Precise Multi-Neuron Abstractions for Neural Network Certification

Mark Niklas Müller[*]
Department of Computer Science
ETH Zurich
Zurich, Switzerland
mark.mueller@inf.ethz.ch

Gleb Makarchuk[*]
Department of Computer Science
ETH Zurich
Zurich, Switzerland
gleb.makarchuk@gmail.com

Gagandeep Singh
VMware Research and UIUC
United States
gagandeepsi@vmware.com
ggnds@illinois.edu

Markus Püschel
Department of Computer Science
ETH Zurich
Zurich, Switzerland
pueschel@inf.ethz.ch

Martin Vechev
Department of Computer Science
ETH Zurich
Zurich, Switzerland
martin.vechev@inf.ethz.ch

Abstract

Formal verification of neural networks is critical for their safe adoption in real-world applications. However, designing a verifier which can handle realistic networks in a precise manner remains an open and difficult challenge.

In this paper, we take a major step in addressing this challenge and present a new framework, called PRIMA, that computes precise convex approximations of arbitrary non-linear activations. PRIMA is based on novel approximation algorithms that compute the convex hull of polytopes, leveraging concepts from computational geometry. The algorithms have polynomial complexity, yield fewer constraints, and minimize precision loss.

We evaluate the effectiveness of PRIMA on challenging neural networks with ReLU, Sigmoid, and Tanh activations. Our results show that PRIMA is significantly more precise than the state-of-the-art, verifying robustness for up to 16%, 30%, and 34% more images than prior work on ReLU-, Sigmoid-, and Tanh-based networks, respectively.

Keywords: Robustness, Certification, Convexity

1 Introduction

With the growing adoption of neural networks (NNs) in many critical real-world domains, it is imperative to guarantee their safety and robustness [47]. While the last few years have seen significant progress in formal verification of NNs, existing methods either do not scale or are too imprecise for proving robustness of realistic networks.

Key challenge: handling non-linearities. Computations in neural networks involve the application of non-linear activations in a layerwise manner. These functions can be applied many thousands of times leading to highly non-linear output regions. The main challenge in neural network verification is in designing methods that can handle the effect of these non-linear functions in a precise and scalable manner.

Exact verifiers [1, 8, 9, 16, 23, 25, 26, 29, 49, 51] typically do not scale to larger networks due to their exponential time complexity. To overcome this, incomplete verifiers [7, 30, 33, 40, 44, 45, 50, 52, 53, 57] overapproximate the effects of non-linear functions by designing suitable convex abstractions. Most of these verifiers [7, 30, 33, 40, 44, 45, 50, 52, 53, 57] are based on single neuron convex approximations, i.e., activations are approximated separately. This approach leads to significant imprecision, as dependencies between neurons in the same layer are ignored. As a result, the approximation error grows exponentially with each layer in the worst-case, leading to imprecise verification results. To mitigate this limitation for ReLU networks, recent work [43, 48] considers multi-neuron approximations that capture neuron interdependencies. Singh et al. [43] group neurons of a layer into small subsets of size $k > 1$ and then compute a convex hull *jointly* approximating the output of k ReLUs. Tjandraatmadja et al. [48] merge the activation layer with the preceding affine layer and compute a convex approximation over the resulting multivariate activation. These approaches yields state-of-the-art precision but are expensive as the NP-hard convex hull problem has to be solved exactly for the small subsets or approximately for large sets, respectively.

This work: precise multi-neuron approximations. In this work, we push the boundaries of the state of the art in precise neural network verification and present a new general verification framework called PRIMA, PRecIse Multi-neuron Abstraction. PRIMA is based on a novel, general method, called Partial Double Description Method (PDDM), for precise and fast approximation of the convex hull problem for polytopes and can be applied to arbitrary specifications expressible as polyhedra such as individual fairness [37], global safety properties [25], acoustic [38], geometric [4], spatial [37], and ℓ_p bounded perturbations [19]. Using PDDM as a subroutine, PRIMA uses our novel Split-Bound-Lift Method for the convex approximation of non-linear activation layers. Based on these techniques, and as we demonstrate in

[*] Equal contribution

our experimental evaluation, PRIMA produces substantially improved precision results for ReLU-, Sigmoid-, and Tanh-based networks.

Main contributions. Our key contributions are:

1. The PDDM method for the sound and precise approximation of the convex hull computation for polytopes, with worst-case polynomial time and space complexity and exactness guarantees in low dimensions.
2. The Split-Bound-Lift Method which can efficiently compute joint constraints over groups of non-linear functions, by decomposing the underlying convex hull problem into lower-dimensional spaces.
3. A combination of these approaches with a sparse neuron grouping technique, trading-off minimizing the number of groups while maximizing the number of discovered dependencies. This results in what is the first multi-neuron abstraction framework for arbitrary, bounded, multivariate non-linear activation functions such as ReLU, Sigmoid, Tanh, and MaxPool. We refer to this novel verification framework as PRIMA.
4. We experimentally evaluate PRIMA on a range of ReLU-, Sigmoid-, and Tanh-based fully connected and convolutional networks and show that it is significantly more precise than the state-of-the-art, improving precision by up to 14%, 30%, and 34% for ReLU-, Sigmoid-, and Tanh-based networks.
5. We release our code as part of the open source ERAN framework at https://github.com/eth-sri/eran.

2 Problem statement

In this section, we first establish the terminology we use to discuss neural networks (NNs), the notion of robustness and how it can be certified. Then, we explain the main challenge in NN robustness certification and the one addressed by our work: the precise and scalable handling of non-linear activation layers.

Notation. We use lower case Latin or Greek letters $a, b, x, \dots, \lambda, \dots$ for scalars, bold for vectors $\boldsymbol{a}$, capitalized bold for matrices $\boldsymbol{A}$, and calligraphic $\mathcal{A}$ or blackboard bold $\mathbb{A}$ for sets. Similarly, we denote scalar functions as $f : \mathbb{R}^d \to \mathbb{R}$ and vector valued functions bold as $\boldsymbol{f} : \mathbb{R}^{d \cdot k} \to \mathbb{R}^k$.

Neural Networks. We focus our discussion on image classification feedforward networks $\boldsymbol{h}(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|}$ that map input samples (images) $\boldsymbol{x} \in \mathcal{X}$ to output activations $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{Y}|}$. A feedforward network is the interleaved composition of affine functions $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}$, such as normalization, linear, convolutional, or average pooling layers, with non-linear activation layers $\boldsymbol{f}(\boldsymbol{x})$, composed from ReLU, Tanh, Sigmoid, or MaxPool:

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}_L \circ \boldsymbol{f}_L \circ \boldsymbol{g}_{L-1} \circ \dots \circ \boldsymbol{f}_1 \circ \boldsymbol{g}_0(\boldsymbol{x}) \tag{1}$$

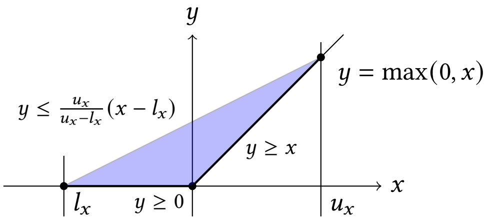


Figure 1. Single-neuron approximation of the ReLU function $f(x) = \max(x, 0)$ with bounded inputs $x \in [l_x, u_x]$. The exact ReLU function (black) is nonconvex. A convex overapproximation is given by the three inequalities describing the blue triangle.

The network $\boldsymbol{h}$ classifies an input sample $\boldsymbol{x}$ by taking the argmax of its output: $c = \arg\max_j h(\boldsymbol{x})_j$.

Adversarial Robustness. An NN $\boldsymbol{h}$ is adversarially robust if the classification of an input $\boldsymbol{x}$ remains unchanged under small perturbations of the input. Formally, it means that $\boldsymbol{h}$ classifies all inputs in the p-norm ball

$$\mathbb{B}^p_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\eta} \mid |\boldsymbol{\eta}|_p \leq \epsilon\} \tag{2}$$

of radius ϵ to the same correct class:

$$\arg\max_j h(\boldsymbol{x})_j = \arg\max_j h(\boldsymbol{x}')_j, \quad \text{for all } \boldsymbol{x}' \in \mathbb{B}^p_\epsilon(\boldsymbol{x}). \tag{3}$$

The parameter ϵ bounds the admissible perturbations $\boldsymbol{\eta}$.

Neural Network Verification. Neural network verification is the task of certifying that a given property holds for all network outputs corresponding to a given input set. While our methods can be used in this general setting, we focus on verifying classification robustness as defined in (3).

Exact verification does not scale [25] due to the thousands of non-linear activations $f : \mathbb{R} \to \mathbb{R}$ that either lead to an exponential blowup of case distinctions (as for ReLU) or complicated shapes (as for Sigmoids). Therefore, state-of-the-art verifiers (e.g., [45, 52]) sacrifice completeness for scalability by overapproximating with convex polyhedra when propagating $\mathbb{B}^p_\epsilon(\boldsymbol{x})$ through the network.

Affine layers map between convex polyhedra and are thus captured exactly. The challenge are the non-linear activations, which introduce errors due to the needed overapproximations. As an example, consider the single-neuron ReLU shown in Figure 1, which maps the input x to $y = \max(0, x)$. If bounds for x are known, $l_x \leq x \leq u_x$, the ReLU can be overapproximated by its convex hull, i.e., the triangle shown. The approximation errors grow exponentially with the network depth and can render these methods ineffective for verifying deeper networks. The challenge here is the trade-off between precision and scalability, discussed next.

Optimal approximation. Given a layer of n neurons, each applying the scalar, univariate, non-linear activation function $f(x)$ and the most precise polyhedral approximation $\mathcal{P}$ of the values for inputs $\boldsymbol{x}$, the most precise convex approximation after applying each f is given by the convex hull $\operatorname{conv}(\{(\boldsymbol{x}, f(\boldsymbol{x})) \mid \boldsymbol{x} \in \mathcal{P} \subseteq \mathbb{R}^n\})$. Computing this hull is intractable due to the exponential cost ($O(v \log(v) + v^n)$ [11]) in the number n of neurons, with the number of vertices v at worst also being exponential ($O(a^n)$ [42]) in the number of neurons, given the number of constraints a.

Neuronwise approximation. For scalability reasons, almost all approximations of non-linear activations [44, 45, 52] operate separately on each neuron, as illustrated in Figure 1 for ReLU. They maintain upper and lower bounds l_x, u_x for each input x and compute convex hulls of all input-output tuples: $\operatorname{conv}(\{(x, f(x)) \mid x \in [l_x, u_x] \subseteq \mathbb{R}\})$. The union of the obtained constraints is the final approximation of the layer. Geometrically, it is the Cartesian product of triangles, which is significantly larger (exponential in n) than the optimal convex hull discussed above.

K-ReLU approximation. A first compromise between the intractable optimal and imprecise but scalable neuronwise approximation for the piecewise-linear ReLU activation was suggested by Singh et al. [43]. Their work computes convex hulls on very small neuron groups of size k with simplified input bounds and combines the resulting constraints from many such groups to approximate the output of the entire layer. However, their approach only considers ReLU activations and relies on solving the convex hull problem exactly for every group in the $2k$-dimensional space. This heavily limits the number and size of groups they can process and therefore how many dependencies they can discover.

Our work. The fundamental challenge in pushing the limits of robustness verification is computing more precise and scalable convex approximations of activation layers. In this work, we address this challenge and introduce a general framework, called PRIMA, which can handle *all* common non-linear activation functions, including ReLU, Tanh, Sigmoid, and MaxPool. PRIMA works by combining a decompositional approach to reduce the dimensionality of the convex hull problems with a novel fast and precise approximate method for convex hull computations. This allows it to process up to two orders of magnitude more neuron groups, which yields much tighter constraints. Overall, PRIMA achieves significantly higher precision as well as a speedup of up to an order of magnitude compared to state-of-the-art methods.

3 Background on Polyhedra

We now introduce the necessary background on polyhedra.

Vertex representation. A polytope $\mathcal{P} \subseteq \mathbb{R}^d$ is the closed convex hull of a set of generators called vertices $\mathcal{R} = \{x_i \in \mathbb{R}^d\}$:

$$\mathcal{P} = P(\mathcal{R}) = \left\{ \sum_i \lambda_i \boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{R},\ \sum_i \lambda_i = 1,\ \lambda_i \in \mathbb{R}_0^+ \right\}.$$

A polyhedral cone $\mathcal{P} \subseteq \mathbb{R}^d$ is the positive linear span of a set of generators called rays $\mathcal{R} = \{x_i \in \mathbb{R}^d\}$:

$$\mathcal{P} = P(\mathcal{R}) = \left\{ \sum_i \lambda_i \boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{R},\ \lambda_i \in \mathbb{R}_0^+ \right\}.$$

The origin is always included in a polyhedral cone. We call this representation of polyhedra vertex- or $\mathcal{V}$-representation.

Halfspace representation. Equivalently, any polyhedron can be described as the set $\mathcal{P} \subseteq \mathbb{R}^d$ satisfying a system of linear inequalities:

$$\mathcal{P} = \mathcal{P}(\boldsymbol{A}, \boldsymbol{b}) \equiv \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}\} \tag{4}$$

with $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$. Geometrically, $\mathcal{P}$ is the intersection of m closed affine halfspaces $\mathcal{H}_i = \{x \in \mathbb{R}^d \mid \boldsymbol{a}_i \boldsymbol{x} \geq b_i\}$ with $\boldsymbol{a}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$. For a polyhedral cone $\boldsymbol{b} = \boldsymbol{0}$. We call this representation halfspace- or $\mathcal{H}$-representation. For convenience, a polytope $\mathcal{P}(\boldsymbol{A}, \boldsymbol{b})$ can be equivalently described in so-called homogenized coordinates $\boldsymbol{x}' = [1, \boldsymbol{x}]$, where it can be expressed as $\mathcal{P}(\boldsymbol{A}') = \{\boldsymbol{x}' \in \mathbb{R}^{d+1} \mid \boldsymbol{A}'\boldsymbol{x}' \geq 0\}$ with the new constraint matrix $\boldsymbol{A}' = [-\boldsymbol{b}, \boldsymbol{A}]$.

We call a face of a d-dimensional polyhedron a vertex if it satisfies d linearly independent constraints[1] with equalityand a facet if it satisfies exactly 1 linearly independent constraint with equality [15].

The rank of a ray or vertex in a d-dimensional polyhedron is the number of linearly independent constraints it satisfies with equality. We call a ray of rank $d-1$ and a vertex of rank d extremal. A ray of rank $d-n$ can be represented as the positive combination of n extremal rays and a vertex of rank $d-n$ as the convex combination of $n+1$ extremal points.

Double description. Polyhedral analysis [18, 34] usually maintains both $\mathcal{H}$-representation and $\mathcal{V}$-representation in a pair $(\boldsymbol{A}, \mathcal{R})$, called double description. This is useful as computing the convex hull in the $\mathcal{V}$-representation is trivial (union of generator sets), but computing intersection is NP-hard. Conversely, computing intersection in $\mathcal{H}$-representation is trivial (union of constraints), but computing the convex hull is NP-hard. The transformation from the $\mathcal{V}$-representation to the $\mathcal{H}$-representation is called the *convex hull* problem and the reverse is called the *vertex enumeration* problem. Both are NP-hard in general and done on demand.

[1] We call a set of constraints $\boldsymbol{a}_i \boldsymbol{x} \geq b_i$ linearly independent, if the $\boldsymbol{a}_i$ are linearly independent.

4 Overview of PRIMA

We now present an overview of PRIMA (for PRecIse Multi-neuron Abstraction), our framework for faster and more precise convex approximation of nonlinear activations. We explain the algorithm step-by-step including its key ideas. Full formalization is provided in subsequent sections.

We assume the setup as outlined in Section 2: (1) an activation layer consisting of n neurons representing non-linear activations $f(x)$ (e.g., ReLU, Tanh, Sigmoid, ...), and (2) an n-dimensional polytope $\mathcal{S}$ constraining the input to the layer and providing neuronwise bounds l_x, u_x.

We obtain the input polytopes $\mathcal{S}$ using a fast LP based verification method and insert PRIMA abstractions to yield the final verifier which we evaluate in Section 7. Note that while we focus on univariate activation functions, PRIMA can also be applied to multivariate activations or support layers that mix different activations.

PRIMA computes a convex overapproximation of the output using the following steps:

1. *Group decomposition:* Decompose the set of n neurons into overlapping groups (subsets) of size k.
2. *Octahedral projection:* For each group compute an octahedral overapproximation $\mathcal{P}$ of the projection of $\mathcal{S}$ to this group.
3. *Split-Bound-Lift Method (SBLM):* For each obtained octahedral input polytope $\mathcal{P}$ for a group of k neurons, compute a convex overapproximation of the output using our novel SBLM method. The SBLM decomposes this problem to lower dimensions where it leverages our novel Partial Double Description Method (PDDM) to compute fast and scalable convex hull approximations and yields an output in $\mathcal{H}$-representation.
4. *Combine constraints:* Finally, take the union of all obtained constraints to obtain the $\mathcal{H}$-representation of the final result.

Our core contribution is SBLM and PDDM, discussed in detail later. Next, we explain the basic workings of each step, identify key ideas, and illustrate the concepts on examples.

Group decomposition. It is infeasible to compute convex hulls for large sets of neurons, thus we restrict to groups of size k, typically $k = 3$–5. The key idea of the grouping is to capture dependencies between neurons and thus achieve higher precision by overlapping them. Considering all possible $\binom{n}{k}$ groups is too expensive; thus we first partition the neurons of a layer into sets of size n_s and then for every set choose a subset of all $\binom{n_s}{k}$ groups that pairwise overlap by at most s, $1 \leq s < k$. This is done by enumerating all and keeping, in one pass, only those satisfying this condition. For example, for $n = 5$ neurons, $k = 3$, and $s = 1$, there are 10 groups, obtaining $\{1, 2, 3\}$ and $\{1, 4, 5\}$ as (a possible) result.

Octahedral projection. Ideally, the next step would project the input polytope $\mathcal{S}$ onto each of the obtained groups with k variables. However, this is generally intractable due to the typically large number of constraints. Therefore, we follow the idea of Singh et al. [43] and overapproximate the projection by a multidimensional octahedron [12]. This is done by first generating all $3^k - 1$ possible non-trivial octahedral constraints. These are of the form $\sum_{i=1}^{k} a_i \cdot x_i \geq b$ for all combinations of $a_i \in \{-1, 0, 1\}^k$. The left hand side b in each case is obtained by minimizing the linear program (LP) given by the constraints of $\mathcal{S}$. Figure 2 visualizes the approximation on a small example for $k = 2$.

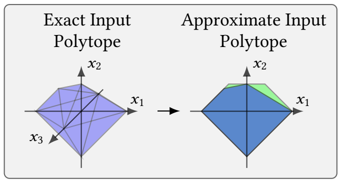


Figure 2. Approximation of the projection of $\mathcal{S} \in \mathbb{R}^3$ (left) to $k = 2$ variables (blue) and its octahedral approximation $\mathcal{P} \in \mathbb{R}^2$ (green).

Split-Bound-Lift Method. The next and most demanding step takes a k-dimensional input polytope for a given k neuron group, and computes a $2k$-dimensional convex overapproximation of the result after applying the corresponding k activations. Our approach is based on three key ideas:

- Decompose the input polytope into regions, which we call quadrants, for which tight or even exact, linear constraints of the activation functions are available.
- Compute the result by extending[2] quadrants with one output variable at a time, bounding this variable using the linear constraints, and computing a convex hull over specific groups of quadrants. Thus, the majority of convex hull computations is done in lower-dimensional spaces, on simpler polyhedra.
- The previous step utilizes our proposed PDDM to compute precise convex hulls, leveraging the concept of duality and our novel PDD polyhedron representation with the associated notion of A-irredundancy.

This decomposition approach, extending quadrants as needed, has two main effects: (i) computing convex hulls approximately using PDDM is exact for polytopes of dimension up to 3 and loses precision only slowly for higher dimensions. Directly computing $2k$-dimensional convex hulls with PDDM will lose more precision than our decomposed method; (ii) a lower-dimensional polytope with fewer constraints, and generally also fewer vertices, significantly reduces the time required for the individual convex hull operations. Even our approximate method scales quartically $(O\{n_a^4 \cdot n_v + n_a^2 \log(n_a^2)\})$ in the number of input constraints n_a and linear in the number of vertices

[2] Extending a polytope by a variables defines it in a space with the corresponding, additional dimension in which it is (initially) unbounded.

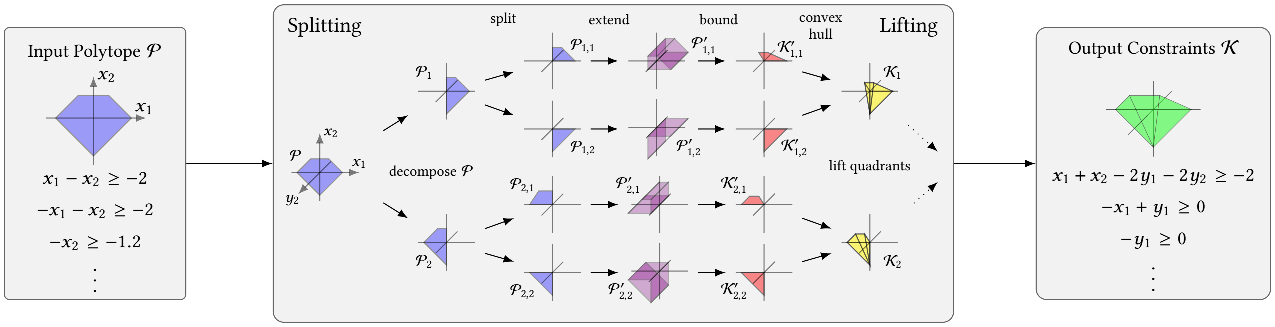


Figure 3. Illustration of the Split-Bound-Lift Method for a group of $k = 2$ neurons and a ReLU activation.

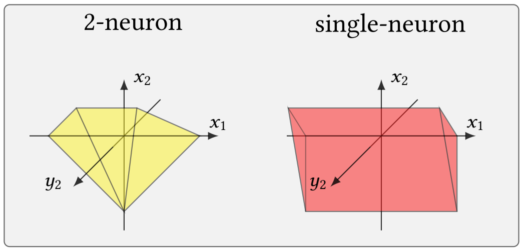


Figure 4. Comparison of 2-neuron and 1-neuron constraints on y_2 for a ReLU activation, given input polytope $\mathcal{P}$.

n_v (see Appendix B for a proof) while optimal exact methods are exponential ($O(n_v \log(n_v) + n_v^{\lfloor d/2 \rfloor})$ [11]) in the number of dimensions and super linear in the number of input vertices. Note that if bounds are not exact on a quadrant, as is the case for non-piecewise-linear activations (e.g., Tanh), the number of vertices doubles for every extended dimension, making exact methods intractable and approximate methods not using the SBLM slow.

We illustrate the Split-Bound-Lift Method in Figure 3 and describe its steps on an example. We assume ReLU activations, $k = 2$, and an octahedral input polytope (left panel in Figure 3) described by:

$$\mathcal{P} = \{x_1 + x_2 \geq -2,\ -x_1 + x_2 \geq -2,\ x_1 - x_2 \geq -2,\\ -x_1 - x_2 \geq -2,\ -x_2 \geq -1.2\}.$$

To partition $\mathcal{P}$ into quadrants, we first chose the ordering $\{y_1, y_2\}$ of output variables and intersect $\mathcal{P}$ first with the halfspaces $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_1 \geq 0\}$ and $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_1 \leq 0\}$ and then $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_2 \geq 0\}$ and $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_2 \leq 0\}$ yielding the second and third column of polytopes in the central panel of Figure 3. For brevity we only follow the bottom path. There the two quadrants $\mathcal{P}_{2,1}$ and $\mathcal{P}_{2,2}$ are described by:

$$\mathcal{P}_{2,1} = \{x_1 - x_2 \geq -2,\ -x_1 \geq 0,\ -x_2 \geq -1.2,\ x_2 \geq 0\},$$
$$\mathcal{P}_{2,2} = \{x_1 + x_2 \geq -2,\ -x_1 \geq 0,\ -x_2 \geq 0\}.$$

Next, in the key part of the algorithm, we lift these quadrants, one output variable at a time. First, we trivially extend all quadrants into the space that includes the y_2 dimension (purple column in Figure 3). There we apply the upper and lower bounds to y_2, which are $y_2 \leq 0$ and $y_2 \geq 0$ for the quadrant $\mathcal{P}_{2,2}$ and, $y_2 \leq x_2$ and $y_2 \geq x_2$ for the quadrant $\mathcal{P}_{2,1}$. This corresponds to applying (a relaxation of) the activation function yielding the two polytopes (red column):

$$\mathcal{K}'_{2,1} = \{x_1 - x_2 \geq -2,\ -x_1 \geq 0,\ -x_2 \geq -1.2,\ x_2 \geq 0,\\ x_2 - y_2 \geq 0,\ -x_2 + y_2 \geq 0\},$$
$$\mathcal{K}'_{2,2} = \{x_1 + x_2 \geq -2,\ -x_1 \geq 0,\ -x_2 \geq 0,\\ -y_2 \geq 0,\ y_2 \geq 0\}.$$

Now their convex hull is computed using the PDDM (explained in detail later), which yields exact results in this case. For the two polytopes $\mathcal{K}'_{2,1}$ and $\mathcal{K}'_{2,2}$, we thus obtain their convex hull (yellow column) as

$$\mathcal{K}_2 = \{x_1 + x_2 - 2y_2 \geq -2, -x_1 \geq 0,\ 0.375x_2 - y_2 \geq -0.75,\\ -x_2 + y_2 \geq 0,\ y_2 \geq 0\}.$$

This already yields tighter bounds than the single neuron approximation (see Figure 4) and completes the first step of lifting. The next and final step of lifting starts with extending $\mathcal{K}_2$, and the analogously computed $\mathcal{K}_1$, into the y_1 dimension, where we apply bounds on y_1 yielding:

$$\mathcal{K}'_1 = \{-x_1 + x_2 - 2y_2 \geq -2, x_1 \geq 0,\ 0.375x_2 - y_2 \geq -0.75,\\ -x_2 + y_2 \geq 0,\ y_2 \geq 0,\ x_1 - y_1 \geq 0,\ -x_1 + y_1 \geq 0\},$$
$$\mathcal{K}'_2 = \{x_1 + x_2 - 2y_2 \geq -2, -x_1 \geq 0,\ 0.375x_2 - y_2 \geq -0.75,\\ -x_2 + y_2 \geq 0,\ y_2 \geq 0,\ -y_1 \geq 0,\ y_1 \geq 0\}.$$

Completing the second step of lifting by computing their convex hull yields the tight 2-neuron constraints

$$\mathcal{K} = \{x_1 + x_2 - 2 \cdot y_1 - 2 \cdot y_2 \geq -2,\ 0.375 \cdot x_2 - y_2 \geq -0.75,\\ -x_1 + y_1 \geq 0,\ -x_2 + y_2 \geq 0,\ y_1 \geq 0,\ y_2 \geq 0\}.$$

Partial Double Description Method (PDDM). We use our novel PDDM to compute a precise, fast and sound over-approximation of convex hulls, scaling at worst only quartic ($O\{n_a^4 \cdot n_v + n_a^2 \log(n_a^2)\}$) in the number of input constraints n_a and linear in the number of vertices n_v (see Appendix B for a proof). Further, for up too 3-dimensional polytopes, the

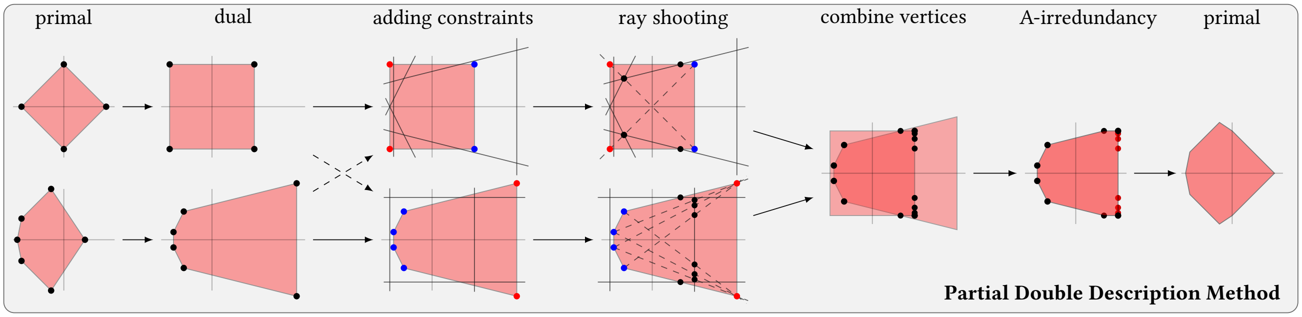


Figure 5. Illustration of the Partial Double Description Method for a 2-dimensional example. The input polytopes (1st column) are translated to their dual representation (2nd column), then all their constraints are added to the other dual polytope (3rd column). Now ray-shooting is used to discover vertices on the rays between points of the old polytope lying inside and outside the new constraints, by intersecting them with these constraints (4th column). These vertices are then combined (5th column), before A-irredundancy is enforced (6th column) and the result is translated back to primal space (7th column).

PDDM computes the exact convex hull. In contrast, the general exact convex hull is exponential ($O(n_v \log(n_v) + n_v^{\lfloor d/2 \rfloor})$ [11]) in the number of dimensions d and superlinear in the number of input vertices. Empirically, we obtain roughly a speed-up of two orders of magnitude for the convex hull computations we consider (Section 7).

The widespread Double Description Method [18, 34] for computing the convex hull of two polyhedra translates them to their dual representation, intersects them by aggregating constraints, and then translates them back to their primal representation. Crucially, every step of adding an additional constraint generates new vertices quadratic in the number of original vertices, leading to an exponential increase.

We introduce the Partial Double Description (PDD), combining an exact $\mathcal{H}$-representation with a soundly under-approximating[3] $\mathcal{V}$-representation and an applicable concept of irredundancy[4] which we call A-irredundancy.

At a high level, the PDDM leverages the PDD, A-irredundancy, and a method called ray-shooting to efficiently add all constraints of one polytope to the other in one step. Importantly, the final number of vertices is then only quadratic in the number of original vertices. Replacing the sequential vertex discovery in the DDM, with a symmetric application of that approach and combining the resulting $\mathcal{V}$-representations, yields a precise and scalable overapproximation of the convex hull in $\mathcal{H}$-representation. We illustrate the Partial Double Description Method in Figure 5 and provide more technical details in Section 5.

Combine constraints. In the last, trivial step, we combine the constraints of the computed output polyhedra for each group to obtain the final convex polyhedral over-approximation of the activation layer.

5 The Partial Double Description Method

In this section, we explain our PDDM for computing convex hull approximations in greater detail. First, we introduce the needed notion of duality and our novel Partial Double Description (PDD) representation for polyhedra. Then, we explain PDDM step by step as illustrated in Figure 5.

PDDM computes the convex hull of two d-dimensional polytopes $\mathcal{P}_1 = \mathcal{P}(\boldsymbol{A}_1, \boldsymbol{b}_1)$ and $\mathcal{P}_2 = \mathcal{P}(\boldsymbol{A}_2, \boldsymbol{b}_2)$, but uses the equivalent homogenized representation (see Section 3) of $(d+1)$-dimensional cones $\mathcal{P}'_1 = \mathcal{P}(\boldsymbol{A}'_1)$ and $\mathcal{P}'_2 = \mathcal{P}(\boldsymbol{A}'_2)$. Vertices in the original polytope now correspond to rays in the cone and in the explanations we will use either term, depending on convenience.

The original polytope can be recovered from the cone, by intersecting it with the hyperplane $x'_0 = 1$, or with $x'_0 = -1$ in dual space (explained next) as visualized in Figure 6.

Duality. The dual $\overline{\mathcal{P}}$ of a polytope $\mathcal{P}$ with a minimal set of extremal vertices $\mathcal{R}$ enclosing the origin but not containing it in its boundary is defined as:

$$\overline{\mathcal{P}} = \{\boldsymbol{y} \in \mathbb{R}^d \mid \boldsymbol{x}^\top \boldsymbol{y} \le 1 \; \forall \boldsymbol{x} \in \mathcal{P}\} \tag{5}$$

$$= \bigcap_{x \in \mathcal{R}} \{\boldsymbol{y} \in \mathbb{R}^d \mid \boldsymbol{x}^\top \boldsymbol{y} \le 1\}. \tag{6}$$

and for polyhedral cones $\mathcal{P}'$ [20]:

$$\overline{\mathcal{P}'} = \{\boldsymbol{y}' \in \mathbb{R}^{d+1} \mid \boldsymbol{x}'^\top \boldsymbol{y}' \le 0 \; \forall \boldsymbol{x}' \in \mathcal{P}'\}. \tag{7}$$

Figure 6 shows an example.

Important for the remaining section are three properties of the transform between primal and dual. (1) It is inclusion reversing: $\mathcal{P} \subset \mathcal{Q}$ if and only if $\overline{\mathcal{Q}} \supset \overline{\mathcal{P}}$. (2) The $\mathcal{V}$-representation of the dual corresponds to the $\mathcal{H}$-representation of the primal and vice versa: $\mathcal{P} = \mathcal{P}(\boldsymbol{A}', \mathcal{R}')$ implies $\overline{\mathcal{P}} = \mathcal{P}(\mathcal{R}'^\top, \boldsymbol{A}'^\top)$, where $(\cdot)^\top$ denotes transpose. (3) The dual of the dual of a polyhedron is the original primal polyhedron $\overline{\overline{\mathcal{P}}} = \mathcal{P}$.

[3] An under-approximation in dual space corresponds to an over-approximation in primal space, due to inclusion reversion.

[4] A generator of a polyhedron is called irredundant if it is extremal. That is if a vertex has rank d or a ray rank $d-1$.

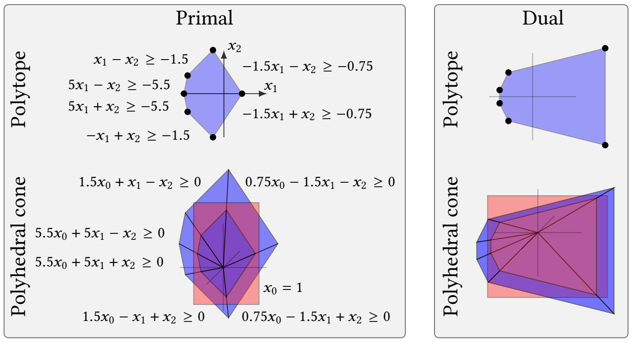

Primal
Dual
Polytope
Polyhedral cone
$x_1 - x_2 \geq -1.5$
$5x_1 - x_2 \geq -5.5$
$5x_1 + x_2 \geq -5.5$
$-x_1 + x_2 \geq -1.5$
$-1.5x_1 - x_2 \geq -0.75$
$-1.5x_1 + x_2 \geq -0.75$
x_1
x_2
$1.5x_0 + x_1 - x_2 \geq 0$
$0.75x_0 - 1.5x_1 - x_2 \geq 0$
$5.5x_0 + 5x_1 - x_2 \geq 0$
$5.5x_0 + 5x_1 + x_2 \geq 0$
$x_0 = 1$
$1.5x_0 - x_1 + x_2 \geq 0$
$0.75x_0 - 1.5x_1 + x_2 \geq 0$

Figure 6. Top row: 2d-polytope in primal (left) and dual (right) space. Bottom row: equivalent polyhedral cones in homogenized coordinates. In red we show the plane the polyhedral cone can be intersected with to recover the polytope.

Partial Double Description. We leverage the duality properties in two ways: we translate the convex hull problem to an intersection problem in dual space and we obtain an $\mathcal{H}$-representation soundly *over-approximating* the convex hull by computing a $\mathcal{V}$-representation soundly *under-approximating* the intersection in dual space. To do so efficiently we introduce the Partial Double Description (PDD) as a relaxation of the Double Description (DD) (Section 3).

Formally, the PDD of a $(d+1)$-dimensional polyhedral cone is the pair $(\boldsymbol{A}', \mathcal{R}')$ with $\boldsymbol{A}' \in \mathbb{R}^{m \times (d+1)}$ and $\mathcal{R}' \in \mathbb{R}^{n \times (d+1)}$ where the $\mathcal{V}$-representation is an under-approximation of the $\mathcal{H}$-representation or more formally, where for any row $\boldsymbol{r} \in \mathcal{R}'$ and $\boldsymbol{a} \in \boldsymbol{A}'$, $\boldsymbol{a} \cdot \boldsymbol{r} \geq 0$ holds.

We call constraints $\boldsymbol{a}'_j \in \boldsymbol{A}'$ *active* for a given ray $\boldsymbol{r}_i \in \mathcal{R}'$, if they are fulfilled with equality, that is $\boldsymbol{a}_j \boldsymbol{r}_i = 0$. We store this relationship as part of the PDD in what we call the incidence matrix $\mathcal{I} \in \{0,1\}^{n \times m}$: $\mathcal{I}_{i,j} = 1$ if $\boldsymbol{a}_j \boldsymbol{r}_i = 0$ and $\mathcal{I}_{i,j} = 0$ else. Further, we define the row-wise inclusion relationship on $\mathcal{I}$: $\mathcal{I}_i \subseteq \mathcal{I}_j$ if $\mathcal{I}_{i,k} \leq \mathcal{I}_{j,k}$, $1 \leq k \leq m$.

Next, we describe PDDM as illustrated in Figure 5.

5.1 Conversion to Dual

Given an input polyhedral cone in PDD representation $(\boldsymbol{A}', \mathcal{R}')$ (1st column in Figure 5), the first step of the PDDM is to convert it trivially to its dual space representation $(\mathcal{R}'^{\top}, \boldsymbol{A}'^{\top})$ [17] (2nd column in Figure 5).

5.2 Intersection

The next step in the PDDM is the intersection in dual space itself (columns 3 to 5 in Figure 5).

The standard approach for the intersection of polyhedra in DD is to sequentially add the constraints of one polytope to the other, computing exact $\mathcal{V}$-representations at every step. This however can increase the number of vertices quadratically in every step resulting in an exponential size of the representation. Instead, we add all constraints jointly in one step by leveraging our PDD. In the following description, we adopt the polytope (not cone) view.

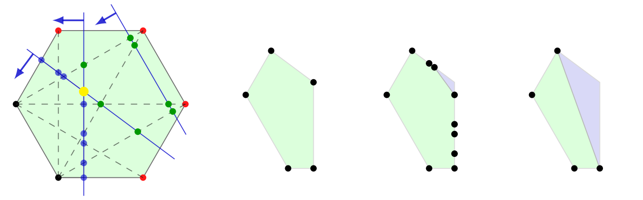

(a) Adding Multiple Constraints (b) Exact Result (c) Discovered Vertices (d) A-Irredundant

Figure 7. Adding a batch of three constraints (blue thick lines) to a polytope in PDD. Vertices are separated into $\mathcal{R}_+$ (black), $\mathcal{R}_0$ (none), and $\mathcal{R}_-$ (red). Ray-shooting discovers new vertices $\mathcal{R}_*$ (blue), avoiding the superfluous green points, but missing an extremal vertex (yellow) (a). Exact intersection (b), result of joint constraint processing (c), and underapproximation after enforcing A-irredundancy (d).

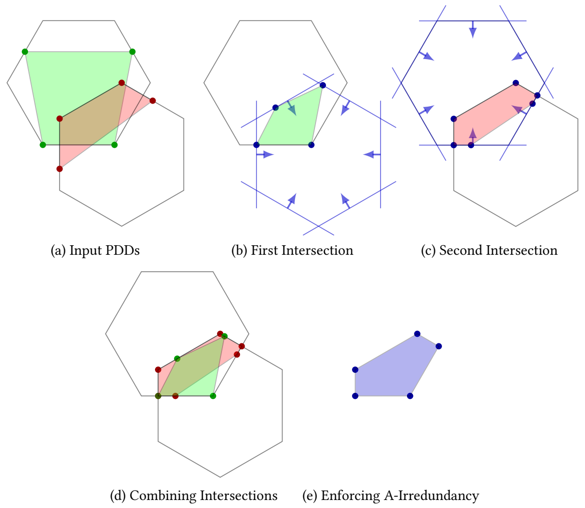
Batch intersection. To intersect a polytope in PDD $(\boldsymbol{A}', \mathcal{R}')$ with a batch of constraints represented by the matrix $\tilde{\boldsymbol{A}}$, we separate the vertices in $\mathcal{R}$ into three sets depending on whether they satisfy all new constraints with inequality ($\mathcal{R}_+$), with equality ($\mathcal{R}_0$), or violate at least one ($\mathcal{R}_-$). An example is shown in Figure 7(a): the three constraints are shown in blue and the vertices as $\mathcal{R}_+$ (black), $\mathcal{R}_0$ (none), and $\mathcal{R}_-$ (red).

Now we employ a technique called ray-shooting [32] to discover the first intersection $\boldsymbol{r}_*$ of a ray $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$ shot from a vertex $\boldsymbol{r}_+ \in \mathcal{R}_+$ "inside" the newly added constraints to a vertex $\boldsymbol{r}_- \in \mathcal{R}_-$ "outside" with a hyperplane $\mathcal{H} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}_i \boldsymbol{x} = 0\}$ corresponding to one of the new constraints $\boldsymbol{a}_i \in \tilde{\boldsymbol{A}}$. Doing so for all combinations of $(\boldsymbol{r}_+, \boldsymbol{r}_-) \in \mathcal{R}_+ \times \mathcal{R}_-$ yields the set of points

$$\mathcal{R}_* = \{\boldsymbol{r}_* = \overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-} \cap \mathcal{H} \mid (\boldsymbol{r}_+, \boldsymbol{r}_-) \in \mathcal{R}_+ \times \mathcal{R}_-\}. \quad (8)$$

The $\mathcal{V}$-representation of the resulting intersection is now the union $\mathcal{R}' \cap \tilde{\boldsymbol{A}} = \mathcal{R}_+ \cup \mathcal{R}_0 \cup \mathcal{R}_*$. In Figure 7 the rays are dashed lines from all black to all red vertices and discover new vertices $\mathcal{R}_*$ (blue). By only using the first intersections, we avoid discovering the green points, however we also do not discover the yellow point, which is an extremal vertex of the true intersection (b), obtaining instead the under-approximation (c).

Boosting precision. Batch intersection is asymmetric: all constraints from one polytope are added to the other to obtain a $\mathcal{V}$-representation of the under-approximation of the intersection. Thus, we perform it in both directions, obtaining two under-approximations. Their convex hull (obtained by the union of vertices) is still an under-approximation of the intersection of the exact $\mathcal{H}$-representations and more precise. This is illustrated in Figure 8, where the exact intersection of the two $\mathcal{H}$-representations (e) is recovered despite the union of the under-approximating input $\mathcal{V}$-representations (a) not covering it. This synergy between PDD and PDDM



(a) Input PDDs

(b) First Intersection

(c) Second Intersection

(d) Combining Intersections

(e) Enforcing A-Irredundancy

Figure 8. Adding individual constraints to the Partial Double Description and applying A-Irredundancy. Input polytope (a), adding the constraint $\boldsymbol{ax} \geq 0$, separating points into $\mathcal{R}_+$ (black), $\mathcal{R}_0$ (none), and $\mathcal{R}_-$ (red) and discovering new points $\mathcal{R}_*$ (blue) (b), and applying A-irredundancy (c).

helps to minimize the precision loss due to using approximations.

In our experimental results this yields a significant boost in precision and for small dimensions $d \leq 4$ of cones even the exact intersections (see Appendix A for the proof).

5.3 Enforcing A-Irredundancy

Despite using batch intersection, the number of vertices can grow quickly when computing multiple convex hulls sequentially. Therefore, some notion of redundancy is needed to efficiently reduce the representation size. The standard definitions of irredundancy are: (1) the set of unique extremal rays of the cone $\mathcal{P}(\boldsymbol{A}')$ are irredundant, and (2) a ray $\boldsymbol{r}_i$ is irredundant if removing it leads to a different cone $\mathcal{P}(\mathcal{R}') \neq \mathcal{P}(\mathcal{R}' \setminus \boldsymbol{r}_i)$. For an exact DD, an irredundant representation can be computed by retaining only the extremal rays with rank $d-1$. However, a PDD $(\boldsymbol{A}', \mathcal{R}')$ usually does not include all or even any extremal rays of the cone $\mathcal{P}(\boldsymbol{A}')$. Consequently, enforcing the first redundancy definition could remove all rays. Enforcing the second irredundancy definition is expensive to compute in the absence of extremal rays.

Therefore, we propose the concept of A-irredundancy, which we define by requiring for all rays $\boldsymbol{r}'_i \in \mathcal{R}'$ that there may not be another generator $\boldsymbol{r}'_j \in \mathcal{R}'$ with a larger (by inclusion) active constraint set:

$$\mathcal{I}_i \not\subseteq \mathcal{I}_j, \quad \text{for all } i, j \in \{1, \ldots, n\}, \; i \neq j.$$

Any ray fulfilling a subset (including the same) constraints with equality as another ray, is removed to obtain an A-irredundant representation. Extremal rays will always be retained as they have the maximum number of active constraints and there are never two with the same active set.

We illustrate the effect of enforcing A-irredundancy in Figure 7 where we use it to obtain the polytope 7(d) and see that the resulting reduction in generator set size can come at the price of precision.

Enforcing A-irredundancy in the 6th column of Figure 5 (removing the red vertices), recovers the minimal set of extremal rays. Translating the resulting PDD back to primal space concludes the PDDM. See C for a proof of the soundness of the PDDM.

6 Split-Bound-Lift Method

In this section, we explain the Split-Bound-Lift Method in greater detail. Recall that we use the SBLM to compute k-neuron abstractions, by approximating the convex hull $\mathrm{conv}(\{(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x})) \mid \boldsymbol{x} \in \mathcal{P} \subseteq [l_x, u_x]^k\})$ for a group of k neurons and their activation functions $\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1), \ldots, f_k(x_k)]^\top$, assuming that their inputs are constraint to the polytope $\mathcal{P}$.

At a high level, we first decompose the input polytope into regions where we can bound all activation functions tightly. Then we extend these regions into the output space and apply linear constraints. Taking the convex hull of the resulting polytopes yields an $\mathcal{H}$-representation encoding the k-neuron abstraction.

To increase the efficiency of this approach, we use a decomposition method we call *splitting* and then recursively extend and bound the resulting polytopes by one output variable at a time, which we call *lifting*, to minimize the dimensionality in which we have to compute convex hulls.

We formalize this approach in Algorithm 1 and explain both splitting and lifting below after stating the prerequisites for the SBLM.

6.1 Prerequisites

We assume an individual activation function $f : \mathbb{R} \to \mathbb{R}$, which can be tightly bounded using the pairs of upper and lower bounds $\mathcal{B}^1$ and $\mathcal{B}^2$ on the closed halfspaces $\mathcal{D}^1$ and $\mathcal{D}^2$ covering $\mathbb{R}$. More formally, we require the intervals

$$\mathcal{D}^i = [c_i, d_i], \quad c_i, d_i \in \overline{\mathbb{R}} \text{ and } c_i \leq d_i$$
$$\mathbb{R} = \mathcal{D}^1 \cup \mathcal{D}^2$$

with the affinely extended real numbers $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ and the bounds on these intervals

$$\mathcal{B}^i = (a_i^-, a_i^+), \quad a_i^{\{+,-\}}(x) = ax + b, \; a, b \in \mathbb{R} \quad s.t.$$
$$a_i^-(x) \leq f(x) \leq a_i^+(x), \quad \forall\, x \in (\mathcal{D}^i \cap [l_x, u_x])_i$$

to be provided. That is, we require a decomposition of the domain $\mathbb{R}$ of the function f into the intervals $\mathcal{D}^i$, and tight linear constraints $\mathcal{B}^i$ on these intervals, assuming the concrete bounds $[\max(l_x, c_i), \min(u_x, d_i)]$ of that interval are available.

Generalization. We focus on the univariate case using only two bounding regions $\mathcal{D}^i$ here. However, the SBLM and by extension PRIMA can be generalized to allow for neuron groups combining different multivariate activation functions $f : \mathbb{D} \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$ which only have to be bounded on their domain $\mathbb{D}$. Further, more than one upper and lower bound $\mathcal{B}^i$ per bounding region as well as an arbitrary number of polyhedral bounding regions $\mathcal{D}^i$ can be specified, as long as their union covers the domain $\mathbb{D} \subseteq \bigcup_i \mathcal{D}^i$ of the individual functions f.

6.2 Splitting the Input Space

To apply the bounds $\mathcal{B}^i$, the input polytope $\mathcal{P}$ has to be decomposed into the regions for which the bounds were specified. These regions correspond to the intersection of $\mathcal{P}$ with the k-cartesian product of the bounding regions $\mathcal{D}^i$.

We choose an ordering of the output variables $\mathcal{I}$ and recursively split $\mathcal{P}$ by intersecting with the bounding regions associated with these output variables.

As every such split is equivalent, we will explain one case assuming the parent polytope Q_1, the output variable $y_i = f(x_i)$, and the corresponding bounding regions $\mathcal{D}_i^1 = \{\boldsymbol{x} \in \mathbb{R}^k \mid x_i \geq c_1\}$ and $\mathcal{D}_i^2 = \{\boldsymbol{x} \in \mathbb{R}^k \mid x_i \leq d_2\}$. We compute the children nodes by intersecting Q_1 with $\mathcal{D}_i^1$ und $\mathcal{D}_i^2$ to obtain $Q_{1,1} = Q_1 \cap \mathcal{D}_i^1$ and $Q_{1,2} = Q_1 \cap \mathcal{D}_i^2$.

Starting with $\mathcal{P}$ at the root and recursively applying this splitting rule to all leaf polytopes for every $y_i \in \mathcal{I}$, generates a polytope tree, which we call the decomposition tree, with 2^k leaf polytopes $\mathcal{P}_{i^k}$, which we call *quadrants*. This is illustrated in the blue portion of the central panel in Figure 3, where $\mathcal{D}^1$ and $\mathcal{D}^2$ are $\mathbb{R}_0^+$ and $\mathbb{R}_0^-$, respectively.

6.3 Bound & Lift

To obtain a k-neuron abstraction in the form of the $\mathcal{H}$-representation of a polytope $\mathcal{K}$, jointly constraining the inputs and outputs of a neuron group, the quadrants thus obtained have to be extended to the output space and bounded, before their convex hull is taken. We call this process *lifting* and propose a recursive approach, lifting sibling polytopes on the decomposition tree until only the desired polytope $\mathcal{K}$ remains.

Again we explain one step of lifting, as they are equivalent. We assume the sibling polytopes $\mathcal{K}_{1,1}$ and $\mathcal{K}_{1,2}$, corresponding to $Q_{1,1}$ and $Q_{1,2}$ in the decomposition tree, with the associated input x_i and output y_i variable, and the pairs of bounds $\mathcal{B}_i^1$ and $\mathcal{B}_i^2$ instantiated for y_i. One step of lifting entails the following:

- Extend $\mathcal{K}_{1,1}$ and $\mathcal{K}_{1,2}$ by the output variable y_i
- Bound y_i on the extended polytopes, by intersecting them with the constraints $\mathcal{B}_i^1$ and $\mathcal{B}_i^2$ to obtain $\mathcal{K}_{1,1}'$ and $\mathcal{K}_{1,2}'$
- Compute their convex hull using the PDDM: $\mathcal{K}_1 = \text{conv}(\mathcal{K}_{1,1}', \mathcal{K}_{1,2}')$

Algorithm 1: Split-Bound-Lift Method (SBLM)

```
Input: Variable ordering $\mathcal{I}$, input polytope $Q$, set of
       bounding regions $\mathcal{D}$ and set of bounds $\mathcal{B}$
Output: Jointly constraining polytope $\mathcal{K}$
if $|\mathcal{I}| > 0$ then
    Get next output variable: $y \leftarrow \mathcal{I}_0$
    foreach $\mathcal{D}_y^i, \mathcal{B}_y^i$ in $\mathcal{D}, \mathcal{B}$ do
        Create split region: $Q_i = Q \cap \mathcal{D}_y^i$
        Apply SBLM: $\mathcal{K}_i \leftarrow \texttt{SBLM}(\mathcal{I}_{1:end}, Q_i, \mathcal{D}, \mathcal{B})$
        Extend into space including $y$: $\mathcal{K}_i \leftarrow \mathcal{K}_i \times \mathbb{R}$
        Apply bounds $\mathcal{B}^i$: $\mathcal{K}_i \leftarrow \mathcal{K}_i \cap \mathcal{B}_y^i$
    end
    Compute convex hull: $\mathcal{K} = \texttt{PDDM}(\{\mathcal{K}_i\}_i)$
    return $\mathcal{K}$
else
    return $\mathcal{P}$
end
```

Applying this lifting rule recursively to the decomposition tree, combines all 2^k quadrants into a single $2k$-dimensional polytope $\mathcal{K}$, jointly constraining the inputs and outputs, thereby concluding the Split-Bound-Lift Method.

This decompositional approach has two benefits: Firstly, computing the convex hull approximately using the PDDM is exact for polytopes of dimension up to 3 and it starts to lose precision as dimensionality increases. Directly computing $2k$-dimensional convex hulls with PDDM will therefore lose more precision than using our decomposed method. Secondly, a lower-dimensional polytope with fewer constraints and generally also fewer vertices reduces the time required for the individual convex hull operations significantly. In fact, if the approximated activation function is not piecewise linear, computing the convex hull of all extended and bounded quadrants directly is intractable for groups as small as $k = 3$, as the number of vertices increases exponentially with k during bounding in that case.

6.4 Instantiation for various functions

In the following paragraphs, we describe instantiations of the SBLM for the ReLU, Tanh, Sigmoid and MaxPool functions.

ReLU. All piecewise linear functions, such as ReLU, can be approximated exactly on the intervals $\mathcal{D}^i$ if chosen as their linear regions. Further, if the neuronwise bounds $[l_x, u_x]$ do not include a point of slope change, the neuron behaves linearly, can be encoded exactly and is excluded from the k-neuron abstraction. Therefore, we can assume $y = max(x, 0)$ with $x \in [l_x, u_x]$ for $l_x < 0 < u_x$. We chose to split into the linear regions $\mathcal{D}^1 = \{x \leq 0\} = [-\infty, 0]$ and $\mathcal{D}^2 = \{y \geq 0\} = [0, \infty]$, where we have the bounds $\mathcal{B}^1 = (y \geq 0,\ y \leq 0)$ and $\mathcal{B}^2 = (y \geq x,\ y \leq x)$ and typically use $k = 3$.

Tanh and Sigmoid. Let f be an S-curve function with domain $[l_x, u_x]$, that is $f''(x) \geq 0$ for $x \leq 0$, $f''(x) \leq 0$

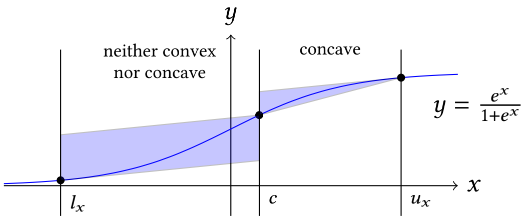


Figure 9. Intervalwise bounds for the Sigmoid function on the intervals $[l_x, c]$ and $[c, u_x]$.

for $x \geq 0$ and $f'(x) > 0$ for $x \in [l_x, u_x]$. Both the Sigmoid function $\sigma(x) = \frac{e^x}{e^x+1}$ and Tanh function $\tanh(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}}$ fulfill these requirements. We split the domain at $c \in [l_x, u_x]$ into $\mathcal{D}^1 = [l_x, c]$ and $\mathcal{D}^2 = [c, u_x]$ to minimize the total area between upper and lower bound in the input-output plane, using the bounds from Singh et al. [45]

$$x \leq f(b) + (x-b)\begin{cases} \frac{f(b)-f(a)}{b-a} & \text{if} \quad b \leq 0 \\ \min(f'(b), f'(a)) & \text{else} \end{cases}$$

$$x \geq f(a) + (x-a)\begin{cases} \frac{f(b)-f(a)}{b-a} & \text{if} \quad a \geq 0 \\ \min(f'(b), f'(a)) & \text{else} \end{cases}$$

denoting the lower bound of an interval as a and the upper one as b. We illustrate these bounds in Figure 9 for the Sigmoid function. Again we choose $k = 3$ in our experiments.

MaxPool. Let MaxPool be the multivariate function $y = \max(x_1, x_2, ..., x_d)$ on the domain $\boldsymbol{x} \in \mathcal{P} \subseteq [l_x, u_x]^d$. Note, that here the generalized formulation is required. We chose the splits $\mathcal{D}^i = \{\boldsymbol{x} \in \mathbb{R}^d | x_i \geq x_j, \ 1 \leq j \leq d, \ i \neq j\}_i$, separating the domain into the d regions where one variable dominates all others. On each of these regions MaxPool can be bounded exactly with $y \leq x_i$ and $y \geq x_i$. During the splitting process, this increased number of bounding regions leads to a decomposition tree where every parent node has d child nodes.

7 Experimental Evaluation

In this section we evaluate the effectiveness of PRIMA and show that it significantly improves over the results of state-of-the-art verifiers on a range of challenging benchmarks yielding up to 14%, 30% and 34% precision gains on ReLU-, Sigmoid-, and Tanh-based networks, respectively. Additionally, we demonstrate the effectiveness and benefits of computing relaxations with the SBLM compared to using the direct, exact convex hull approach.

7.1 Setup

All neural network certification benchmarks were run on a 20 Core 2.20GHz Intel Xeon Silver 4114 CPU with 100 GB of main memory. We use Gurobi 9.0 for solving MILP and LP problems [22].

Table 1. Neural network architectures used in experiments.

Dataset	Model	Type	Neurons	Layers	Activation
MNIST	5 × 100[5]	FC	510	5	ReLU
	6 × 100	FC	600	6	Tanh/Sigm
	8 × 100[5]	FC	810	8	ReLU
	9 × 100	FC	900	9	Tanh/Sigm
	5 × 200[5]	FC	1 010	5	ReLU
	6 × 200	FC	1 200	6	Tanh/Sigm
	8 × 200[5]	FC	1 610	8	ReLU
	`ConvSmall`	Conv	3 604	3	ReLU/Tanh/Sigm
	`ConvBig`	Conv	34 688	6	ReLU
CIFAR10	`ConvSmall`	Conv	4 852	3	ReLU
	`ConvBig`	Conv	62,464	6	ReLU
	`ResNet`	Residual	107,496	13	ReLU

7.2 Benchmarks

We use the same set of fully-connected and convolutional network architectures[5] on MNIST and CIFAR10 as Singh et al. [43], described in Table 1. The MNIST `ConvBig` and all three CIFAR10 networks are trained to be robust against adversarial examples using DiffAI [33], Wong [54] and PGD [31] training, respectively. All other networks are normally trained. We evaluate our method's performance for verifying the challenging L_∞ perturbations on the first 1000 images of the MNIST and CIFAR10 test sets. We note that PRIMA can also be used for verifying other specifications such as individual fairness [37], global safety properties [25], acoustic [38], geometric [4], and spatial [37] based perturbations.

We note that complete verification methods do not scale to the combinations of admissible perturbation size and networks we consider (with the exception of the `ResNet`). Therefore, for ReLU we compare PRIMA with a range of state-of-the-art incomplete verifiers [7, 43–46, 51, 52, 57] notably also the ReLU-specialized `kPoly` [43]. For Tanh and Sigmoid activations, fewer verifiers are available and thus we compare with the state-of-the-art incomplete verifier DeepPoly [45].

7.3 ReLU activation

For our experiments, we use a setup similar to `kPoly` in [43]. During verification, we use DeepPoly to determine the octahedral inputs required to compute approximations with our framework. All constraints produced by the SBLM are added to the LP encoding of the network. After all layers are processed, an LP solver with a 100 second timeout is used to prove the property.

For fully-connected networks, we further refine neuron bounds on each layer as described by Singh et al. [46]. The key idea is to tighten the neuronwise lower and upper bounds by formulating and solving an LP. For neurons in the second

[5] The networks referred to as 6×·00 and 9×·00 in previous work only include 5 and 8 hidden layers, respectively, and have therefore been renamed.

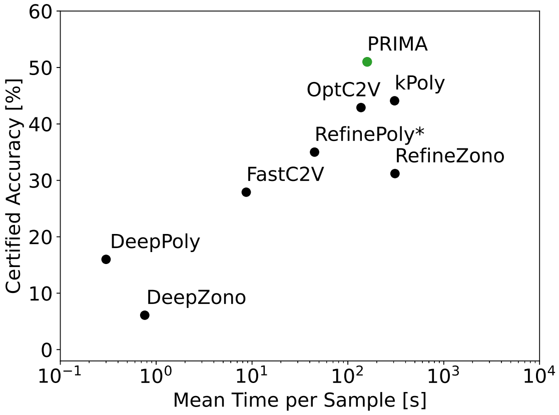


Figure 10. Comparison of PRIMA precision and runtime with other certifiers evaluated on the first 1000 images (100 for RefinePoly) on the MNIST 5 × 100 network. `CROWN` uses the same abstractions as DeepPoly and is therefore omitted to avoid cluttering.

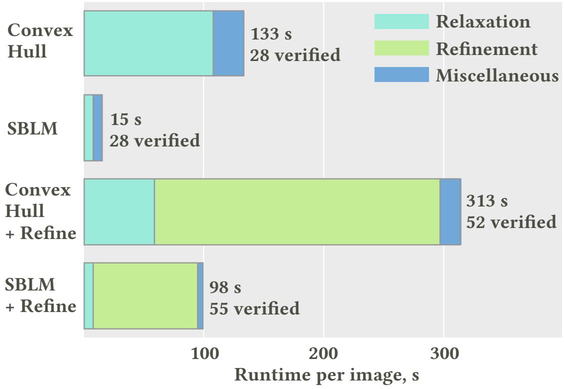


Figure 11. Runtime comparison of using SBLM vs. exact convex hull for computing relaxations in PRIMA. Evaluated on 100 images and the MNIST 5 × 100 ReLU network.

ReLU layer, a MILP encoding from Tjeng et al. [49] is used to tighten the bounds further. We note that encoding more layers with MILP does not scale on these networks. For convolutional networks we encode the fully connected layers (1 in the case of `ConvSmall` and 2 in the case of `ConvBig`) using a MILP encoding from Tjeng et al. [49] and add those constraints to the LP encoding the network and use GPUPoly [35].

SBLM is significantly more scalable to bigger groups of neurons than the naive convex hull approach. Already for $k = 4$ ReLU neurons, the standard convex hull computation takes several minutes for a single group, whereas SBLM computes approximations in under 50 milliseconds. Nevertheless, we discovered that the best strategy to leverage this speedup is to evaluate a large number of small groups. In all our experiments, we consider groups of size $k = 3$ and vary the size of sparse groupings n_s, as shown in Table 2.

Comparison with state-of-the-art. Figure 10 shows a scatter plot comparing the runtime and performance of PRIMA with other state-of-the-art verifiers on the robustness certification of a 5 × 100 normally trained ReLU network. We conduct the same comparison on the remaining networks in figure 13 in appendix D and observe similar results. We note that adversarially trained networks sacrifice accuracy for ease of certification, making normally trained networks both more relevant and more challenging. Existing complete verifiers can not certify individual samples in as much as 15 minutes while fast incomplete verifiers like DeepPoly verify only ≈ 20% of the images. In contrast, PRIMA verifies 51% in less than 160 seconds per image. The closest verifiers in terms of precision are `kPoly` and `OptC2V` which verify 44% and 43% of samples and take around 140 and 310 seconds, respectively. Based on these observations, we compare PRIMA with `kPoly` and `OptC2V` on the remaining benchmarks.

Comparison with `kPoly` and `OptC2V`. For all normally trained networks, PRIMA is significantly more accurate than both `kPoly` and `OptC2V`, verifying between 44 and 162 more regions than the better of the two, and sometimes significantly faster. These results are summarized in Table 2. For the DiffAI trained `ConvBig` MNIST network, we verify 4 more regions than `OptC2V`. We note again that these networks are comparatively easier to verify and therefore we gain less precision. However, the relatively easier proofs come at the cost of reduced accuracy, making them less relevant for real-world applications. For example in Table 2, the accuracy of DiffAI trained `ConvBig` is the lowest among all MNIST networks. For both PGD trained CIFAR10 networks, PRIMA verifies between 10 and 42 more regions than `kPoly` and `OptC2V` while being around 4-times faster. On the provably trained `ResNet`, PRIMA is faster and verifies a marginally more precise than `kPoly`, however this network is so heavily regularized that even complete verification is tractable. Summarized, PRIMA is usually faster than `kPoly` [43] and `OptC2V` [48], especially on larger networks, while always being more precise, sometimes substantially so.

Interestingly, the tight approximation of network outputs computed by PRIMA also enables an efficient discovery of adversarial examples. If the LP solver fails to prove the property, it returns the sample leading to the worst-case bounds. If this point classifies to an incorrect class, the region is proven unsafe. For example, the adversarially trained `ConvSmall` CIFAR10 network can be proven to be safe for 441 images and unsafe for 148, leaving only 41 regions where the original sample is classified correctly, but robustness is unknown. We note that this functionality is not supported by `kPoly`.

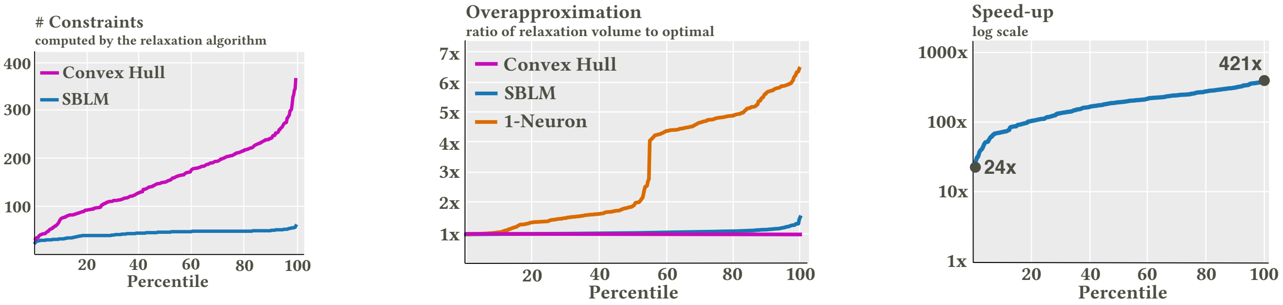
7.4 Effectiveness of SBLM approximation

Computing approximations with SBLM has two main advantages compared to the direct convex hull approach: It

Table 2. Number of verified adversarial regions of the first 1000 and runtime for PRIMA, `OptC2V` and `kPoly`. Depending on the network, natural (NOR), adversarial (PGD [31]), or provable (DiffAI [33], Wong [54]) training was used.

Dataset	Model	Training	Accuracy	ϵ	n_s	`kPoly` [43]		`OptC2V` [48]		PRIMA (ours)		# Upper Bound
						# Verified	Time	# Verified	Time	# Verified	Time	
MNIST	5 × 100	NOR	960	0.026	50	441	307	429	137	510	159	842
	8 × 100	NOR	947	0.026	50	369	171	384	759	428	301	820
	5 × 200	NOR	972	0.015	50	574	187	601	403	690	224	901
	8 × 200	NOR	950	0.015	50	506	464	528	3451	624	394	911
	`ConvSmall`	NOR	980	0.120	100	347	477	436	55	598	42	746
	`ConvBig`	DiffAI	929	0.300	100	736	40	771	102	775	15	804
CIFAR10	`ConvSmall`	PGD	630	2/255	100	399	86	398	105	441	20	482
	`ConvBig`	PGD	631	2/255	50	459	346	-	-	469	97	613
	`ResNet`	Wong	290	8/255	20	245	91	-	-	249	64	290

- `OptC2V` [48] is omitted as the code has not been released and no results were reported on these networks.



(a) Number of constraints for individual k-neuron abstractions.

(b) Volume of the constraint polytope in comparison to the exact convex hull.

(c) Speedup realized by using SBLM compared to exact convex hull.

Figure 12. Case study: Analysis of the distribution of the number of discovered constraints, abstraction volume and runtime over all (≈ 400) individual 3-neuron groups processed during verification of a single MNIST image on 5 × 100 ReLU network.

is significantly faster and produces fewer constraints, making the resulting LP easier to solve, while barely losing any precision.

For example, the runtime analysis in Figure 11 shows that using SBLM instead of the exact convex hull results in the same number of verified images while giving an eight-fold speed-up. If we additionally perform time-intensive neuron refinement, then using SBLM reduces the runtime by about 70% while verifying 3% more images. This accuracy improvement is due to the LP solver obtaining tighter neuron-wise bounds and terminating quicker when using a larger number of the more diverse SBLM constraints. It also illustrates that, using SBLM, we can use neuron refinement and still achieve faster runtimes than using the standard, exact convex hull computation without refinement.

Even the group individual constraints are only marginally less precise then when using exact convex hull computations. For example, verifying the 5 × 100 network for the first image with PRIMA and comparing the 3-ReLU relaxations computed with the exact convex hull and SBLM to the trivial 1-ReLU approximations (Figure 12), we observe the following: SBLM is on average 200 times faster than the exact convex hull, while being only marginally less precise. That is, the volume of the constraint polytopes in the 6-dimensional input-output space of individual neuron groups is only a few percent larger using SBLM. In contrast, both convex hull and SBLM yield on average three times smaller volumes than 1-ReLU approximations.

This combination of a much faster computation and a four-fold reduction in returned constraints allows us to consider many more neuron groups and consequently discover more diverse constraints when using SBLM. This enables the LP solver to compute tighter bounds faster, despite only using approximate constraints.

Table 3. Number of verified adversarial regions and runtime of PRIMA vs. DeepPoly for Tanh/Sigmoid on 100 images from the MNIST dataset.

Act.	Model	Acc.	ϵ	DeepPoly		PRIMA	
				Ver.	Time	Ver.	Time
Tanh	6×100	97	0.006	38	0.3	61	72.5
	9×100	98	0.006	18	0.4	52	183.0
	6×200	98	0.002	39	0.6	68	170.0
	ConvSm	99	0.005	16	0.4	30	27.8
Sigm	6×100	99	0.015	30	0.3	53	96.9
	9×100	99	0.015	38	0.5	56	336.4
	6×200	99	0.012	43	1.0	73	267.0
	ConvSm	99	0.014	30	0.5	51	50.0

7.5 Tanh and Sigmoid activations

While using the exact convex hull algorithm for ReLU relaxations is merely slow, it becomes infeasible for non-piecewise-linear activations such as Tanh and Sigmoid. Computing the constraints for a single group of $k = 3$ neurons can take minutes, whereas SBLM takes 10 milliseconds. This dramatic speedup is a result of the SBLM's decompositional approach, solving the problem in lower dimensions (see Section 6), significantly reducing its complexity. Note that both methods compute only approximations for these cases, as the underlying intervalwise bounds are not exact.

We evaluate our method on the MNIST dataset for normally trained, fully-connected and convolutional networks. We choose an ϵ for the B_ϵ^∞ region such that the current state-of-the-art verifier for Tanh and Sigmoid activations, DeepPoly, verifies less than 50% of adversarial regions. We remark that DeepPoly is based on the same principles and has similar precision as other state-of-the-art verifiers for these activations such as `CNN-Cert` [7] and `CROWN` [57].

We use sparse groups of size $n_s = 10$, compute relaxations jointly for $k = 3$ neurons, and again refine neuronwise lower and upper bounds for fully-connected networks. We verify between 14% and 34% more regions than the current state-of-the-art, in some cases doubling the number of verified samples, while maintaining a reasonable runtime comparable to that for ReLU networks (Table 3).

8 Related Work

The importance of certifying the robustness of NNs to input perturbations has created a surge of research activity in recent years. The approaches that deliver deterministic guarantees can be divided into exact and incomplete methods. Incomplete methods are much faster and more scalable than exact methods, but at the price of reduced precision, i.e., they may fail to certify a property even if it holds.

Complete methods are mostly based on satisfiability modulo theory (SMT) [16, 23, 25, 26] or the branch-and-bound approach [1, 8, 9, 29, 49], often implemented using mixed integer linear programming (MILP). These methods offer exactness guarantees but are based on solving NP-hard optimization problems, which makes them intractable even for small networks. Incomplete methods can be divided into propagation based approaches, computing bounds by substituting variables or propagating an abstract element [21, 33, 35, 44, 45, 52, 57] and those that generate polynomially-solvable optimization problems [10, 30, 36, 43, 48, 55] such as linear programming (LP) or semidefinite programming (SDP) optimization problems. Randomized smoothing [13, 28, 39] can provide probabilistic bounds but not a certificate, incurs significant runtime costs at inference and generalization to arbitrary safety properties is still an open problem.

A new avenue towards more precision are methods [43, 48] breaking the so-called convex barrier [40] by considering activation functions jointly instead of separately. However, their scalability is limited by the need to solve NP-hard convex hull problems. There are many approaches for solving the convex hull problem for polyhedra exactly [2, 3, 5, 14, 15, 18, 24, 34], in contrast to the few approximate methods either sacrificing soundness [6, 27, 41, 58] or still exhibiting exponential complexity [56]. Ray-shooting was used by Maréchal and Périn [32] to find a subset of irredundant constraints to speed up an irredundancy computation in the constraint representation.

Our work follows the line of convex barrier-breaking methods, generalizing the concept to arbitrary bounded, multivariate activation functions. In contrast to prior work, we decompose the underlying convex hull problem into lower-dimensional spaces where we solve it approximately, combining the Double Description Method [34] with a novel relaxed Double Description and irredundancy formulation, and a novel ray-shooting-based approach to add multiple constraints jointly. The resulting speed-ups make PRIMA tractable for non-piecewise-linear activation functions, a first for convex barrier braking methods.

9 Conclusion

We presented PRIMA, a general framework that substantially advances the state-of-the-art in precise neural network verification by providing efficient multi-neuron abstractions for arbitrary, bounded, multivariate non-linear activation functions. Our key idea is to decompose the bottleneck convex hull computation into lower-dimensional spaces, solve it approximately, and leverage the resulting speedup to evaluate more neuron groups, discovering more diverse constraints. Our evaluation shows significant improvements both in precision and speed over prior state-of-the-art for ReLU-, Sigmoid- and Tanh-based networks.

References

[1] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. 2020. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming* (2020), 1–37.

[2] David Avis and Komei Fukuda. 1991. A basis enumeration algorithm for linear systems with geometric applications. *Applied Mathematics Letters* 4, 5 (1991), 39–42.

[3] David Avis and Komei Fukuda. 1992. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry* 8, 3 (1992), 295–313.

[4] Mislav Balunović, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. 2019. Certifying geometric robustness of neural networks. *Advances in Neural Information Processing Systems 32* (2019).

[5] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. 1993. *The quickhull algorithm for convex hull.* Technical Report. Technical Report GCG53, The Geometry Center, MN.

[6] Jon Louis Bentley, Franco P Preparata, and Mark G Faust. 1982. Approximation algorithms for convex hulls. *Commun. ACM* 25, 1 (1982), 64–68.

[7] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. 2019. CNN-Cert: An Efficient Framework for Certifying Robustness of Convolutional Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI).*

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

[9] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Pushmeet Kohli, P Torr, and P Mudigonda. 2020. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research* 21, 2020 (2020).

[10] Rudy R Bunel, Oliver Hinder, Srinadh Bhojanapalli, and Krishnamurthy Dvijotham. 2020. An efficient nonconvex reformulation of stagewise convex optimization problems. *Advances in Neural Information Processing Systems* 33 (2020).

[11] Bernard Chazelle. 1993. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry* 10, 4 (1993), 377–409.

[12] Robert Clarisó and Jordi Cortadella. 2007. The octahedron abstract domain. *Science of Computer Programming* 64, 1 (2007), 115–139.

[13] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. 2019. Certified Adversarial Robustness via Randomized Smoothing. In *Proceedings of the 36th International Conference on Machine Learning.*

[14] George Bernard Dantzig. 1998. *Linear programming and extensions.* Vol. 48. Princeton university press.

[15] Herbert Edelsbrunner. 2012. *Algorithms in combinatorial geometry.* Vol. 10. Springer Science & Business Media.

[16] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis.* Springer, 269–286.

[17] Komei Fukuda. 2020-07-10. Polyhedral Computation. https://doi.org/10.3929/ethz-b-000426218

[18] Komei Fukuda and Alain Prodon. 1995. Double description method revisited. In *Franco-Japanese and Franco-Chinese Conference on Combinatorics and Computer Science.* Springer, 91–111.

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

[20] Blagoy Genov. 2015. The convex hull problem in practice: improving the running time of the double description method. (2015).

[21] Sven Gowal, Krishnamurthy Dj Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. 2019. Scalable verified training for provably robust image classification. In *Proceedings of the IEEE International Conference on Computer Vision.* 4842–4851.

[22] Gurobi Optimization, LLC. 2018. Gurobi Optimizer Reference Manual. http://www.gurobi.com

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

[24] Michael Joswig. 2003. Beneath-and-beyond revisited. In *Algebra, Geometry and Software Systems.* Springer, 1–21.

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

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

[27] Hamid R Khosravani, António E Ruano, and Pedro M Ferreira. 2013. A simple algorithm for convex hull determination in high dimensions. In *2013 IEEE 8th International Symposium on Intelligent Signal Processing.* IEEE, 109–114.

[28] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2018. Certified Robustness to Adversarial Examples with Differential Privacy. *2019 IEEE Symposium on Security and Privacy (S&P)* (2018).

[29] Jingyue Lu and M. Pawan Kumar. 2020. Neural Network Branching for Neural Network Verification. In *International Conference on Learning Representations.* https://openreview.net/forum?id=B1evfa4tPB

[30] Zhaoyang Lyu, Ching-Yun Ko, Zhifeng Kong, Ngai Wong, Dahua Lin, and Luca Daniel. 2019. Fastened crown: Tightened neural network robustness certificates. *arXiv preprint arXiv:1912.00574* (2019).

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[32] Alexandre Maréchal and Michaël Périn. 2017. Efficient elimination of redundancies in polyhedra using raytracing.

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

[34] Theodore S Motzkin, Howard Raiffa, Gerald L Thompson, and Robert M Thrall. 1953. The double description method. *Contributions to the Theory of Games* 2, 28 (1953), 51–73.

[35] Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2020. Neural Network Robustness Verification on GPUs. *arXiv preprint arXiv:2007.10868* (2020).

[36] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. 2018. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems.* 10877–10887.

[37] Anian Ruoss, Mislav Balunović, Marc Fischer, and Martin Vechev. 2020. Learning certified individually fair representations. *arXiv preprint arXiv:2002.10312* (2020).

[38] Wonryong Ryou, Jiayu Chen, Mislav Balunovic, Gagandeep Singh, Andrei Dan, and Martin Vechev. 2020. Fast and effective robustness certification for recurrent neural networks. *arXiv preprint arXiv:2005.13300* (2020).

[39] Hadi Salman, Greg Yang, Jerry Li, Pengchuan Zhang, Huan Zhang, Ilya Razenshteyn, and Sebastien Bubeck. 2019. Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers. *arXiv preprint arXiv:1906.04584* (2019).

[40] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. 2019. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems.* 9835–9846.

[41] Hossein Sartipizadeh and Tyrone L Vincent. 2016. Computing the approximate convex hull in high dimensions. *arXiv preprint arXiv:1603.04422* (2016).

[42] Raimund Seidel. 1995. The upper bound theorem for polytopes: an easy proof of its asymptotic version. *Computational Geometry* 5, 2 (1995), 115–116.

[43] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. 2019. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems*. 15098–15109.

[44] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. *Advances in Neural Information Processing Systems* 31 (2018), 10802–10813.

[45] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–30.

[46] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. Boosting Robustness Certification of Neural Networks. In *International Conference on Learning Representations*.

[47] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proc. International Conference on Learning Representations (ICLR)*.

[48] Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. 2020. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *arXiv preprint arXiv:2006.14076* (2020).

[49] Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2017. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356* (2017).

[50] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020. Verification of Deep Convolutional Neural Networks Using ImageStars. In *Proc. Computer Aided Verification (CAV)*, Shuvendu K. Lahiri and Chao Wang (Eds.). 18–42.

[51] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*. 6367–6377.

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

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

[54] Eric Wong, Frank R Schmidt, Jan Hendrik Metzen, and J Zico Kolter. 2018. Scaling provable adversarial defenses. *arXiv preprint arXiv:1805.12514* (2018).

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

[56] Zong-Ben Xu, Jiang-She Zhang, and Yiu-Wing Leung. 1998. An approximate algorithm for computing multidimensional convex hulls. *Applied mathematics and computation* 94, 2-3 (1998), 193–226.

[57] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Advances in neural information processing systems*.

[58] Jinhong Zhong, Ke Tang, and A Kai Qin. 2014. Finding convex hull vertices in metric space. In *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1587–1592.

A Partial Double Description Method Proofs

To proof the exactness guarantee given for the intersection of two polytopes using our batch intersection and precision boosting approach, described in Section 5.2, we first have to proof the following results on batch intersection (Algorithm 2) and precision boosting (Algorithm 3):

Algorithm 2: Batch Intersection

```
Result: Intersected cone $(\boldsymbol{A} \cup \tilde{\boldsymbol{A}}, \tilde{\mathcal{R}})$
Input: Cone $(\boldsymbol{A}, \mathcal{R})$, constraint matrix $\tilde{\boldsymbol{A}}$
Initialize $\mathcal{R}_{-}, \mathcal{R}_0, \mathcal{R}_{+}, \mathcal{R}_{*} = \emptyset, \emptyset, \emptyset, \emptyset$
for $r$ in $\mathcal{R}$ do
    if $min(\tilde{\boldsymbol{A}}\boldsymbol{r}) < 0$ then
        Add $r$ to $\mathcal{R}_{-}$
    else if $min(\tilde{\boldsymbol{A}}\boldsymbol{r}) > 0$ then
        Add $r$ to $\mathcal{R}_{+}$
    else
        Add $r$ to $\mathcal{R}_0$
end
for $r_{+}$ in $\mathcal{R}_{+}$ do
    for $r_{-}$ in $\mathcal{R}_{-}$ do
        Compute $r_{*}$ using ray-shooting from $r_{+}$ to $r_{-}$
        Add $r_{*}$ to $\mathcal{R}_{*}$
    end
end
Construct new PDD $(\boldsymbol{A} \cup \tilde{\boldsymbol{A}}, \mathcal{R}_0 \cup \mathcal{R}_{+} \cup \mathcal{R}_{*})$
Make PDD A-irredundant
return PDD
```

Batch intersection following Algorithm 2 of a cone in DD with a matrix of constraints yields the following guarantee for the resulting PDD:

Theorem A.1. *Given a Double Description $(\boldsymbol{A}, \mathcal{R})$ of a polyhedral cone and the constraint matrix $\boldsymbol{A'}$, adding all constraints jointly as per Algorithm 2 is guaranteed to yield a double description $(\boldsymbol{A} \cup \boldsymbol{A'}, \mathcal{R}')$ enumerating* all *extremal rays r' of the $\boldsymbol{A} \cup \boldsymbol{A'}$-induced cone with one of the following properties:*

1. *r' is extremal in the $\boldsymbol{A}$-induced cone.*
2. *r' is of rank $d - 2$ in the $\boldsymbol{A}$-induced cone.*

Proof. We can formally divide the rays of the new PDD $\mathcal{R}'$ into the two non-overlapping sets:

- $\mathcal{R}_{+} \cup \mathcal{R}_0$ – Rays in $\mathcal{R}$ not violating any constraint $a \in \boldsymbol{A'}$
- $\mathcal{R}_{*}$ – Rays discovered by ray-shooting

Since $(\boldsymbol{A}, \mathcal{R})$ is a DD of the $\boldsymbol{A}$-induced cone it enumerates all extremal rays. If r' is extremal in both the $\boldsymbol{A}$-induced and the $\boldsymbol{A} \cup \boldsymbol{A'}$-induced cones it is included in $\mathcal{R}$ and does not violate any constraints. Therefore, it is included in the first group above and will be part of $\mathcal{R}'$, which concludes the proof of the first point. Any ray of rank $d - 2$ can by definition be represented as a positive combination of two extremal rays, that is rays of rank $d - 1$. As we assume ray r' to be extremal in the $\boldsymbol{A} \cup \boldsymbol{A'}$-induced cone and therefore have rank $d - 1$ it necessarily intersects at least one constraint $\boldsymbol{a} \in \boldsymbol{A'}$ and is extremal to the $\boldsymbol{A} \cup \boldsymbol{a}$-induced cone. Consequently exactly one of the extremal rays used to construct it has to lie on either side of thy hyperplane induced by constraint $\dashv$. Therefore, they will be included in the sets $\mathcal{R}_{+}$ and $\mathcal{R}_{-}$ and the intersection will be discovered as part of the ray-shooting, concluding the proof of the second point. □

Algorithm 3: PDD Intersection

```
Result: Intersected cone $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2, \mathcal{R}')$
Input: Cones $(\boldsymbol{A}_1, \mathcal{R}_1)$ and $(\boldsymbol{A}_2, \mathcal{R}_2)$
Compute $(\boldsymbol{A}', \mathcal{R}_1') = (\boldsymbol{A}_1 \cup \boldsymbol{A}_2, \mathcal{R}_1)$ with Algorithm 2
Compute $(\boldsymbol{A}', \mathcal{R}_2') = (\boldsymbol{A}_2 \cup \boldsymbol{A}_1, \mathcal{R}_2)$ with Algorithm 2
Construct new PDD $(\boldsymbol{A}', \mathcal{R}_1' \cup \mathcal{R}_2')$
Make PDD A-irredundant
return PDD
```

Using this is result, we can proof the following guarantee for intersections of two cones in DD using our batch intersection and precision boosting approach, described in Section 5.2 and Algorithm 3:

Theorem A.2. *Given the Double Description $(\boldsymbol{A}_1, \mathcal{R}_1)$ and $(\boldsymbol{A}_2, \mathcal{R}_2)$ of two polyhedral cones adding all their constraints jointly to the other as per Algorithm 2 and then taking the convex hull of the two resulting polytopes is guaranteed to yield a double description $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2, \mathcal{R}')$ enumerating* all *extremal rays r' of the $\boldsymbol{A}_1 \cup \boldsymbol{A}_2$-induced cone with one of the following properties:*

1. *r' is extremal in the $\boldsymbol{A}_1$-induced cone.*
2. *r' is extremal in the $\boldsymbol{A}_2$-induced cone.*
3. *r' is of rank $d - 2$ in the $\boldsymbol{A}_1$-induced cone.*
4. *r' is of rank $d - 2$ in the $\boldsymbol{A}_2$-induced cone.*

Proof. s The proof follows directly from applying Lemma A.1 both ways and the insight that every extremal ray discovered by either will be included in the final generating set $\mathcal{R}'$. □

Using these results, we can in turn proof the following exactness guarantee for the intersection of two polyhedral cones of up to dimension 4 in DD using the approach described in Section 5.2:

Theorem A.3. *Given the Double Description $(\boldsymbol{A}_1, \mathcal{R}_1)$ and $(\boldsymbol{A}_2, \mathcal{R}_2)$ of two polyhedral cones P_1 and P_2 of dimension $d \leq 4$, the PDD of their intersection $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2, \mathcal{R}')$ as computed with Algorithm 3 is in fact a DD with an irredundant generating set $\mathcal{R}'$.*

Proof. For briefness sake, we will only show the proof for $d = 4$ here. Let $\mathcal{R}^*$ be the set of extremal rays of the $\boldsymbol{A}_1 \cup \boldsymbol{A}_2$-induced polyhedral cone. Consequently $\boldsymbol{r}^* \in \mathcal{R}^*$ has the rank $d - 1 = 3$ in this cone and therefore it fulfills 3 linearly independent constraints in $\boldsymbol{A}_1 \cup \boldsymbol{A}_2$ with equality. This leads to the following four exhaustive options:

1. All 3 constraints are part of $\boldsymbol{A}_1$, $\boldsymbol{r}^*$ is extremal in P_1
2. All 3 constraints are part of $\boldsymbol{A}_2$, $\boldsymbol{r}^*$ is extremal in P_2
3. 2 constraints are part of $\boldsymbol{A}_1$ and one of $\boldsymbol{A}_2$, $\boldsymbol{r}^*$ is of rank $d - 2 = 2$ in P_1
4. 2 constraints are part of $\boldsymbol{A}_2$ and one of $\boldsymbol{A}_1$, $\boldsymbol{r}^*$ is of rank $d - 2 = 2$ in P_2

As all of those are enumerated by Algorithm A.2, $\mathcal{R}'$ will include all extremal rays of the $\boldsymbol{A}_1 \cup \boldsymbol{A}_2$-induced cone. In this case A-irredundancy is equivalent with irredundancy, concluding the proof. □

B PDDM Complexity

In this section, we analyse the complexity of our approximate Partial Double Description Method for the computation of an over-approximation to the convex hull:

Theorem B.1. *Given the DD of two d-dimensional, bounded polytopes generated by at most n_v vertices or equivalently induced by at most n_a constraint, computing a sound over-approximation of their convex hull has a worst-case time complexity of:*

$$O(n_v \cdot n_a^4 + n_a^2 \log(n_a^2)). \tag{9}$$

Proof. The PDDM can be broken down in its six components illustrated in Figure 5:

1. Conversion from primal to dual representation (Section 5.1)
2. Adding the constraints of one polytope to the other, or more concretely separation of vertices into the three sets $\mathcal{R}_+$, $\mathcal{R}_0$, and $\mathcal{R}_-$ (Section 5.2)
3. Discovery of new vertices via ray-shooting (Section 5.2)
4. Combining the vertices of the two intersection directions (Section 5.2)
5. Enforcing of A-irredundancy (Section 5.3)
6. Conversion from dual to primal representation (Section 5.1)

The primal dual conversions and the combining of vertices are computed in constant time, as this only involves computing the transpose and concatenation which can be done by changing the indexing of the corresponding matrices. Therefore, we will focus on the remaining three steps, which are all conducted in dual space.

In the following we assume the setting, where we compute the convex hull of the two d-dimensional, bounded polytopes which in duals space are defined by $\mathcal{P}_1 = (\boldsymbol{A}_1, \mathcal{R}_1)$ and $\mathcal{P}_2 = (\boldsymbol{A}_2, \mathcal{R}_2)$. For convenience sake, we assume the number of vertices to be $n_v = \max(|\mathcal{R}_1|, |\mathcal{R}_2|)$ and number of constraints $n_a = \max(|\boldsymbol{A}_1|, |\boldsymbol{A}_2|)$. Note that their roles are swapped compared to a primal space representation.

Adding Constraints and Separating Vertices. Recall that in dual space we compute the intersection of the two polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$. The first step to intersect $\mathcal{P}_1$ with $\mathcal{P}_2$ is to split all points in $\mathcal{R}_1$ into the three groups $\mathcal{R}_+$, $\mathcal{R}_0$, and $\mathcal{R}_-$ defined in Section 5.2 depending on whether the lie inside, on the border or outside of the polytope defined by $\boldsymbol{A}_2$. This requires (at worst) evaluating $\boldsymbol{a}_i \boldsymbol{r}_j - b_i \geq 0$ for all $\boldsymbol{r}_j \in \mathcal{R}_1$ and $\boldsymbol{a}_i, b_i \in \boldsymbol{A}_2$. The addition and comparison involved are dominated by the d-dimensional dot-product between $\boldsymbol{a}_i$ and $\boldsymbol{r}_j$, leading to a total complexity of this step of order $O(d \cdot n_a \cdot n_v)$. Note that columns corresponding to the new constraints are added to the incidence matrix, which can be populated without any extra computation with 0s for the vertices in $\mathcal{R}_+$ and 1s where constraints are satisfied with equality for vertices in $\mathcal{R}_0$.

Ray-Shooting. Recall that to discover new generating vertices, the first intersections between the rays shot from all generating vertices of $\mathcal{P}_1$ lying inside $\mathcal{P}_2$, $\boldsymbol{r}_+ \in \mathcal{R}_+$, to all points lying outside $\mathcal{P}_2$, $\boldsymbol{r}_- \in \mathcal{R}_-$, and all constraints in $\boldsymbol{A}_2$ are computed. At worst there are no points in group $\mathcal{R}_0$ and all vertices are spread equally between $\mathcal{R}_+$ and $\mathcal{R}_-$, leading to $\frac{n_v^2}{4}$ rays to be intersected with n_a constraints where each intersection corresponds to computing a ratio of dot-products and is order $O(d)$. Selecting the first intersection for each ray is linear in the intersection number. Consequently, the ray-shooting process overall is $O((d+1) \cdot n_a \cdot n_v^2)$. Note that this adds new rows corresponding to the new vertices $\mathcal{R}_*$ to the incidence matrix, which can then be populated with 0s except for the column(s) associated with the constraint of the first intersection without any additional computations.

Enforcing A-Irredundancy. The intermediate state prior to enforcing A-irredundancy contains at most $n = 2(n_v + \frac{n_v^2}{4})$ vertices, consisting of the at most n_v vertices in $\mathcal{R}_+$ and the at most $\frac{n_v^2}{4}$ vertices in $\mathcal{R}_*$, discovered during ray shooting, for both intersection directions. To enforce A-irredundancy, vertices are first sorted in descending order by the number of active constraints which is order $O(n \log(n))$. Then starting with the first vertex, row-wise inclusion of the corresponding incidence matrix rows is checked for all following elements. Each check is $O(n_a)$ and $\frac{n^2-n}{2}$ checks have to be performed in the worst case that is, if no element is removed. This leads to an overall complexity of $O(n_a \cdot n_v^4 + n_v^2 \log(n_v^2))$ for enforcing A-irredundancy.

PDDM Complexity. Putting the three elements together and observing $d < n_v$ for any d-dimensional, bounded polytope, we observe that both the ray-shooting and the separation of vertices get dominated by the last step of enforcing A-irredundancy. Swapping the roles of n_v and n_a to derive an expression in terms of primal space entities, we arrive at an overall complexity of $O(n_v \cdot n_a^4 + n_a^2 \log(n_a^2))$. □

C PDDM Soundness

In this section, we proof the soundness of the Partial Double Description Method. Computing sound over-approximations of the convex hull of two polytopes in primal space, by inclusion-inversion, is equivalent to computing sound under-approximation of the intersection of their dual space representations.

Since the primal-dual conversion employed in the PDDM is exact, showing a sound under-approximating intersection of two polytopes in PDD in dual space is sufficient for overall soundness.

Enforcing A-irredundancy on a polytope $\mathcal{P}$ to yield $\mathcal{Q}$ can only remove generators, yielding $\mathcal{Q} \subseteq \mathcal{P}$. It follows directly that $\mathcal{Q}$ is a sound under-approximation, if $\mathcal{P}$ is.

If both polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$ generated by the vertex sets obtained for the two directions of batch intersection are sound under-approximations of the true intersection of the exact $\mathcal{H}$-representations, it follows that their union $\mathcal{P}$ is also a sound under-approximation. Hence, the soundness of the PDDM follows if we can show soundness of the batch intersection step.

Batch Intersection.

Theorem C.1. *The intersection $\mathcal{P}' = (\boldsymbol{A}', \mathcal{R}_p')$ of a polytope $\mathcal{P}$ in PDD $(\boldsymbol{A}_p, \mathcal{R}_p)$ with the exact constraints $\boldsymbol{A}_q$ of a polytope $\mathcal{Q}$ computed with the Batch Intersection, described in Section 5.2 and detailed in Algorithm 2, is a sound under-approximation of the intersection of the two exact $\mathcal{H}$-representations $\boldsymbol{A}_p$ and $\boldsymbol{A}_q$:*

$$\{\boldsymbol{x} \in \mathbb{R}^d | \boldsymbol{A}'\boldsymbol{x} \geq 0\} = \{\boldsymbol{x} \in \mathbb{R}^d | \boldsymbol{A}_p\boldsymbol{x} \geq 0 \land \boldsymbol{A}_q\boldsymbol{x} \geq 0\}$$

$$\left\{ \sum_{\boldsymbol{r}_i \in \mathcal{R}_p'} \lambda_i \boldsymbol{r}_i \middle| \sum_i \lambda_i \leq 1 \right\} \subseteq \{\boldsymbol{x} \in \mathbb{R}^d | \boldsymbol{A}_p\boldsymbol{x} \geq 0 \land \boldsymbol{A}_q\boldsymbol{x} \geq 0\}$$

Proof. Recall that a PDD consists of an exact $\mathcal{H}$-representation and an under-approximate $\mathcal{V}$-representation. The intersection of two polytopes in $\mathcal{H}$-representation is simply the union of all constraints, allowing for an exact intersection of the $\mathcal{H}$-representations. Hence, it remains to show that the resulting $\mathcal{V}$-representation $\mathcal{R}'{}_p$ is a sound under-approximation of the $\mathcal{H}$-representation $\boldsymbol{A}'$. For this it is sufficient to show that, by construction, every vertex included in the $\mathcal{V}$-representation $\mathcal{R}_p'$ satisfies all constraints of the $\mathcal{H}$-representation $\boldsymbol{A}'$.

Recall that $\mathcal{R}_p'$ is the union of three groups of vertices (see Section 5.2):

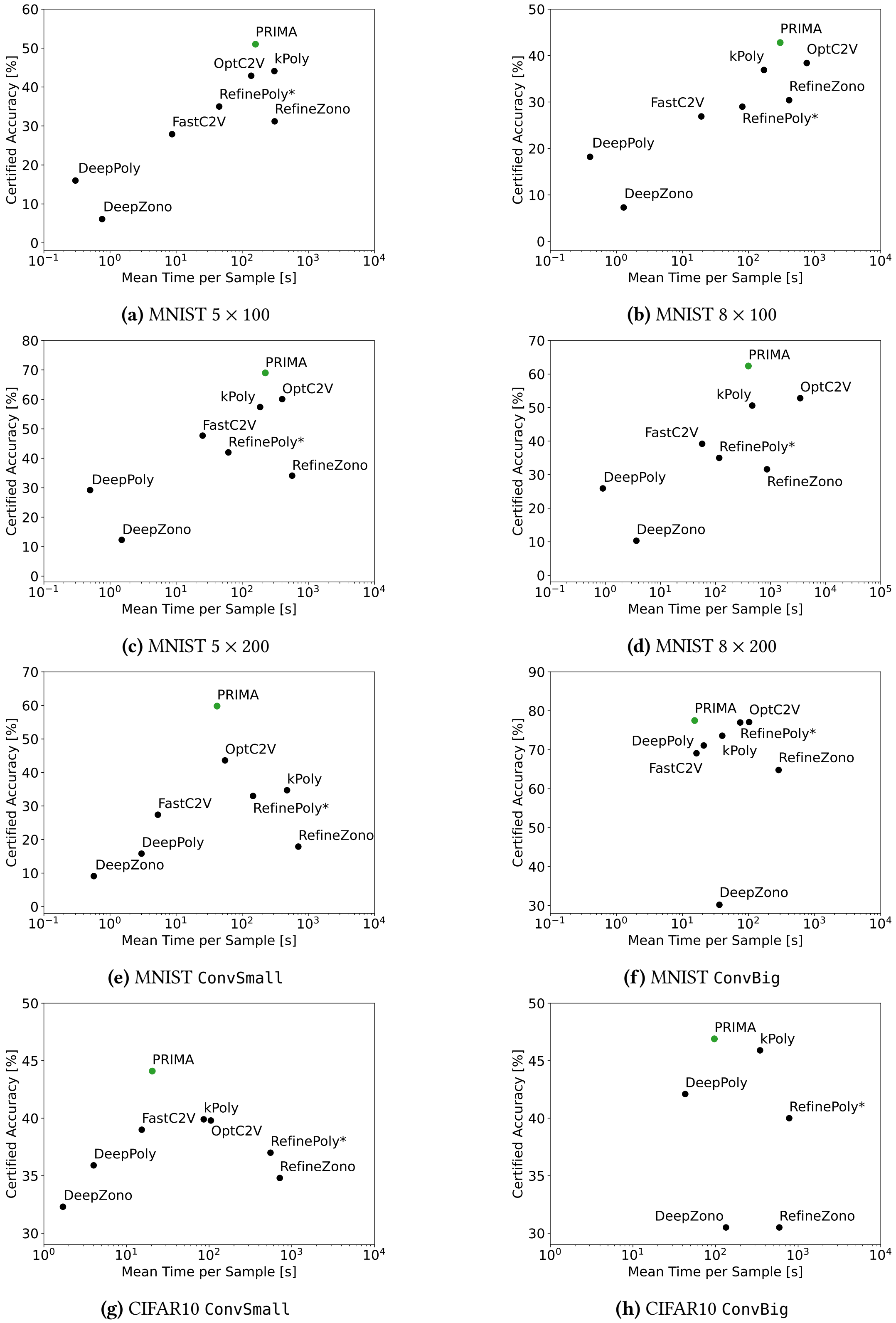
- $\mathcal{R}_+$ vertices of the generating set $\mathcal{R}_p$ that satisfy all constraints in $\boldsymbol{A}_q$ strictly,
- $\mathcal{R}_0$ vertices of the generating set $\mathcal{R}_p$ that satisfy all constraints in $\boldsymbol{A}_q$, at least one with equality,
- $\mathcal{R}_*$ the first intersections $\boldsymbol{r}_*$ of rays from a vertex in $\boldsymbol{r}_+ \in \mathcal{R}_+$ to a vertex in $\boldsymbol{r}_- \in \mathcal{R}_-$ (vertices in $\mathcal{R}_p$ not satisfying at least one constraints in $\boldsymbol{A}_q$) with the hyperplanes defined by the constraints in $\boldsymbol{A}_q$. Since $\boldsymbol{r}_-$ lies outside $\mathcal{Q}$ while $\boldsymbol{r}_+$ lies inside, an intersection $\boldsymbol{r}_*$ is guaranteed to exist and lie between the two. Therefore by convexity of $\mathcal{P}$, $\boldsymbol{r}_*$ satisfies all constraints of $\boldsymbol{A}_p$. Further, since it is the first intersection, $\boldsymbol{r}_*$ satisfies all constraints in $\boldsymbol{A}_q$, at least one with equality.

Consequently, all vertices in the generating set $\mathcal{R}_p'$ satisfy all constraints of both $\mathcal{P}$ and $\mathcal{Q}$. It follows that they generate a polytope that is a subset of the intersection $\mathcal{Q} \cap \mathcal{P}$ and therefore soundly under-approximating it, concluding the soundness proof. □

D Extended Results

In this section, we present additional experimental results omitted in the main part due to space constraints. In Figure 13 we illustrate the runtime accuracy trade-off of PRIMA in comparison to previous state-of-the-art methods including `OptC2V` [48] and `kPoly` [43]. We evaluate all methods on the first 1000 samples of the corresponding test sets with the exception of RefinePoly, where we only use the first 100 samples. We use results reported in the literature where available and rerun experiments with the hyper-parameters recommended when an evaluation on the same network, samples and perturbation-size is not available.

We observe that PRIMA is always the most accurate method and faster than either `kPoly` or `OptC2V` and often both, especially on large networks.



(a) MNIST 5 × 100

(b) MNIST 8 × 100

(c) MNIST 5 × 200

(d) MNIST 8 × 200

(e) MNIST ConvSmall

(f) MNIST ConvBig

(g) CIFAR10 ConvSmall

(h) CIFAR10 ConvBig

Figure 13. Comparison of the runtime accuracy trade-off of PRIMA (ours), OptC2V [48], FastC2V [48], kPoly [43], RefinePoly [45], DeepPoly [46], RefineZono [46] and DeepZono [44] on different networks, evaluated on the first 1000 samples (100 for RefinePoly) of the corresponding test sets. Further up and to the left is better.