Reachability Analysis of Deep ReLU Neural Networks using Facet-Vertex Incidence

Xiaodong Yang, Tomoya Yamaguchi, Hoang-Dung Tran
Bardh Hoxha, Taylor T Johnson and Danil Prokhorov

ABSTRACT

Deep Neural Networks (DNNs) have demonstrated to be powerful machine learning models for approximating complex functions. In this work, we provide an exact reachability analysis method for DNNs with Rectified Linear Unit (ReLU) activation functions. At its core, our set-based method utilizes a facet-vertex incidence matrix, which represents a complete encoding of the combinatorial structure of convex sets. When a safety violation is detected, our approach provides backtracking which determines the complete input set that caused the safety violation. The performance of our method is evaluated and compared to other state-of-the-art methods by using the ACAS Xu flight controller and other benchmarks.

ACM Reference Format:
Xiaodong Yang, Tomoya Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T Johnson and Danil Prokhorov. 2018. Reachability Analysis of Deep ReLU Neural Networks using Facet-Vertex Incidence. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The versatility and ease-of-use of DNNs has made them popular in a number of industrial systems, including safety-critical systems such as autonomous systems. However, a major barrier in realizing trustful autonomy is the lack of scalable methods for safety verification. Recently, there has been significant effort to develop methods to establish robustness and formal guarantees of Learning-Enabled Components (LECs) [6, 8, 12, 13, 17, 20, 24, 27–30, 32, 33, 36–38].

One of the main approaches for safety verification is through Reachability Analysis. There are two primary classes of reachability analysis methods to formally analyze DNNs, over-approximation and exact analysis. The approximation methods are mainly based on Mixed-Integer Linear Programs (MILPs) [6, 19, 21], zonotopes [8], abstract domain [28], global optimisation [25], regressive polynomial [5], network conversion [15], unified framework [4] and linearization [34, 38]. These methods provide soundness but not completeness guarantees. An exception is the work in [6], where the authors provide a MILP-based method is sound and complete but is limited to neural networks with only one output. The common strategy of the over-approximation methods is to apply a conservative convex domain to over-approximate the multiple linearities of the ReLU function $\max(0, x)$ w.r.t. the input space in each neuron. Thus when the input space spans both the positive and negative domains where the ReLU exhibits different linearities, it does not need to be separately considered. Such over-approximation methods generate a single reachable domain w.r.t. an input domain and are capable of efficiently analyzing large scale neural networks, but the conservativeness of the over approximation is normally large.

The methods for exact analysis are based on Satisfiability Modulo Theory (SMT) [17, 18], Interval Arithmetic [32, 33] and Set-based methods [2, 31, 36]. The strategy of Reluplex [17] is to extend the simplex method to handle the piece-wise linear ReLU activation function. The Marabou [7, 18] method is an extension of Reluplex. It supports arbitrary piecewise-linear activation functions, parallel computation, etc. Another work, Neurify [32, 33] is based on interval arithmetic to over approximate the bounds on the outputs. The method iteratively refines the over-approximation of the output reachable sets by bisecting its input range, and it doesn't terminate until a counterexample is found or the network is verified safe. In [2, 31, 36], the authors conduct reachability analysis by using polytope and star set representations. Different from the SMT and Interval Arithmetic methods, they compute the exact output reachable sets of a neural network. This provides more information to the practitioner on the neural network's behavior. The advantage of such reachability analysis is not only limited to the verification of neural network, but also can facilitate the training of adversarially-robust neural networks [1, 22, 35].

In this paper, we propose a set-based method for exact reachability analysis of DNNs. We utilize a Facet-Vertex Incidence Matrix (FVIM), which represents a complete encoding of the combinatorial structure of convex sets. The FVIMs are mathematical strucutures with very useful properties for set manipulation. The vertices of the set can be directly utilized to determine whether the input set spans both negative and positive input ranges of the ReLU function in neurons, such that the LP used in [2, 31, 36] can be avoided and the problem can be solved more efficiently. When the input set spans the two input ranges of the ReLU function, it can also be quickly split into two subsets. This speeds up the computation of the reachable set significantly in comparison with the aforementioned exact methods. An additional feature of the proposed method is that it enables output-to-input backtracking. This enables the computation of the complete input set that caused a safety violation. This is very useful for debugging and retraining the network. Finally, the performance of our method is evaluated and compared to other state-of-the-art methods by using the ACAS Xu DNNs and other benchmarks.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

2 PRELIMINARIES

2.1 Neural Network Verification

DNNs consists of one input layer, multiple hidden layers, and one output layer. Each layer contains multiple neurons which are interconnected with neurons in the next layer by weights and bias in a feed-forward way. The output of each neuron is associated with three components: its input weight ω, input bias b and the activation function f, namely:

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

where ω_{ij} and b_j are respectively the weight and bias from the jth neuron of the previous layer to the ith neuron of the current layer, and x_j is an input to this neuron and also the output of jth neuron in the front layer, and y_i is the output of the ith neuron. In this paper, we consider networks that contain ReLU activation functions which are defined as $\mathrm{ReLU}(x) = \max(0, x)$. Let $W_{(k,k\text{-}1)}$, b_k denote the weight matrix, the bias vector between the (k-1)th layer and kth layer, and X_k be its input consisting of elements x_j, then the output of the kth layer will be Eq. 1. For the first hidden layer, its input X_1 is equal to the input to the network X_0. The output of one layer is also an input of the next layer. Therefore, given an input X_0, the output Y_k of the kth layer will be Eq. 2.

$$L(X_k) = \mathrm{ReLU}(W_{(k,k\text{-}1)} X_k + b_k) \tag{1}$$

$$Y_k = L_k(L_{k-1}(\dots(L_1(X_0)))) \tag{2}$$

Definition 2.1 (Reachable Sets and Reachable Domain of Neural Networks). *Given a neural network $\mathcal{N}$ and an input set $P \in \mathbb{R}^{n_{in}}$, a* ***reachable set*** *$S \in \mathbb{R}^{n_{out}}$ indicates a set where $\forall y \in S, \exists x \in P$ and $y = \mathcal{N}(x)$. The computation of reachable sets is $\{S_1, S_2, \dots, S_l\} = Reach(\mathcal{N}, P)$. The* ***reachable domain*** *$\mathcal{D}$ is the union of all the computed reachable sets $\bigcup_{i=1}^{l} S_i$ where $\forall x \in P, y = \mathcal{N}(x)$ and $y \in \mathcal{D}$ besides the condition for reachable sets.*

The input set P_0 is constructed with the norm L_∞ bound and the input to the kth layer is denoted as P_k. We also denote the linear transformation by $W_{(k,k\text{-}1)}$, b_k on P_k as a function $\mathcal{T}_k(P_k)$ and the following operation from the ReLU functions of neurons as $\mathcal{R}_k(\cdot)$. Therefore, the whole process of the kth layer w.r.t. the input set P_k can be denoted as

$$O_k = \mathcal{R}_k(\mathcal{T}_k(P_k)) \tag{3}$$

The $\mathcal{T}_k$ function outputs one affine transformed set P'_k w.r.t. an input set P_k. While the $\mathcal{R}_k$ may yield multiple output sets due to the the different linearities of the ReLU function w.r.t. the input set. Finally, O_k is passed on as an input set to the next layer.

The reachable sets are also associated with the *linear region*s of neural networks [10, 23, 26]. A *linear region* of a piecewise linear function $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ refers to a maximum convex subset of an input set in $\mathbb{R}^n$, on which the function F is linear. As described above, a set is either affine transformed by weights and bias, or divided into subsets to match the different linearities of ReLU functions in each neuron. Thus the DNN exhibits unique linearity for each output reachable set over a subspace of the input set. Those subspaces are *linear region*s. Our method takes advantage of this property in the back-tracing process, where it tracks back to the input space given a particular output set. The details will be introduced in Section 5.

The verification of a neural network using reachability analysis is defined as follows.

Definition 2.2 (Verification of Neural Networks). *Given a neural network $\mathcal{N}$, an input set $P_0 \in \mathbb{R}^{n_{in}}$ and an unsafe domain $\mathcal{U} \in \mathbb{R}^{n_{out}}$. The verification problem is to determine whether $(\bigcup Reach(\mathcal{N}, P)) \cap \mathcal{U} = \emptyset$.*

2.2 Facet-Vertex Incidence Matrix

A convex polytope is a convex subset $P \in \mathbb{R}^d$ and can be represented by the vertex representation (V-rep) or the half-space representation (H-rep). The V-rep represents P as a finite number of extreme points (vertices), whereas the H-rep represents P as the intersection of a set of closed halfspaces (in terms of linear inequalities).

In [36], the reachability analysis of neural networks utilizes both the V-rep and the H-rep to compute reachable sets. The V-rep of P is utilized to conduct affine transformations between layers, and the H-rep is utilized for the computation of new subsets while considering the linear constraints from the input range in ReLU neurons. The iterative process, which requires the back-and-forth conversion between representations, is called the enumeration problem in computational geometry and has a high computational complexity. In [2, 30], the authors use the Star set based on the H-rep and can avoid the conversion issue. However, the computation for each neuron involves a large number of LP processes, which has a significant impact on its efficiency. Other works [8, 27] utilize zonotope representations, which are centrally symmetric polytope representations. The issue with these approaches is that with large inputs, the over-approximation in every layer is overly conservative and this increases exponentially with every layer.

In this work we utilize a Facet-Vertex Incidence Matrix to encode the complete combinatorial structure of a polytope. The FVIM enables quick operations over convex polytopes. For example, the containment relation is completely encoded. This enables fast affine transformations and can be used to derive vertex adjacency for fast division by hyperplanes. In the rest of the section, we provide a formal definition of the FVIM. For details, see [9, 11].

Consider a 3-dimensional cube as illustrated in Fig. 1. The graph below with blue blocks describes the containment relation between the facets and vertices and is equivalent to the FVIM shown in Eq. 5. In FVIM, f_* denotes the facet/plane and v_* denotes the vertex of the cube. For instance, f_1 is the $2\text{-}face : plane_{1,2,3,4}$.

We define polytopes using FVIM $\mathcal{F}$ and a vertices V as follows:

$$P = \langle \mathcal{F}, V \rangle \tag{4}$$

In order to define a Facet-Vertex Incidence Matrix, we first need to define the notion of a supporting hyperplane and face of a polytope.

Definition 2.3 (Supporting Hyperplane). *A hyperplane H denoted by $a^\top x = b$ is a* ***supporting hyperplane*** *of polytope P if one of its closed halfspaces, $a^\top x \leq b$ or $a^\top x \geq b$ contains P.*

Definition 2.4 (Face & Facet). *The* ***face*** *of a d-dimensional polytope P is an intersection of P with a supporting hyperplane. When the dimension of $\mathrm{aff}(P \cap H)$ is k, the face is denoted as* ***k-face****. The function $\mathrm{aff}(S)$ indicates the affine hull of S, which is the smallest affine set that contains S. P contains {$0\text{-}face$, $1\text{-}face$,..., $(d-1)\text{-}face$} where the $0\text{-}face$ is named* ***vertex*** *and the $(d-1)\text{-}face$ is named* ***facet****. The cardinality of the set of all $k\text{-}faces$ is denoted as* $\boldsymbol{f_k(P)}$.

Definition 2.5 (Facet-Vertex Incidence Matrix). *Given a full-dimensional polytope $P \in \mathbb{R}^d$, the* ***facet-vertex Incidence matrix*** *is a matrix $M \in \{0,1\}^{f_{d-1}(P) \times f_0(P)}$ where an entry $M(F,v) = 1$ indicates the facet F contains the vertex v, and an entry $M(F,v) = 0$ indicates that it does not.*

$$\text{FVIM} = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{array} \begin{array}{c} \begin{array}{cccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{5}$$

3 OPERATIONS ON POLYTOPES

Our method relies on two basic operations for processing polytopes through a neural network. This includes *affine transformation* by weights and *division* by a hyperplane H. In the following, we present each operation in detail.

3.1 Affine Transformation

An affine transformation can map a polytope to a higher-dimensional or lower-dimensional space. A polytope P is called full-dimensional if it is a d-dimensional object in $\mathbb{R}^d$. Otherwise, if P is in $\mathbb{R}^{n>d}$, then it is not full-dimensional.

Remark 3.1. *Polytopes that differ only by an affine transformation are combinatorially equivalent. The main reason is that an affine transformation only changes vertices but preserves polytopes' combinatorial structure [9, 11].*

Therefore, the FVIM remains unchanged under affine transformation regardless if it is on a higher or lower dimensional space. Let W, b denote the weights and bias of an affine transformation. Then the computation of FVIM $\mathcal{F}'$ and vertices V' of the output polytope P', under an affine transformation, can be formulated as in Eq. 6. This process computes the exact affine transformed polytope as represented by the function $\mathcal{T}$ in Eq. 3.

$$\mathcal{F} = \mathcal{F}', \ V' = WV + b \tag{6}$$

3.2 Division by Hyperlanes

Another common operation when processing politopes is the *divison* by hyperplanes. A hyperplane H divides the input range of a ReLU function into two sub-ranges over which the function exhibits different linearities. *Division* is only considered when the set P intersects H. As the polytope representation has vertices V of P, the determination whether an intersection is needed can be made by checking whether the distribution of vertices V is on one side of a hyperplane $\mathcal{H}$. For example, let $\mathcal{H}$ be $\mathbf{a}^\top \mathbf{x} + b = 0$. To compute the distribution of vertices, we can substitute $\mathbf{x}$ with V. Formally, we define a set of *positive* vertices as $V^+ = \{v \in V | \mathbf{a}^\top \mathbf{v} + b > 0\}$ and *negative* vertices as $V^- = \{v \in V | \mathbf{a}^\top \mathbf{v} + b < 0\}$. Here we make an assumption that no vertices are located on the hyperplane H. In practice, this is an extremely unlikely occurrence due to floating point computation. When computing a division by hyperplane H, on of the following three cases may occur:

(1) Both V^+ and V^- are non-empty. In this case, the hyperplane H intersects with P. The polytope P will be divided by H into two non-empty polytopes P_{sub}^{+H} and P_{sub}^{-H}. The *positive polytope* w.r.t. H, P_{sub}^{+H} is defined as $\forall \mathbf{x} \in P, \ \mathbf{a}^\top \mathbf{x} + b \geq 0$ On the other hand the *negative polytope* w.r.t. H, is defined as $\forall \mathbf{x} \in P, \ \mathbf{a}^\top \mathbf{x} + b \leq 0$.
(2) Only set V^+ in non-empty. In this case, polytope P is on the positive closed halfspace of H and we have P^{+H}.
(3) Only set V^- in non-empty. In this case, polytope P is on the negative closed halfspace of H and we have P^{-H}.

3.2.1 Identification of a new Facet.

When a hyperplane H intersects with P and we have both P_{sub}^{+H} and P_{sub}^{-H}, the intersection creates one common facet with multiple vertices for both subsets. The derivation of the facet is as follows. In [9, 11], the authors show that an intersection of a polytope with an affine subspace (hyperplane) is a polytope. Therefore, the intersection generates one polytope P_b where $P_b \in P_{sub}^{+H}$ and $P_b \in P_{sub}^{-H}$. In the following, we show that P_b is a new facet.

Theorem 1. *Given a d-dimensional polytope P located in $\mathbb{R}^{n \geq d}$ space, and a hyperplane $H \in \mathbb{R}^{n-1}$ where $P \not\subseteq H$, then the intersection of P with H is a one $(d$-$1)$-dimensional polytope P_b. P_b is also a common facet of subsets P_{sub}^{+H} and P_{sub}^{-H}.*

Proof. In dimension theory [14], given a vector space V, and U,W, two subspaces of V, we have that

$$\dim(U + W) = \dim(U) + \dim(W) - \dim(U \cap W)$$

where $dim(\cdot)$ returns the dimension of the object. Since $P \not\subseteq H$, we have $\dim(P + H) = n$. By substituting U,W with P,H, we have that

$$\dim(P \cap H) = \dim(P) + \dim(H) - \dim(P + H) = d - 1$$

Therefore P_b is $(d$-$1)$-dimensional polytope. Here, we use a result from polytope theory which states that a face of polytope is equivalent to a polytope [11]. We can conclude that a polytope P_b generated from the intersection is a common facet of both subsets. □

3.2.2 Identification of New Vertices.

When a hyperplane H intersects with P, new vertices are generated from the intersection of H with the edges of P. Therefore, to compute the new vertices, the edges of P first need to be identified from the FVIM. For a regular d-dimensional polytope, if $d < 4$, then two vertices are adjacent and form an edge if and only if they are contained by at least $(d - 1)$ common facets. But this criterion does not apply to higher-dimensional polytopes where vertices may be nonadjacent under the same condition.

However, with the assumption above that no vertices of P lie on the hyperplane, the criterion becomes suitable for the higher-dimensional polytopes in our reachability analysis. First, we can show that all the polytopes in our analysis are *simple polytope*s.

Definition 3.1 (Simple polytope). *A d-dimensional polytope is* ***simple polytope*** *if each vertex is contained in exactly d facets.*

The input set created with the norm L_∞ bound is a hyperrectangle. The hyperrectangle is a *simple polytope* which is closed under affine-transofrmation [11]. According to Theorem 2, the subsets from the division of a *simple polytope* are also *simple polytopes*. In terms of the polytope theory, two vertices of a d-dimensional *simple polytope* are adjacent and form an edge if they are contained in

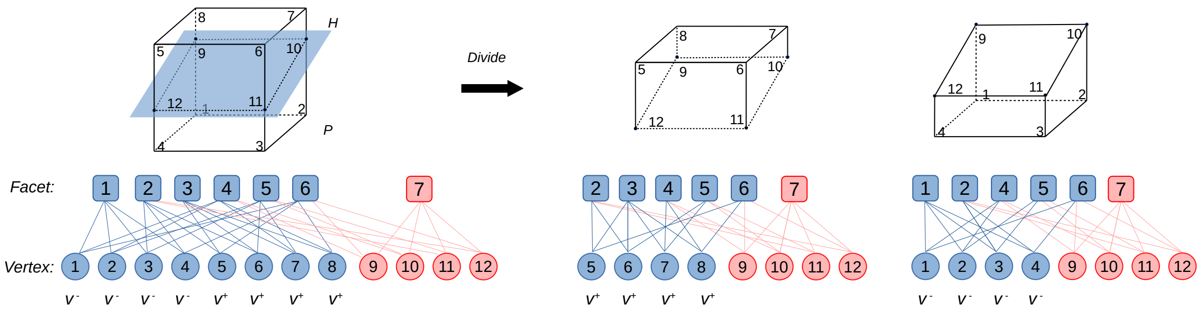

Figure 1: Division of a 3-dimensional cube P by a hyperplane H. The graph which describe the containment relation between facets and vertices is equivalent with the FVIM. The blue blocks denote the faces of P and the red blocks denote the new faces generated from the intersection.

exactly (d-1) common facets. Suppose a pair of vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ form an edge E of P, the new vertex $\mathbf{v}$ generated from $E \cap H$ can be computed by Eq. 7 where λ is an unknown scalar.

$$\begin{cases} \mathbf{v} = \mathbf{v}_1 + \lambda(\mathbf{v}_2 - \mathbf{v}_1) \\ \mathbf{a}^\top \mathbf{v} + b = 0 \end{cases} \tag{7}$$

Theorem 2. *Given a d-dimensional simple polytope P and a hyperplane H which intersects with P without any vertices of P located on H, then the two subsets from the division are simple polytopes*

Proof. Suppose there are l edges $E_{i,i\in\{1,\ldots,l\}} \in P$ intersecting with H. Thus l new vertices v are generated respectively and $v_i \in E_i,\ v_i \in H$. Since P is a *simple polytope*, each edge is contained in exactly (d-1) facets. Then, each new-generated vertex v is also contained in (d-1) facets of P. As Theorem 1 states, the intersection creates a common facet for both subsets. Then each v is also contained by this facet. Therefore, all vertices in two subsets are adjacent to exactly d facets,and the subsets are *simple polytopes*. □

3.2.3 Derivation of Two Subsets. After identifying the new facet and vertices generated from the intersection, the next step is to dividing the original polytope P into two subsets according to the containment relation between facets and vertices. An example of a division of the 3-dimensional cube is shown in Fig. 1. The intersection of P with H is the new facet $plane_{\{9,10,11,12\}}$ which is denoted as facet $\{7\}$ in the graph. *Negative vertices* $\{1, 2, 3, 4\}$ and *positive vertices* $\{5, 6, 7, 8\}$ w.r.t. H are first identified. As P is a *simple* polytope, vertices that are contained in exactly 2 common facets are adjacent and form an edge of P. Thus, the edges $E_{1,8}$, $E_{2,7}$, $E_{3,6}$ and $E_{4,5}$ are identified intersecting with H and generate new vertices v_9, v_{10}, v_{11} and v_{12}. Their values can be computed according to Eq. 7. Their containment relation with facets is inherited from the relation between the edges with facets. For instance, v_9 is from the intersection of $E_{1,8}$ with H, and $E_{1,8}$ are contained in facets $plane_{1,4,5,8}$ and $plane_{1,2,7,8}$. Therefore, v_9 is also contained in these two facets. This inheritance relation is described by the red-line connection between Facet $\{4, 6\}$ and Vertex $\{9\}$ in the graph. Finally, the graph is split in terms of *positive* and *negative* vertices as well as the new vertices, generating two compact and exact FVIMs for the subsets and completing the division.

4 REACHABILITY ANALYSIS WITH FVIM

In the previous section, we described the *affine transformation* and *division* operations, two basic operations for reachability analysis with FVIM. In this section, we present the application of these operations in one layer of the neural network and then extend them to the entire network. The process is illustrated in Fig. 2. In each layer, first we apply an affine transformation to the input polytope P_k by the weights and bias Fig. 2 (❶). Then, we apply a *divison* operation on the resulting polytope to process it through the ReLU activation function Fig. 2 (❷).

Suppose a neuron layer contains n neurons, and let $\mathbf{x} \in \mathbb{R}^n$ be an input. In the i^{th} neuron, the input range of its ReLU function is divided by $\mathbf{x}_i = 0$ into two domains over which the function exhibits different linearities. The $\mathbf{x}_i = 0$ can also be formulated as a hyperplane $H_i : \mathbf{a}^\top \mathbf{x} = 0$ where $\mathbf{a}_i = 1$ and the rest are zeros. When an input polytope P intersects with H_i which indicates that the polytope spans two input domains of the ReLU function in the ith neuron, this polytope will be divided by H_i according to the *division* operation. Then two subsets $P^{-H_i}_{sub}$ and $P^{+H_i}_{sub}$ are obtained. This process can be formulated as

$$\mathbf{Q}_i = \mathcal{E}_i(P) \tag{8}$$

where $\mathbf{Q}_i$ can be one or two output polytopes for the three intersection cases. For the output *negative* subset that is located in the negative closed halfspace of H_i, the $\mathbf{x}_i$ dimension of all points belonging to it is set to zero in terms of the property of the ReLU function. This process is equivalent to a linear transformation and can be implemented with Eq. 6. Let I_k denote the input polytope to the neuron layer. Then by sequentially processing I_k w.r.t. each neuron as shown in Eq. 8, we can consider all combinations of linearities. This reachability analysis, for the k-th layer, which was initially described by $\mathcal{R}$ in Eq. 3, can be reformulated as a sequence of $\mathcal{E}(\cdot)$ functions as follows:

$$O_k = \mathcal{R}_k(I_k) = \mathcal{E}_n(\ldots(\mathcal{E}_2(\mathcal{E}_1(I_k))) \tag{9}$$

where n is the number of neurons. An example is demonstrated in Fig. 2 (❷). The layer contains 2 ReLU neurons. First, $\mathcal{E}_1(\cdot)$ is applied to the input polytope as shown in (a). The polytope is divided by the hyperplane H_1 into two polytopes P_1 and P_2 which are $P^{+H_1}_{sub}$ and $P^{-H_1}_{sub}$. Then the $\mathbf{x}_1$ dimension of all element points in $P^{-H_1}_{sub}$ is set to zero, generating a new polytope P_3 (red solid line) in (b). Then the output P_1 and P_3 will be the inputs to the function $\mathcal{E}_2(\cdot)$. As shown in (c). P_1 and P_3 are divided by H_2 into $\{P_4, P_5\}$ and $\{P_6, P_7\}$, respectively. P_4 and P_7 are in $P^{-H_2}_{sub}$ and the $\mathbf{x}_2$ dimension of points is set to zero as shown in (d), creating P_8 and P_9 (origin), respectively. Finally the output polytopes of this layer w.r.t. I_k are $\{P_5, P_6, P_8, P_9\}$,

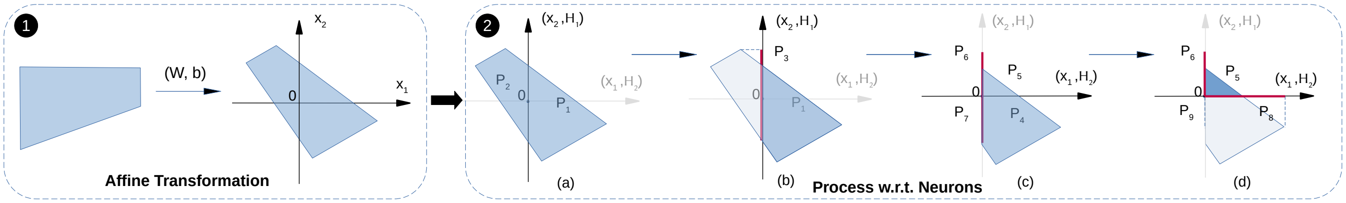

Figure 2: A simple layer that contains 2 neurons to demonstrate the processing of an input polytope

and the whole process can be formulated as

$$\{P_5, P_6, P_8, P_9\} = \mathcal{E}_2(\mathcal{E}_1(\mathcal{I}_k))$$

The output polytopes of the current layer will be the inputs to the next layer. Let the process of the k^{th} layer in Eq. 9 be denoted as a function $\mathcal{L}_k(\cdot)$, then given an input set I, and number of layers l, the computation of reachable sets can be defined as:

$$O = \mathcal{L}_l(\ldots \mathcal{L}_2(\mathcal{L}_1(I))) \tag{10}$$

The algorithm for the computation of reachable sets is shown in Algorithm 1. Its implementation is highly parallelizable because (1) the input sets to each layer are independent in the *for* loop in Line 5, (2) the processing of each set w.r.t. one neuron is also independent in the *for* loop in Line 14. The details of functions are as following:

(1) The **Reach**() function corresponds to Eq. 10. Given an input set to the neural network, it computes all the reachable sets.
(2) The **singleLayer**() function computes reachable sets of the target layer given an input set.
(3) The **affineTransform**() function corresponds to Eq. 6. It computes the affine transformation on the input set with the weight matrix and bias vector between layers.
(4) The **Divide**() function corresponds to Eq. 8, which is illustrated in Fig. 2. Here we only present one case where P' intersects with the hyperplane.
(5) The **Project**() function is used to set the target dimension of the elements of a polytope to zeros, which is equivalent to a linear transformation.

Given an input set to the neural network that has n ReLU neurons, the maximum number of reachable sets is bounded by 2^n.

5 BACKTRACKING

Given an input set, the algorithm introduced in the previous section can compute the exact reachable sets for neural networks. However, our method also supports *backtracking*, which is the process of computing the input sets associated with a subset of the output domain. This is particularly useful in verification, since we can use this process to find input areas which are associated with possible violations. We track the connection between the polytopes processed in layers and their corresponding *linear regions* in the input space. Here we revise the representation of polytopes in Eq. 4 by replacing the set of vertices V with an affine transformation of points in the input space:

$$P = \langle \mathcal{F}, V_i, M, d \rangle \tag{11}$$

V_i is a set of points and represents a convex subset of the input space to the network. The vertices of P can be computed with V_i by $V = MV_i + d$. This linear relation between V_i and V indicates that the convex subset is the *linear region* of P. The *linear region* of the input set to the network can be constructed with $V_i = V$, M as an identity matrix and d as a zeros vector.

Algorithm 1 Reachable set computation of a neural network

```
Input: $I$   # input set to the neural network
Output: $O$   # reachable sets of the $n$th layer
 1: procedure $O$ = Reach($I$)
 2:     $\mathcal{I}_k = I$   # $\mathcal{I}_k$: input sets to the $k$th layer
 3:     for $k = 1 : layers$ do   # layers: the number of layers
 4:         $O_k = empty$   # $O_k$: output set of the $k$th layer
 5:         for $P$ in $\mathcal{I}_k$ do
 6:             $\mathcal{S}_k$ = singleLayer($k$, $P$)
 7:             Add $\mathcal{S}_k$ to $O_k$
 8:         $\mathcal{I}_k = O_k$
 9:     return $O = O_k$
10: procedure $\mathcal{S}_k$ = singleLayer($k$, $P$)
11:     $\mathcal{S}_k$ = affineTransform($k$, $P$)
12:     for $i = 1 : m$ do   # $m$: the number of neurons
13:         $\mathcal{S}_{temp} = empty$
14:         for $P'$ in $\mathcal{S}_k$ do
15:             $P'^{+}, P'^{-}$ = Divide($H_i$, $P'$)
16:             $P'^{-}$ = Project($i$, $P'^{-}$)
17:             Add $P'^{+}, P'^{-}$ to $\mathcal{S}_{temp}$
18:         $\mathcal{S}_k = \mathcal{S}_{temp}$
19:     return $\mathcal{S}_k$
```

With the represenation of polytopes as a tuple $\langle \mathcal{F}, V_i, M, d \rangle$, the *affine transformation* in Eq. 6 will be slightly different. As shown in Eq. 12, the new operation on polytopes will update M and d instead and preserve V_i. Hence, the *linear region* of P can be tracked. The updated polytope $\langle \mathcal{F}', V_i', M', d' \rangle$ is calculated as follows:

$$\mathcal{F}' = \mathcal{F}, \quad V_i{}' = V_i, \quad M' = WM, \quad d' = Wd + b \tag{12}$$

We note that the *division* operation in Eq. 8 is not affected. When checking the distribution of vertices of P w.r.t. a hyperplane H, its vertices can be computed by $V = MV_i + d$. As P only differs from its *linear region* by an affine transformation, they have the same FVIM $\mathcal{F}$. Therefore the *division* on $\mathcal{F}$ not only divides P as well as its *linear region*.

5.1 Identification of Unsafe Input Subspace

Let the output unsafe domain be $\mathcal{U} : \{\mathbf{y} \,|A\mathbf{y} + \mathbf{v} \le 0\}$. Suppose we have a reachable set P=$\langle F, V_i, M, d \rangle$. First, the vertices of P are computed by $V = MV_i + d$ and we can inspect whether P intersects with $\mathcal{U}$. If not, the input subspace characterized by V_i is safe.

In case of an intersection, a series of *division* operations will be conducted for the backtracking. In each *division*, the hyperplane is constructed from one of the linear inequalities of $\mathcal{U}$. One linear inequality $\mathbf{a}^\top \mathbf{y} + v \le 0$ is essentially a negative closed halfspace of a hyperplane $H : \mathbf{a}^\top \mathbf{y} + v = 0$. The P_{sub}^{-H} generated from the *division* with H contains all element points that satisfy this linear inequality. Then P_{sub}^{-H} is the input set to the *division* with the next linear inequality. We can obtain the final unsafe set after considering all linear inequalities of $\mathcal{U}$, and its V_i represents the unsafe input subspace. This process in represented in Algorithm 2 in Appendix and the computation in Line 3 is parallelizable.

6 EVALUATION

6.1 Safety Verification of ACAS Xu Networks

In this section, we evaluate our method against the Marabou [18], NNV-exact [31], nnenum [2], Venus [3] and Neurify [32] methods, which conduct a sound and complete verification. We also include ERAN [27], which is an over-approximation. We include it to demonstrate that performance-wise, over-appoximation does not provide a significant improvment in speed as oposed to exact methods.

We verify safety of the DNN benchmarks proposed in [16] for the Airborne Collision System X Unmanned (ACAS Xu). These benchmarks have been widely tested by the verification community [2, 18, 30–32]. The ACAS Xu networks are used to approximate a large lookup table that converts sensor measurements into maneuver advisories. The memory intensive lookup table was successfully reduced to a DNN. The set of networks contain 45 fully-connected DNNs for different combinations of discretized parameters. Each one has five inputs, five outputs, and six hidden layers. Each layer consists of 50 ReLU neurons. There are 300 hidden neurons total in each network. There are ten safety properties 1-10 which specify input bounds P and unsafe domains $\mathcal{U}$ in the output. The details can be found in the Appendix of [17].

We tested all 45 networks for properties 1,2,3 and 4, with 180 instances in total . The hardware configuration is Intel Core i9-10900K CPU @3.7GHz×, 10-core and 20-thread Processor, 128GB Memory, 64-bit Ubuntu 18.04. The result is represented in the cactus plot are shown in Fig. 3. Our method is the only method that can verify all the 180 instances without timing out (120s). Although Neurify [32] is faster than our methods on some instances by a few seconds, there are instances where it performs significantly worse. In some instances, Neurify takes up to one hour. For properties 5-10, we select an individual network that is commonly used by other publications. The result is shown in Table 1. Marabou is not included since their published code does not support disjunctive conditions for the unsafe domain that is required in these properties.

In addition, our approach can also identify the subspace of the input space that leads to the safety violation. For instance, the inputs that make the network N_{1_2} violate Property 2 are computed as shown in Fig. 4 in Appendix.

6.2 Reachability Analysis of Microbenchmarks

To further evaluate our approach, we compare them on a set of microbenchmarks that are proposed in [6]. These benchmarks consist of neural networks are created from the unwindings of a closed-loop controller and a plant model. Here we check these networks' safety on some synthetic unsafe domains. Neurify is not included as we fail to apply their published code to these new networks. The performance on each benchmark is shown in Table 2. The neural networks consist of thousands of neurons, as indicated in column m in Table 2. As shown in Table 2, our approach computes the reachable sets for large networks. For Marabou, the *HD Error* indicates a high-degradation error. The *Error* means Marabou returns a false verification result for the instance. The *LP Error* for the nnenum indicates that the algorithm is terminated after encountering a LP-solver error.

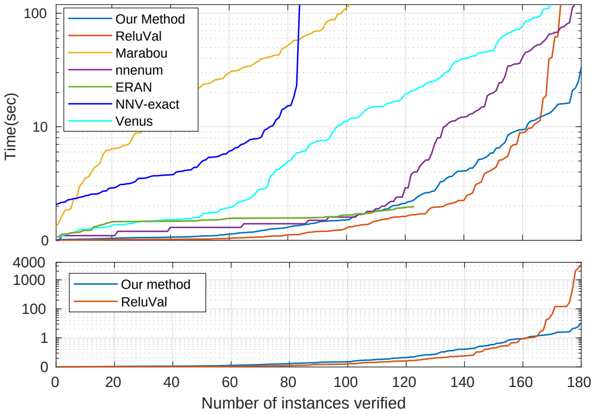

Figure 3: Test on ACAS Xu networks Property 1-4 with timeout. The timeout for each instance is set to 120 seconds. ERAN fails to complete all instances. The figure below is without timeout.

Table 1: Evaluation on properties 5-10 with 1hr timeout.

P	Net	Result	Our Method	nnenum	NNV -Exact	Neurify	ERAN	Venus
5	N_{11}	SAFE	1.7	4.3	84.1	4.8	-	174.6
6.1	N_{11}	SAFE	4.7	20.9	TT	1.2	6.0	325.7
6.2	N_{11}	SAFE	5.6	24.1	TT	0.5	-	402.4
7	N_{19}	UNSAFE	1757.0	3730.5	TT	344.5	-	5298.0
8	N_{29}	UNSAFE	5.2	0.1	TT	24.4	-	0.7
9	N_{33}	SAFE	6.9	26.0	TT	146.2	-	926.8
10	N_{45}	SAFE	1.1	4.0	TT	0.4	10.3	78.4

Table 2: Performance results on the microbenchmarks. The label x, k and m respectively denote the number of input, the number of layers and total number of ReLU neurons. The running time is in *sec*. The timeout is set to 10 minutes.

ID	x	k	m	Our Method	Marabou	NNV	nnenum
N_{11}	3	9	1427	65.5	567.7	TIMEOUT	LP Error
N_{12}	3	14	2292	96.5	TIMEOUT	TIMEOUT	LP Error
N_{13}	3	19	3057	128.9	HD Error	TIMEOUT	LP Error
N_{14}	3	24	3822	150.9	HD Error	TIMEOUT	LP Error
N_{15}	3	127	6845	2.3	Error	383.5	12.7

7 CONCLUSION AND FUTURE WORK

In this paper, we presented a polytope-based method to conduct exact reachability analysis of feed-forward neural networks with ReLU activation. Experimental results indicate that, in most cases, our method is faster than state-of-the-art methods. In the future, we will consider applying our approach for reachability analysis and safety verification of learning-enabled control systems with machine learning components.

REFERENCES

[1] Alexandre Araujo, Laurent Meunier, Rafael Pinot, and Benjamin Negrevergne. 2019. Robust neural networks using randomized adversarial training. *arXiv preprint arXiv:1903.10219* (2019).

[2] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. 2020. Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer.

[3] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. 2020. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis.. In *AAAI*. 3291–3299.

[4] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. 2018. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*. 4790–4799.

[5] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 157–168.

[6] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*. Springer, 121–138.

[7] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. 2020. An abstraction-based framework for neural network verification. In *International Conference on Computer Aided Verification*. Springer, 43–65.

[8] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[9] Branko Grünbaum. 2013. *Convex polytopes*. Vol. 221. Springer Science & Business Media.

[10] Boris Hanin and David Rolnick. 2019. Complexity of linear regions in deep networks. *arXiv preprint arXiv:1901.09021* (2019).

[11] Martin Henk, Jürgen Richter-Gebert, and Günter M Ziegler. 2004. 16 basic properties of convex polytopes. *Handbook of discrete and computational geometry* (2004), 255–382.

[12] Patrick Henriksen and A Lomuscio. 2019. *Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search*. Ph.D. Dissertation. Imperial College London.

[13] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 3–29.

[14] Witold Hurewicz and Henry Wallman. 2015. *Dimension Theory (PMS-4)*. Vol. 4. Princeton university press.

[15] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 169–178.

[16] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th*. IEEE, 1–10.

[17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

[18] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. 2019. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 443–452.

[19] Panagiotis Kouvaros and Alessio Lomuscio. 2018. Formal verification of cnn-based perception systems. *arXiv preprint arXiv:1811.11373* (2018).

[20] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J Kochenderfer. 2019. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758* (2019).

[21] Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351* (2017).

[22] Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*. 3578–3586.

[23] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*. 2924–2932.

[24] Luca Pulina and Armando Tacchella. 2010. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*. Springer, 243–257.

[25] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. 2018. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242* (2018).

[26] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. 2017. Bounding and counting linear regions of deep neural networks. *arXiv preprint arXiv:1711.02114* (2017).

[27] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*. 10825–10836.

[28] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 41.

[29] Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. 2019. Parallelizable reachability analysis algorithms for feed-forward neural networks. In *Proceedings of the 7th International Workshop on Formal Methods in Software Engineering*. IEEE Press, 31–40.

[30] Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. 2019. Star-Based Reachability Analsysis for Deep Neural Networks. In *23rd International Symposium on Formal Methods (FM'19)*. Springer International Publishing.

[31] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. 2020. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. *arXiv preprint arXiv:2004.05519* (2020).

[32] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*. 6369–6379.

[33] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1599–1614.

[34] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. 2018. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699* (2018).

[35] Eric Wong and Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*. PMLR, 5286–5295.

[36] Weiming Xiang, Hoang Tran, and Taylor Johnson. 2017. Reachable set computation and safety verification for neural networks with ReLU activations. *arXiv preprint arXiv:1712.08163* (2017).

[37] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. 2018. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems* 29, 11 (2018), 5777–5783.

[38] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*. 4944–4953.

A APPENDIX

Algorithm 2 Identification of unsafe input subspaces

```
Input: $\mathcal{O}, \mathcal{U}$    # a set of all reachable sets, unsafe output domain
Output: $\mathcal{I}$    # a set of subspaces that leads to safety violation
procedure $\mathcal{I}$ = UNSAFEINPUT($\mathcal{O}, \mathcal{U}$)
    $\mathcal{U} \rightarrow \mathcal{H}$  # get hyperplanes of its linear inequalities
    for $P$ in $\mathcal{O}$ do
        for $H_i$ in $\mathcal{H}$ do
            $P'^{+}, P'^{-}$ = Divide($H_i$, $P$);    $P = P'^{-}$;
        Add $P'$ to $\mathcal{I}$
    return $\mathcal{I}$
```

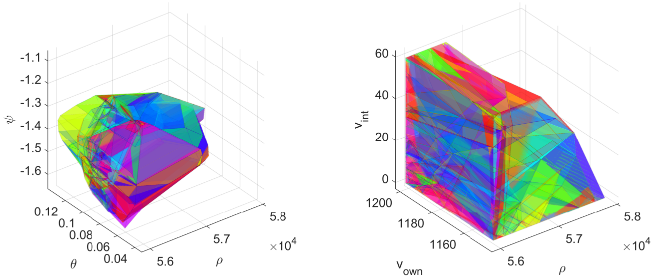

Figure 4: Complete input sets that lead to the property ϕ_2 violation on the network N_{1_2}. It is a union of 102 polytopes. Any element in these polytopes will yield property violations.