Hierarchical Verification for Adversarial Robustness

Cong Han Lim [1] **Raquel Urtasun** [1 2] **Ersin Yumer** [1]

Abstract

We introduce a new framework for the exact point-wise ℓ_p robustness verification problem that exploits the layer-wise geometric structure of deep feed-forward networks with rectified linear activations (ReLU networks). The activation regions of the network partition the input space, and one can verify the ℓ_p robustness around a point by checking all the activation regions within the desired radius. The GeoCert algorithm (Jordan et al., 2019) treats this partition as a generic polyhedral complex in order to detect which region to check next. In contrast, our LayerCert framework considers the *nested hyperplane arrangement* structure induced by the layers of the ReLU network and explores regions in a *hierarchical* manner. We show that, under certain conditions on the algorithm parameters, LayerCert *provably* reduces the number and size of the convex programs that one needs to solve compared to GeoCert. Furthermore, our LayerCert framework allows the incorporation of lower bounding routines based on convex relaxations to further improve performance. Experimental results demonstrate that LayerCert can significantly reduce both the number of convex programs solved and the running time over the state-of-the-art.

1. Introduction

Deep neural networks have been demonstrated to be susceptible to adversarial perturbations of the inputs (e.g., Szegedy et al. (2013); Biggio et al. (2013); Goodfellow et al. (2014)). Hence, it is important to be able to measure how vulnerable a neural network may be to such noise, especially for safety-critical applications. We study the problem of pointwise *exact* verification for ℓ_p-norm adversarial robustness for trained deep feed-forward networks with ReLU activation functions. The point-wise ℓ_p robustness with respect to an input $x \in \mathbb{R}^n$ and a classifier $\mathrm{c} : \mathbb{R}^n \to [C] \coloneqq \{1, 2, 3, \ldots, C\}$ is defined as

$$\epsilon^*(x; c) \coloneqq \min_{\|v\|_p \leq \epsilon} \epsilon \text{ s.t. } \mathrm{c}(x + v) \neq \mathrm{c}(x). \qquad (1)$$

The goal of exact or complete robustness verification is to check if $\epsilon > r$ for some desired radius r. The choices of p studied in the literature are typically $1, 2,$ and ∞; our work applies to all $p \geq 1$. Solving Problem (1) exactly (or within a factor of $1 - o(1) \ln n$) is known to be NP-hard (Weng et al., 2018). Developing methods that perform well in practice would require a better understanding of the mathematical structure of neural networks.

In this work, we approach the problem from the angle of how to directly exploit the geometry induced in input space by the neural network. Each activation pattern (i.e., whether each neuron is on or off) corresponds to a polyhedral region in the input space, and the decision boundary within each region is *linear*. A natural geometric approach to the verification problem is then to check regions in order of their distance. We illustrate this in Figure 1. We can terminate this process either when we have reached the desired verification radius or when we have exceeded the distance to the closest decision boundary found. In the latter case the distance to that boundary is the solution to Problem (1).

Jordan et al. (2019) proposed the first algorithm for this distance-based exploration of the regions. Their GeoCert algorithm navigates the regions in order of distance by maintaining a priority queue containing all the polyhedral faces that make up the frontier of all regions that have been visited. The priority associated with each face is computed via an optimization problem that can be solved by a generic convex programming solver. Under a limited time budget, GeoCert finds a stronger computational lower bound for $\epsilon^*(x; c)$ compared to a complete method that directly uses mixed-integer programming (Tjeng et al., 2019).

In this paper we introduce the *LayerCert* framework that makes use of the *layer-wise* structure of neural networks. The first layer of ReLUs induce a hyperplane arrangement structure, and each subsequent layer induces one within each region of the hyperplane arrangement from the previous layer. This forms a nested hyperplane pattern and a

---

Work done as part of first author's AI Residency Program. [1]Uber Advanced Technologies Group, Toronto ON, Canada [2]Department of Computer Science, University of Toronto, Toronto ON, Canada. Correspondence to: Cong Han Lim <conghan@uber.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

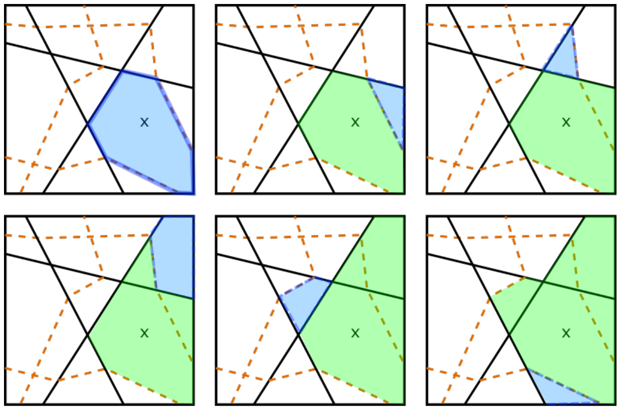

Figure 1. How geometric methods explore the input space. Each square illustrates one step in the process of exploring regions of increasing ℓ_2 distance from the initial point x. The box represents the input space to a ReLU network, the inner black lines the regions induced by three first layer ReLUs, and brown lines the regions by another three ReLUs in the second layer. The blue regions are being processed during that step, while the green regions have already been processed.

hierarchy of regions/subregions that our algorithm will use to navigate the input space. This hierarchical approach has two main advantages over GeoCert:

1. *Provable* reduction in the number and size of convex programs solved when using a convex distance-based priority function.

2. The ability to incorporate convex-relaxation-based *lower bounding/incomplete methods* for Problem (1) to reduce the number of regions that are processed.

We demonstrate the first advantage by studying a simplified version of LayerCert (LayerCert-Basic), which introduces our hierarchical approach to navigating the regions but does not include the use of additional subroutines to prune the search space. By making use of the nested hyperplane structure, LayerCert *provably* reduces the number of convex programs and the sizes of each program that need to be solved compared to GeoCert when using the same convex distance priority function. This is done by identifying a minimal set of programs that are required and by amortizing the work associated with a single region in GeoCert across multiple levels of the hierarchy in LayerCert.

The second advantage comes from the fact that each region R in the i-th level of the hierarchy is associated with a set of children regions in the $(i+1)$-th level that is contained entirely within R. This allows us to use incomplete verifiers over just this region to determine if the region might intersect with a decision boundary. If the verifier returns that no such overlap exists, we can then safely remove the region and all its children from further consideration. One straightforward way to do this is to use efficient reachability methods such as interval arithmetic (Xiang et al., 2018). In this work we also develop a novel method that leverages *linear lower bounds* on the neural network function (Weng et al., 2018; Zhang et al., 2018) to construct a half-space that is guaranteed not to contain any part of the decision boundary. This can restrict the search space significantly. Furthermore, we can warm start LayerCert by projecting the input point onto the half-space.

Using the experimental setup of Jordan et al. (2019), we compare the number of programs solved and overall wall-clock time of different variants of GeoCert and LayerCert. Each LayerCert method uses a different combination of lower-bounding methods. For GeoCert, we consider different choices of priority functions. In addition to the standard ℓ_p distance priority function, Jordan et al. (2019) describes a non-convex priority function that incorporates a Lipschitz term. This Lipschitz variant modifies the order in which regions are processed and also provides an alternative warm-start procedure. Our LayerCert variants consistently outperform GeoCert using just the ℓ_p distance priority function and in most experiments our lower-bounding techniques outperform the Lipschitz variant of GeoCert.

Notation. We use $[k]$ to denote the index set $\{1, 2, \ldots, k\}$. Superscripts are used to index distinct objects, with the exception of $\mathbb{R}^n$ and $\mathbb{R}^+$ to denote the n-dimensional Euclidean space and the nonnegative real numbers respectively. We use subscripts for vectors and matrices to refer to entries in the objects and subscripts for sets to refer to distinct sets. We use `typewriter_fonts` to denote subroutines.

2. Related work

Besides GeoCert (Jordan et al., 2019), the majority of exact or complete verification methods are based on branch-and-bound (e.g. Katz et al. (2017); Wang et al. (2018); Tjeng et al. (2019); Anderson et al. (2019a); Lu & Kumar (2020)), and Bunel et al. (2018; 2019) provide a detailed overview. These methods construct a search tree over the possible individual ReLU activations and use upper and lower bounds to prune the tree. The upper bounds come from adversarial examples, while the lower bounds are obtained by solving a relaxation of the original problem, which we briefly discuss in the next paragraph. Since our focus in this work is on methods that directly leverage the geometry of the neural network function in the input space, we leave a detailed comparison against these methods to future work.

Instead of exactly measuring or verifying, we can instead overapproximate or relax the set reachable by an ϵ-ball around the input point (e.g. Dvijotham et al. (2018); Xiang et al. (2018); Singh et al. (2018); Weng et al. (2018); Wong

& Kolter (2018); Zhang et al. (2018); Singh et al. (2019b)). These *incomplete* approaches give us a *lower bound* for Problem (1). These can only certify that there is no perturbation that changes the class within some radius r for some r that can be much smaller than $\epsilon^*(x; c)$ The majority of the works focus on different convex relaxations of the ReLU activations (see Salman et al. (2019) for a discussion of the tightness of the different approaches), though recent works have started going beyond single ReLU relaxations (Singh et al., 2019a; Anderson et al., 2019b). Another line of work studies how to efficiently bound the Lipschitz constant of the neural network function (Szegedy et al., 2013; Bartlett et al., 2017; Balan et al., 2017; Weng et al., 2018; Combettes & Pesquet, 2019; Fazlyab et al., 2019; Zou et al., 2019).

The complexity of geometric methods (and many other exact methods) for robustness verification can be upper bounded by a constant × (number of activation regions) × (complexity of solving a convex program). There have been several works in recent years that study the number of these activation regions in ReLU networks. The current tightest upper and lower bounds for the maximum number of nonempty activation regions, which are exponential in the number of ReLU neurons, is given by Serra et al. (2018). Hanin & Rolnick (2019) provide an upper bound on the expected number of regions (exponential in the lesser of the input dimension and the number of ReLU neurons).

3. Hierarchical Structure of ReLU Networks

In this section, we describe the structure induced by the ReLU neurons in the input space and how this leads naturally to a hierarchy of regions that we use in our approach.

3.1. Hyperplane Arrangements

Definition 3.1. *(Hyperplanes and hyperplane arrangements) A hyperplane $H \subset \mathbb{R}^n$ is an $(n-1)$-dimensional affine space that can be written as $\{x \mid a^\intercal x = b\}$ for some $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. A hyperplane arrangement $\mathcal{H}$ is a finite set of hyperplanes.*

Definition 3.2. *(Halfspaces) Given a hyperplane $H := \{x \mid a^\intercal x = b\}$, the halfspaces $H^{\leq}$ and $H^{\geq}$ correspond to the sets $\{x \mid a^\intercal x \leq b\}$ and $\{x \mid a^\intercal x \geq b\}$, respectively. A polyhedron is a finite intersection of halfspaces.*

Definition 3.3. *(Patterns and regions in hyperplane arrangements) Given a hyperplane arrangement $\mathcal{H}$ and a pattern $P : \mathcal{H} \to \{-1, +1\}$, the corresponding region is*

$$R_+ := \bigcap_{P(H)=-1} H^{\leq} \cap \bigcap_{P(H)=+1} H^{\geq}.$$

We say that another pattern $Q : \mathcal{H} \to \{-1, +1\}$ (or region R_Q) is a neighbor of P (R_P resp.) if P and Q differ only on a single hyperplane.

3.2. Geometric Structure of Deep ReLU Networks

A ReLU network with L hidden layers for classification with C classes and n_i neurons in the i-th layer for $i \in [L]$ can be represented as follows:

$$\begin{aligned} x^0 &:= x && \text{input}, \\ z^i &:= W^{i-1}x^{i-1} + b^{i-1} && \text{for } i \in [L+1], \\ x^i &:= \text{relu}(z^i) && \text{for } i \in [L], \\ f(x) &:= W^L x^L + b^L = z^{L+1} && \text{output}, \end{aligned} \tag{2}$$

where W^i, b^i describe the weight matrix and bias in the i-th layer and the ReLU function is defined as $\text{relu}(v))_i := \max(0, v_i)$. The length of b^i is n^{i+1} for $i \in [L-1]$ and $b^0 \in \mathbb{R}^n$ and $b^L \in \mathbb{R}^C$. The classification decision is given by $\text{argmax}_i(f(x))_i$, and the decision boundary between classes i and j is the set $\{x \mid f_i(x) - f_j(x) = 0\}$.

Note that layers like batch normalization layers, convolutional layers, and average pooling layers can be included in this framework by using the appropriate weights and biases. We can also have a final softmax layer since it does not affect the decision boundaries as it is a symmetric monotonically increasing function.

Definition 3.4. *(Full activation patterns) Let n_i denote the number of neurons in the i-th layer. A (full) activation pattern $A = (A_1, \dots, A_L)$ is a collection of functions $A_i : [n_i] \to \{-1, +1\}$. Two activation patterns are neighbors if they differ on exactly one layer for exactly one neuron.*

We can define a *neighborhood graph* over the set of activation patterns where each node presents a pattern and we add an edge between neighboring patterns.

Definition 3.5. *(Activation patterns and the input space) For an input x, the (full) activation pattern of x is A^x where*

$$A_i^x(j) := \begin{cases} +1 & \text{if } z_j^i \geq 0, \\ -1 & \text{if } z_j^i < 0, \end{cases} \tag{3}$$

where the z_j^i terms are defined according to (2). Conversely, given an activation pattern A, the corresponding activation region *is $R_A := \{x \mid A^x = A\}$.*

Given an activation pattern A and some $x \in R_A$, the corresponding z^{i+1} terms for $i \in [L]$ are given by

$$z^{i+1} = W^i I^{A_i} z^i + b^i \tag{4}$$

where I^{A_i} is the diagonal $n_i \times n_i$ binary matrix such that

$$I_{j,k}^{A_i} := \begin{cases} 1 & \text{if } j = k \text{ and } A_i(j) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

By letting

$$c^i := \sum_{j=0}^{i} \left(\left(\prod_{k=j+1}^{i} W^k I^{A_k} \right) b^j \right)$$

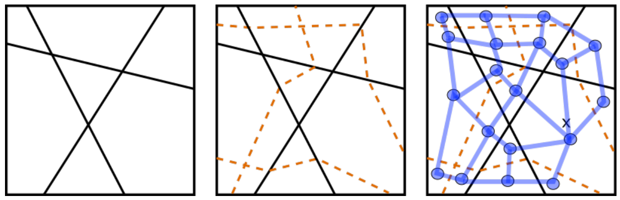

Figure 2. Left: the activation regions of a ReLU network with one hidden layer forms a hyperplane arrangement. Middle: the activation regions of a ReLU network with two hidden layers. Note that within the regions defined by the previous layer, the lines induce a hyperplane arrangement. Right: The neighborhood graph of the regions.

we can expand Eq. (4) as

$$z^{i+1} = \left(\prod_{j=1}^{i} W^j I^{A_j}\right) W^0 x + c^i. \quad (5)$$

Hence, each z^i term and $f(x)$ can be expressed as a linear expression over x involving W^i, I^{A_i}, and b^i terms. This also allows us to write R_A in the form of linear inequalities over x as a polyhedron

$$\{x \mid A_i(j) z^i_j \geq 0 \text{ for } i \in [L], j \in [n_i]\} \quad (6)$$

Since the neural network function f is linear within each activation region, each decision boundary is also linear within each activation region. This allows us to efficiently compute the classification decision boundaries.

The set of activation regions for a network with one hidden layer corresponds to the regions of a hyperplane arrangement (where each row of W^0 and the corresponding entry in b^0 defines a hyperplane). With each additional layer, we take the regions corresponding to the previous layer and add a hyperplane arrangement to each region. Thus, this leads to a *nested hyperplane arrangement*. Figure 2 illustrates this structure and the corresponding neighborhood graph.

In addition to full activation patterns, it is useful to consider the patterns for all neurons up to a particular layer.

Definition 3.6. *(Partial activation patterns and regions) Given some $l < L$, an l-layer partial activation pattern $A = (A_1, \ldots, A_l)$ is a collection of functions $A_i : [n_i] \to \{-1, +1\}$. The corresponding partial activation region R_A is $\{x \mid A_i(j) z^i_j \geq 0$ for $i \in [L], j \in [n_i]\}$.*

The partial activation regions naturally induce a hierarchy of regions. We can describe the relationship between the regions in the different levels in the following terms:

Definition 3.7. *For a l-layer activation pattern $A = (A_1, \ldots, A_l)$, let $\mathrm{parent}(A) := (A_1, \ldots, A_{l-1})$. The terms* child *and* descendant *are defined analogously.*

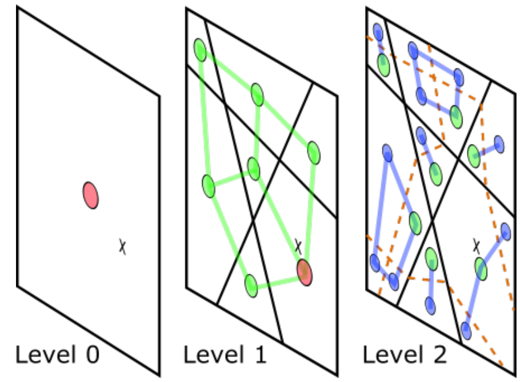

Figure 3. The hierarchical structure induced by the partial activation regions. Each partial region is marked by a circle node. The nodes of the same color represent the closest subregion within a parent region to the input point x. For example, the upper left most green nodes in levels 1 and 2 are connected.

Definition 3.8. *Two l-layer partial activation patterns are siblings if they share the same parent pattern. They are neighboring siblings if they differ on exactly one neuron in the l-th layer and agree everywhere else.*

We can use Defintion 3.7 and 3.8 to define a *hierarchical search graph* with $L + 1$ levels. The nodes in the l-th level represent the l-layer activation patterns. We connect two activation patterns in the same level if they are neighboring siblings. We connect a pattern A to $\mathrm{parent}(A)$ if R_A is the region closest to the input point x out of all its siblings. We introduce a single node in the level 0 and connect it to the first-layer activation pattern that contains x. Figure 3 illustrates this hierarchy of activation regions and the corresponding hierarchical search graph. In Sections 4 and 5, we describe how to leverage this hierarchical structure to design efficient algorithms for verification.

4. Exploring Activation Regions Geometrically

In this section, we first describe the GeoCert algorithm (Jordan et al., 2019) that provides a method for navigating the activation regions in order of increasing distance from the input. We subsequently consider the hierarchy of partial regions and describe LayerCert-Basic that leverages the geometric structure to provably reduce the number and size of convex programs compared to GeoCert. We then introduce the full LayerCert framework in Section 5.

4.1. Prior Work: GeoCert

GeoCert performs a process akin to breadth-first search over the neighbourhood graph (see Figure 2). In each iteration, GeoCert selects the activation region corresponding to the

Algorithm 1 GeoCert

Input: x, y (label of x), U (upper bound)
$A^x \leftarrow$ activation pattern of x
$Q \leftarrow$ empty priority queue
$Q.\mathtt{push}((0, A^x))$
$S \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  $(d, A) \leftarrow Q.\mathtt{pop}()$
  if $A \in S$ **then**
    continue
  $S \leftarrow S \cup \{A'\}$
  if $U \leq d$ **then**
    return U
  $U \leftarrow \min(\mathtt{decision\_bound}(A, x, y), U)$
  for $A' \in N(A) \setminus S$ **do**
    if $\mathrm{Face}(A, A')$ is nonempty **then**
      $d' \leftarrow \mathtt{priority}(x, \mathrm{Face}(A, A'))$
      $Q.\mathtt{push}((d', A'))$

closest unexplored node neighbouring an explored node and then computes the distance to all regions neighbouring the selected region.

We provide a formal description of GeoCert in Algorithm 1 and demonstrate an iteration of the algorithm in Figure 4. Setting the input U term to a radius r solves the robustness verification problem for that radius, while setting it sufficiently high measures the robustness (i.e., Problem (1)).

We describe the subroutines in detail below.

Measuring the distance to a distance boundary restricted to a region. The subroutine $\mathtt{decision\_bound}$ with inputs A, x, y solves the problem

$$\min_{v \in R_A} \|x - v\|_p \quad \text{s.t.} \quad f_y(v) - f_j(v) = 0 \text{ for } j \neq y \tag{7}$$

or returns ∞ if infeasible. This is equivalent to computing the ℓ_p projection of x onto the respective set. We can use algorithms specifically designed for projection onto sets such as Dykstra's method (Boyle & Dykstra, 1986). We can also use generic convex optimization solvers to handle a wider range of priority functions.

Computing the priority function. From (6), we can write each activation region R_A as a polyhedron $\{x \mid A_i(j) z_j^i \geq 0 \text{ for } i \in [L], j \in [n_i]\}$. A neighbouring region R'_A that differs only on the a-th neuron on the b-th layer will intersect with R_A within the hyperplane $H = \{x \mid z_a^b \geq 0\}$. We name the set $R_A \cap H = R'_A \cap H$ as $\mathrm{Face}(A, A')$ since this set is a face of both A and A'. As with R_A and (6), we can write $\mathrm{Face}(A, A')$ as

$$\{x \mid A_i(j) z_j^i \geq 0 \text{ for } i \in [L], j \in [n_i], z_b^a = 0\} \tag{8}$$

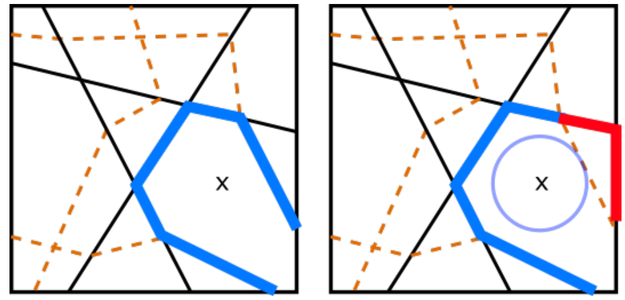

Figure 4. One iteration of GeoCert . The ball represents a level set of the priority function and the blue line the set of boundary faces of the explored regions. After popping the nearest unexplored activation region off a priority queue, GeoCert computes the distance of previously unseen faces in that region (marked in red).

where z_j^i can be expressed in terms of the x variables using the expression in (5). Given some function $q : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$, the subroutine $\mathtt{priority}$ with inputs $x, \mathrm{Face}(A, A')$ solves the following optimization problem:

$$\min_{v \in \mathrm{Face}(A, A')} q(x, v). \tag{9}$$

For ℓ_p-robustness, a natural choice of q is the ℓ_p distance function. Jordan et al. (2019) also propose an alternative variant of GeoCert where they incorporate an additional $\min_{y \neq j} \frac{f_y(v) - f_j(v)}{L}$ term in the priority function, where L denotes an upper bound on the Lipschitz constant. This makes the priority function nonconvex. We will refer to the ℓ_p priority variant of GeoCert as just GeoCert and specifically use the term GeoCert-Lip for the Lipschitz variant.

4.2. Our Approach – LayerCert

Instead of exploring the neighborhood graph of full activation regions, we develop an algorithm that makes use of the nested hyperplane arrangement and the graph induced by it. See Definition 3.7 and Figure 3 for a description of this graph. We first give a description of a basic form of LayerCert in Algorithm 2, followed by theoretical results about this method.

In each iteration, LayerCert processes the next nearest *partial activation pattern* by computing the distances to all *sibling patterns* to the queue. This is in contrast to GeoCert which computes the distance to all neighbouring patterns of the next nearest *full* activation pattern.

The new subroutine in LayerCert-Basic is $\mathtt{next\_layer}$, which takes an l-layer activation pattern A and an input vector v and returns the $l+1$ layer activation pattern that is a child of A and contains v:

$$\mathtt{next\_layer}(A, v) := (A_1, \ldots, A_l, A_{l+1}^v)$$

where A^v is the full activation pattern of v (Definition 3.5).

Algorithm 2 LayerCert-Basic

Input: x, y (label of x), U (upper bound)
$A^x \leftarrow$ activation pattern of x at first layer
$Q \leftarrow$ empty priority queue
$Q.\texttt{push}((0, A^x, x))$
$S \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  $(d, A, v) \leftarrow Q.\texttt{pop}()$
  if $U \leq d$ **then**
    return U
  if A is a full activation pattern **then**
    $U \leftarrow \min(\texttt{decision\_bound}(A, x, y), U)$
  else
    $Q.\texttt{push}((d, \texttt{next\_layer}(A, v), v))$
  for $A' \in N_{\text{current\_layer}}(A) \setminus S$ **do**
    if Face(A, A') is nonempty **then**
      $v', d' \leftarrow \texttt{priority}(x, \text{Face}(A, A'))$
      $Q.\texttt{push}((d', A', v'))$
      $S \leftarrow S \cup \{A'\}$

Algorithm 3 LayerCert Framework

Input: x, y (label of x), U (upper bound)
$d, A, v, M \leftarrow \texttt{restriction}(x, U)$
$Q \leftarrow$ empty priority queue
$Q.\texttt{push}((d, A, v))$
$S \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  $(d, A, v) \leftarrow Q.\texttt{pop}()$
  if $U \leq d$ **then**
    return U
  if A is a full activation pattern **then**
    $U \leftarrow \min(\texttt{decision\_bound}(A, x, y), U)$
  else
    if $\texttt{contains\_db}(A, x, U) =$ 'maybe' **then**
      $Q.\texttt{push}((d, \texttt{next\_layer}(A, v), v))$
  for $A' \in N_{\text{current\_layer}}(A) \setminus S$ **do**
    if Face$(A, A') \cap M$ is nonempty **then**
      $v', d' \leftarrow \texttt{priority}(x, \text{Face}(A, A') \cap M)$
      $Q.\texttt{push}((d', A', v'))$
      $S \leftarrow S \cup \{A'\}$

We prove in the appendix that when the priority function is a convex distance function, LayerCert-Basic visits full activation patterns in ascending order of distance, which implies that it returns the correct solution.

Theorem 4.1. *(Correctness of LayerCert-Basic) If the priority function used is a convex distance function, LayerCert processes full activation patterns in ascending order of distance and returns the distance of the closest point with a different class from x.*

As a corollary of Theorem 4.1, LayerCert-Basic and GeoCert equipped with the same priority function visit the activation patterns in the same order (allowing for permutations of patterns with the exact same priority).

The primary difference in computational difficulty between the two methods is that the number and complexity of `priority` computations is reduced in LayerCert. There are three main reasons for this. First, LayerCert-Basic adds A to the set of seen patterns S once we have processed any neighbour of A. Hence, we only do a single `priority` computation for each A. This is not necessarily the case in GeoCert. Secondly, the convex programs corresponding to nodes further up the hierarchy have less constraints since they only need to consider all neurons to the respective layer. Finally, in LayerCert-Basic we do not need to compute `priority` between two l-layer partial activation patterns that differ on exactly a single neuron that is not in layer l (i.e. these patterns are not siblings). GeoCert in contrast computes the priority between any two neighbours as long as the corresponding Face is non-empty. This allows us to amortize the number of convex programs to compute for a single full activation region in GeoCert over multiple levels in LayerCert — Consider a full activation pattern $A = (A_1, \ldots, A_L)$ and the set of all ancestor patterns $B^1, \ldots, B^{L-1}$ where $B^i \coloneqq (A_1, \ldots, A_i)$. We have $\sum_i^L n_i$ potential neighbours for GeoCert when processing pattern A. For LayerCert-Basic, we have up to n_i siblings when processing each B^i, for a total of $\sum_i^L n_i$ patterns when processing $B^1, \ldots, B^{L-1}, A$.

These facts can be used to prove our main theoretical result about the complexity of LayerCert-Basic. The proof of these and the main theorem are in the appendix.

Theorem 4.2. *(Complexity of LayerCert-Basic) Given an input x, suppose the distance to the nearest adversary returned by LayerCert/GeoCert is not equal to the distance of any activation region from x. Suppose we formulate the convex problems associated with `decision_bound` and `next_layer` using Formulations (6) and (8). We can construct an injective mapping from the set of convex programs solved by LayerCert to the corresponding set in GeoCert such that the constraints in the LayerCert program is a subset of those in the corresponding GeoCert program.*

5. The LayerCert Framework

We now describe how our hierarchical approach is amenable to the use of convex relaxation-based lower bounds to prune the search space. For simplicity, in this section we will assume that we are performing verification with respect to a *targeted* class y'. The general LayerCert framework is presented in Algorithm 3. The two new subroutines in the algorithm are `contains_db` and `restriction`.

Let $B_{p,r}(x)$ denote the ℓ_p-ball of radius r around x. The subroutine contains_db(A, x, r) returns 'false' when $R_A \cap B_{p,r}(x)$ is *guaranteed not* to intersect with a decision boundary. This means that we can remove A and all its descendants from consideration. Otherwise, it returns 'maybe' and we proceed on to the next level.

The routine restriction(x, r) explicitly computes a convex set M that is a superset of the part of the decision boundary contained in $B_{p,r}(x)$. This both restricts the search directions we need to consider and also allows us to *warm start* the algorithm by projecting the initial point onto this set. In the following we describe some possible choices for these subroutines. In the appendix we discuss a version of LayerCert that recursively applies a modified version of restriction to aggressively prune the search space.

Pruning partial activation regions. We can use incomplete verifiers (see Section 2 for references) to check if the region $B_{p,U}$(x) *might* contain a decision boundary. These methods work by implicitly or explicitly computing some overapproximation of the set

$$\{f_y(v) - f_{y'}(v) \mid v \in B_{p,U}(x)\}. \tag{10}$$

If all values in (10) are strictly positive, then we know that the decision boundary cannot be in $R_A \cap B_{p,U}(x)$.

Many incomplete verifiers work by constructing convex relaxations of each nonlinear ReLU function. If a neuron is guaranteed to be always on or always off over all points in the region of interest, we can use this to tighten the convex relaxation. This allows us to incorporate information from the current partial activation region into incomplete verifiers. For our experiments, we choose to use the efficient interval arithmetic approach (first applied in the context of neural network verification by Xiang et al. (2018)). Below we describe a version of the method that incorporates information about partial regions:

$$\begin{aligned} X^0 &:= B_{p,U}(x), \\ Z^i &:= W^{i-1}X^{i-1} + b^{i-1} && \text{for } i \in [L+1], \\ X^i &:= \text{relu}(\text{Box}_{A_i}(Z^i)) && \text{for } i \in [L], \end{aligned}$$

where $\text{Box}_{A_i}(Z^i)$ is the set of vectors v satisfying

$$\begin{aligned} &\min_{w \in Z^i} w_j \le v_j \le \max_{w \in Z^i} w_j && \text{if } A_i(j) = +1, \\ &v_j = 0 && \text{if } A_i(j) = -1. \end{aligned}$$

We can use the fact that $f(B_{p,U}(x) \cap R_A) \subseteq Z^{L+1}$ to check if we need to explore the descendants of A.

Warm starts via restricting search area. Certain incomplete verifiers implicitly generate enough information to allow us to efficiently compute a convex set M that contains the decision boundary. We will describe how to do this for verifiers based on simple linear underestimates of $f_y - f_{y'}$ in the ball $B_{p,U}(x)$ such as Fast-Lin (Weng et al., 2018) and CROWN (Zhang et al., 2018). These methods construct a linear function $g \le f_y - f_{y'}$ by propagating linear and upper bounds of the ReLUs through the layers. The resulting set $\{v \mid g(v) \le 0\}$ is a halfspace that we can efficiently project onto. Figure 5 illustrates this concept. For the purposes of this paper we use the lower bound from the CROWN verifier (Zhang et al., 2018) to compute a halfspace M.

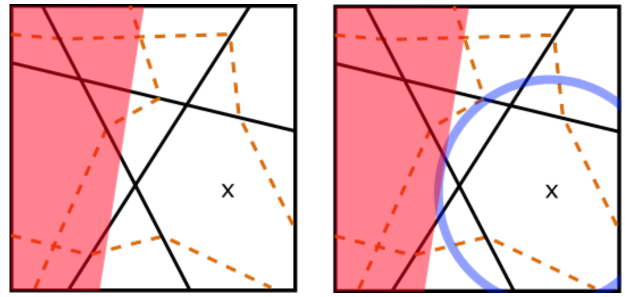

Figure 5. Computing a set that contains the decision boundary and warm starting. The red shaded region corresponds to the region that contains any potential decision boundary. By identifying this region, we can warm start the algorithm by initializing the search at the closest point in the red region.

Once we have computed M, we can use it throughout our algorithm. In particular, if a face does not overlap with M, we can remove it from consideration in our algorithm. We discuss this in more detail in the appendix.

Instead of just using linear approximations, we can also use tighter approximations such as the linear programming relaxation that models single ReLUs tightly (see for example Salman et al. (2019)) These result in a smaller convex set M that is still guaranteed to contain the decision boundary but overlaps with less regions, resulting in a reduction in the search space. The drawback of using such methods is that the representation of M can get significantly more complicated, which in turn increases the cost of solving Problems (7) and (9).

6. Experimental Evaluation

We use an experimental setup similar to the one used in Jordan et al. (2019). The two experiments presented here are taken directly from them, except for an additional neural network in the first and a larger neural network in the second. Additional results are presented in the appendix.

We consider a superset of the networks used in GeoCert. The fully connected ReLU networks we use in the paper have two to five hidden layers, with between 10 to 50 neurons in each layer. As with Jordan et al. (2019), we train our

	[10, 50, 10]		2 x [20]		3 x [20]		4 x [20]		5 x [20]		2 x [30]	
	#LPs	Time (s)	#LPs	Time (s)	#LPs	Time (s)	#LPs	Time (s)	#LPs	Time (s)	#LPs	Time (s)
GeoCert	48.8	3.68	488.0	22.85	3297.7	212.32	275.2	26.46	82.0	9.50	92.0	6.51
GeoCert-Lip	39.4	4.19	183.4	11.11	1055.0	90.23	154.7	18.36	61.3	8.79	55.1	4.91
LayerCert	28.4	1.56	227.7	8.60	1617.8	62.53	111.8	5.62	44.0	3.36	45.9	2.28
LayerCert-IA	26.4	1.34	208.1	7.72	1258.9	44.17	90.7	3.52	38.6	2.66	45.6	2.23
LayerCert-CROWN	26.8	1.46	116.7	5.80	648.1	24.34	73.7	3.43	31.1	2.36	39.1	1.93
LayerCert-Both	25.4	1.38	116.7	5.82	613.0	22.63	67.2	2.83	29.9	2.08	39.1	1.91

	3 x [30]		4 x [30]		5 x [30]		2 x [50]		3 x [50]		4 x [50]	
	#LPs	Time (s)	#LPs	Time (s)	#LPs	Time (s)	#LPs	Time (s)	#LPs	Time (s)	#LPs	Time (s)
GeoCert	867.4	88.92	18.0	1.48	392.7	96.08	1739.7	252.82	1567.1	410.47	814.8	336.03
GeoCert-Lip	253.5	41.88	16.4	1.61	216.2	70.47	506.9	117.66	923.8	316.74	510.0	269.39
LayerCert	440.4	25.52	12.1	0.68	176.7	22.00	1620.0	111.61	3131.7	239.81	826.3	99.61
LayerCert-IA	408.5	21.69	10.7	0.56	164.4	19.25	1778.5	111.63	3400.4	216.51	771.9	83.38
LayerCert-CROWN	184.2	9.76	11.5	0.63	122.2	14.84	378.4	24.49	1959.2	158.46	438.6	57.67
LayerCert-Both	184.2	9.74	10.5	0.56	117.0	13.81	378.3	24.21	1843.9	127.97	422.2	53.81

Table 1. Average number of convex programs and running time over 100 inputs for 12 different networks for ℓ_∞-distance. The green shaded entries indicate the best performing method for each neural network under each metric. The gray shaded entries indicates the algorithm timed out before an exact solution to Problem (1) could be found for *at least one input*. Whenever a timeout occurs, we use a time of 1800 seconds in its place, which leads to an underestimate of the true time.

networks to classify the digits 1 and 7 in the MNIST dataset and used the same training parameters. We consider ℓ_∞ distance here and also ℓ_2 distance in the appendix.

Methods evaluated. We test the following variants of GeoCert and LayerCert. All variants of LayerCert use an ℓ_p distance priority function.

- GeoCert with ℓ_p distance priority function.
- GeoCert-Lip: GeoCert with $\|x - v\|_p + \min_{y\neq j} \frac{f_y(v)-f_j(v)}{L}$ priority function, where L denotes an upper bound on the Lipschitz constant found by the Fast-Lip method (Weng et al., 2018).
- LayerCert-Basic.
- LayerCert with interval arithmetic pruning.
- LayerCert with warmstart in the initial iteration using CROWN (Zhang et al., 2018) to underestimate $f_y - f_j$.
- LayerCert-Both: with both interval arithmetic pruning and CROWN-based warm start.

We initialize each algorithm with a target verification radius of 0.3. Each method terminates when we have exactly computed the answer to (1) or when it has determined that the lower bound is at least the radius.

Implementation details. Our implementation of GeoCert (Jordan et al., 2019) starts with the original code provided by the authors at `https://github.com/revbucket/geometric-certificates` and modifies it to make use of open-source convex programming solvers to enable further research to a wider community, as the original code uses Gurobi (Gurobi Optimization, 2019), which is a commercial software. We use CVXPY (Diamond & Boyd, 2016) to model the convex programs and the open-source ECOS solver (Domahidi et al., 2013) as the main solver. Since ECOS can occasionally fail, we also use OSQP (Stellato et al., 2017) and SCS (O'donoghue et al., 2016) as backup solvers. In our tests, ECOS and OSQP are significantly faster than SCS which we use only as a last resort.

We implemented LayerCert in Python using the packages numpy, PyTorch, numba (for the lower bounding methods), and the aforementioned packages for solving convex programs. Our experiments were performed on an Ubuntu 18.04 server on an Intel Xeon Gold 6136 CPU with 12 cores. We restricted the algorithms to use only a single core and do not allow the use of the GPU. All settings, external packages, and compute are identical for all the methods compared.

6.1. Exact measurement experiments.

We randomly picked 100 '1's and '7's from MNIST and collected the wall-clock time and number of linear programs solved. Since some instances can take a very long time to solve fully, we set a time limit of 1800s for each instance.

Averaged metrics. We measure (1) the average of the wall-clock time taken to measure the distances and (2) the average of number of linear programs solved. The first metric is what matters in practice but is heavily dependent

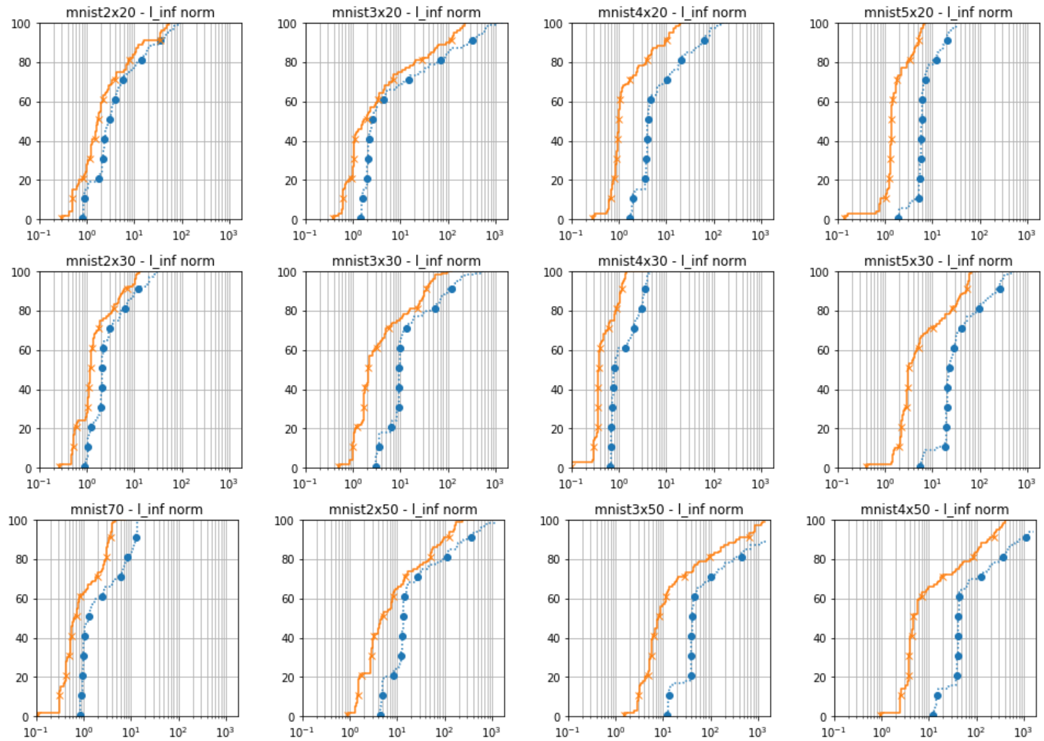

Figure 6. Performance profiles comparing LayerCert-Both (orange, 'x' markers) with GeoCert-Lip (blue, dotted, circle markers) over 12 different networks using ℓ_∞ norm. Marks are placed for every 10 input points.

on the machine and the solver used, whereas the second metric provides a system-independent proxy for time. We present the results in Table 1. The basic GeoCert method is consistently outperformed in both metrics by our methods, while GeoCert-Lip always improves on GeoCert in terms of number of LPs and often in terms of run time. The method with the fastest run time is always one of the three LayerCert variants that use lower bounds. With the exception of the $3 \times [50]$ network where all methods timed out, the method with the least number of convex programs solved is always LayerCert-Both. Plain LayerCert always outperforms both GeoCert methods in terms of timing, while it often outperforms GeoCert-Lip in number of LPs. In some cases only one of the two lower bounding methods helps significantly, though the use of both methods at most includes a small overhead. Thus, we recommend in general using both.

Performance profiles. Performance profiles (Dolan & Moré, 2002) are a commonly-used technique to benchmark the performance of different optimization methods over *sets of instances*. A performance profile is a plot of a cumulative distribution, where the x-axis denotes the amount of time each method is allowed to run for, and the y-axis the number of problems that can be solved within that time period. If the performance profile of a method is consistently above the performance profile of another method, it indicates that the former method is always able to solve more problems regardless of the time limit. We illustrate the performance profiles for GeoCert-Lip and LayerCert-Both in Figure 6. LayerCert-Both consistently dominates GeoCert-Lip.

7. Conclusion

We have developed a novel hierarchical framework LayerCert for exact robustness verification that leverages the nested hyperplane arrangement structure of ReLU networks. We prove that a basic version of LayerCert is able to reduce the number and size of the convex programs over GeoCert using the equivalent priority functions. We showed that LayerCert is amenable to the use of lower bounding methods that use convex relaxations to both prune and warm start the algorithm. Our experiments showed that LayerCert can significantly outperform variants of GeoCert.

Acknowledgments

We would like to thank the anonymous reviewers for suggestions that helped greatly to improve the presentation of this work. We would also like to thank Eilyan Bitar, Ricson Cheng, and Jerry Liu for helpful discussions.

References

Anderson, G., Pailoor, S., Dillig, I., and Chaudhuri, S. Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pp. 731–744, Phoenix, AZ, USA, June 2019a. Association for Computing Machinery. ISBN 978-1-4503-6712-7.

Anderson, R., Huchette, J., Tjandraatmadja, C., and Vielma, J. P. Strong Mixed-Integer Programming Formulations for Trained Neural Networks. In Lodi, A. and Nagarajan, V. (eds.), *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, pp. 27–42. Springer International Publishing, 2019b. ISBN 978-3-030-17953-3.

Balan, R., Singh, M., and Zou, D. Lipschitz properties for deep convolutional networks. *arXiv preprint arXiv:1701.05217*, 2017.

Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pp. 6240–6249, 2017.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 387–402. Springer, 2013.

Boyle, J. P. and Dykstra, R. L. A Method for Finding Projections onto the Intersection of Convex Sets in Hilbert Spaces. In Dykstra, R., Robertson, T., and Wright, F. T. (eds.), *Advances in Order Restricted Statistical Inference*, Lecture Notes in Statistics, pp. 28–47, New York, NY, 1986. Springer. ISBN 978-1-4613-9940-7.

Bunel, R., Turkaslan, I., Torr, P. H., Kohli, P., and Kumar, M. P. A Unified View of Piecewise Linear Neural Network Verification. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 4795–4804, USA, 2018. Curran Associates Inc. event-place: Montréal, Canada.

Bunel, R., Lu, J., Turkaslan, I., Torr, P. H. S., Kohli, P., and Kumar, M. P. Branch and Bound for Piecewise Linear Neural Network Verification. *arXiv:1909.06588 [cs, stat]*, November 2019. arXiv: 1909.06588.

Combettes, P. L. and Pesquet, J.-C. Lipschitz certificates for neural network structures driven by averaged activation operators. *arXiv preprint arXiv:1903.01014*, 2019.

Diamond, S. and Boyd, S. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Dolan, E. D. and Moré, J. J. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, January 2002. ISSN 1436-4646.

Domahidi, A., Chu, E., and Boyd, S. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pp. 3071–3076, 2013.

Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., and Kohli, P. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18)*, pp. 162–171, Corvallis, Oregon, 2018. AUAI Press.

Fazlyab, M., Robey, A., Hassani, H., Morari, M., and Pappas, G. J. Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks. *arXiv:1906.04893 [cs, math, stat]*, June 2019. arXiv: 1906.04893.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Gurobi Optimization, L. Gurobi optimizer reference manual, 2019. URL http://www.gurobi.com.

Hanin, B. and Rolnick, D. Deep relu networks have surprisingly few activation patterns. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 359–368. Curran Associates, Inc., 2019.

Jordan, M., Lewis, J., and Dimakis, A. G. Provable Certificates for Adversarial Examples: Fitting a Ball in the Union of Polytopes. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 14059–14069. Curran Associates, Inc., 2019.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In Majumdar, R. and Kunčak, V. (eds.), *Computer Aided Verification*, Lecture Notes in Computer Science, pp. 97–117, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63387-9.

Lu, J. and Kumar, M. P. Neural Network Branching for Neural Network Verification. In *International Conference on Learning Representations*, 2020.

O'donoghue, B., Chu, E., Parikh, N., and Boyd, S. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

Salman, H., Yang, G., Zhang, H., Hsieh, C.-J., and Zhang, P. A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 9832–9842. Curran Associates, Inc., 2019.

Serra, T., Tjandraatmadja, C., and Ramalingam, S. Bounding and Counting Linear Regions of Deep Neural Networks. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4558–4566, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR.

Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and Effective Robustness Certification. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10802–10813. Curran Associates, Inc., 2018.

Singh, G., Ganvir, R., Püschel, M., and Vechev, M. Beyond the Single Neuron Convex Barrier for Neural Network Certification. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 15072–15083. Curran Associates, Inc., 2019a.

Singh, G., Gehr, T., Püschel, M., and Vechev, M. An Abstract Domain for Certifying Neural Networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, January 2019b. ISSN 2475-1421.

Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. OSQP: An operator splitting solver for quadratic programs. *ArXiv e-prints*, November 2017.

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

Tjeng, V., Xiao, K. Y., and Tedrake, R. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*, 2019.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1599–1614, Baltimore, MD, August 2018. USENIX Association. ISBN 978-1-939133-04-5.

Weng, L., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Daniel, L., Boning, D., and Dhillon, I. Towards Fast Computation of Certified Robustness for ReLU Networks. In *International Conference on Machine Learning*, pp. 5276–5285, July 2018.

Wong, E. and Kolter, Z. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *International Conference on Machine Learning*, pp. 5286–5295, July 2018.

Xiang, W., Tran, H.-D., and Johnson, T. T. Output Reachable Set Estimation and Verification for Multilayer Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, November 2018. ISSN 2162-2388.

Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. Efficient Neural Network Robustness Certification with General Activation Functions. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 4939–4948. Curran Associates, Inc., 2018.

Zou, D., Balan, R., and Singh, M. On lipschitz bounds of general convolutional neural networks. *IEEE Transactions on Information Theory*, 2019.