} = \emptyset$, $\tilde{I} = [\tilde{\Theta}_1 \ \cdots \ \tilde{\Theta}_k]$ ▷ intermediate star input and output sets
  for $j = 1 : k$ **do**
    $[lb_i,\ ub_i] = \tilde{\Theta}_j.getRange(i)$ ▷ get exact range of the j^{th} input
    $R_1 = \emptyset$, $M = [e_1\ e_2\ \cdots e_{i-1}\ 0\ e_{i+1}\ \cdots\ e_n]$
    if $lb_i \geq 0$ & $ub_i \leq 1$ **then** $R_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{Z}_j \rangle$
    if $lb_i \geq 1$ **then**
      $\tilde{c}'_j = \tilde{c}_j, \tilde{c}'_j(i) = 1,\ \tilde{V}'_j = \tilde{V}_j, \tilde{V}'_j(i, :) = 0$
      $\tilde{Z}'_j = \tilde{Z}_j, \tilde{Z}'_j.l(i) = 1, \tilde{Z}'_j.G(i, :) = 0$
      $R_1 = \langle \tilde{c}'_j, \tilde{V}'_j, \tilde{P}'_j, \tilde{Z}'_j \rangle$
    if $ub_i \leq 0$ **then** $R_1 = M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j, M\tilde{Z}_j \rangle$
    if $lb_i < 0$ & $ub_i \leq 1$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$,
      $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{Z}_j \rangle$
      $R_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
    if $0 \leq lb_i \leq 1$ & $ub_i \geq 1$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \leq 1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$,
      $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] \geq 1 = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j, \tilde{Z}''_j \rangle$
      $\tilde{c}''_j = \tilde{c}_j, \tilde{c}''_j(i) = 1, \tilde{V}''_j = \tilde{V}_j, \tilde{V}''_j(i, :) = 0,$
      $\tilde{Z}''_j = \tilde{Z}_j, \tilde{Z}''_j.l(i) = 1, \tilde{Z}''_j.G(i, :) = 0$
      $R_1 = \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$
    if $lb_i \leq 0$ & $ub_i \geq 1$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge 0 \leq x[i] \leq 1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$
      $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{Z}_j \rangle$
      $\tilde{\Theta}'''_j = \tilde{\Theta}_j \wedge x[i] \geq 1 = \langle \tilde{c}'''_j, \tilde{V}'''_j, \tilde{P}'''_j, \tilde{Z}'''_j \rangle$
      $\tilde{c}'''_j = \tilde{c}_j, \tilde{c}'''_j(i) = 1, \tilde{V}'''_j = \tilde{V}_j, \tilde{V}'''_j(i, :) = 0,$
      $\tilde{Z}'''_j = \tilde{Z}_j, \tilde{Z}'''_j.l(i) = 1, \tilde{Z}'''_j.G(i, :) = 0$
      $R_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j \cup \tilde{\Theta}'''_j$
    $\tilde{R} = \tilde{R} \cup R_1$

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

Algorithm 4.3 Improved star-based reachability for a satlins layer.

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

procedure R = LayerReach$(I, W, b, method)$
  $R = \emptyset$
  parfor $i = 1 : N$ **do** ▷ parallel for loop
    $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$
    $R_1 = reachSatlin(I_1,\ method)$, $R = R \cup R_1$
  end parfor
procedure R_1 = reachSatlins$(I_1,\ method)$
  $In = I_1$
  $[lb, ub] = In.Z.getRanges$ ▷ estimate ranges of all input variables
  $map = find(ub \leq -1)$ ▷ list of neglected neurons
  $In.c(map, 1) = -1,\ In.V(map, :) = 0$ ▷ update the input star set
  $In.Z.l(map, 1) = -1,\ In.Z.G(map, :) = 0$ ▷ update the outer-zonotope
  $map = find(lb \geq 1)$ ▷ list of neglected neurons
  $In.c(map, 1) = 1,\ In.V(map, :) = 0$ ▷ update the input star set
  $In.Z.l(map, 1) = 1,\ In.Z.G(map, :) = 0$ ▷ update the outer-zonotope
  $map = find(lb < 1\ ||\ ub > -1)$ ▷ construct computation map
  $m = length(map)$ ▷ minimized number of step operations
  for $i = 1 : m$ **do**
    if $method = exact$ **then**
      $In = stepSatlins(In, map(i))$ ▷ stepSatlins operation
    else if $method = approx$ **then**
      $In = approxStepSatlins(In, map(i))$ ▷ approxStepSatlins operation
  $R_1 = In$
procedure $\tilde{R}$ = stepSatlins$(\tilde{I}, i)$
  $\tilde{R} = \emptyset$, $\tilde{I} = [\tilde{\Theta}_1 \ \cdots \ \tilde{\Theta}_k]$ ▷ intermediate star input and output sets
  for $j = 1 : k$ **do**
    $[lb_i,\ ub_i] = \tilde{\Theta}_j.getRange(i)$ ▷ get exact range of the j^{th} input
    $R_1 = \emptyset$, $M = [e_1\ e_2\ \cdots e_{i-1}\ 0\ e_{i+1}\ \cdots\ e_n]$
    if $lb_i \geq -1$ & $ub_i \leq 1$ **then** $R_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{Z}_j \rangle$
    if $lb_i \geq 1$ **then**
      $\tilde{c}'_j = \tilde{c}_j, \tilde{c}'_j(i) = 1, \tilde{V}'_j = \tilde{V}_j, \tilde{V}'_j(i, :) = 0$
      $\tilde{Z}'_j = \tilde{Z}_j, \tilde{Z}'_j.l(i) = 1, \tilde{Z}'_j.G(i, :) = 0$
      $R_1 = \tilde{\Theta}'_j = \langle \tilde{c}'_j, \tilde{V}'_j, \tilde{P}_j, \tilde{Z}'_j \rangle$
    if $ub_i \leq -1$ **then**
      $\tilde{c}'_j = \tilde{c}_j, \tilde{c}'_j(i) = -1, \tilde{V}'_j = \tilde{V}_j, \tilde{V}'_j(i, :) = 0$
      $\tilde{Z}'_j = \tilde{Z}_j, \tilde{Z}'_j.l(i) = -1, \tilde{Z}'_j.G(i, :) = 0$
      $R_1 = \tilde{\Theta}'_j = \langle \tilde{c}'_j, \tilde{V}'_j, \tilde{P}_j, \tilde{Z}'_j \rangle$
    if $lb_i < -1$ & $ub_i \leq 1$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq -1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$,
      $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < -1 = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j, \tilde{Z}''_j \rangle$
      $\tilde{c}''_j = \tilde{c}_j, \tilde{c}''_j(i) = -1, \tilde{V}''_j = \tilde{V}_j, \tilde{V}''_j(i, :) = 0$
      $\tilde{Z}''_j = \tilde{Z}_j, \tilde{Z}''_j.l(i) = -1, \tilde{Z}''_j.G(i, :) = 0$
      $R_1 = \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$
    if $-1 \leq lb_i \leq 1$ & $ub_i \geq 1$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \leq 1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$,
      $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] \geq 1 = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j, \tilde{Z}''_j \rangle$
      $\tilde{c}''_j = \tilde{c}_j, \tilde{c}''_j(i) = 1, \tilde{V}''_j = \tilde{V}_j, \tilde{V}''_j(i, :) = 0,$
      $\tilde{Z}''_j = \tilde{Z}_j, \tilde{Z}''_j.l(i) = 1, \tilde{Z}''_j.G(i, :) = 0$
      $R_1 = \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$
    if $lb_i < -1$ & $ub_i > 1$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge -1 \leq x[i] \leq 1 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$
      $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < -1 = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j, \tilde{Z}''^{}_j \rangle$
      $\tilde{c}''_j = \tilde{c}_j, \tilde{c}''_j(i) = -1, \tilde{V}''_j = \tilde{V}_j, \tilde{V}''_j(i, :) = 0,$
      $\tilde{Z}''_j = \tilde{Z}_j, \tilde{Z}''_j.l(i) = -1, \tilde{Z}''_j.G(i, :) = 0$
      $\tilde{\Theta}'''_j = \tilde{\Theta}_j \wedge x[i] \geq 1 = \langle \tilde{c}'''_j, \tilde{V}'''_j, \tilde{P}'''_j, \tilde{Z}'''_j \rangle$
      $\tilde{c}'''_j = \tilde{c}_j, \tilde{c}'''_j(i) = 1, \tilde{V}'''_j = \tilde{V}_j, \tilde{V}'''_j(i, :) = 0,$
      $\tilde{Z}'''_j = \tilde{Z}_j, \tilde{Z}'''_j.l(i) = 1, \tilde{Z}'''_j.G(i, :) = 0$
      $R_1 = \tilde{\Theta}'_j \cup \tilde{\Theta}''_j \cup \tilde{\Theta}'''_j$
    $\tilde{R} = \tilde{R} \cup R_1$

procedure $\tilde{R}$ = APPROXSTEPSATLINS$(\tilde{I}, i)$
  $\tilde{I} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$
  $[l, u] = \tilde{\Theta}.getRange(i)$ ▷ get actual range of the i^{th} input
  $M = [e_1\ e_2\ \cdots e_{i-1}\ 0\ e_{i+1}\ \cdots\ e_n]$
  if $l \geq -1\ \&\ u \leq 1$ **then** $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$
  if $l \geq 1$ **then**
    $\tilde{c}' = \tilde{c}, \tilde{c}(i) = 1,\ \tilde{V}' = \tilde{V}, \tilde{V}(i,:) = 0,\ \tilde{Z}' = \tilde{Z}, \tilde{Z}.l(i) = 1, \tilde{Z}.G(i,:) = 0$
    $\tilde{R} = \tilde{\Theta}' = \langle \tilde{c}', \tilde{V}', \tilde{P}, \tilde{Z}' \rangle$
  if $u \leq -1$ **then**
    $\tilde{c}' = \tilde{c}, \tilde{c}(i) = -1,\ \tilde{V}' = \tilde{V}, \tilde{V}(i,:) = 0,\ \tilde{Z}' = \tilde{Z}, \tilde{Z}.l(i) = -1, \tilde{Z}.G(i,:) = 0$
    $\tilde{R} = \tilde{\Theta}' = \langle \tilde{c}', \tilde{V}', \tilde{P}, \tilde{Z}' \rangle$
  if $l < -1\ \&\ 0 < u \leq 1$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d},\ \alpha = [\alpha_1, \alpha_2, \cdots, \alpha_m]^T$ ▷ input set's predicate
    $\alpha' = [\alpha_1, \cdots, \alpha_m, \alpha_{m+1}]^T$ ▷ new variable α_{m+1}
    $C_1 = [0\ 0\ \cdots\ 0\ \text{-}1],\ d_1 = 1$ ▷ $\alpha_{m+1} \geq -1 \Leftrightarrow C_1\alpha' \leq d_1$
    $C_2 = [\tilde{V}(i,:)\ \text{-}1],\ d_2 = -\tilde{c}(i)$ ▷ $\alpha_{m+1} \geq x[i] \Leftrightarrow C_2\alpha' \leq d_2$
    ▷ $\alpha_{m+1} \leq \frac{(u+1)(x[i]-u)}{u-l} + u \Leftrightarrow C_3\alpha' \leq d_3$
    $C_3 = [\frac{-u-1}{u-l} \times \tilde{V}(i,:)\ \ 1],\ d_3 = \frac{-u(u+1)}{u-l} \times \tilde{c}[i] + u$
    $C_0 = [\tilde{C}\ 0_{m\times 1}],\ d_0 = \tilde{d}$
    $C' = [C_0; C_1; C_2; C_3],\ d' = [d_0; d_1; d_2; d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$ ▷ output set's predicate
    $c' = M\tilde{c},\ V' = M\tilde{V},\ V' = [V'\ e_i]$ ▷ $y[i] = satlins(x[i]) = \alpha_{m+1}$
    $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$
  if $-1 \leq l < 1\ \&\ u > 1$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d},\ \alpha = [\alpha_1, \alpha_2, \cdots, \alpha_m]^T$ ▷ input set's predicate
    $\alpha' = [\alpha_1, \cdots, \alpha_m, \alpha_{m+1}]^T$ ▷ new variable α_{m+1}
    $C_1 = [0\ 0\ \cdots\ 0\ 1],\ d_1 = 1$ ▷ $\alpha_{m+1} \leq 1 \Leftrightarrow C_1\alpha' \leq d_1$
    $C_2 = [-\tilde{V}(i,:)\ 1],\ d_2 = \tilde{c}(i)$ ▷ $\alpha_{m+1} \leq x[i] \Leftrightarrow C_2\alpha' \leq d_2$
    $C_3 = [\frac{1-l}{u-l} \times \tilde{V}(i,:)\ \text{-}1],\ d_3 = \frac{l(1-l)}{u-l}\tilde{c}(i) - l$ ▷ $\alpha_{m+1} \geq \frac{(1-l)(x[i]-l)}{u-l} + l \Leftrightarrow C_3\alpha' \leq d_3$
    $C_0 = [\tilde{C}\ 0_{m\times 1}],\ d_0 = \tilde{d}$
    $C' = [C_0; C_1; C_2; C_3],\ d' = [d_0; d_1; d_2; d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$ ▷ output set's predicate
    $c' = M\tilde{c},\ V' = M\tilde{V},\ V' = [V'\ e_i]$ ▷ $y[i] = satlins(x[i]) = \alpha_{m+1}$
    $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$
  if $l < -1\ \&\ u > 1$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d},\ \alpha = [\alpha_1, \alpha_2, \cdots, \alpha_m]^T$ ▷ input set's predicate
    $\alpha' = [\alpha_1, \cdots, \alpha_m, \alpha_{m+1}]^T$ ▷ new variable α_{m+1}
    $C_1 = [0\ 0\ \cdots\ 0\ \text{-}1],\ d_1 = 1$ ▷ $\alpha_{m+1} \geq -1 \Leftrightarrow C_1\alpha' \leq d_1$
    $C_2 = [0\ 0\ \cdots\ 0\ 1],\ d_2 = 1$ ▷ $\alpha_{m+1} \leq 1 \Leftrightarrow C_2\alpha' \leq d_2$
    $C_3 = [-\frac{2}{1-l}\tilde{V}(i,:)\ 1],\ d_3 = \frac{2\tilde{c}[i]}{1-l} - 1$ ▷ $\alpha_{m+1} \leq \frac{2x[i]}{1-l} - 1 \Leftrightarrow C_3\alpha' \leq d_3$
    $C_4 = [\frac{2}{u+1} \times \tilde{V}(i,:)\ \text{-}1],\ d_4 = -\frac{2}{u+1}\tilde{c}(i) + 1$ ▷ $\alpha_{m+1} \geq \frac{2(x[i]+1)}{u+1} - 1 \Leftrightarrow C_4\alpha' \leq d_4$
    $C_0 = [\tilde{C}\ 0_{m\times 1}],\ d_0 = \tilde{d}$
    $C' = [C_0; C_1; C_2; C_3; C_4],\ d' = [d_0; d_1; d_2; d_3; d_4]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$ ▷ output set's predicate
    $c' = M\tilde{c},\ V' = M\tilde{V},\ V' = [V'\ e_i]$ ▷ $y[i] = satlins(x[i]) = \alpha_{m+1}$
    $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$

Algorithm 4.4 Improved star-based reachability for a leaky ReLU layer.

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

procedure R = LayerReach$(I, W, b, \gamma, method)$
  $R = \emptyset$
  parfor $i = 1 : N$ **do** ▷ parallel for loop
    $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i, WZ_i \rangle$
    $R_1 = reachLeakyReLU(I_1, \gamma, \ method)$, $R = R \cup R_1$
  end parfor
procedure R_1 = reachLeakyReLU$(I_1, \gamma, \ method)$
  $In = I_1$, $M_i = [e_1 \ e_2 \ \cdots e_{i-1} \ \gamma \times e_i \ e_{i+1} \ \cdots \ e_n]$
  $[lb, ub] = In.Z.getRanges$ ▷ estimate ranges of all input variables
  $map = find(ub < 0)$ ▷ list of neglected neurons
  $In.c = M_{map} In.c$, $In.V = M_{map} In.V$ ▷ update the input star set
  $In.Z.l = M_{map} In.Z.l$, $In.Z.G = M_{map} In.Z.G$ ▷ update the outer-zonotope
  $map = find(lb < 0 \ \& \ ub > 0)$ ▷ construct computation map
  $m = length(map)$ ▷ minimized number of step operations
  for $i = 1 : m$ **do**
    if $method = exact$ **then**
      $In = stepLeakyReLU(In, map(i))$ ▷ stepLeakyReLU operation
    else if $method = approx$ **then**
      $In = approxStepLeakyReLU(In, map(i))$ ▷ approxStepLeakyReLU operation
  $R_1 = In$
procedure $\tilde{R}$ = stepLeakyReLU$(\tilde{I}, i, \gamma)$
  $\tilde{R} = \emptyset$, $\tilde{I} = [\tilde{\Theta}_1 \ \cdots \ \tilde{\Theta}_k]$ ▷ intermediate star input and output sets
  for $j = 1 : k$ **do**
    $[lb_i, \ ub_i] = \tilde{\Theta}_j.getRange(i)$ ▷ get exact range of the j^{th} input
    $R_1 = \emptyset$, $M = [e_1 \ e_2 \ \cdots e_{i-1} \ \gamma \times e_i \ e_{i+1} \ \cdots \ e_n]$
    if $lb_i \geq 0$ **then** $R_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{Z}_j \rangle$
    if $ub_i \leq 0$ **then** $R_1 = M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j, M\tilde{Z}_j \rangle$
    if $lb_i < 0 \ \& \ ub_i > 0$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{Z}_j \rangle$,
      $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{Z}_j \rangle$
      $R_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
    $\tilde{R} = \tilde{R} \cup R_1$
procedure $\tilde{R}$ = approxStepLeakyReLU$(\tilde{I}, i, \gamma)$
  $\tilde{I} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$
  $[l, u] = \tilde{\Theta}.getRange(i)$ ▷ get actual range of the i^{th} input
  $M = [e_1 \ e_2 \ \cdots e_{i-1} \ \gamma \times e_i \ e_{i+1} \ \cdots \ e_n]$
  if $l \geq 0$ **then** $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{Z} \rangle$
  if $u \leq 0$ **then** $\tilde{R} = M * \tilde{\Theta} = \langle M\tilde{c}, M\tilde{V}, \tilde{P}, M\tilde{Z} \rangle$
  if $l < 0 \ \& \ u > 0$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}$, $\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_m]^T$ ▷ input set's predicate
    $\alpha' = [\alpha_1, \cdots, \alpha_m, \alpha_{m+1}]^T$ ▷ new variable α_{m+1}
    $C_1 = [\tilde{V}(i,:) \ \text{-1}]$, $d_1 = -\gamma\tilde{c}(i)$ ▷ $\alpha_{m+1} \geq \gamma x[i] \Leftrightarrow C_1\alpha' \leq d_1$
    $C_2 = [\tilde{V}(i,:) \ \text{-1}]$, $d_2 = -\tilde{c}[i]$ ▷ $\alpha_{m+1} \geq x[i] \Leftrightarrow C_2\alpha' \leq d_2$
    $C_3 = [\frac{-u}{u-l} \times \tilde{V}(i,:) \ \ 1]$, $d_3 = \frac{ul}{u-l} \times (1 - \tilde{c}[i])$ ▷ $\alpha_{m+1} \leq \frac{u(x[i]-l)}{u-l} \Leftrightarrow C_3\alpha' \leq d_3$
    $C_0 = [\tilde{C} \ 0_{m\times 1}]$, $d_0 = \tilde{d}$
    $C' = [C_0; C_1; C_2; C_3]$, $d' = [d_0; d_1; d_2; d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$ ▷ output set's predicate
    $M = [e_1 \ e_2 \ \cdots e_{i-1} \ 0 \ e_{i+1} \ \cdots \ e_n]$
    $c' = M\tilde{c}$, $V' = M\tilde{V}$, $V' = [V' \ e_i]$ ▷ $y[i] = leakyReLU(x[i]) = \alpha_{m+1}$
    $\tilde{R} = \langle c', V', P', \tilde{Z} \rangle$

5 Evaluation

In this section, we re-evaluate the improved star-based reachability algorithms in comparison to Marabou [16] which is an improvement of Reluplex [15], the zonotopes [26], and the abstract domain [27] methods implemented in our NNV tool [31]. The implementation of the zonotope and new abstract domain methods allows us to the visualize of 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 online[5].

5.1 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 [14]. 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.**
  - 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.

[5] https://github.com/verivital/nnv/tree/master/code/nnv/examples/Submission/FM2019_Journal

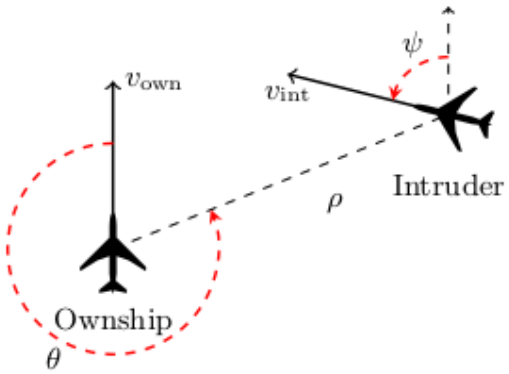

Fig. 3: Vertical view of a generic example of the ACAS Xu benchmark set. [15]

- The desired output property is that the score for COC is not the minimal score.
- It has 5 input constraints: $1500 \leq \rho \leq 1800$, $\theta \leq |0.06|$, $\psi = 0$, $v_{own} \geq 1000$, $700 \leq v_{int} \leq 800$.

Properties ϕ_3 and ϕ_4 require that the COC is not the minimal score. Therefore, if COC is the minimal score, the networks violate their safety specification. Mathematically, the unsafe specification can be written as $COC \leq WeakLeft \wedge COC \leq WeakRight \wedge COC \leq StrongLeft \wedge COC \leq StrongRight$.

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.[6] The verification results are presented in Tables 1 and 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. In the following, we use '×' to state the timing improvement of our approach compared with Marabou or Reluplex.

Star set approach. The experimental results show that the improved exact star method is on average 12× times faster than Marabou. It is also 27× and

[6] In [30], 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	Marabou		Exact-Star			Approx-Star			Zonotope			Abstract Domain		
	Res.	VT	Res.	VT	Imp.	Res.	VT	Imp.	Res.	VT	Imp.	Res.	VT	Imp.
N_{11}	SAFE	3636	SAFE	383.68	**9.5**×	UNK	1.16	**3135**×	UNK	0.04	**90925**×	UNK	0.10	**36370**×
N_{12}	SAFE	3418	SAFE	244.63	**14**×	UNK	0.86	**3974**×	UNK	0.02	**170900**×	UNK	0.06	**56957**×
N_{13}	SAFE	1795	SAFE	55.96	**32.1**×	UNK	0.95	**1889**×	UNK	0.02	**89750**×	UNK	0.07	**25643**×
N_{14}	SAFE	187	SAFE	13.68	**13.7**×	SAFE	0.17	**1100**×	UNK	0.02	**9350**×	UNK	0.08	**2338**×
N_{15}	SAFE	124	SAFE	18.25	**6.8**×	SAFE	0.24	**517**×	UNK	0.02	**6200**×	UNK	0.06	**2067**×
N_{16}	SAFE	37	SAFE	4.60	**8**×	SAFE	0.09	**411**×	UNK	0.02	**1850**×	UNK	0.07	**529**×
N_{17}	UNSAFE	5	UNSAFE	2.17	**2.3**×	UNK	0.07	**71**×	UNK	0.02	**250**×	UNK	0.05	**100**×
N_{18}	UNSAFE	2	UNSAFE	1.73	**1.2**×	UNK	0.05	**40**×	UNK	0.02	**100**×	UNK	0.04	**50**×
N_{19}	UNSAFE	2	UNSAFE	1.55	**1.3**×	UNK	0.03	**67**×	UNK	0.01	**200**×	UNK	0.05	**40**×
N_{21}	SAFE	1001	SAFE	67.63	**14.8**×	UNK	0.55	**1820**×	UNK	0.02	**50050**×	UNK	0.06	**16683**×
N_{22}	SAFE	480	SAFE	22.24	**21.6**×	UNK	0.36	**1333**×	UNK	0.02	**24000**×	UNK	0.05	**9600**×
N_{23}	SAFE	1103	SAFE	38.68	**28.5**×	UNK	0.66	**1671**×	UNK	0.02	**55150**×	UNK	0.05	**22060**×
N_{24}	SAFE	22	SAFE	1.71	**12.9**×	SAFE	0.05	**440**×	UNK	0.02	**1100**×	UNK	0.05	**440**×
N_{25}	SAFE	58	SAFE	8.45	**6.9**×	SAFE	0.26	**223**×	UNK	0.02	**2900**×	UNK	0.06	**967**×
N_{26}	SAFE	23	SAFE	1.64	**14**×	SAFE	0.10	**230**×	UNK	0.02	**1150**×	UNK	0.06	**383**×
N_{27}	SAFE	57	SAFE	4.16	**13.7**×	SAFE	0.12	**475**×	UNK	0.02	**2850**×	UNK	0.07	**814**×
N_{28}	SAFE	13	SAFE	1.74	**7.5**×	SAFE	0.06	**217**×	UNK	0.02	**650**×	UNK	0.05	**260**×
N_{29}	SAFE	2	SAFE	1.21	**1.7**×	SAFE	0.02	**100**×	SAFE	0.01	**200**×	UNK	0.03	**67**×
N_{31}	SAFE	97	SAFE	19.99	**4.9**×	SAFE	0.22	**441**×	UNK	0.02	**4850**×	UNK	0.05	**1940**×
N_{32}	SAFE	1973	SAFE	210.03	**9.4**×	UNK	0.86	**2294**×	UNK	0.02	**98650**×	UNK	0.07	**28186**×
N_{33}	SAFE	344	SAFE	35.06	**9.8**×	SAFE	0.62	**555**×	UNK	0.02	**17200**×	UNK	0.05	**6880**×
N_{34}	SAFE	106	SAFE	8.64	**12.3**×	UNK	0.78	**136**×	UNK	0.02	**5300**×	UNK	0.07	**1514**×
N_{35}	SAFE	41	SAFE	3.96	**10.4**×	SAFE	0.13	**315**×	UNK	0.02	**2050**×	UNK	0.05	**820**×
N_{36}	SAFE	120	SAFE	7.90	**15.2**×	UNK	0.34	**353**×	UNK	0.02	**6000**×	UNK	0.07	**1714**×
N_{37}	SAFE	8	SAFE	1.11	**7.2**×	SAFE	0.04	**200**×	UNK	0.02	**400**×	UNK	0.06	**133**×
N_{38}	SAFE	57	SAFE	2.87	**20**×	SAFE	0.17	**335**×	UNK	0.02	**2850**×	UNK	0.06	**950**×
N_{39}	SAFE	55	SAFE	4.83	**11.4**×	SAFE	0.11	**500**×	UNK	0.02	**2750**×	UNK	0.05	**1100**×
N_{41}	SAFE	94	SAFE	8.77	**10.7**×	UNK	0.22	**427**×	UNK	0.02	**4700**×	UNK	0.04	**2350**×
N_{42}	SAFE	1724	SAFE	83.21	**20.7**×	UNK	0.52	**3315**×	UNK	0.02	**86200**×	UNK	0.06	**28733**×
N_{43}	SAFE	1257	SAFE	121.59	**10.3**×	UNK	0.82	**1533**×	UNK	0.02	**62850**×	UNK	0.06	**20950**×
N_{44}	SAFE	19	SAFE	2.19	**8.7**×	SAFE	0.07	**271**×	UNK	0.02	**950**×	UNK	0.06	**317**×
N_{45}	SAFE	14	SAFE	1.40	**10**×	SAFE	0.06	**233**×	UNK	0.02	**700**×	UNK	0.04	**350**×
N_{46}	SAFE	118	SAFE	11.27	**10.5**×	SAFE	1.01	**1168**×	UNK	0.02	**5900**×	UNK	0.07	**1686**×
N_{47}	SAFE	54	SAFE	3.50	**15.4**×	SAFE	0.09	**600**×	UNK	0.02	**2700**×	UNK	0.06	**900**×
N_{48}	SAFE	36	SAFE	2.66	**13.5**×	SAFE	0.07	**514**×	UNK	0.02	**1800**×	UNK	0.06	**600**×
N_{49}	SAFE	33	SAFE	4.03	**8.2**×	SAFE	0.19	**174**×	UNK	0.02	**1650**×	UNK	0.06	**550**×
N_{51}	SAFE	764	SAFE	34.87	**21.9**×	UNK	0.47	**1626**×	UNK	0.02	**38200**×	UNK	0.05	**15280**×
N_{52}	SAFE	120	SAFE	7.28	**16.5**×	SAFE	0.22	**546**×	UNK	0.02	**6000**×	UNK	0.03	**4000**×
N_{53}	SAFE	146	SAFE	8.53	**17.1**×	SAFE	0.39	**374**×	UNK	0.02	**7300**×	UNK	0.05	**2920**×
N_{54}	SAFE	42	SAFE	3.02	**13.9**×	SAFE	0.14	**300**×	UNK	0.02	**2100**×	UNK	0.06	**700**×
N_{55}	SAFE	27	SAFE	4.36	**6.2**×	SAFE	0.24	**113**×	UNK	0.02	**1350**×	UNK	0.06	**450**×
N_{56}	SAFE	81	SAFE	4.75	**17.1**×	SAFE	0.24	**338**×	UNK	0.02	**4050**×	UNK	0.06	**1350**×
N_{57}	SAFE	5	SAFE	0.89	**5.6**×	SAFE	0.02	**250**×	SAFE	0.02	**250**×	UNK	0.04	**125**×
N_{58}	SAFE	157	SAFE	9.14	**17.2**×	SAFE	0.18	**872**×	UNK	0.02	**7850**×	UNK	0.06	**2617**×
N_{59}	SAFE	8	SAFE	1.05	**7.6**×	SAFE	0.05	**160**×	UNK	0.01	**800**×	UNK	0.02	**400**×
VT		**19466**		**1480.60**			**14.01**			**0.84**			**2.56**	
SAFE	**42/45**		**42/45**			**29/45**			**2/45**			**0/45**		
Imp.					**12**×			**769**×			**19622**×			**6731**×

Table 1: Verification results for property ϕ_3 on 45 ACAS Xu networks in which VT is the verification time in seconds. The exact-star method is on average 12× faster than Marabou (27× faster than Reluplex which is not presented here). The approx-star is on average 769× faster than Marabou (1408× 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.

29.8× times faster than Reluplex on property ϕ_3 and ϕ_4 respectively when we use parallel computing. Impressively, the approximate star method can achieve on average 769× (or 1408×) (for property ϕ_3) and 544× (or 961×) (for property ϕ_4) faster than Marabou (or 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 intersection operations can be done quickly. The improvement also come from the utilization of the zonotope pre-filtering step. In comparison with the original star set method, the

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}	SAFE	1929.00	SAFE	76.95	**25**×	UNK	0.42	**4643**×	UNK	0.02	**111334**×	UNK	0.04	**47639**×
N_{12}	SAFE	2012.00	SAFE	47.54	**42**×	UNK	0.54	**3709**×	UNK	0.02	**120003**×	UNK	0.05	**37812**×
N_{13}	SAFE	956.00	SAFE	36.29	**26**×	UNK	0.61	**1571**×	UNK	0.02	**47960**×	UNK	0.06	**16898**×
N_{14}	SAFE	79.00	SAFE	3.51	**23**×	UNK	0.14	**571**×	UNK	0.02	**4455**×	UNK	0.06	**1360**×
N_{15}	SAFE	299.00	SAFE	26.58	**11**×	UNK	0.21	**1434**×	UNK	0.02	**18200**×	UNK	0.03	**8772**×
N_{16}	SAFE	201.00	SAFE	12.80	**16**×	SAFE	0.24	**850**×	UNK	0.02	**12201**×	UNK	0.05	**3965**×
N_{17}	UNSAFE	2.00	UNSAFE	2.03	**1**×	UNK	0.05	**37**×	UNK	0.01	**141**×	UNK	0.04	**45**×
N_{18}	UNSAFE	2.00	UNSAFE	2.10	**1**×	UNK	0.06	**32**×	UNK	0.01	**133**×	UNK	0.04	**50**×
N_{19}	UNSAFE	1.00	UNSAFE	1.94	**1**×	UNK	0.04	**23**×	UNK	0.02	**65**×	UNK	0.03	**39**×
N_{21}	SAFE	239.00	SAFE	15.69	**15**×	UNK	0.42	**566**×	UNK	0.02	**14571**×	UNK	0.05	**4535**×
N_{22}	SAFE	273.00	SAFE	14.79	**18**×	UNK	0.40	**687**×	UNK	0.02	**16097**×	UNK	0.06	**4615**×
N_{23}	SAFE	47.00	SAFE	3.23	**15**×	SAFE	0.12	**406**×	UNK	0.02	**2871**×	UNK	0.03	**1447**×
N_{24}	SAFE	23.00	SAFE	3.55	**6**×	SAFE	0.22	**106**×	UNK	0.02	**1312**×	UNK	0.03	**702**×
N_{25}	SAFE	95.00	SAFE	11.95	**8**×	SAFE	0.42	**226**×	UNK	0.02	**5401**×	UNK	0.05	**1840**×
N_{26}	SAFE	71.00	SAFE	5.65	**13**×	SAFE	0.37	**192**×	UNK	0.02	**3749**×	UNK	0.07	**1090**×
N_{27}	SAFE	25.00	SAFE	2.47	**10**×	SAFE	0.12	**208**×	UNK	0.02	**1413**×	UNK	0.07	**376**×
N_{28}	SAFE	121.00	SAFE	8.41	**14**×	UNK	1.24	**97**×	UNK	0.02	**6042**×	UNK	0.07	**1764**×
N_{29}	SAFE	6.00	SAFE	1.29	**5**×	SAFE	0.04	**160**×	UNK	0.01	**429**×	UNK	0.03	**215**×
N_{31}	SAFE	220.00	SAFE	14.23	**15**×	SAFE	0.32	**686**×	UNK	0.02	**12501**×	UNK	0.05	**4407**×
N_{32}	SAFE	237.00	SAFE	27.10	**9**×	SAFE	0.21	**1117**×	UNK	0.02	**14725**×	UNK	0.03	**7339**×
N_{33}	SAFE	36.00	SAFE	3.70	**10**×	SAFE	0.12	**308**×	UNK	0.02	**2271**×	UNK	0.03	**1228**×
N_{34}	SAFE	32.00	SAFE	4.18	**8**×	SAFE	0.13	**248**×	UNK	0.02	**2020**×	UNK	0.03	**958**×
N_{35}	SAFE	212.00	SAFE	16.68	**13**×	SAFE	0.71	**300**×	UNK	0.02	**11232**×	UNK	0.05	**4514**×
N_{36}	SAFE	50.00	SAFE	5.57	**9**×	SAFE	0.31	**163**×	UNK	0.02	**2713**×	UNK	0.07	**702**×
N_{37}	SAFE	49.00	SAFE	4.07	**12**×	SAFE	0.11	**436**×	UNK	0.02	**2314**×	UNK	0.07	**753**×
N_{38}	SAFE	48.00	SAFE	2.98	**16**×	UNK	0.28	**174**×	UNK	0.02	**2124**×	UNK	0.05	**1062**×
N_{39}	SAFE	144.00	SAFE	13.04	**11**×	SAFE	0.44	**330**×	UNK	0.02	**6914**×	UNK	0.06	**2307**×
N_{41}	SAFE	30.00	SAFE	2.73	**11**×	SAFE	0.07	**405**×	SAFE	0.02	**1933**×	UNK	0.03	**1182**×
N_{42}	SAFE	77.00	SAFE	4.96	**16**×	SAFE	0.32	**240**×	UNK	0.02	**4698**×	UNK	0.05	**1632**×
N_{43}	SAFE	163.00	SAFE	9.79	**17**×	SAFE	0.32	**503**×	UNK	0.02	**9874**×	UNK	0.05	**3491**×
N_{44}	SAFE	52.00	SAFE	4.94	**11**×	UNK	0.57	**91**×	UNK	0.02	**2786**×	UNK	0.06	**920**×
N_{45}	SAFE	53.00	SAFE	3.91	**14**×	SAFE	0.22	**242**×	UNK	0.02	**3086**×	UNK	0.07	**788**×
N_{46}	SAFE	18.00	SAFE	7.96	**2**×	SAFE	0.30	**59**×	UNK	0.02	**983**×	UNK	0.06	**327**×
N_{47}	SAFE	16.00	SAFE	1.26	**13**×	SAFE	0.09	**169**×	UNK	0.02	**968**×	UNK	0.06	**288**×
N_{48}	SAFE	68.00	SAFE	6.10	**11**×	SAFE	0.18	**382**×	UNK	0.02	**3787**×	UNK	0.06	**1144**×
N_{49}	SAFE	51.00	SAFE	8.74	**6**×	SAFE	0.17	**295**×	UNK	0.02	**3190**×	UNK	0.05	**1038**×
N_{51}	SAFE	174.00	SAFE	19.88	**9**×	SAFE	0.22	**782**×	UNK	0.02	**9626**×	UNK	0.05	**3733**×
N_{52}	SAFE	48.00	SAFE	13.37	**4**×	SAFE	0.15	**318**×	UNK	0.02	**3066**×	UNK	0.04	**1306**×
N_{53}	SAFE	39.00	SAFE	5.16	**8**×	SAFE	0.27	**144**×	UNK	0.02	**2301**×	UNK	0.05	**823**×
N_{54}	SAFE	43.00	SAFE	3.61	**12**×	SAFE	0.19	**225**×	UNK	0.02	**2497**×	UNK	0.04	**1043**×
N_{55}	SAFE	61.00	SAFE	5.29	**12**×	SAFE	0.16	**371**×	UNK	0.02	**3646**×	UNK	0.06	**1102**×
N_{56}	SAFE	55.00	SAFE	3.01	**18**×	SAFE	0.19	**294**×	UNK	0.02	**3300**×	UNK	0.04	**1359**×
N_{57}	SAFE	8.00	SAFE	1.13	**7**×	SAFE	0.07	**114**×	UNK	0.02	**478**×	UNK	0.05	**163**×
N_{58}	SAFE	48.00	SAFE	3.09	**16**×	SAFE	0.28	**172**×	UNK	0.02	**2642**×	UNK	0.06	**742**×
N_{59}	SAFE	57.00	SAFE	3.77	**15**×	SAFE	0.14	**404**×	UNK	0.02	**3266**×	UNK	0.05	**1084**×
VT		**8470**		**477.01**			**12.20**			**0.78**			**2.19**	
SAFE	**42/45**		**42/45**			**32/45**			**1/45**			**0/45**		
Imp.					**12**×			**544**×			**10785**×			**3969**×

Table 2: Verification results for property ϕ_4 on 45 ACAS Xu networks in which VT is the verification time in seconds. The exact-star method is on average 12× faster than Marbou (and 29.8× faster than Reluplex). The approx-star is on average 544× faster than Marabou (and 961× 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.

improved exact method reduces the total verification time by 5× for property ϕ_3 and by 2.6× for property ϕ_4 (the original exact method verifies 45 networks with 7457 seconds for ϕ_3 and 1157 seconds for ϕ_4). Similarly, the improved approximate method reduces the total verification time by 3.7× for property ϕ_3 and by 2.5× for property ϕ_4 (the original approximate method verifies 45 networks with 52 seconds for ϕ_3 and 30 seconds for ϕ_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 [29]

whose the verification results are not presented in this paper. For example, when verifying property ϕ_3, the polyhedron method reaches time-out set as 30 minutes on more than 20 networks.

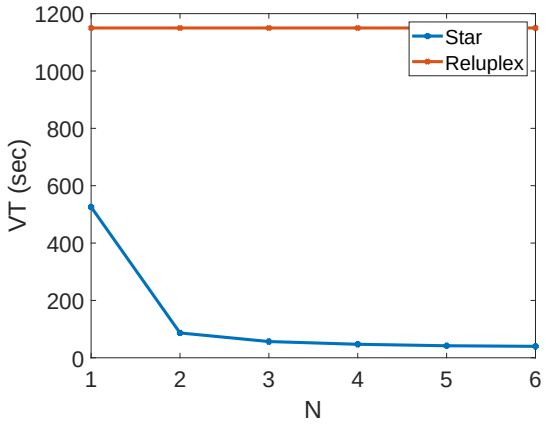

Fig. 4: Verification times for property ϕ_4 on N_{13} network with different number of cores.

Figure 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 ϕ_4 on N_{13}, our approach takes 525.8 seconds which is 2.2× faster than Reluplex (with 1150 seconds). With only 2 cores, our verification time drops quickly to 86.7 seconds which is 13.3× faster than Reluplex. When 4 cores are used, the verification time decreases to 42 seconds which is 27.4× 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 [26]. 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 ϕ_3 and 1/45 ($\approx 2.22\%$) networks for property ϕ_4. In comparison with our approximate star method, we can verify 29/45 ($\approx 64.44\%$) networks for property ϕ_3 and 32/45 ($\approx 71.11\%$) networks for property ϕ_4. This shows the fact that our method is significantly less conservative than the zonotope approach.

Abstract-domain based method *[27]* 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 [30], 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 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 "back-tracking" method. We have experienced that the lower and upper bounds obtained by the new abstract-domain method may be very conservative.

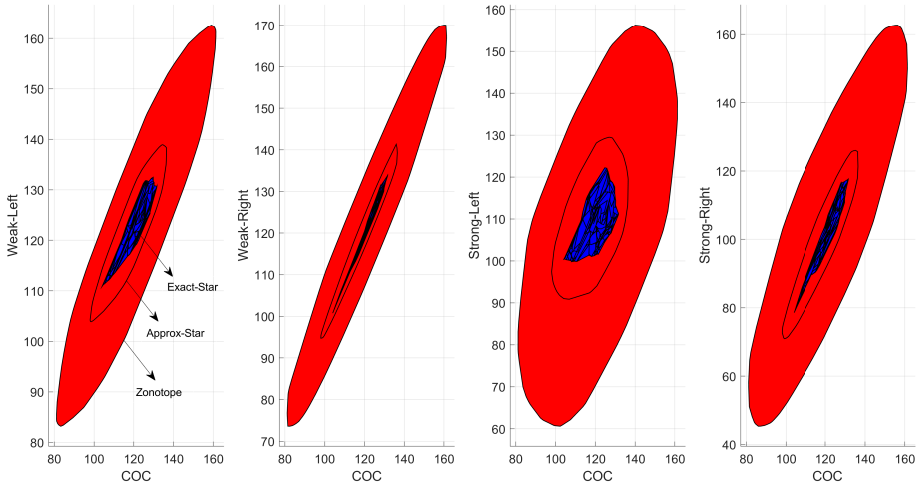

Fig. 5: Reachable sets of N_{41} network w.r.t property ϕ_4 with different methods.

Benefits of computing the reachable set. The computed reachable sets are useful for intuitively observe the complex behavior of the network. For example, Figure 5 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 verifi-

cation 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, 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 a adversarial input generator [5, 10] 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 2, 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 6 is a part of the input set that contains all counter inputs that make the neural network unsafe.

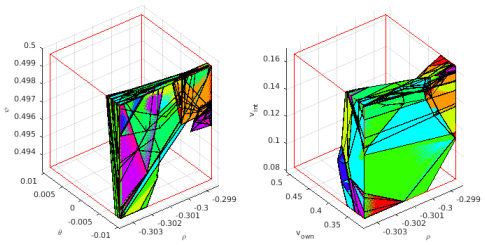

Fig. 6: The (normalized) complete counter input set for property ϕ_4' on N_{2_8} network is a part of the normalized input set (red boxes).

5.2 Robustness Certification of Image Classification ReLU 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

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 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.

the robustness of a network corresponding 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 [19] (50000 are used for training, while 10000 are used for testing). The trained networks have 784 inputs and a single output with expected value from 0 to 9. We trained a variety of feed-forward neural networks with varying sizes. The number of neurons that we considered ranged from 140 Neurons to 4000 neurons. Throughout our experiments, we used a fixed number of neurons for each layer. We utilized mean squared error as the loss function and utilized the Adam optimization algorithm for training [7]. On average, it took 12 epochs for the training to converge to an accuracy greater than 95%. 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 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 [26] and the abstract domain [27] 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.

[7] The hyperparameters we utilized were $\beta_1 = 0.9$, $\beta_2 = 0.999$, $lr = 0.001$

5.3 Robustness Certification of Image Classification DNNs with LeakyReLU, Satlin and SatLins activation functions

We further evaluate our method on DNNs with LeakyReLU, Satlin, and Satlins activation function via verifying the robustness of image classification DNNs under brightening attack [28]. We train three image classification networks on the MNIST data set using Matlab Deep Learning Toolbox. Each network has 784 (flattening each image with 28X28 resolution) input neurons and gives ten output neurons (for ten different classes; each neuron outputs corresponding to each of the classes; the max valued class being the prediction). The input dataset is normalized with $mean = 33.3184$ and $std = 1$ for networks with LReLU activation, otherwise no normalization is considered. The accuracy of each of the networks is between 96 and 98 percent.

To certify the robustness of the networks, we selected 100 images that are correctly classified by the respective networks and performed the brightening attack on these images. Then we calculated the reachable output sets for each of the perturbed images to determine if the correctly classified output always has the maximum value in the set compared to the others. The output layer activation $'softmax'$ is ignored for the approximate reachable set calculations.

In a brightening attack, some pixels are changed independently in the image to make it brighter or darker to fool the network, i.e., to misclassify the image. In this case study, to implement this attack, the value of that pixel x_i is reduced to the new value ${x'}_i$ if it is greater than a threshold value d such that $0 \leq {x'}_i \leq \delta \cdot x_i$. Here δ is a very small number and indicates the size of the input set that can be created by a specific attack. When d is small, the number of pixels in the image that are attacked is large and vice versa. The verification results of the networks corresponding to different values of d and δ are presented in Table 4, 5 and 6.

	Robustness Results (in Percent)								
	$\delta = 0.01$			$\delta = 0.05$			$\delta = 0.1$		
	Approx-Star	*Abs-Dom*	*Approx-Zono*	*Approx-Star*	*Abs-Dom*	*Approx-Zono*	*Approx-Star*	*Abs-Dom*	*Approx-Zono*
$d = 254$	91.00	91.00	91.00	90.00	88.00	87.00	86.00	82.00	82.00
$d = 250$	70.00	70.00	70.00	64.00	58.00	53.00	48.00	31.00	26.00
$d = 245$	66.00	66.00	66.00	57.00	49.00	47.00	45.00	27.00	22.00
	Verification Times (in Seconds)								
$d = 254$	1.17	49.38	1.34	3.98	55.07	1.35	18.07	57.89	0.86
$d = 250$	1.11	44.52	0.50	7.95	49.87	1.49	34.20	65.73	2.76
$d = 245$	1.13	43.12	0.65	9.25	50.15	1.80	44.23	74.60	3.17

Table 4: Verification results of 5-layered LeakyReLU network.

From the comparative tables, we can infer that all the reachability approaches that were considered exhibit almost similar robustness performance(in terms of the correct number of class predictions) when δ is very small, i.e., when the attack on the pixel is minimal, when the threshold value d decreases and δ value increases the robustness performance changes. It is important to emphasize that in this case study, we solve LP to find the ranges of neurons of the networks for the new abstract domain approach (in the ACAS Xu case study, we use only the estimated range for this method). Because the ranges of neurons are optimized,

	Robustness Results (in Percent)								
	$\delta = 0.01$			$\delta = 0.05$			$\delta = 0.1$		
	Approx-Star	*Abs-Dom*	*Approx-Zono*	*Approx-Star*	*Abs-Dom*	*Approx-Zono*	*Approx-Star*	*Abs-Dom*	*Approx-Zono*
$d = 254$	89.00	89.00	89.00	86.00	85.00	81.00	83.00	82.00	74.00
$d = 245$	60.00	60.00	59.00	48.00	47.00	26.00	37.00	33.00	7.00
$d = 234$	58.00	58.00	57.00	44.00	42.00	19.00	26.00	26.00	2.00
	Verification Times (in Seconds)								
$d = 254$	32.86	27.56	0.95	29.16	29.32	1.13	33.09	30.48	1.33
$d = 245$	22.05	21.67	0.83	32.16	32.17	2.56	37.68	28.54	5.50
$d = 234$	24.94	22.96	1.04	40.16	27.05	3.46	35.43	29.34	3.80

Table 5: Verification results of 4-layered Satlin network.

	Robustness Results (in Percent)								
	$\delta = 0.01$			$\delta = 0.05$			$\delta = 0.1$		
	Approx-Star	*Abs-Dom*	*Approx-Zono*	*Approx-Star*	*Abs-Dom*	*Approx-Zono*	*Approx-Star*	*Abs-Dom*	*Approx-Zono*
$d = 254$	86.00	86.00	82.00	83.00	83.00	66.00	80.00	80.00	51.00
$d = 245$	57.00	57.00	40.00	42.00	41.00	4.00	27.00	23.00	2.00
$d = 234$	51.00	51.00	34.00	40.00	37.00	1.00	19.00	14.00	0.00
	Verification Times (in Seconds)								
$d = 254$	50.14	49.36	1.55	59.87	52.35	1.69	56.43	50.81	2.26
$d = 245$	31.95	32.07	1.83	50.79	46.09	3.54	92.11	61.21	4.08
$d = 234$	31.21	30.36	1.98	56.34	49.65	3.58	80.83	65.43	3.79

Table 6: Verification results of 5-layered Satlins network.

the abstract domain method is more precise. However, at the same time, its verification time grows.

From the Tables, we ascertain that the approximate star method is more robust compared to the abstract domain and Zonotope in terms of prediction. Interestingly, for the LeakyReLU network, the approximate star method is much faster than the abstract domain method. This impressive improvement is due to the optimization in the implementation of the approximate reachability algorithm for the LeakyReLU network in which instead of performing a sequence of approximate stepReLU operations, we perform these operations in parallel. Though the zonotope method takes less time to analyze robustness, it is very clear that it can only predict 2% of the images correctly when the brightening attack on pixels is higher(i.e., lesser d and higher δ). The prediction accuracy of the abstract domain approach with optimized ranges of neurons is better than the Zonotope but lesser than approx-star. Overall, the approx-star method achieves a more consistent level of robustness performance for varying disturbance levels than others. Importantly, by parallelizing the approximate stepReLU operations, we can speed up the approximate star method by several orders of magnitudes.

6 Conclusion and Future Work

We have proposed two reachability analysis algorithms for DNNs with piecewise linear activations using star sets, one that is exact (sound and complete) but has scalability challenges and one that over-approximates (sound) with better scalability. The exact algorithm can compute and visualize the exact behaviors of DNNs. The exact method is more efficient than standard polyhedra approaches,

and faster than SMT-based approaches when running on multi-core platforms. The over-approximate algorithm is much faster than the exact one, and notably, it is much less conservative than recent zonotope and abstract-domain based approaches. Our algorithms are applicable for real world applications as shown in the safety verification of ACAS Xu DNNs and robustness certification of image classification DNNs. In ongoing and future work, we are extending the proposed methods for convolutional neural networks (CNNs) [28] and recurrent neural networks (RNNs), improving scalability for other types of activation functions such as tanh and sigmoid, and continuing to improve the NNV software framework [31].

Acknowledgments

The material presented in this paper is based upon work supported by the Air Force Office of Scientific Research (AFOSR) through contract number FA9550-18-1-0122, and the Defense Advanced Research Projects Agency (DARPA) through contract number FA8750-18-C-0089. The U.S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of AFOSR or DARPA.

References

1. Akintunde, M., Lomuscio, A., Maganti, L., Pirovano, E.: Reachability analysis for neural agent-environment systems. In: Sixteenth International Conference on Principles of Knowledge Representation and Reasoning (2018)
2. Akintunde, M.E., Kevorchian, A., Lomuscio, A., Pirovano, E.: Verification of rnn-based neural agent-environment systems. In: Proceedings of the 33th AAAI Conference on Artificial Intelligence (AAAI19). Honolulu, HI, USA. AAAI Press. To appear (2019)
3. Bak, S., Duggirala, P.S.: Simulation-equivalent reachability of large linear systems with inputs. In: International Conference on Computer Aided Verification. pp. 401–420. Springer (2017)
4. Bak, S., Tran, H.D., Hobbs, K., Johnson, T.: Improved geometric path enumeration for verifying ReLU neural networks. In: Proceedings of the 32nd International Conference on Computer Aided Verification. Springer (2020)
5. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., Criminisi, A.: Measuring neural net robustness with constraints. In: Advances in neural information processing systems. pp. 2613–2621 (2016)
6. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
7. Dutta, S., Jha, S., Sanakaranarayanan, S., Tiwari, A.: Output range analysis for deep neural networks. arXiv preprint arXiv:1709.09130 (2017)

8. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: International Symposium on Automated Technology for Verification and Analysis. pp. 269–286. Springer (2017)
9. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In: Security and Privacy (SP), 2018 IEEE Symposium on (2018)
10. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
11. Heilweil, R.: Tesla needs to fix its deadly Autopilot problem (2020), https://www.vox.com/recode/2020/2/26/21154502/tesla-autopilot-fatal-crashes
12. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., 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)
13. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: International Conference on Computer Aided Verification. pp. 3–29. Springer (2017)
14. Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. arXiv preprint arXiv:1810.04240 (2018)
15. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)
16. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)
17. Kouvaros, P., Lomuscio, A.: Formal verification of cnn-based perception systems. arXiv preprint arXiv:1811.11373 (2018)
18. Kvasnica, M., Grieder, P., Baotić, M., Morari, M.: Multi-parametric toolbox (mpt). In: International Workshop on Hybrid Systems: Computation and Control. pp. 448–462. Springer (2004)
19. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
20. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., van der Laak, J.A., Van Ginneken, B., Sánchez, C.I.: A survey on deep learning in medical image analysis. Medical image analysis **42**, 60–88 (2017)
21. Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E.: A survey of deep neural network architectures and their applications. Neurocomputing **234**, 11–26 (2017)
22. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. arXiv preprint arXiv:1706.07351 (2017)
23. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2574–2582 (2016)
24. Muoio, D.: The self-driving Uber in the Arizona crash was hit crossing an intersection on yellowUber crashes (2017), https://www.businessinsider.com/uber-self-driving-car-accident-arizona-police-report-2017-3
25. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: International Conference on Computer Aided Verification. pp. 243–257. Springer (2010)

26. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems. pp. 10825–10836 (2018)
27. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 41 (2019)
28. Tran, H.D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. In: 32nd International Conference on Computer-Aided Verification (CAV) (2020)
29. Tran, H.D., Musau, P., Lopez, D.M., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Parallelizable reachability analysis algorithms for feed-forward neural networks. In: 7th International Conference on Formal Methods in Software Engineering (FormaliSE2019), Montreal, Canada (2019)
30. Tran, H.D., Musau, P., Lopez, D.M., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Star-based reachability analsysis for deep neural networks. In: 23rd International Symposisum on Formal Methods (FM'19). Springer International Publishing (October 2019)
31. Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: 32nd International Conference on Computer-Aided Verification (CAV) (2020)
32. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in Neural Information Processing Systems. pp. 6369–6379 (2018)
33. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. arXiv preprint arXiv:1804.10829 (2018)
34. Weng, T.W., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Boning, D., Dhillon, I.S., Daniel, L.: Towards fast computation of certified robustness for relu networks. arXiv preprint arXiv:1804.09699 (2018)
35. Xiang, W., Musau, P., Wild, A.A., Lopez, D.M., Hamilton, N., Yang, X., Rosenfeld, J.A., Johnson, T.T.: Verification for machine learning, autonomy, and neural networks survey. CoRR **abs/1810.01989** (2018)
36. Xiang, W., Tran, H.D., Johnson, T.T.: Reachable set computation and safety verification for neural networks with relu activations. arXiv preprint arXiv:1712.08163 (2017)
37. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE transactions on neural networks and learning systems (99), 1–7 (2018)
38. Xiang, W., Tran, H.D., Johnson, T.T.: Specification-guided safety verification for feedforward neural networks. AAAI Spring Symposium on Verification of Neural Networks (2019)
39. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems. pp. 4944–4953 (2018)