Star-Based Reachability Analysis of Deep Neural Networks

Conference Paper · June 2019

CITATIONS
29

READS
2,106

8 authors, including:

Dung Hoang Tran
University of Nebraska at Lincoln
65 PUBLICATIONS **1,920** CITATIONS
SEE PROFILE

Diego Manzanas Lopez
Vanderbilt University
27 PUBLICATIONS **670** CITATIONS
SEE PROFILE

Patrick Musau
Vanderbilt University
27 PUBLICATIONS **735** CITATIONS
SEE PROFILE

Luan Viet Nguyen
University of Texas at Arlington
40 PUBLICATIONS **928** CITATIONS
SEE PROFILE

Star-Based Reachability Analysis of Deep Neural Networks

Hoang-Dung Tran[1], Diago Manzanas Lopez[1], Patrick Musau[1], Xiaodong Yang[1], Luan Viet Nguyen[2], Weiming Xiang[1], and Taylor T. Johnson[1]

[1] Institute for Software Integrated Systems, Vanderbilt University, TN, USA
[2] Department of Computer and Information Science, University of Pennsylvania, PA, USA

Abstract. This paper proposes novel reachability algorithms for both exact (sound and complete) and over-approximation (sound) analysis for deep neural networks (DNNs). The approach uses star sets as a symbolic representation of sets of states, which are known in short as stars and provide an effective representation of high-dimensional polytopes. Our star-based reachability algorithms can be applied to several problems in analyzing the robustness of machine learning methods, such as safety and robustness verification of DNNs. Our star-based reachability algorithms are implemented in a software prototype called the neural network verification (NNV) tool that is publicly available for evaluation and comparison. Our experiments show that when verifying ACAS Xu neural networks on a multi-core platform, our exact reachability algorithm is on average about 19 times faster than Reluplex, a satisfiability modulo theory (SMT)-based approach. Furthermore, our approach can visualize the precise behavior of DNNs because the reachable states are computed in the method. Notably, in the case that a DNN violates a safety property, the exact reachability algorithm can construct a complete set of counterexamples. Our star-based over-approximate reachability algorithm is on average 118 times faster than Reluplex on the verification of properties for ACAS Xu networks, even without exploiting the parallelism that comes naturally in our method. Additionally, our over-approximate reachability is much less conservative than DeepZ and DeepPoly, recent approaches utilizing zonotopes and other abstract domains that fail to verify many properties of ACAS Xu networks due to their conservativeness. Moreover, our star-based over-approximate reachability algorithm obtains better robustness bounds in comparison with DeepZ and DeepPoly when verifying the robustness of image classification DNNs.

1 Introduction

Deep neural networks (DNNs) have become one of the most powerful techniques to deal with challenging and complex problems such as image processing [15] and natural language translation [9, 16] due to its learning ability on large data sets. Recently, the power of DNNs has inspired a new generation of intelligent autonomy which makes use of DNNs-based learning enable components such as

autonomous vehicles [5] and air traffic collision avoidance systems [11]. Although utilizing DNNs is a promising approach, assuring the safety of autonomous applications containing neural network components is difficult because DNNs usually have complex characteristics and behavior that are generally unpredictable. Notably, it has been proved that well-trained DNNs may not be robust and are easily to be fooled by a slight change in the input [18]. Several recent incidents in autonomous driving (e.g., Tesla and Uber) raises an urgent need for techniques and tools that can formally verify the safety and robustness of DNNs before utilizing them in safety-critical applications.

Safety verification and robustness certification of DNNs have attracted a huge attention from different communities such as machine learning [1,2,13,17,20,24, 25,28], formal method [6,10,12,19,22,26,27], and security [7,23,24]. Analyzing the behavior of a DNN can be categorized into exact and over-approximate analyses. For the exact analysis, the SMT-based [12] and polyhedron-based approaches [22,26] are notable representatives. For the over-approximate analysis, the mixed-integer linear program (MILP) [6], interval arithmetic- [23, 24], zonotope- [20], input partition- [27], linearization- [25], and abstract-domain- [21] 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 [12]. 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.

In this paper, we propose a fast and scalable approach for the *exact* and *over-approximate* reachability analysis of DNN with ReLU activation functions using the concept of *star* set [3], or shortly "star". Star fits perfectly for the reachability analysis of DNNs due to its following essential characteristics: 1) an efficient (exact) representation of large input sets; 2) fast and cheap affine mapping operations; 3) inexpensive intersections with half-spaces and checking empty. *By utilizing star, we avoid the expensive affine mapping operation in polyhedron-based approach [22]* and thus, reduce the verification time significantly. Our approach performs reachability analysis for feedforward DNNs layer-by-layer. In the case of exact analysis, the output reachable set of each layer is a union of a set of stars. Based on this observation, *the star-based exact reachability algorithm naturally can be designed for efficient execution on multi-core platforms* where each layer can handle multiple input sets at the same time. In the case of over-approximate analysis, the output reachable set of each layer

is a single star which can be constructed by doing *point-wise* over-approximation of the reachable set at all neurons of the layer.

We evaluate the proposed algorithms in comparison with the polyhedron approach [22], Reluplex [12], zonotope [20] and abstract domain [21] approaches on safety verification of the ACAS Xu neural networks [11] and robust certification of image classification DNN. The experimental results show that our exact reachability algorithm can achieve 19 times faster than Reluplex when running on multi-core platform and > 70 times faster than the polyhedron approach. Notably, our exact algorithm 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 averagely 118 times faster than Reluplex. It successfully verifies many safety properties of ACAS Xu networks while the zonotope and abstract domain approaches fail due to their large over-approximation errors. Our over-approximate reachability algorithm also provides a better robustness certification for image classification DNN in comparison with the zonotope and abstract domain approaches. 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; 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 $W_{k,k-1}$, 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:

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

where W_k and b_k are the weight matrix and bias vector 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:*

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

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 if and only if $P(\alpha)$ is empty.

Proposition 1. *Any bounded convex polyhedron $\mathcal{P} \triangleq \{x \mid Cx \leq d, x \in \mathbb{R}^n\}$ can be represented as a star.*

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.*

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

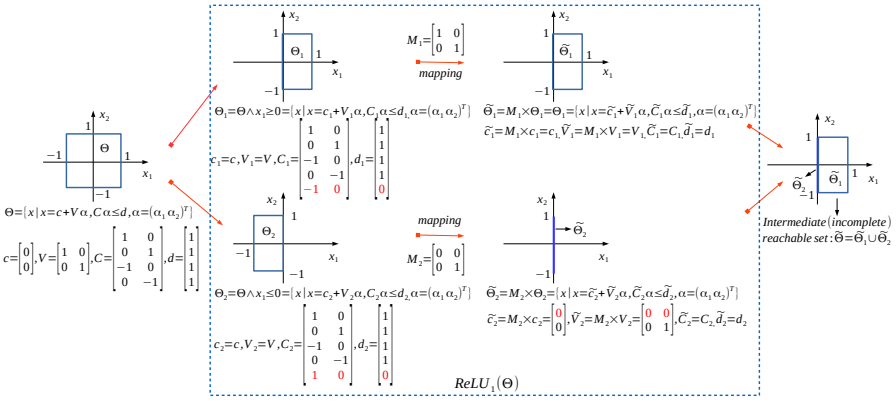


Fig. 1: An example of a stepReLU operation on a layer with two neurons.

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.*

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

3 Star-Based Reachability Analysis of FNNs

3.1 Exact and Complete Analysis

Since any bounded convex polyhedron can be represented as a star, 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, 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 $\mathcal{R}_L = ReLU_n(ReLU_{n-1}(\cdots ReLU_1(\Theta)))$..

The stepReLU operation on the i^{th} neuron, i.e., $ReLU_i(\cdot)$, works as follows. First, the input star set Θ is decomposed into two subsets $\Theta_1 = \Theta \wedge x_i \geq 0$ and $\Theta_2 = \Theta \wedge x_i < 0$. Note that from Proposition 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.

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 which can be done efficiently by solving n-linear programming problems.

The star-based exact reachability algorithm 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 $\Theta_i = \langle c_i, V_i, P_i\rangle$, the main procedure maps the element with the layer weight matrix W and bias vector b which results a new star $I_1 = \langle Wc_i + b, WV_i, P_i\rangle$. 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 operations on the new star I_1, i.e., iteratively calls *stepReLU* 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.

Lemma 1. *The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN computed by Algorithm 3.1 is $\mathcal{O}(2^N)$.*

Lemma 2. *The worst-case complexity of the number of constraints of a star in the reachable set of an N-neuron FNN computed by Algorithm 3.1 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.*

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.

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 $\mathcal{C}_\Theta = \cup_{i=1}^{k} \langle c, V, \bar{P}_i\rangle$, $\bar{P}_i \neq \emptyset$.*

Algorithm 3.1 Star-based exact reachability analysis for one 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)
  $R = \emptyset$
  parfor $i = 1 : N$ **do** ▷ parallel for loop
    $I_1 = W * \Theta_i + b = \langle Wc_i + b, WV_i, P_i \rangle$
    $R_1 = reachReLU(I_1)$, $R = R \cup R_1$
  end parfor
procedure R_1 = reachReLU(I_1)
  $In = I_1$
  $[lb, ub] = In.getRange$ ▷ get ranges of all input variables
  $map = find(lb < 0)$ ▷ construct computation map
  $m = length(map)$ ▷ minimized number of stepReach operations
  for $i = 1 : m$ **do** ▷ n is the number of neurons of the layer
    $In = stepReLU(In, map(i), lb(i), ub(i))$ ▷ stepReLU operation
  $R_1 = In$
procedure $\tilde{R}$ = stepReLU$(\tilde{I}, i, lb_i, ub_i)$
  $\tilde{R} = \emptyset$, $\tilde{I} = [\tilde{\Theta}_1 \ \cdots \ \tilde{\Theta}_k]$ ▷ intermediate star input and output sets
  for $j = 1 : k$ **do**
    $\mathcal{R}_1 = \emptyset$, $M = [e_1 \ e_2 \ \cdots e_{i-1} \ 0 \ e_{i+1} \ \cdots \ e_n]$
    if $lb_i \geq 0$ **then** $\mathcal{R}_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j \rangle$
    if $ub_i \leq 0$ **then** $R_1 = M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j \rangle$
    if $lb_i < 0$ & $ub_i > 0$ **then**
      $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \rangle$, $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j \rangle$
      $\mathcal{R}_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
    $\tilde{R} = \tilde{R} \cup R_1$

3.2 Over-approximate Analysis

Although the exact reachability analysis can compute the exact behavior of 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 propose an over-approximation reachability algorithm for FNNs. The main benefit of this approach is that the reachable set of each layer is only a single star that can be constructed efficiently by using an over-approximation of the ReLU activation function at all neurons in the layer. Importantly, our star-based over-approximate reachability algorithm is much less conservative than the zonotope-based [20] and abstract domain [21] based approaches in the way of approximating the ReLU activation function, shown in Figure 2. The zonotope-based approach [20] over-approximates the ReLU activation function by a minimal parallelogram while the abstract-domain approach [21] 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 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 [21], for exam-

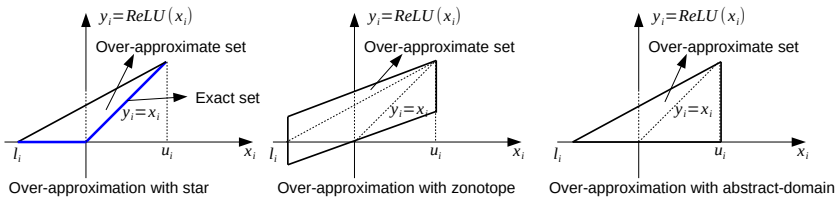


Fig. 2: Over-approximation of ReLU functions with different approaches.

ple, in Figure 2, these constraints are $y_i \geq 0,\ y_i \leq u_i(x_i - l_i)/(u_i - l_i)$. To obtain a tighter over-approximation, our approach uses three constraints for the output y_i instead. The over-approximation rule for a single neuron is given as follows,

$$\begin{cases} y_i = x_i & \text{if } l_i \geq 0 \\ y_i = 0 & \text{if } u_i \leq 0 \\ y_i \geq 0,\ y_i \leq \frac{u_i(x_i - l_i)}{u_i - l_i},\ y_i \geq x_i & \text{if } l_i < 0 \text{ and } u_i > 0 \end{cases}$$

where l_i and u_i is the lower and upper bounds of x_i.

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 performing the above over-approximation rule. The over-approximate reachability algorithm for a single layer of FNN using star set given in Algorithm 3.2 works as follows. Given a star input set Θ, the algorithm computes the affine mapping of the input set using the layer's weight matrix and bias vector. The resulting star set is the input of a sequence of n $approximate-stepReLU$ operations. An approximate-stepReLU operation first computes the lower and upper bounds of the state variable $x[i]$ w.r.t the i^{th} neuron. If the lower bound is not negative (line 11), 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 (line 12), 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 (line 13), the approximate-stepReLU operation introduces a new variable α_{m+1} to capture the over-approximation of ReLU function at the i^{th} neuron. As a result, the obtained intermediate reachable set has one more variable and three more linear constraints in the predicate in comparison with the corresponding input set. From this observation, we can see that 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.

Algorithm 3.2 Star-based over-approximate reachability analysis for one layer.

Input: $I = \Theta = \langle c, V, P \rangle$, W, b ▷ star input set, weight matrix, bias vector
Output: R ▷ over-approximate reachable set

procedure R = ApproxLayerReach(I, W, b)
  $I_1 = W * \Theta + b = \langle Wc + b, WV, P \rangle$
  $In = I_1$
  for $i = 1 : n$ **do** ▷ n is the number of neurons of the layer
    $In = approxStepReLU(In, i)$ ▷ i^{th} approximate-stepReLU operation
  $R_1 = In$
procedure $\tilde{R}$ = ApproxStepReLU$(\tilde{I}, i)$
  $\tilde{I} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P} \rangle$
  $[l_i, u_i] = \tilde{\Theta}.getRange(i)$ ▷ range of $x[i]$, i.e., $l_i \leq x[i] \leq u_i$
  $M = [e_1\ e_2\ \cdots e_{i-1}\ 0\ e_{i+1}\ \cdots\ e_n]$
  if $l_i \geq 0$ **then** $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P} \rangle$
  if $u_i \leq 0$ **then** $\tilde{R} = M * \tilde{\Theta} = \langle M\tilde{c}, M\tilde{V}, \tilde{P} \rangle$
  if $l_i < 0$ & $u_i > 0$ **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}
    $C_1 = [0\ 0\ \cdots\ 0\ \text{-}1] \in \mathbb{R}^{1 \times m+1}$, $d_1 = 0$ ▷ $\alpha_{m+1} \geq 0 \Leftrightarrow C_1\alpha' \leq d_1$
    $C_2 = [V[\tilde{i}, :]\ \text{-}1] \in \mathbb{R}^{1 \times m+1}$, $d_2 = -\tilde{c}[i]$ ▷ $\alpha_{m+1} \geq x[i] \Leftrightarrow C_2\alpha' \leq d_2$
    $C_3 = [\frac{-u_i}{u_i - l_i} \times V[\tilde{i}, :]\ \ 1]$, $d_3 = \frac{u_i l_i}{u_i - li} \times (1 - \tilde{c}[i])$ ▷ $\alpha_{m+1} \leq \frac{u_i(x[i] - l_i)}{u_i - l_i} \Leftrightarrow C_3\alpha' \leq d_3$
    $C_0 = [\tilde{C}\ 0_{m \times 1}]$, $d_0 = \tilde{d}$
    $C' = [C_0; C_1; C_2; C_3]$, $d' = [d_0; d_1; d_2; d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$ ▷ output set's predicate
    $c' = M\tilde{c}$, $V' = M\tilde{V}$, $V' = [V'\ e_i]$ ▷ $y[i] = ReLU(x[i]) = \alpha_{m+1}$
    $\tilde{R} = \langle c', V', P' \rangle$

3.3 Reachability Algorithm for FNNs

The reachability analysis of a FNN is done layer-by-layer in which the output set of the previous layer is the input set of the next layer. The reachability algorithm for a FNN is summarized in Algorithm 3.3.

4 Evaluation

In this section, we evaluate the proposed star-based reachability algorithms in comparison to existing state-of-the-art approaches including exact (sound and complete) SMT-based (Reluplex [12]) and polyhedron-based [22] approaches, as well as over-approximate approaches, such as those using zonotopes [20] and abstract domains [21]. To clarify intuitively the benefit of our approach, we re-implement the zonotope- and abstract-domain based approaches in our tool called NNV. This allows the visualization of the over-approximate reachable set of these approaches. The evaluation and comparison are done by verifying safety of the ACAS Xu DNNs [11] and the robustness of image classification

Algorithm 3.3 Reachability analysis for a FNN.

Input: $I = \Theta = \langle c, V, P \rangle$, $L = [L_1\ L_2\ \cdots\ L_k]$, $scheme$ ▷ star input set, network's layers, reachabiltiy scheme
Output: R ▷ reachable set

```
1: procedure R = REACH(I, L, scheme)
2:     In = I
3:     for i = 1 : k do ▷ k is the number of layers on the network
4:         if scheme = exact then In = L_i.LayerReach(In)
5:         else if scheme = approx then In = L_i.ApproxLayerReach(In)
6:     R = In
```

DNN against adversarial attacks. All results presented in this section and their corresponding scripts are available online[3].

4.1 Safety Verification for ACAS Xu DNNs

The ACAS Xu networks are DNN-based advisory controllers that map the sensor measurements to advisories in the Airborne Collision Avoidance System X [11]. It consists of 45 DNNs which are trained to replace the traditional memory-consuming lookup table. Each DNN denoted by N_{x_y} has 5 inputs, 5 outputs, and 6 hidden layers of 50 neurons. The detail about ACAS Xu networks and their safety properties are given in Appendix. The experiments in this case study are done using Amazon Web Services Elastic Computing Cloud (EC2), on a powerful `m5a.24xlarge` instance with 96 cores and 384 GB of memory. The verification results are presented in Table 1. We used 90 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. The detail of the reachable set computation time and the safety checking time can be found in the verification results.

Exact star-based method. The experimental results show that the exact star-based approach is on average > 70 times faster than the polyhedron-based approach and 18.9 times faster than Reluplex when using parallel computing. Impressively, it can even achieve 61.8 faster than Reluplex when verifying property ϕ_2 on N_{5_1} network. This improvement comes from the fact that star set

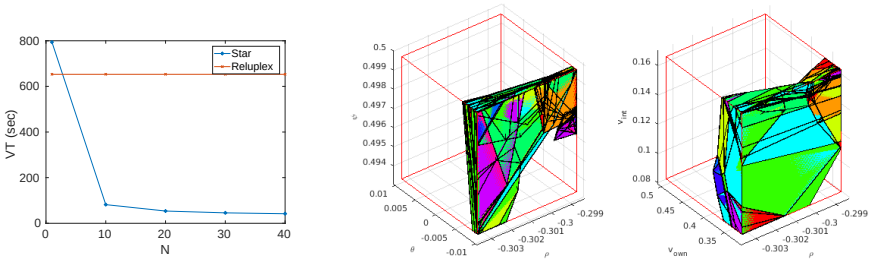
[3] https://github.com/verivital/nnv/tree/master/nnv0.1/examples/Submission/FM2019

Prop.	ID	Exact methods						Over-approximation methods								
		Res.	Rel	Poly	Star			Zonotope			Abstract-Domain			Star		
			VT	VT	VT	r^1_{ES}	r^2_{ES}	Res.	VT	r_Z	Res.	VT	r_{AD}	Res.	VT	r_{AS}
ϕ_1	N_{1_1}	safe	5986	TO	1481.2	**4.04x**	—	safe	0.07	**85514x**	safe	6.22	**962.4x**	safe	13.79	**434.1x**
ϕ_2	N_{1_3}	safe	1102	TO	77.3	**14.3x**	**> 47x**	UN	0.06	**18367x**	UN	6.3	**174.9x**	UN	6.01	**183.4x**
	N_{2_1}	safe	1173	TO	51.57	**22.8x**	**> 70x**	UN	0.062	**18919x**	UN	5.72	**205.1x**	safe	5.91	**198.5x**
	N_{2_2}	safe	634	TO	35.8	**17.7x**	**> 101x**	UN	0.084	**7548x**	UN	5.83	**108.8x**	safe	5.8	**109.3x**
	N_{2_3}	safe	1014	TO	36.1	**28.1x**	**> 100x**	UN	0.073	**13890x**	UN	5.89	**172.2x**	safe	5.79	**175.7x**
	N_{5_1}	safe	1097	TO	17.76	**61.8x**	**> 202x**	UN	0.081	**13543x**	UN	5.85	**187.5x**	safe	5.76	**190.5x**
ϕ_3	N_{1_5}	safe	393	1520.29	33.39	**11.8x**	**45.5x**	UN	0.0796	**4937x**	UN	6.1	**64.4x**	safe	5.89	**66.7x**
	N_{2_2}	safe	451	TO	43.66	**10.3x**	**> 83x**	UN	0.056	**8054x**	UN	5.57	**81x**	safe	5.66	**79.7x**
	N_{2_3}	safe	293	2759	37.23	**7.8x**	**74.1x**	UN	0.08	**3663x**	UN	6.04	**74.7x**	safe	5.66	**79.7x**
	N_{2_8}	safe	653	1152.6	31.04	**21x**	**37.1x**	safe	0.102	**6401x**	safe	6.36	**102.7x**	safe	7.47	**87.4x**
	N_{2_9}	safe	61	233	6.33	**9.63x**	**36.8x**	safe	0.065	**938.5x**	safe	5.76	**10.6x**	safe	5.88	**10.4x**
	N_{3_7}	safe	357	1115.6	14	**25.5x**	**79.7x**	UN	0.085	**4200x**	safe	5.88	**60.7x**	safe	6	**59.5x**
	N_{3_8}	safe	149	770	15.2	**9.8x**	**50.7x**	UN	0.08	**1862x**	UN	5.53	**26.9x**	safe	5.83	**25.6x**
	N_{3_9}	safe	715	1664.4	40.9	**17.5x**	**40.7x**	UN	0.076	**9408x**	UN	6.25	**114.4x**	safe	6.13	**116.6x**
	N_{4_9}	safe	489	1098.7	22.2	**22x**	**49.49x**	safe	0.049	**9980x**	safe	5.3	**92.3x**	safe	5.57	**87.8x**
	N_{5_1}	safe	585	1005.3	18.43	**31.74x**	**54.5x**	UN	0.069	**8479x**	UN	5.8	**100.9x**	safe	5.83	**100.3x**
	N_{5_7}	safe	42	275.1	7.69	**5.5x**	**35.8x**	safe	0.054	**778x**	safe	5.5	**7.6x**	safe	5.54	**7.6x**
Average time improvement						**≈ 18.9x**	**> 70x**			**≈ 12734x**			**≈ 150x**			**≈ 118.4x**

Table 1: Safety verification results of ACAS Xu networks. Notation: TO is 'Time-out', **Rel** states for 'Reluplex', **Poly** is the Polyhedron method, UN states for unknown, VT is the verification time in seconds, and r^1_{ES}, r^2_{ES} are the verification time improvement of the exact, star-based method compared with Reluplex and the polyhedron-based methods; r_Z, r_{AD} and r_{AS} respectively are the verification time improvement of the zonotope-, abstract domain- and over-approximate star-based methods compared with Reluplex. The computation time limitation for polyhedron-based method (run on Amazon cloud) was set to be 1 hour while for Reluplex, it was set at 24 hours.

that is very efficient in affine mapping and intersection with half-spaces which are crucial operations for reachable set computation and safety checking. Therefore, the exact star-based method is much more efficient than the polyhedron-based approach [22]. In addition, the exact star-based algorithm is well-designed and optimized (i.e., minimize the number of stepReLU operations) for efficiently running on multi-core platforms while Reluplex does not exploit the power of parallel-computing. Figure 3a describes the benefits of parallel computing. The figure shows that when a single core is used for a verification task, our approach takes 790.07 seconds which is a little bit slower than Reluplex with 653 seconds. However, our verification time drops quickly to 80.45 seconds, which is 8 times faster than Reluplex, when we use 10 cores for the computation.

Zonotope-based method [20]. 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 less than 0.1 seconds, for example, the zonotope-based method successfully verifies property ϕ_3 on N_{5_7} network in 0.054 seconds and the corresponding reachable set



(a) Verification times for property ϕ_3 on N_{2_8} network with different number of cores.

(b) The (normalized) complete counter input set for property ϕ_3' on N_{2_8} network is a part of the normalized input set (red boxes).

Fig. 3: Verification time reduction with parallel computing and complete counter input set construction.

is depicted in Figure 4a. Although the zonotope-based method is time-efficient (on average 12734 times faster than Reluplex), it fails to verify the safety of many networks due to its huge over-approximation error. For example, Figure 4b describes the reachable set obtained by the zonotope-based method for N_{3_8} network w.r.t property ϕ_3. As shown in the figure, the obtained reachable set is too conservative and can not be used for safety verification of the network. The main reason that makes the zonotope approach fast is that to do reachability analysis, we need to compute the lower and upper bounds of all state $x[i]$ of all neurons in each layer. This information can be obtained straightforwardly in the zonotope method while in the other approaches, i.e., abstract-domain and star-based approaches, this is equivalent to solving n linear optimization problems where n is the number of neurons at that layer. The time for solving these optimization problems increase over layers since the number of constraints in the reachable set increases. Therefore, despite the a large over-approximation error, the zonotope-based method is *time-efficient* when dealing with large and deep neural networks.

Abstract-domain based method [21] The over-approximation method using abstract-domain is 150 times faster than Reluplex on average. It is also much less conservative than the zonotope-based method as can be seen from Figure 4. However, the reachable set computed by the abstract-domain based method is still too conservative which makes this approach fail to verify the safety properties of many ACAS Xu networks.

Over-approximate star-based method. The experiments show that our over-approximate star-based approach can obtain tight reachable sets for many networks compared to the exact sets. Therefore, our over-approximate approach successfully verifies safety properties of most of ACAS Xu networks. Notably, it is on average 118.4 times faster than Reluplex. Impressively, it is 434 times faster than Reluplex when verifying property ϕ_1 on N_{1_1} network. In comparison with the zonotope and abstract-domain approaches, our method is timing-comparable with the abstract-domain method and slower than the zonotope method. How-

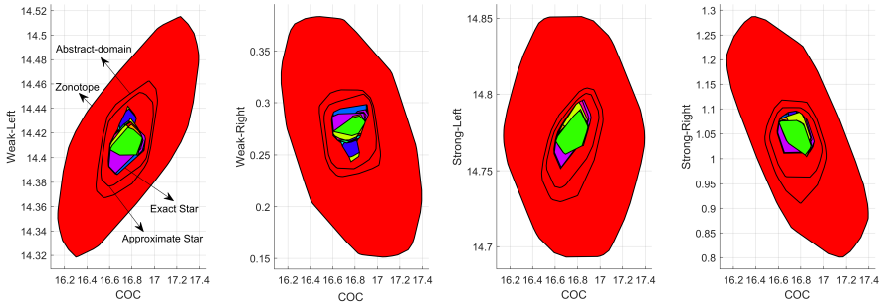
ever, our results is much less conservative than those obtained by the zonotope and abstract-domain methods which are shown in Figure 4. This makes our approach applicable for safety verification of many ACAS Xu networks where the zonotope and abstract-domain methods cannot verify.

Benefits of computing the reachable set. The reachable set computed in our NNV tool are useful for intuitively checking the safety properties of the network. For example, Figure 4a describes the behaviors of N_{5_7} network corresponding to property ϕ_3 requiring that the output COC is not the minimal score. From the figure, one can see that the COC is not the minimal score and thus, property ϕ_3 holds on N_{5_7} 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 abstract-domain-based methods. Last but not least, the reachable set is useful in the case that we need to verify a set of safety properties corresponding to the same input set. In this case, once the reachable set is obtained, it can be re-used to check different safety properties without rerunning the whole verification procedure as Reluplex does, and thus helps saving a significant amount of time.

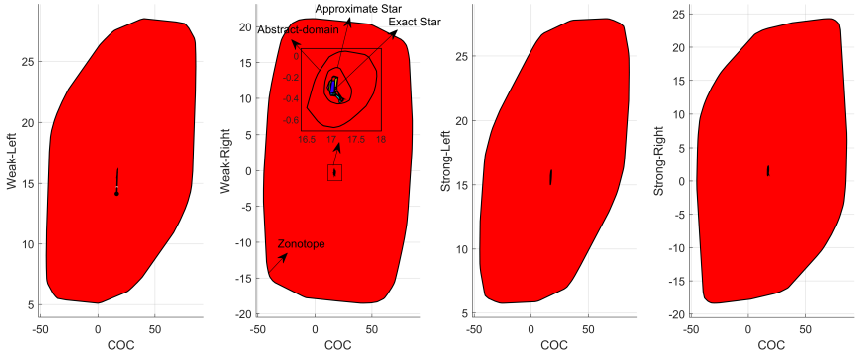
Complete counter example input set construction. Another strong advantage of our approach in comparison with other existing approaches is, in the case that a neural network violates its safety specification, our exact, star-based method can construct a *complete counter input set* that leads the neural network to the unsafe region. The complete counter input set can be used as a adversarial input generator [4, 8] 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 $\phi_3' \triangleq \neg(COC \geq 15.8 \wedge StrongRight \leq 15.09)$ on N_{2_8} network with the same input constraints as in property ϕ_3. Using the available reachable set of N_{2_8} network, we can verify that the above property ϕ_3' 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 3b is a part of the input set that contains all counter inputs that make the neural network unsafe.

4.2 Maximum Robustness Certification of Image Classification DNNs

Robustness certification of DNNs becomes more an 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



(a) Reachable sets of N_{5_7} network w.r.t property ϕ_3 with different methods. All methods successfully verify the property.



(b) Reachable sets of N_{3_8} network w.r.t property ϕ_3 with different methods. The zonotope (used in DeepZ [20]) and abstract-domain (used in DeepPoly [21]) methods cannot verify the property due to large over-approximation error.

Fig. 4: Conservativeness of the reachable sets obtained by different methods.

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 95%) using the well-known MNIST data set consisting of 60000 images of handwritten digits with a resolution of 28×28 pixels [14]. The trained networks have 784 inputs and a single output with expected value from 0 to 9. We find the maximum robustness verification value δ_{max} for the networks on an image of digit one with the assumption that there is a δ_{max}-bounded disturbance modifying the (normalized) values of the input vector x at all pixels of the image, i.e., $|x[i] - x'[i]| \leq \delta_{max}$. The result are presented in Table 2. We note that the polyhedron and Reluplex approaches are not applicable for these networks because they cannot deal with high dimensional input space. The table shows that our approximate star approach produces larger upper bounds of the robustness values of the networks with

Net	Parameters	Tol	δ_{max}			
			Zonotope	**Approximate-Star**	**Abstract-Domain**	**Exact-Star**
N_1	k=5, N = 140	0.00001	0.00248	0.00269	0.00255	0.0036
N_2	k=5, N = 250	0.0001	0.0046	0.0055	0.0051	TimeOut
N_3	k=2, N = 1000	0.0001	0.0045	0.0053	0.0049	TimeOut
N_4	k = 1, N = 2000	0.0001	0.0015	0.0015	0.0015	TimeOut
N_5	k = 1, N = 4000	0.0001	0.0019	0.0019	0.0019	TimeOut

Table 2: 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.

many layers. For single layer networks, our approach gives the same results as the zonotope [20] and the abstract domain [21] methods. The exact-star method only can find the maximum robustness value for small network, i.e., N_1 and it reaches timeout (set as 1 hour) for the other networks.

5 Conclusion and Future Work

We have proposed two reachability analysis algorithms for DNNs using star sets, one that is exact (sound and complete) but has worse scalability 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 polyhedron 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 future work, we are extending the proposed methods for the convolutional neural networks (CNN) and recurent neural networks (RNN), as well as improving scalability for other types of activation functions such as tanh and sigmoid.

Acknowledgments

We thank Gagandeep Singh from ETH Zurich for his help on explaining DeepZ and DeepPoly methods as well as running his tool ERAN. We also thank Shiqi Wang from Columbia University for his explanation about his interval propagation method and Guy Katz from The Hebrew University of Jerusalem for his explanation about ACAS Xu networks and Reluplex. The discussion with Gagandeep Singh, Shiqi Wang and Guy Katz is the main inspiration of our work in this paper.

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. 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)
5. 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)
6. Dutta, S., Jha, S., Sanakaranarayanan, S., Tiwari, A.: Output range analysis for deep neural networks. arXiv preprint arXiv:1709.09130 (2017)
7. 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)
8. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
9. 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)
10. 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)
11. Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. arXiv preprint arXiv:1810.04240 (2018)
12. 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)
13. Kouvaros, P., Lomuscio, A.: Formal verification of cnn-based perception systems. arXiv preprint arXiv:1811.11373 (2018)
14. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
15. 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)
16. 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)
17. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. arXiv preprint arXiv:1706.07351 (2017)

18. 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)
19. 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)
20. 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)
21. 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)
22. 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)
23. 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)
24. 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)
25. 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)
26. 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)
27. 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)
28. 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)

A Appendix: Proofs

A.1 Proof of Proposition 1

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 $P(\alpha) \triangleq C\alpha \leq d$.

A.2 Proof of Proposition 2

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.

A.3 Proof of Proposition 3

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 $\bar{P} = P \wedge P', P'(\alpha) \triangleq H \times V_m \times \alpha \leq g - H \times c$.

A.4 Proof of Lemma 1

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 produced by Algorithm 3.1 is $2^{n_{L_1}} \times \cdots \times 2^{n_{L_k}} = 2^{n_{L_1}+\cdots+n_{L_k}} = 2^N$.

A.5 Proof of Lemma 2

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 $\mathcal{O}(N)$.

A.6 Proof of Theorem 1

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 safety specification. 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.

A.7 Proof of Theorem 2

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 $\mathcal{C}_\Theta = \cup_{i=1}^{k} \langle c, V, \bar{P}_i \rangle,\ \bar{P}_i \neq \emptyset$.

B Appendix: ACAS Xu Case Study Details

ACAS Xu 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.

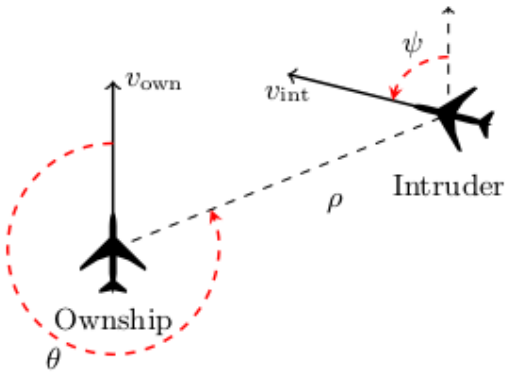


Fig. 5: Vertical view of a generic example of the ACAS Xu benchmark set. [12]

B.1 Properties

We present a list of properties that are checked to be satisfied or unsatisfied by ACAS Xu benchmarks. [12]

- **Property ϕ_1.**
  - If the intruder is distant and is significantly slower than the ownship, the score of a *COC* advisory will always be below a certain fixed threshold.
  - The desired output property is that the score for COC is at most 1500.
  - It has 3 input constraints: $\rho \geq 55947.691$, $v_{own} \geq 1145$, $v_{int} \leq 60$.
- **Property ϕ_2 (is the property ϕ_3 in Reluplex).**
  - 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_3 (is the property ϕ_4 in Reluplex).**
  - If the intruder is directly ahead and is moving away from the ownship but at a lower speed than that of the ownship, the score for *COC* will not be minimal.
  - The desired output property is that the score for COC is not the minimal score.
  - It has 5 input constraints: $1500 \leq \rho \leq 1800$, $\theta \leq |0.06|$, $\psi = 0$, $v_{own} \geq 1000$, $700 \leq v_{int} \leq 800$.