Parallelizable Reachability Analysis Algorithms for Feed-Forward Neural Networks

1st Hoang-Dung Tran
Vanderbilt University
Nashville, USA
trhoangdung@gmail.com

2nd Patrick Musau
Vanderbilt University
Nashville, USA
patrick.musau@vanderbilt.edu

3rd Diego Manzanas Lopez
Vanderbilt University
Nashville, USA
diego.manzanas.lopez@vanderbilt.edu

4thXiaodong Yang
Vanderbilt University
Nashville, USA
xiaodong.yang@vanderbilt.edu

5th Luan Viet Nguyen
University of Pennsylvania
Philadelphia, USA
luanvn@seas.upenn.edu

6th Weiming Xiang
Vanderbilt University
Nashville, USA
xiangwming@gmail.com

7th Taylor T. Johnson
Vanderbilt University
Nashville, USA
taylor.johnson@vanderbilt.edu

***Abstract*—Artificial neural networks (ANN) have displayed considerable utility in a wide range of applications such as image processing, character and pattern recognition, self-driving cars, evolutionary robotics, and non-linear system identification and control. While ANNs are able to carry out complicated tasks efficiently, they are susceptible to unpredictable and errant behavior due to irregularities that emanate from their complex non-linear structure. As a result, there have been reservations about incorporating them into safety-critical systems. In this paper, we present a reachability analysis method for feed-forward neural networks (FNN) that employ rectified linear units (ReLUs) as activation functions. The crux of our approach relies on three reachable-set computation algorithms, namely *exact* schemes, *lazy-approximate* schemes, and *mixing* schemes. The exact scheme computes an exact reachable set for FNN, while the lazy-approximate and mixing schemes generate an over-approximation of the exact reachable set. All schemes are designed efficiently to run on parallel platforms to reduce the computation time and enhance the scalability. Our methods are implemented in a MATLAB® toolbox called, *NNV*, and is evaluated using a set of benchmarks that consist of realistic neural networks with sizes that range from tens to a thousand neurons.Notably, *NNV* successfully computes and visualizes the exact reachable sets of the real world ACAS Xu deep neural networks (DNNs), which are a variant of a family of novel airborne collision detection systems known as the ACAS System X, using a representation of tens to hundreds of polyhedra.**

I. Introduction

Artificial neural networks (ANN) have demonstrated an effective and powerful ability to achieve success in numerous contexts such as image classification [1], speech, and character recognition [2], [3], financial market forecasting, autonomous vehicles [4], and the design of neural network based airborne collision avoidance systems such as ACAS Xu [5]. Due to their proficient ability to learn complex non-linear functions from large sets of data, ANNs have quickly become a popular methodology for carrying out various sophisticated tasks. Despite their success, it has been observed that even a well-trained ANN can exhibit incorrect and errant behavior by slightly perturbing its inputs [6]. This revelation has stimulated a wealth of safety verification techniques that seek to reason about the correctness of an ANNs behavior. Regrettably, the safety verification problem for ANNs is notoriously difficult, due to the non-convex, non-linear, and typically large nature of their structure. In fact, it has been demonstrated that verifying even a simple property about an ANN's behavior is NP-complete [7]. Thus there is an immediate need for methods and advanced software tools that can efficiently deal with the complexity and scale of real-world ANNs.

In light of these challenges, the central focus of this paper is the provision of reachable set computation methods for trained feed-forward neural networks (FNN) with ReLU activation functions. By computing the reachable set of a neural network's outputs, one can verify various safety specifications, and by visualizing the projection of the reachable set onto a specific subspace, one can further reason about the behavior of a particular network. Unlike many of the existing verification approaches, our method can compute both an exact reachable set, and an over-approximation of the reachable set for a given FNN. Furthermore, the algorithms proposed in this work, have been tailored to exploit the benefits of parallel computing. Thus, the computation time required to compute the reachable set for a given neural network diminishes linearly with the number of processors (cores) utilized in the computation process. In our framework, we express the reachable set of a FNN at the output layer as a union of convex polyhedra.

The three procedures that we consider for computing the reachable set of FNNs are: an exact scheme, a lazy-approximate scheme, and a mix of both schemes, which we denote as mixing schemes. The *exact scheme* computes a layer-by-layer explicit reachable set for a FNN. At each layer, the exact scheme performs a sequence of *stepReLU operations*, which compute an *intermediate (incomplete) reachable set* for a specific neuron. This is done by examining the output of the ReLU activation function on the input field of the considered neuron. The reachable set at a given layer's output is obtained after a sequence of stepReLU operations is completed, and is not dependent on the order of stepReLU operations. To optimize the reachable set computation, our scheme pre-processes the input set at each layer in order to minimize the number of stepReLU operations that need to be executed. Additionally, the reachable set for each input polyhedron for a given layer is computed independently and in parallel.

Secondly, the *lazy-approximate scheme*, is an over-approximation procedure specifically designed to compute *safe ranges* for the outputs of a FNN. In particular, the output range of each layer is bounded by a hyper-rectangle which can be constructed efficiently by solving n linear programming problems, where n is the number of neurons of the layer. The lazy-approximate scheme, does not perform stepReLU operations, and as a result, it is much faster than the exact scheme. However, the error resulting from the over-approximations accumulate very quickly for deep FNNs, resulting in very conservative output ranges for networks with many layers. Therefore, the approximate scheme is primarily suitable for FNNs with a limited number of layers. However, it can deal with large numbers of neurons in these layers. It is also worth to noting that for a FNN with a small input space, we can first partition the input space into a smaller set of polyhedra, and then compute a tight over-approximation of the reachable set in parallel using the lazy-approximate scheme.

The third scheme, called the *mixing scheme*, is designed to maximize the advantages of the exact and lazy-approximate approaches and operates as follows. 1) The exact scheme is used to compute a reachable set layer-by-layer until the number of polyhedra representing this set exceeds a user-defined upper bound, N_{max}. Once this occurs, these polyhedra are efficiently *clustered and merged* into N_{max} hyper-rectangles such that the union of N_{max} hyper-rectangles represents a *tight over-approximation* of the reachable set. In fact, we cluster $\{N \mid N > N_{max}\}$, polyhedra of the reachable set into N_{max} groups based on whether the considered polyhedra overlap, and then over-approximate the union of the polyhedra in these groups using one hyper-rectangle. 2) After the clustering and merging steps, the lazy-approximate scheme is invoked to compute the reachable set of the rest layers in parallel, using an input set that is a union of N_{max} hyper-rectangles. The conservativeness of the reachable set obtained using this methodology is highly dependent on the user-defined upper bound N_{max}. Generally, when N_{max} is increased, the over-approximation of the reachable set approaches the exact reachable set. The main benefit of this mixing scheme is that the number of polyhedra used in representing the reachable set can be tuned by a user which has a great influence the required computation time.

We evaluate our methods by considering two practical problems including safety verification, and local adversarial robustness of FNNs. For safety verification, our exact scheme successfully verifies the safety of the real-world ACAS Xu DNNs [5]. Notably, our method can visualize explicitly the behaviors of these networks using the computed reachable sets which is a notable advance of our approach in comparison with Reluplex, an SMT-solver based approach. Our techniques also successfully prove the local adversarial robustness of neural networks with up to a thousand neurons.

Contributions. In summary, the main contributions of this study are as follows.

1) The provision of three Parallelizable schemes designed to efficiently compute the reachable set of an FNN with ReLU activation functions,
2) An end-to-end design and implementation of these schemes in a MATLAB® toolbox that is publicly available for verifying complex FNNs,
3) and a thorough experimental evaluation of our proposed methods in comparison to the other existing state-of-the art approaches using a set of benchmarks consisting of practical FNNs.

The rest of the paper is organized as follows. Section II presents the background of the safety verification problem of FNNs. Section III addresses the exact reachable set computation scheme. Section II studies the lazy-approximate scheme, and discusses how to cluster and merge a number of polyhedra of a reachable set into a user-specified number of hyper-rectangles, and then presents the mixing scheme. The experimental results are illustrated in Section V. The related works are discussed in Section VI, and Section VII concludes the paper.

II. Preliminaries

A FNN consists of an input layer, an output layer, and multiple hidden layers. Each layer is comprised of neurons that are connected to the neurons of the preceding layer labeled using weights. An example of FNN is shown in Figure 1 where the input layer L_1 has two neurons, the hidden layer L_2 has three neurons, and the output layer L_3 has two neurons.

Evaluation of FNN. The output of a FNN, given a specific input vector 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. Formally, the output of a neuron i is defined by:

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

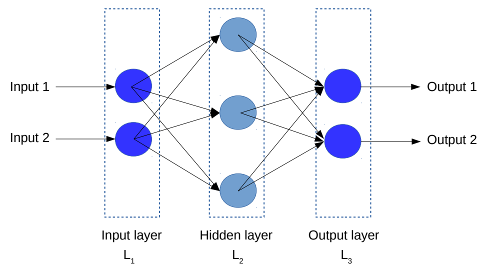

Figure 1: An example of FNN.

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 consider FNN with ReLU activation functions which is defined as $ReLU(x) = max(0, x)$.

Example II.1 (Evaluation of FNN)**.** The example shown in Figure 1 has the following weight matrices and bias vectors:

$$W_{2,1} = \begin{bmatrix} 2 & 0 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}, \; b_2 = \begin{bmatrix} 0.5 \\ -1 \\ -0.5 \end{bmatrix},$$
$$W_{3,2} = \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}, \; b_3 = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}.$$

The weight matrix $W_{2,1}$ expresses the weighted connections between the neurons of the hidden layer L_2 and the neurons of the input layer L_1, while the weight matrix $W_{3,2}$ describes the weighted connections between the neurons of the output layer L_3 and the neurons of the hidden layer L_2.

Assuming that the input vector for the input layer is $x = [1 \; 1]^T$, and that we use ReLUs for the activation functions of the neural network, then the vector containing the values of the hidden layer's neurons is:

$$y_2 = ReLU(W_{2,1} \times x + b_2) = [2.5 \; 0 \; 1.5]^T$$

The vector y_2 is then fed to the output layer L_3 as its input vector. Finally the output vector for the output layer is:

$$y_3 = ReLU(W_{3,2} \times y_2 + b_3) = [0 \; 3]^T$$

Reachability analysis of FNN. The example described above describes how to compute the output values for a FNN using a specific input vector. In this paper, we are interested in computing the reachable set R of the output of a FNN with a given input set I. Specifically, we consider the reachability analysis of a FNN with a bounded convex polyhedron input set defined as

$$I = \{x \mid Ax \leq b, x \in \mathbb{R}^n\},$$

where x is the input vector, and n is the dimension of the input space, i.e., the number of inputs of the FNN. Since the ReLU activation function is non-linear, and typically FNNs have a large number of layers and neurons, performing reachability analysis for a FNN is non-trivial.

Safety verification of FNN. The ultimate goal of conducting reachability analysis for a FNN is to verify whether the outputs of the FNN violate some safety property S defined by users. In our framework, the safety properties we consider are a set of linear constraints on the outputs of the FNN defined as:

$$S = \{y \mid Cy \leq d, y \in \mathbb{R}^m\},$$

where y is the output vector, and m is the dimension of the output space, i.e., the number of outputs of the FNN. From the reachable set R of the outputs computed by reachability analysis algorithms, we can verify safety properties for FNNs by checking the intersection between the determined output reachable set with an unsafe region, i.e., $\neg S$, where $\neg$ is the symbol for logical negation. Formally, the FNN is called safe if and only if $R \cap \neg S = \emptyset$.

III. Exact Reachability Analysis

In our approach, the exact reachability analysis is done layer-by-layer. Given an input set I_i, the reachable set of a layer L can be obtained precisely in two steps. First, an affine map $\bar{I}_i$ of the input set I_i is computed with the weight matrix W and bias vector b of the layer. Mathematically, the affine map $\bar{I}_i$ is defined by:

$$\bar{I}_i = \{y \mid y = Wx + b, \; x \in I_i\}.$$

After calculating the affine map of the input set, the reachable set of the layer R_L is obtained by applying the ReLU activation function on the affine-mapped set $\bar{I}_i$,

$$R_L = ReLU(\bar{I}_i).$$

This second step is done by executing a sequence of stepReLU operations $ReLU_j()$ which is defined below.

A. stepReLU operation

The basic idea of stepReLU operations is illustrated in Figure 2 for a two-dimensional affine-mapped set $\bar{I}_i$. The stepReLU operation works as follows. First, the affine mapped set $\bar{I}_i$ is decomposed into two sets $\Psi_1 = \bar{I}_i \wedge x_1 \geq 0$ and $\Psi_2 = \bar{I}_i \wedge x_1 < 0$. Since the later set has $x_1 < 0$, applying the ReLU activation function on the first element x_1 of a vector $x = [x_1 \;\; x_2]^T \in \Psi_2$ will lead to a new vector $x' = [0 \;\; x_2]^T$. Also, applying the ReLU activation function on the first element x_1 of a vector $x \in \Psi_1$ does not change the set since we have $x_1 \geq 0$. As a result, the intermediate reachable set after applying the $ReLU_1$ operation on the affine-mapped set $\bar{I}_i$ is $\tilde{R} = \tilde{R}_1 \cup \tilde{R}_2$, where $\tilde{R}_1 = \Psi_1$, $\tilde{R}_2$ is the projection of Ψ_2 on the hyper-plane $x_1 = 0$. The intermediate reachable set $\tilde{R}$ is then fed to the $ReLU_2$ operation as an input. The final reachable set of the layer is a union of three polyhedra $R_L = ReLU(\bar{I}_i) = R_1 \cup R_2 \cup R_3$ as shown in the figure. For a layer with n neurons, the reachable

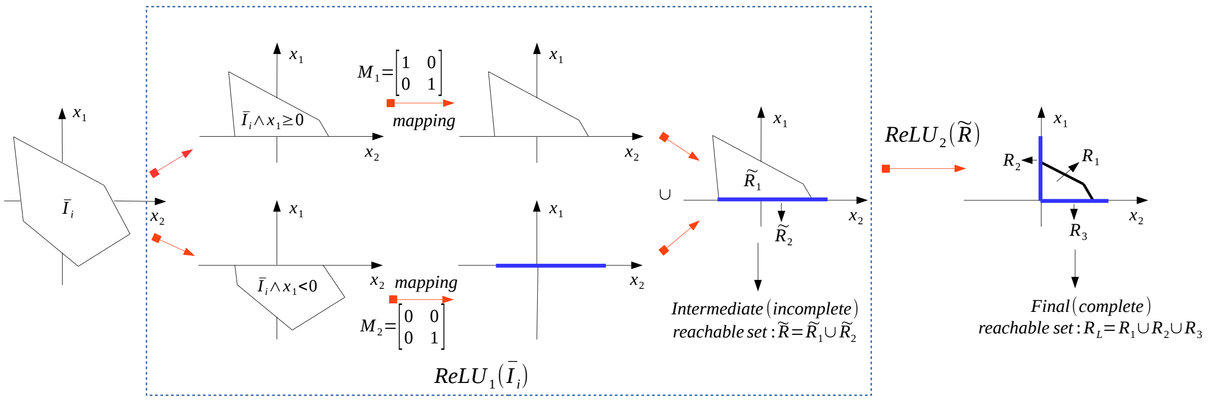
x_1
x_2
$\bar{I}_i$
$\bar{I}_i \wedge x_1 \geq 0$
$\bar{I}_i \wedge x_1 < 0$
$M_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
mapping
$M_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$
mapping
$\tilde{R}_1$
$\tilde{R}_2$
$\cup$
Intermediate (incomplete) reachable set : $\tilde{R} = \tilde{R}_1 \cup \tilde{R}_2$
$ReLU_1(\bar{I}_i)$
$ReLU_2(\tilde{R})$
R_1
R_2
R_3
Final (complete) reachable set : $R_L = R_1 \cup R_2 \cup R_3$

Figure 2: stepReLU operation.

set of the layer can generally be obtained by executing a sequence of n stepReLU operations as follows.

$$R_L = ReLU_n(ReLU_{n-1}(\cdots ReLU_1(\bar{I}_i))).$$

One can verify that changing the order of the stepReLU operations affects the intermediate reachable set but it does not change the final reachable set of the layer.

Optimize stepReLU operation. From a single polyhedron input set, a stepReLU operation can produce two polyhedra, e.g., $ReLU_1(\bar{I}_i)$ in Figure 2. The number of polyhedra in computation increases with the number of stepReLU operations. Therefore, to reduce the computation cost, it is crucial to minimize the number of necessary stepReLU operations and the number of polyhedra in the intermediate reachable sets. The solution for this optimization problem is to find the smallest hyper-rectangle, i.e., a box, that bounds the affine-mapped set $\bar{I}_i$, from this hyper-rectangle we determine the ranges of all elements x_i in the vector $x = [x_1 \quad x_2 \quad \cdots \quad x_n]^T \in \bar{I}_i$. Finding these ranges is trivial since it is equivalent to solving n simple linear programming problems. From the ranges of all elements of the vector x, if we know that the minimum value of the element x_i is larger than zero, then we can neglect the stepReLU operation on the i^{th} neuron. To minimize the number of polyhedra in the intermediate reachable sets, in each stepReLU operation, if it produces two polyhedra, we need to check whether or not one polyhedron is a subset of another to delete the subset polyhedron.

The stepReLU operation is stated in Algorithm 1 in which $\tilde{I}$ is the input set (i.e., the output of the preceding stepReLU operation), i is the index of the current neuron we want to perform the stepReLU operation, and x_{min}, x_{max} are the lower and upper bounds of the element $x[i]$, respectively.

Algorithm 1 stepReLU operation with multiple inputs

```
1: procedure $\tilde{R}$ = STEPRELU($\tilde{I}, i, x_{min}, x_{max}$)
2:     % $\tilde{I}$: intermediate input set
3:     % $i$: index of current neuron
4:     % $x_{min}$: lower-bound of $x[i]$
5:     % $x_{max}$: upper-bound of $x[i]$
6:     % $\tilde{R}$: intermediate output set
7:     $n = length(\tilde{I})$
8:     $\tilde{R} = \emptyset$
9:     for $j = 1 : n$ do
10:        $I_1 = I(j)$
11:        $R_1 = \emptyset$
12:        if $x_{min} \geq 0$ then
13:            $R_1 = I_1$
14:        if $x_{max} < 0$ then
15:            $R_1 = projection\ of\ I_1\ on\ x[i] = 0$
16:        if $x_{min} < 0$ & $x_{max} \leq 0$ then
17:            $Z_1 = I_1 \wedge x[i] \geq 0$
18:            $Z_2 = I_1 \wedge x[i] < 0$
19:            $Z_2' = projection\ of\ Z_2\ on\ x[i] = 0$
20:            if $Z_2'\ is\ a\ subset\ of\ Z_1$ then
21:                $R_1 = Z_1$
22:            else
23:                $R_1 = Z_1 \cup Z_2'$
24:        $\tilde{R} = \tilde{R} \cup R_1$
```

B. Parallel-exact reachability analysis

Algorithm 1 computes the intermediate reachable set at each neuron. To compute the reachable set of a layer, we need to perform a sequence of stepReLU operations. The last step of the sequence calculates the reachable set of the layer. Algorithm 2 is the reachable set computation of one layer using the stepReLU operation. First, it calculates the affine map of the input set using the layer's weight matrix and bias vector. Then, it performs the *reachReLU* procedure which basically executes a sequence of stepReLU operations. To optimize the stepReLU operations, the *reachReLU* procedure finds the lower and upper bounded of a vector x in the affine-mapped set. Then, it constructs a computation map which minimizes the number of stepReLU operations needs to be executed. Since a layer can have multiple polyhedra as input sets, we can perform the reachable set computation for each input set in parallel. To derive the reachable set for an FNN, we perform the reachable set computation layer-by-layer. The reachable set of the last layer is the output reachable set of the FNN.

Algorithm 2 Parallel-exact reachable set computation for one layer

Input: I, W, b % input set, weight matrix, bias vector
Output: R % reachable set

```
1: procedure $R_1$ = REACHRELU($I_1$)
2:     $lb \leftarrow lower-bound\ of\ x \in I_1$
3:     $ub \leftarrow upper-bound\ of\ x \in I_1$
4:     $map = find(lb < 0)$ % computation map
5:     $m = length(map)$ % number of stepReach operations
6:     $In = I_1$
7:     for $i = 1 : m$ do
8:         $In = stepReLU(In, map(i), lb(map(i)), ub(map(i)));$
9:     $R_1 = In;$
10: procedure $R$ = LAYERREACH($I, W, b$)
11:     $n = length(I)$
12:     $R = \emptyset$
13:     parfor $i = 1 : n$ do % parallel for loop
14:         $I_1 = I(i).affineMap(W) + b$
15:         $R_1 = reachReLU(I_1)$
16:         $R = R \cup R_1$
17:     end parfor
```

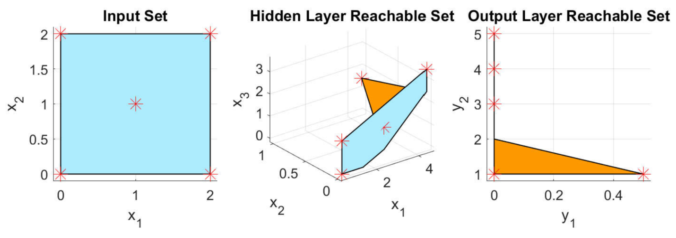

Figure 3: Exact reachable set of the FFN in Example II.1.

Example III.1 (Exact reachability analysis of FNN)**.** In this example, we perform the reachability analysis for the FNN in Example II.1 with the input set $I = \{0 \leq x[1] \leq 2 \wedge 0 \leq x[2] \leq 2\}$. Note that, the vector $x = [1 \quad 1]^T$ is the center of the input set. Using Algorithm 2, the exact reachable sets of the hidden layer $L2$ and the output layer $L3$ are computed in which each layer's exact reachable set consists of two polyhedra as depicted in Figure 3. It can be seen that the output $y = [0 \quad 3]^T$ corresponding to the input $x = [1 \quad 1]^T$ lies in the output reachable set.

IV. Approximate Reachability Analysis

A. Lazy-approximate reachability analysis

We have investigated the exact scheme for reachability analysis. In this section, we discuss a lazy approximation approach for reachability analysis of FNN with ReLU activation functions. This lazy-approximate scheme does not perform any stepReLU operations. Instead, it finds the smallest hyper-rectangle that bounds an affine-mapped set $\bar{I}_i$. Since $ReLU(x) \geq 0$, the output ranges of a layer can be quickly derived by the lower-bound and upper-bound information in each dimension of the state vector x in the obtained hyper-rectangle. Figure 4 illustrates the idea of the lazy-approximate scheme. Algorithm 3 is the pseudo-code of the parallel, lazy-approximate scheme.

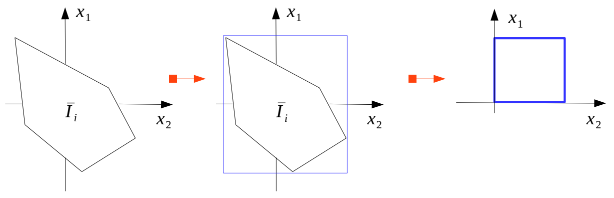

Figure 4: Lazy-approximate scheme.

Algorithm 3 Parallel-lazy reachable set approximation for one layer

Input: I, W, b % input set, weight matrix, bias vector
Output: R % an over-approximate reachable set

```
1: procedure $B$ = LAZYRELU($I_1$)
2:     $lb \leftarrow lower-bound\ of\ x \in I_1$
3:     $ub \leftarrow upper-bound\ of\ x \in I_1$
4:     $m = length(lb)$
5:     for $i = 1 : m$ do
6:         if $lb(i) \geq 0$ then
7:             $lb(i) = 0$
8:         if $ub(i) \geq 0$ then
9:             $ub(i) = 0$
10: procedure $R$ = LAYERLAZYREACH($I, W, b$)
11:     $n = length(I)$
12:     $R = \emptyset$
13:     parfor $i = 1 : n$ do % parallel for loop
14:         $I_1 = I(i).affineMap(W) + b$
15:         $R_1 = lazyReLU(I_1)$
16:         $R = R \cup R_1$
17:     end parfor
```

Example IV.1 (Lazy output range analysis of FNN)**.** In this example, we perform a lazy reachability analysis for the FNN in Example II.1 with the same input set as in the Example III.1. The resulted over-approximate reachable sets of the hidden and output layers are depicted in Figure 5. We can see that the output ranges of the hidden layer can be derived precisely from its lazy, over-approximate reachable set. However, the output ranges of the output layer are conservatively approximated by its corresponding lazy, over-approximate reachable set. We observe that the error in over-approximation is accumulated quickly over layers. Therefore, the lazy-scheme is only useful for FNN with a small number of layers.

Lazy-approximate reachability analysis with input partition. As shown in Figure 5, the ranges of the first and second outputs are $[0, \quad 2.5]$ and $[0, \quad 5]$, which are much larger than the exact ranges computed, i.e., $[0, \quad 0.5]$ and $[1, \quad 2]$ as depicted in Figure 3. But, we can improve these over-approximations by partitioning the input set into smaller sets, and then performing the lazy-approximate scheme on these partitioned input sets.

Example IV.2 (Output range analysis of FNN with lazy-ap-

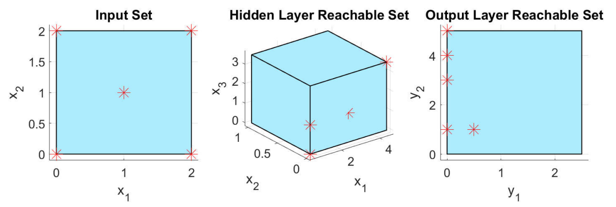

Figure 5: Reachable set of the FFN in Example III.1 using lazy-approximate scheme.

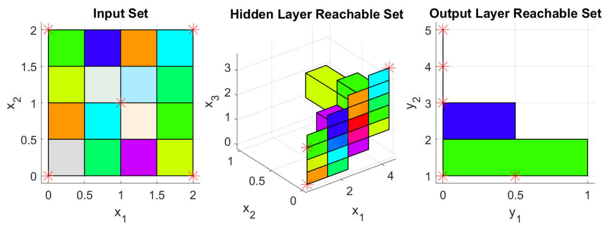

Figure 6: Reachable set of the FFN in Example III.1 using lazy-approximate scheme and input partition.

proximate scheme and input partition)**.** In this example, we partition uniformly the input set $I = \{0 \leq x[1] \leq 2 \wedge 0 \leq x[2] \leq 2\}$ into 16 smaller sets. We perform Algorithm 3 on these partitioned sets. The over-approximate reachable sets of the hidden and output layers are given in Figure 6. As shown in the figure, the output ranges of the FNN precisely match the one obtained by the exact scheme.

We have discussed in detail the lazy-approximate scheme. Next, we consider a combination of both schemes to control the number of polyhedra in the reachable set while still obtaining a over-approximate reachable set with an acceptable conservativeness.

B. *Mixing reachability analysis*

The mixing reachability analysis scheme is proposed with three main objectives: 1) controlling the number of polyhedra in the reachable set; 2) reducing the reachable set computation time, and 3) producing an acceptable over-approximation of the actual reachable set. For the first objective, the mixing scheme lets users choose a maximum number of polyhedra N_{max} to represent the reachable set. The mixing scheme computes the exact reachable set layer-by-layer and observes the number of polyhedra N in the reachable set. When $N > N_{max}$, the mixing scheme performs clustering and merging algorithm to merge N polyhedra into N_{max} hyper-rectangles. Then, it continues computing a reachable set for the rest layers using the lazy-approximate scheme with the N_{max} hyper-rectangles as input sets. The conservativeness of the resulting reachable set comes from two sources. The first source is the error caused by merging N polyhedra into N_{max} hyper-rectangles. The second source is the error from using the lazy-approximate scheme to compute the reachable set. We need to minimize

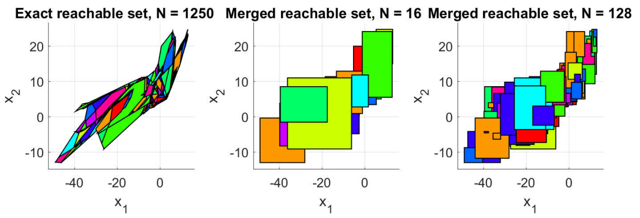

Figure 7: Clustering and merging reachable set.

these errors in order to improve the conservativeness of the result. In this paper, we propose an efficient clustering and merging algorithm by analyzing how much the polyhedra overlap to reduce error. The second source of error remains a challenge.

Assume we want to merge N polyhedra into N_{max} hyper-rectangles such that the union of these hyper-rectangles tightly over-approximates the union of the N polyhedra. Our clustering and merging algorithm works as follows. First, we obtain N smallest hyper-rectangles that bound the N polyhedra. Then, based on the lower-bound lb and upper-bound ub vectors of these hyper-rectangles, we compute the overlapness OLN between the N polyhedra which is defined by

$$OLN_{i,j} = \sqrt{(lb_i - lb_j)^2 + (ub_i - ub_j)^2}, \qquad (1)$$

where $lb_i(ub_i), lb_j(ub_j) \in \mathbb{R}^n$ are lower-bound (upper-bound) vectors of polyhedron i and j respectively, and n is the number of neurons of the layer.

Using the overlapness data and the well-known clustering algorithm *K-means* [8], we cluster N polyhedra into N groups and merge all polyhedra in each group into one hyper-rectangle by interfering the lower-bound and upper-bound information of all hyper-rectangles bounding the polyhedra in the group.

Example IV.3 (Clustering and merging polyhedra)**.** In this example, we perform the clustering and merging algorithm on a reachable set with 1250 polyhedra [9]. These polyhedra are clustered and merged into 16 and 128 hyper-rectangles. Figure 7 shows the result of our algorithm. We note that even using a small number of hyper-rectangles, we can tightly over-approximate the actual reachable set with more than one thousand polyhedra. Another benefit of this algorithm is its speed. The conservativeness in the merging process can be reduced by increasing the number of hyper-rectangles representing the reachable set.

Conservativeness and computation time reduction. Although the clustering and merging algorithm can reduce the first source of error significantly, the second source of error in the mixing scheme can still lead to a very conservative reachable set. This is due to the over-approximation error of the lazy-approximate scheme, which accumulates over the layers. In general, in any approximation scheme we use, e.g., zonotope-based approximation scheme [10], the

Algorithm 4 Mixing scheme for reachable set approximation for one layer

```
Input: $I$, $W$, $b$, $N_{max}$ % Input set, weight matrix, bias vector, maximum number of polyhedra
Output: $R$ % an reachable set
1: procedure $R$ = MIXINGREACH($I, W, b, N_{max}$)
2:     $n = length(I)$
3:     if $n < N_{max}$ then
4:         $R = layerReach(I, W, b)$ % Algorithm 2
5:     else
6:         $I_1 = cluster\_merge(I, N_{max})$ % clustering and merging
7:         $R = layerLazyReach(I_1, W, b)$ % Algorithm 3
```

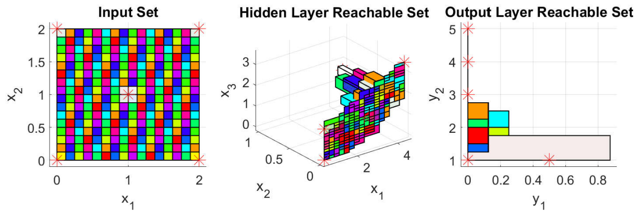

Figure 8: Mixing reachability analysis of the FNN in Example III.1.

accumulation of the over-approximation error is inevitable. Therefore, choosing an appropriate number of polyhedra N_{max} to represent the reachable set is crucial. This number should be chosen in a way that the lazy-approximate scheme is only invoked in the last layer or last two layers to guarantee an acceptable reachable set. It is worth noting that from our experiments in the exact scheme, the reachable set computation time of the last two layers of an FNN usually constitutes $> 50\%$ the total reachable set computation time. This is because: 1) the last two layers take a vast number of polyhedra as input sets, and 2) the constraints of each polyhedron in the input sets are large. Note that the number of constraints of a reachable set increases over the layers due to the intersection operations in the computation process, e.g., see Algorithm 1. Since the lazy-approximate scheme is usually much faster than the exact scheme, choosing N_{max} is a trade-off problem between conservativeness and computation time reduction. If we use the lazy-approximate scheme for the last layer, we usually reduce $\approx 30\%$ of the total reachable set computation time while still precisely deriving the output ranges of a FNN. Algorithm 4 summarizes the main steps of the mixing scheme.

Example IV.4 (Mixing reachability analysis of FNN)**.** In this example, we reuse the FNN in Example III.1 in which the input set is partitioned into 256 smaller sets. We want to compute the reachable set of this FNN with these partitioned input sets using the mixing scheme in which the maximum allowable number of polyhedra is chosen to be $N_{max} = 100$. In this case, if we use the exact scheme to compute the reachable set, the numbers of polyhedra in the reachable set of the hidden and output layers are 276 and 281 respectively. Using the mixing scheme with $N_{max} = 100$, the number of polyhedra of the reachable sets of the hidden and output layers is reduced to 100. The final result is displayed in Figure 8, noticing that some polyhedra are overlapping others.

V. Evaluation

We implement our approach in a MATLAB® toolbox called, *NNV* using the set library in MPT toolbox [11]. Our methods are evaluated using the computation time, scalability and conservativeness via a set of real-world FNN with several to a thousand neurons. The experimental results presented in this section are reproducible. The scripts for reproducing the results are available at https://github.com/verivital/nnv/tree/master/nnv0.1/examples/Submission/FORMALISE2019.

A. Safety verification for ACAS Xu DNNs

In this case study, we verify the safety of ACAS Xu DNNs[1] using our exact scheme. The ACAS Xu networks consist of 45 DNNs which are trained to replace a traditional memory-consuming lookup table that maps the sensor measurements to advisories in the *Airborne Collision Avoidance System X* [5], [7]. Each DNN denoted by N_{x_y} has 5 inputs, 5 outputs, 6 hidden layers in which each layer consists of 50 neurons. A vertical view of a generic example of the ACAS Xu benchmark set is given in Figure 9. The inputs of ACAS Xu neural networks 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.

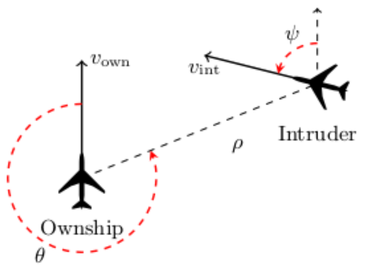

Figure 9: Vertical view of a generic example of the ACAS Xu benchmark set.

[1]https://github.com/guykatzz/ReluplexCav2017

We present two properties that are checked to be satisfied or unsatisfied by ACAS Xu benchmarks using our exact scheme. We refer readers to [7] for more properties.

- **Property ϕ_1.** (Which is property Φ_3 in [7])
  - 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_2.** (Which is property Φ_4 in [7])
  - 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: $1500 \leq \rho \leq 1800$, $\theta \leq |0.06|$, $\psi = 0$, $v_{own} \geq 1000$, $700 \leq v_{int} \leq 800$.

We use the exact scheme on a computer with 4 cores to rigorously compute the exact, output reachable sets of the ACAS Xu networks. Table I shows the verification results of our exact scheme. Our results demonstrate that the number of polyhedra in the output reachable set N_p varies with different properties and may be very large. By pre-processing the input set, our exact scheme can reduce a vast amount of stepReLU operations N_r. The reachable set computation time (RT) dominates the verification time (VT) in our approach. The benefit of our approach is that once the whole output reachable set is obtained, it can be used to verify different properties (defined on the same input conditions) quickly without re-running the analysis. Notably, unlike Reluplex [7], an SMT-solver based approach, out approach can visualize explicitly the behavior of a DNN which is convenient for intuitively checking the safety property. For instance, Figure 10 illustrates the projected, normalized reachable set of the output of the N_{2_9} network for property ϕ_2 which requires that the score (i.e., value) of the COC output is not the minimal score compared with others. We can quickly observe from the figure that the property ϕ_2 holds on N_{2_9} network.

To show the benefit of our parallelizable exact scheme, we evaluate the exact scheme by checking the property ϕ_1 on the network N_{2_9} using a different number of cores. Figure 11 shows that the reachable set computation time reduces significantly as the number of cores increases which implies that our approach is very promising for computing exact reachable set of DNNs using multi-core platforms.

Although there are scalable over-approximation approaches that can be used to compute the reachable set of DNNs [10], [12], the exact reachable set is still important since in many cases, the over-approximation approaches cannot verify the safety properties. For example, Sherlock [12] can be used to compute the output ranges of N_{2_9} network which are $14.66 \leq WeakRight \leq 14.7$, $-0.101 \leq WeakLeft \leq -0.0419$, $14.9470 \leq COC \leq 15.138$, $15.0195 \leq StrongRight \leq 15.0735$, and $1.1817 \leq StrongLeft \leq 1.2634$. Using this output range information, Sherlock can verify property ϕ_2 requiring that COC output is not the minimal score compared with others. However, if we want to verify whether or not the reachable set of the network reaches the unsafe region defined by $\phi_2' \triangleq 15.03 \leq StrongRight \leq 15.04 \wedge 1.24 \leq StrongLeft \leq 1.22$, Sherlock's reachable set contains the unsafe region, and thus the safety of the network is unknown in this case. In contrast, using our exact reachable set, we can prove that the reachable set of the network does not reach the unsafe region ϕ_2', and thus the network is still safe in this case.

Property	FNN	Safe/Unsafe	Exact scheme				
			N_p [a]	N_r	*RT (sec)* [b]	*ST (sec)*	*VT (sec)*
ϕ_1	N_{2_4}	safe	345	6599	4635.7	2.17	4637.87
	N_{2_9}	safe	188	3193	2135.53	2.7454	2138.28
	N_{5_9}	safe	107	2489	1036.06	0.6399	1036.7
ϕ_2	N_{2_9}	safe	16	436	248.8	0.2452	249.05
	N_{3_8}	safe	295	3975	3281.47	1.9107	3283.38
	N_{5_7}	safe	34	1838	522.04	0.7288	522.77

[a] N_p is number of polyhedra in the output reachable set,
N_r is the number of stepReLU operations reduced.
[b] RT is the reachable set computation time, ST is the safety checking time,
$VT = RT + ST$ is the total verification time.

Table I: Verification results of ACAS XU networks. This experiment is done on a computer with following configurations: Intel Core i7-6700 CPU @ 3.4GHz $\times$ 8 Processor, 62.8 GiB Memory, 64-bit Ubuntu 16.04.3 LTS OS

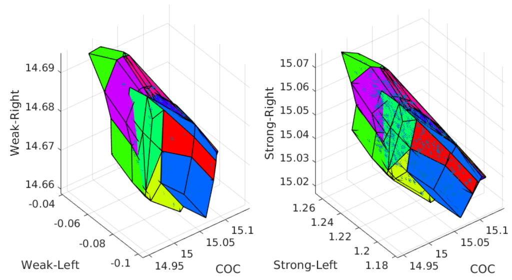

Figure 10: A projected (and normalized) output reachable set for property ϕ_2 on ACAS XU N_{2_9} network.

B. Local adversarial robustness of DNNs

Adversarial robustness of DNNs has become a hot research topic recently since many safety-critical applications rely on the image classification DNNs that have been shown to be vulnerable to adversarial inputs [13]. Adversarial attacks that slightly perturb a correctly classified input can lead to misclassification by a network. A network is said to be $\delta-locally-robust$ 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' [7].

We train a set of image classification DNNs with different

FNN	Cores	Exact			Approximate			Approximate & Partition			Mixing		
		T(sec)	*R(%)*	*Output*	*T(sec)*	*R(%)*	*Output*	*T(sec)*	*R(%)*	*Output*	*T(sec)*	*R(%)*	*Output*
MNIST$_1$ $i = 784, o = 1, l = 6, n = 141$	1	243.57	0	[0.9099, 0.9561]	0.0051	0	[0, 10.22]	0.267	0	[0,6.527]	163.01	0	[0, 2.308]
	2	153.33	37.05					0.7153	-168		118.74	27.16	
	4	142.07	41.67					0.778	-191		114.335	29.86	
MNIST$_2$ $i = 784, o = 1, l = 5, n = 250$	1	684.6	0	[0.9901, 0.9934]	0.0062	0	[0, 5.37]	0.34	0	[0.3446, 2.0788]	72.73	0	[0.6196, 1.4329]
	2	328.5	52.02					0.65	-91		51.23	29.55	
	4	222.8	67.46					0.8	-135		45.13	37.95	
MNIST$_3$ $i = 784, o = 1, l = 2, n = 1000$	1	Timeout			0.0486	0	[0.7788, 1.2575]	9.5771	0	[0.9060, 1.1409]	Timeout		
	2							6.567	31.43				

[a] i is the number of inputs, o is the number of outputs, l is the number of layers, n is the total number of neurons.
[b] T is the reachable set computation time, R is the time reduction in percentage, *Output* is the output reachable set.

Table II: Local adversarial robustness of MNIST networks. This experiment is done on a computer with following configurations: Intel Core i7-6700 CPU @ 3.4GHz $\times$ 8 Processor, 62.8 GiB Memory, 64-bit Ubuntu 16.04.3 LTS OS

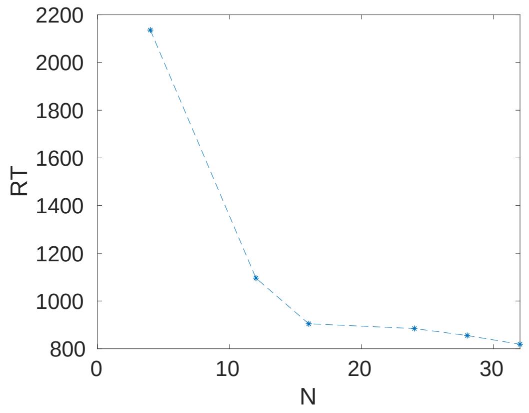

Figure 11: Reachable set computation time (RT) (seconds) for property ϕ_1 on N_{2_9} network with different number of cores (N). This experiment is done on a computer with following configurations: Intel Xenon(R) CPU E5-2620 V4@2.10GHz $\times$ 32, 125.8 GiB Memory, 64-bit Ubuntu 16.04.5 LTS OS.

architectures using the well-known MNIST data set [14] with an accuracy of 95%. The MNIST data set consists of 60000 images of handwritten digits with a resolution of 28×28 pixels. The trained networks have 784 inputs and 1 output. We verify the local adversarial robustness of trained networks on an image of the digit one with the assumption that there is an adversarial attack on a set of pixels of the image. The attack modifies the (normalized) values of the input vector x at these pixels i by some bounded disturbance δ, i.e., $|x[i] - x'[i]| \leq \delta$.

The robustness verification results are demonstrated in Table II with assumption that there are 7 pixels attacked with the bounded disturbance $\delta = 0.7$. The results demonstrate that the computation times of the exact and mixing schemes are reduced significantly with parallel computing. The more cores we use, the more the computation time is reduced. For the lazy-approximate scheme, parallel computing does not help much since there is only one input set (a hyper-rectangle) for all layers in computation. Notably, for the $MNIST_1$ and $MNIST_2$ networks, using parallel computing takes more time than a single core. This is because the time for opening a new parallel pool in Matlab dominates the computation time. We can see that the lazy-approximate scheme and its combination with input partition are much faster than the exact and the mixing schemes. However, the computed output reachable sets are much more conservative when dealing with *deep* networks, i.e., $MNIST_1$ and $MNIST_2$. While the exact scheme can prove that the $MNIST_1$ and $MNIST_2$ networks are robust since the computed outputs are close to 1, the lazy-approximate scheme and its combination with input partitioning and the mixing scheme cannot. The mixing scheme displays its benefit in the case of $MNIST_2$ where it is much faster than the exact scheme while still can prove the robustness of the network. For the $MNIST_3$ network, the exact and mixing scheme reach the time limitation for computation which is set 10 minutes. In this case, the lazy-approximate scheme and its combination with input partition show their strengths by being able to prove the robustness of the networks with very small computation times.

VI. Related Work

Neural network verification has attracted significant attention in numerous research communities such as formal methods, computer vision, machine learning, and security. In recent years, several verification methods and software tools have been proposed for verifying the adversarial robustness and safety of DNNs. Within neural network verification there are two main approaches. The first technique is the exact approach that explicitly analyzes the behavior of neural networks. Typically these techniques consist of an exact reachable set computation for FNNs [9] or SMT-solver based approaches [7], [15]. The second set of techniques are the over-approximation approaches that typically perform reachable set estimation using zonotopes [10], symbolic intervals [16], simulation-based approximation [17] and fast computation of certified robustness for networks employing the use of ReLUs [18].

Our proposed exact scheme lies in the first approaches. Unlike the SMT-solver based approaches, the exact scheme rigorously computes the reachable set of a neural network and thus can visualize the behavior of the network. The main difference between our exact scheme and the one proposed in [9] is that our exact scheme computes the reachable set of FNNs by performing a sequence of stepReLU operations which is more efficiently executed on parallel platforms. Therefore, our exact-scheme can be sped-up significantly on a multi-core computer or clusters. We also note that the SMT-based approach was not designed to work in parallel. Our lazy-approximate and mixing schemes are over-approximation approaches. The conservativeness of our over-approximation schemes are evaluated by comparing the over-approximate reachable set with the exact one. Using the exact scheme, evaluating the conservativeness of the existing over-approximation techniques [10], [16], [18] is feasible and will be worth consideration in the our future work.

VII. Conclusion

In this paper, we have proposed three parallel-computing based reachability analysis schemes for FNNs with ReLU activation functions. From thorough experiments, we have shown that our methods are applicable to many practical problems. The exact scheme has successfully verified safety properties of the real-world ACAS Xu DNNs and can also be used to verify the local adversarial robustness of image classification DNNs. The lazy-approximate scheme and its combination with the input partition technique are useful for output range analysis of an FNN with one or two layers that may consist of a huge number of neurons. We note that in practice, there may be properties that cannot be verified by only using the output range information, and hence, the exact scheme is the only choice in this case. The mixing scheme is suitable for output range analysis of DNNs since it is faster than the exact scheme and produces less conservative results than the lazy-approximate scheme. In future work, we are going to extend the proposed reachability analysis for FNN with nonlinear activation functions, recurrent neural networks, and convolution neural networks.

Acknowledgments

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS 1464311, CNS 1713253, SHF 1527398, and SHF 1736323, the Air Force Office of Scientific Research (AFOSR) through contract numbers FA9550-15-1-0258, FA9550-16-1-0246, and FA9550-18-1-0122, and the Defense Advanced Research Projects Agency (DARPA) through contract number FA8750-18-C-0089. The U.S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of AFOSR, DARPA, or NSF.

References

[1] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[5] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *arXiv preprint arXiv:1810.04240*, 2018.

[6] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[7] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[8] P. S. Bradley and U. M. Fayyad, "Refining initial points for k-means clustering." in *ICML*, vol. 98. Citeseer, 1998, pp. 91–99.

[9] W. Xiang, H.-D. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with relu activations," *arXiv preprint arXiv:1712.08163*, 2017.

[10] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai 2: Safety and robustness certification of neural networks with abstract interpretation," in *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.

[11] M. Kvasnica, P. Grieder, M. Baotić, and M. Morari, "Multi-parametric toolbox (mpt)," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2004, pp. 448–462.

[12] S. Dutta, S. Jha, S. Sanakaranarayanan, and A. Tiwari, "Output range analysis for deep neural networks," *arXiv preprint arXiv:1709.09130*, 2017.

[13] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[14] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[15] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[16] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," *arXiv preprint arXiv:1804.10829*, 2018.

[17] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multi-layer neural networks," *arXiv preprint arXiv:1708.03322*, 2017.

[18] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, "Towards fast computation of certified robustness for relu networks," *arXiv preprint arXiv:1804.09699*, 2018.