PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations

MARK NIKLAS MÜLLER[*], ETH Zurich, Switzerland
GLEB MAKARCHUK[*], ETH Zurich, Switzerland
GAGANDEEP SINGH, UIUC and VMware Research, United States
MARKUS PÜSCHEL, ETH Zurich, Switzerland
MARTIN VECHEV, ETH Zurich, Switzerland

Formal verification of neural networks is critical for their safe adoption in real-world applications. However, designing a precise and scalable verifier which can handle different activation functions, realistic network architectures and relevant specifications remains an open and difficult challenge.

In this paper, we take a major step forward in addressing this challenge and present a new verification framework, called Prima. Prima is both (i) general: it handles any non-linear activation function, and (ii) precise: it computes precise convex abstractions involving *multiple* neurons via novel convex hull approximation algorithms that leverage concepts from computational geometry. The algorithms have polynomial complexity, yield fewer constraints, and minimize precision loss.

We evaluate the effectiveness of Prima on a variety of challenging tasks from prior work. Our results show that Prima is significantly more precise than the state-of-the-art, verifying robustness to input perturbations for up to 20%, 30%, and 34% more images than existing work on ReLU-, Sigmoid-, and Tanh-based networks, respectively. Further, Prima enables, for the first time, the precise verification of a realistic neural network for autonomous driving within a few minutes.

CCS Concepts: • **Theory of computation → Abstraction**; **Program verification**; • **Computing methodologies → Neural networks**.

Additional Key Words and Phrases: Robustness, Convexity, Polyhedra, Abstract Interpretation

ACM Reference Format:
Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2022. PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. *Proc. ACM Program. Lang.* 6, POPL, Article 43 (January 2022), 32 pages. https://doi.org/10.1145/3498704

1 INTRODUCTION

The growing adoption of neural networks (NNs) in many safety critical domains highlights the importance of providing formal, deterministic guarantees about their safety and robustness when deployed in the real world [Szegedy et al. 2014]. While the last few years have seen significant

[*]Equal contribution

Authors' addresses: 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, UIUC and VMware Research, United States, ggnds@illinois.edu, gasingh@vmware.com; Markus Püschel, Department of Computer Science, ETH Zurich, Switzerland, pueschel@inf.ethz.ch; Martin Vechev, Department of Computer Science, ETH Zurich, Zurich, Switzerland, martin.vechev@inf.ethz.ch.

2475-1421/2022/1-ART43
https://doi.org/10.1145/3498704

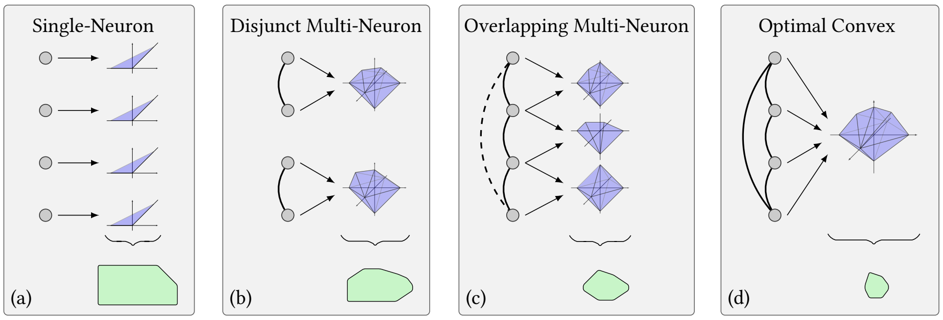


Fig. 1. Illustration of the tightness of different abstraction strategies, for a layer of four neurons (grey dots). Strong interdependencies between neurons that can be captured directly or indirectly are shown as solid or dashed lines, respectively. Individual single-neuron, multi-neuron or optimal convex abstractions are illustrated in blue and the resulting overall layer-wise abstraction in green.

progress in formal verification of NNs, existing deterministic methods (see Urban and Miné [2021] for a survey) still either do not scale to or are too imprecise when handling realistic networks.

Key challenge: handling non-linearities. Neural networks interleave affine and non-linear activation layers (e.g., ReLU, Sigmoid), leading to highly non-linear behaviours. Because affine layers can be captured exactly using linear constraints, the key challenge in neural network verification rests in designing methods that can handle the effect of these non-linear activations in a precise and scalable manner.

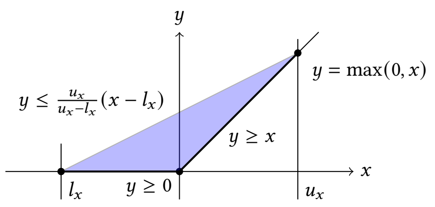


Fig. 2. Convex single-neuron approximation (blue) of a ReLU (black) with bounded inputs $x \in [l_x, u_x]$.

Exact verification, e.g., [Anderson et al. 2019, 2020; Bunel et al. 2020b; Ehlers 2017; Katz et al. 2017; Singh et al. 2019c; Tjeng et al. 2017; Wang et al. 2018, 2021], has, in the worst-case, exponential complexity in the (large) number of non-linear activations due to a combinatorial blow-up of case distinctions (e.g., for ReLUs) and complex shapes for general activations (e.g., for Sigmoids). Therefore exact verifiers typically only handle piecewise linear activations and do not scale to larger networks.

To overcome this limitation, state-of-the-art verifiers, e.g., [Singh et al. 2019a,b; Tjandraatmadja et al. 2020; Weng et al. 2018; Xu et al. 2020a; Zhang et al. 2018], often sacrifice completeness for scalability and leverage abstract interpretation [Cousot 1996] to over-approximate the effect of each activation layer with convex polyhedra. Naturally, the scalability and precision of these incomplete methods are tied to the particular polyhedral fragment they utilize.

Below, we contrast different state-of-the-art abstraction approaches with our work by comparing the strong inter-neuron dependencies they can capture directly or indirectly, illustrated as solid or dashed lines, respectively, in Figure 1 for a layer of four neurons. Individual abstractions are visualized in blue and the resulting layer-wise shape in green.

Optimal convex approximation. Assume a layer of n neurons, each applying the scalar, univariate, non-linear activation function $f\colon \mathbb{R} \to \mathbb{R}$ and the most precise polyhedral abstraction $\mathcal{P}$ of the layer's inputs $\boldsymbol{x}$. The most precise convex abstraction of the layer output is then given by the convex hull of all input-output vector pairs $\operatorname{conv}(\{(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x})) \mid \boldsymbol{x} \in \mathcal{P} \subseteq \mathbb{R}^n\})$, illustrated in Figure 1 (d), where all interactions are fully captured. Computing this $2n$-dimensional convex hull, however, is intractable due to the exponential cost $O(n_v \log(n_v) + n_v^n)$ [Chazelle 1993] in the number of neurons n, where the number of vertices $n_v = O(n_c^n)$ of the input polytope $\mathcal{P}$ is at worst also exponential in n [Seidel 1995] (n_c is the number of constraints of the input polytope $\mathcal{P}$).

Single-Neuron approximation. Most incomplete verifiers are fundamentally based on single-neuron convex abstractions, i.e., activations are approximated separately. The tightest single-neuron abstractions maintain upper and lower bounds l_x, u_x for each input x and compute convex hulls of all input-output tuples: $\text{conv}(\{(x, f(x)) \mid x \in [l_x, u_x] \subseteq \mathbb{R}\})$, as illustrated in Figure 2 for a ReLU. The union of the obtained constraints is the final abstraction of the layer. Geometrically, it is the Cartesian product of the convex hulls for each ReLU. This abstraction is significantly larger in volume (exponential in n) than the optimal convex hull discussed earlier, the key reason being that the interdependencies between neurons in the same layer are ignored, as illustrated in Figure 1 (a). Thus, the approximation error can grow exponentially with each layer, accumulating significant imprecision.

Multi-Neuron approximation. To mitigate this limitation for ReLU networks, recent works [Palma et al. 2021; Singh et al. 2019a; Tjandraatmadja et al. 2020] introduced multi-neuron abstractions as a first compromise between the optimal but intractable layer-wise and the imprecise but scalable neuron-wise abstraction. Singh et al. [2019a] partition the neurons of an activation layer into small sets of size $n_s \leq 5$, form groups of $k \leq 3$ neurons for each partition, approximate the group's input with octahedra [Clarisó and Cortadella 2007], and then compute exact convex hulls *jointly* approximating the *output* of k ReLUs for this input. These exact convex hull computations are computationally expensive and yield complex constraints, limiting the approach to only a few, mostly disjoint neuron groups, and restricting the number of captured dependencies, see Figure 1 (b). Tjandraatmadja et al. [2020] and Palma et al. [2021] merge the activation layer with the preceding affine layer and compute a convex approximation over the resulting multivariate activation layer for a hyperbox approximation of its input. This coarse input abstraction effectively restricts their approach to interactions over a single affine layer at a time. While both approaches currently yield state-of-the-art precision, they are limited to ReLU activations and lack scalability as they require small instances of the NP-hard convex hull problem to be solved exactly or large instances to be solved partially. They also do not address the problem of capturing enough neuron-interdependencies within a layer to come as close as possible to the optimal convex abstraction.

This work: precise multi-neuron approximations. In this work, we present the first general verification framework for networks with arbitrary, bounded, multivariate activation functions called PRIMA (PRecIse Multi-neuron Abstraction). PRIMA builds on the group-wise approximations from Singh et al. [2019a] and leverages the key insight that most interdependencies between neurons can be captured by considering a large number of relatively small, overlapping neuron-groups. While not achieving the tightness of the optimal convex approximation, PRIMA yields much tighter layer-wise approximations than previous methods, as shown in Figure 1 (c).

The key technical contributions of our work are: (i) PDDM (Partial Double Description Method) – a general, precise, and fast convex hull approximation method for polytopes that enables the consideration of many neuron groups, and (ii) SBLM (Split-Bound-Lift Method) – a novel decomposition approach that builds upon the PDDM to quickly compute multi-neuron constraints. While we combine these methods with abstraction refinement approaches in PRIMA, we note that they are also of general interest (beyond neural networks) and can be used independently of each other.

PRIMA can be applied to any network with bounded, multivariate activation functions and arbitrary specifications expressible as polyhedra such as individual fairness [Ruoss et al. 2020b]; global safety properties [Katz et al. 2017]; and acoustic [Ryou et al. 2020], geometric [Balunović et al. 2019], spatial [Ruoss et al. 2020a], and ℓ_p-norm bounded perturbations [Gehr et al. 2018]. Our experimental evaluation shows that PRIMA achieves state-of-the-art precision on the majority of our ReLU-based classifiers while remaining competitive on the rest. For Sigmoid- and Tanh-based networks, PRIMA significantly outperforms prior work on all benchmarks. Further, PRIMA enables, for the first time, precise and scalable verification of a realistic architecture for autonomous driving containing > 100k neurons in a regression setting. Finally, while Prima is incomplete, it can be used for boosting the scalability of state-of-the-art complete verifiers [Singh et al. 2019c; Wang et al. 2021] for ReLU-based networks that benefit from more precise convex abstractions.

Main contributions. Our key contributions are:

(1) PDDM, a precise method for approximating the convex hull of polytopes, with worst-case polynomial time- and space-complexity and exactness guarantees in low dimensions.
(2) Split-Bound-Lift Method, a technique which efficiently computes joint constraints over groups of non-linear functions, by decomposing the underlying convex hull problem into lower-dimensional spaces.
(3) Prima, a novel verifier combining these approaches with a sparse neuron grouping technique and abstraction refinement, to obtain the first multi-neuron verifier for arbitrary, bounded, multivariate non-linear activations (e.g., ReLU, Sigmoid, Tanh, and MaxPool).
(4) An evaluation of Prima on a range of activations and network architectures (e.g., fully connected, convolutional, and residual). We show that Prima is significantly more precise than state-of-the-art, with gains of up to 20%, 30%, and 34% for ReLU-, Sigmoid-, and Tanh-based networks, while being effective in a regression setting, scaling to large networks, and enabling verification in real-world settings such as autonomous driving.

We release our code as part of the open-source framework ERAN at https://github.com/eth-sri/eran.

2 BACKGROUND

In this section, we establish the terminology we use to discuss polyhedra, neural networks (NNs) and their verification.

Notation. We use lower case Latin or Greek letters $a, b, x, \ldots, \lambda, \ldots$ 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 $f\colon \mathbb{R}^d \to \mathbb{R}$ and vector valued functions bold as $\boldsymbol{f}\colon \mathbb{R}^d \to \mathbb{R}^k$.

Neural networks. We focus our discussion on networks $\boldsymbol{h}(\boldsymbol{x})\colon \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|}$ that map input samples (images) $\boldsymbol{x} \in \mathcal{X}$ to numerical scores $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{Y}|}$. For a classification task, the network $\boldsymbol{h}$ classifies an input $\boldsymbol{x}$ by applying argmax to its output: $c(\boldsymbol{x}) = \arg\max_j \boldsymbol{h}(\boldsymbol{x})_j$. While our methods can refine the abstraction of activation functions in arbitrary neural architectures [Xu et al. 2020a], for simplicity, we discuss a feedforward architecture which is an 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 $\boldsymbol{f}(\boldsymbol{x})$ such as ReLU, Tanh, Sigmoid, or MaxPool:

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

2.1 Neural Network Verification

Prima is an optimization-based verification approach and supports any safety specification (pre- and post-condition) which can be expressed as a convex polyhedron. Examples of such specifications include but are not limited to individual fairness [Ruoss et al. 2020b], global safety properties [Katz et al. 2017], acoustic [Ryou et al. 2020], geometric [Balunović et al. 2019], spatial [Ruoss et al. 2020a], and ℓ_p-norm bounded perturbations [Gehr et al. 2018].

At its core, Prima is based on accumulating linear constraints encoding the whole network for a given (convex) pre-condition, defining a linear optimization objective representing the property to be verified, and finally using an LP solver to derive a bound on this objective. If this bound satisfies a predetermined threshold (that depends on the property), the property is verified.

While all affine layers (e.g., linear, convolutional, and normalization layers) can be encoded exactly using linear constraints, non-linearities have to be over-approximated via constraints in their input-output space. That is, for an activation layer $\boldsymbol{f}\colon \mathbb{R}^n \to \mathbb{R}^d$ and a given set of inputs

$\mathcal{P}_{\text{in}} \subseteq \mathbb{R}^n$, we need to derive *sound* output constraints, that represent a set $\mathcal{P}_{\text{in-out}} \subseteq \mathbb{R}^{d+n}$ which includes all possible input-output pairs that can be obtained by applying f to the inputs in $\mathcal{P}_{\text{in}}$.

We show an over-approximation for a *single* ReLU in Figure 2. In the concrete, the ReLU maps input x to $y = \max(0, x)$. If the bounds $0 > l_x \leq x \leq u_x > 0$ are known, the best convex approximation is given by the blue triangle. In this work we present novel methods to compute tighter shapes by considering *multiple neurons jointly* in a higher dimensional space.

2.2 Overview of Convex Polyhedra

We now introduce the necessary background on polyhedra. A polyhedron can be represented as the convex hull of its extremal points, called the vertex- or $\mathcal{V}$-representation, or as the subspace satisfying a set of linear constraints, called the halfspace constraint or $\mathcal{H}$-representation. Simultaneously maintaining both representations of the same polyhedron is called double description.

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

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

where $\mathbb{R}_0^+$ are the positive real numbers including 0. A polyhedral cone $\mathcal{P} \subseteq \mathbb{R}^d$ is the positive linear span of a set of generators called rays $\mathcal{R} = \{x_i \in \mathbb{R}^d\}$ and always includes the origin:

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

Halfspace representation. Alternatively, a polyhedron can be described as the set $\mathcal{P} \subseteq \mathbb{R}^d$ satisfying a system of linear inequalities (or constraints) defined by $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$:

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

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 we have $\boldsymbol{b} = \boldsymbol{0}$. For convenience, a polyhedron $\mathcal{P}(\boldsymbol{A}, \boldsymbol{b})$ 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 $\boldsymbol{A}' = [-\boldsymbol{b}, \boldsymbol{A}]$.

A k-face $\mathcal{F}$ of a d-dimensional polyhedron is a k-dimensional subset $\mathcal{F} \subseteq \mathcal{P}$ satisfying $d - k$ linearly independent constraints[1] with equality. We call a 0-face a vertex and a $(d-1)$-face a facet [Edelsbrunner 2012]. 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. Polyhedra static analysis [Fukuda and Prodon 1995; Motzkin et al. 1953; Singh et al. 2017] usually maintains both representations ($\mathcal{H}$ and $\mathcal{V}$) 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 intersections is NP-hard. Conversely, computing intersections in the $\mathcal{H}$-representation is trivial (union of constraints), but computing the convex hull is NP-hard. The transformation from the $\mathcal{V}$- to the $\mathcal{H}$-representation is called the *convex hull* problem and the reverse the *vertex enumeration* problem. Both are NP-hard in general.

Inclusion. We define the inclusion of a polytope $\mathcal{Q}$ in a polytope $\mathcal{P}$ as: $\mathcal{Q} \subseteq \mathcal{P}$ or equivalently, $\forall \boldsymbol{x} \in \mathcal{Q}, \boldsymbol{x} \in \mathcal{P}$. In this setting, we say $\mathcal{P}$ over-approximates $\mathcal{Q}$ and $\mathcal{Q}$ under-approximates $\mathcal{P}$.

[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.

3 OVERVIEW OF PRIMA

We now present an overview of Prima, our framework for faster and more precise verification of neural networks with arbitrary, bounded, multivariate, non-linear activations. We provide a complete formal description of its main components PDDM and SBLM in Sections 4 and 5, and of Prima in Section 6. In our explanations, we follow the setup outlined in Section 1: an activation layer consisting of n neurons representing non-linear activations $f(x)$ (e.g., ReLU, Tanh, Sigmoid).

Computing a convex approximation of a whole layer. Conceptually, given an n-dimensional polytope $\mathcal{S}$ constraining the input to the activation layer, Prima computes a set of multi-neuron constraints, forming a convex over-approximation of this layer, as follows:

(1) *Group decomposition:* Decompose the set of n activations in the layer into overlapping groups (subsets) of size k.
(2) *Octahedral projection:* For each such group i, compute an octahedral over-approximation $\mathcal{P}^i$ of the projection of $\mathcal{S}$ to the input-space of group i.
(3) *Split-Bound-Lift Method (SBLM):* Then, for each polytope $\mathcal{P}^i$, compute a joint convex over-approximation $\mathcal{K}^i$ of the group output in the $\mathcal{H}$-representation using our novel SBLM method. This method decomposes the problem into lower dimensions and leverages our novel Partial Double Description Method (PDDM) with polynomial complexity to compute fast and scalable convex hull approximations. Both SBLM and PDDM are also key to making Prima applicable to non-piecewise-linear activations.
(4) *Combine constraints:* Finally, take the intersection of all output constraints $\mathcal{K}^i$ (a union of all constraints) to obtain an over-approximation of the entire layer output.

Verification is performed by solving an LP problem which combines the generated multi-neuron constraints with an LP encoding of the whole network (evaluated in Section 7). We now explain the basic workings of each step and illustrate the key concepts on a running example.

Group decomposition. Computing convex hulls for large sets of activations (e.g., a whole layer) is infeasible. Thus, we consider groups of size k, typically $k = 3$ or 4. The key idea here is to capture dependencies between activation inputs and outputs ignored by neuron-wise approximations and thus achieve tighter approximations. The tightness increases with the number of groups and, importantly, the degree of overlap between them. Considering all possible $\binom{n}{k}$ groups for every layer is too expensive; thus we define the parameters partition size n_s and group overlap s for tuning the cost and precision of our approximations. We first partition the activations of a layer into sets of size n_s (sorting by volume of the single neuron abstraction) and then for every set[2] choose a subset of all $\binom{n_s}{k}$ groups that pairwise overlap by at most s, $0 \leq s < k$.

Octahedral projection. Projecting the layer-wise input polytope $\mathcal{S}$ onto the input dimensions of every group is generally intractable due to the high dimensionality and large number of constraints. Therefore, we follow the idea of [Singh et al. 2019a] and over-approximate the projection. Empirically we find that multidimensional octahedra [Clarisó and Cortadella 2007], yielding $3^k - 1$ input constraints per group of k neurons, provide a good trade-off between accuracy and complexity. Such a projection is illustrated in Figure 3 for a layer of $n = 3$ neurons and $k = 2$.

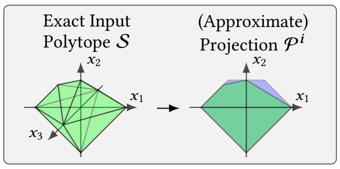


Fig. 3. Exact projection of $\mathcal{S} \in \mathbb{R}^3$ (left) to $k = 2$ variables (green) and its octahedral over-approximation $\mathcal{P}^i$ (blue).

[2]For piecewise-linear activations, typically n_s is chosen large enough such that there is only one set.

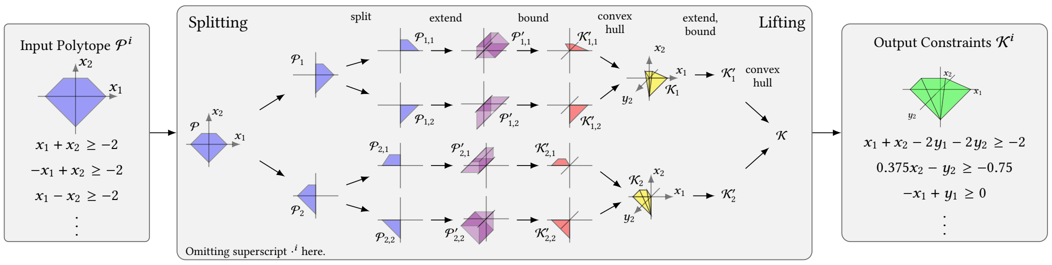


Fig. 4. Illustration of the Split-Bound-Lift Method for a group of $k = 2$ neurons and a ReLU activation.

3.1 Split-Bound-Lift Method

The next and most demanding step takes a k-dimensional input polytope for a given k-activation group, and computes a $2k$-dimensional convex over-approximation of the output of the corresponding k activations. We introduce a new technique, called Split-Bound-Lift Method, and illustrate its workings in Figure 4 on an example. We assume ReLU activations, group size $k = 2$, and an octahedral input polytope $\mathcal{P}^i$ (left panel in Figure 4) described by

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

Our method has three main components explained next.

Split the input polytope. We first split $\mathcal{P}^i$ into regions, which we call quadrants, for which tight or even exact, linear bounds of the activation functions are available. Choosing the right splits is essential for ensuring tight approximations. For piecewise-linear activation functions (like ReLU), splitting into their linear regions even yields exact bounds in every quadrant, leading to the tightest approximations. For our example with ReLU activations, this corresponds to splitting along hyperplanes where the input variables x_1 and x_2 are 0. We (randomly) choose the ordering $\{y_1, y_2\}$ of output variables and split $\mathcal{P}^i$ (in the following we omit the superscript i) along the corresponding hyperplanes. That is, we first intersect $\mathcal{P}$ with the halfspaces $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_1 \geq 0\}$ and $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_1 \leq 0\}$, obtaining $\mathcal{P}_1$ and $\mathcal{P}_2$, and then $\mathcal{P}_1$ and $\mathcal{P}_2$ with $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_2 \geq 0\}$ and $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_2 \leq 0\}$. These intersections generate a tree of polytopes visualized in the first three columns in the central panel of Figure 4 with the quadrants as leafs (third column). For brevity, we only follow the bottom half. There, the two quadrants $\mathcal{P}_{2,1}$ and $\mathcal{P}_{2,2}$ are described by

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

In the second part of the algorithm, we lift these quadrants step-by-step from the space of only their inputs to the space of both their inputs and outputs. We will now describe one step of lifting consisting of extending, bounding and computing a convex hull.

Extend and bound the quadrants. We extend[3] the quadrants one output variable at a time, which, as we will see later, enables significant gains in speed while reducing the approximation error. In our example, we first trivially extend all quadrants from the (x_1, x_2)-space to the (y_2, x_1, x_2)-space (fourth column in Figure 4). Next, we bound the quadrants in the added dimension using the linear bounds (parametrically defined, see Section 5) corresponding to applying (an approximation of) the activation in the quadrant. Here, $y_2 \leq x_2$ and $y_2 \geq x_2$ for the quadrant $\mathcal{P}_{2,1}$ (since $x_2 \geq 0$) and $y_2 \leq 0$ and $y_2 \geq 0$ for the quadrant $\mathcal{P}_{2,2}$ (since $x_2 \leq 0$). Note that in this case the bounds we apply on every

[3]Extending a d-dimensional polytope by a variable defines it in the $d+1$-dimensional space, where it is (initially) unbounded in the dimension of the added variable.

quadrant are exact, yielding the two polytopes (fifth column) with 0 volume in their $3d$-space (in general, the bounds need not be exact):

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

Approximate convex hull. Next, we compute the convex hull of $\mathcal{K}'_{2,1}$ and $\mathcal{K}'_{2,2}$. Instead of using an exact method, we utilize our PDDM to compute precise over-approximations, leveraging the concept of duality, ideas from computational geometry and our novel PDD polyhedron representation (explained below and in more detail in Section 4). Note that because the considered quadrants are only extended one variable at a time, the computation takes place in $3d$ despite the group-output being in the $4d$ (y_1, y_2, x_1, x_2)-space. This yields two main benefits: (i) precision – directly computing $2k$-dimensional convex hulls with PDDM will lose more precision than our decomposed method, because PDDM is exact for polytopes of dimension up to three and loses precision only slowly for higher dimensions, and (ii) speed – a lower-dimensional polytope with fewer constraints and generally also fewer vertices significantly reduces the time required for the individual convex hull computations.

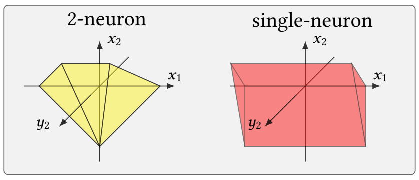


Fig. 5. Comparison of 2-neuron and 1-neuron constraints projected into y_2-x_1-x_2-space for a ReLU activation, given input polytope $\mathcal{P}^i$.

Importantly, our approximate method scales quadratically as $O\{n_a^4 \cdot n_v + n_a^2 \log(n_a^2)\}$ in the number of input constraints n_a and linear in the number of vertices n_v (see Theorem 4.5) while optimal exact methods are in $O(n_v \log(n_v) + n_v^{\lfloor d/2 \rfloor})$ [Chazelle 1993], i.e., exponential in the number of dimensions and superlinear in the number of input vertices.

Note that for non-piecewise-linear functions (e.g., Tanh or Sigmoid), the number of vertices doubles when extending by a dimension. This makes exact methods intractable and approximate methods not using the decompositional SBLM approach (that is, extending by all dimensions at the same time) slow (see our evaluation in Section 7).

We now obtain the convex hull (sixth column) of the two polytopes $\mathcal{K}'_{2,1}$ and $\mathcal{K}'_{2,2}$ which is exact in our $3d$ case:

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

We compute $\mathcal{K}_1$ analogously, thus completing the first step of lifting. The next and in this case final step of lifting starts with extending $\mathcal{K}_1$ and $\mathcal{K}_2$ by y_1 into the (y_1, y_2, x_1, x_2)-space, where we apply bounds on y_1 yielding (in $4d$ and thus not illustrated as figure)

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

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

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

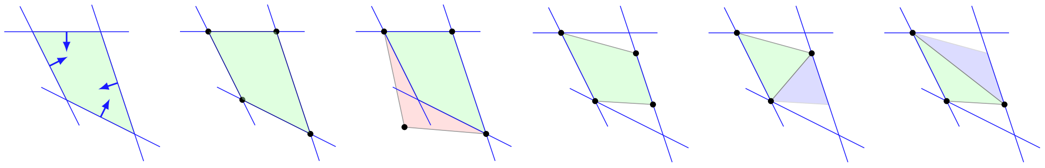

(a) Constraints - $\boldsymbol{A}$
(b) Generators - $\mathcal{R}$
(c) Unsound $\mathcal{V}$-representation
(d) Sound $\mathcal{V}$-representation
(e) A-irredundant a)
(f) A-irredundant b)

Fig. 6. Illustration of the Partial Double Description. Input constraints $\boldsymbol{A}$ (a), exact vertex enumeration $\mathcal{R}_{DD}$ (b), unsound partial vertex enumeration violating the PDD definition (c), partial or approximate vertex enumeration $\mathcal{R}_{PDD}$ (d), and A-irredundant versions of the partial vertex enumeration (e).

Naturally, the region $\mathcal{K}$ is tighter than the tightest single-neuron approximations (triangle relaxation, discussed earlier). We illustrate this point by comparing their projections into the (y_2, x_1, x_2)-space in Figure 5.

3.2 Partial Double Description Method (PDDM)

We now introduce the new PDDM for computing *fast, precise, and sound* over-approximations of the convex hull of two polyhedra. This is in contrast to existing approximation methods, which either optimize for closer approximations [Bentley et al. 1982; Khosravani et al. 2013; Sartipizadeh and Vincent 2016; Zhong et al. 2014] but sacrifice the soundness required for verification, or have exponential complexity [Xu et al. 1998], making them too expensive for our application.

Double description method. The well-known Double Description Method (DDM) [Fukuda and Prodon 1995; Motzkin et al. 1953] for computing the convex hull of two polyhedra in Double Description works as follows: (i) translate both polyhedra to their dual representation (explained in Section 4), (ii) intersect them in dual space by adding the constraints of one to the other, one-at-a-time, computing full Double Descriptions at every intermediate step, and (iii) translate the result back to primal space. Every step of adding an additional constraint generates quadratically many new vertices, leading to an overall increase exponential in the number of constraints (in dual space).

Partial double description. We introduce the Partial Double Description (PDD), which guarantees soundness and also allows an approximate much cheaper intersection in dual space. We combine an exact $\mathcal{H}$-representation, as their intersection is trivial, with an under-approximating[4] $\mathcal{V}$-representation, as their exact intersection carries exponential cost. We illustrate this in Figure 6, where we show the constraints $\boldsymbol{A}$ describing a polytope in (a), the corresponding exact $\mathcal{V}$-representation in (b), an unsound approximate $\mathcal{V}$-representation in (c), and three sound ones in (d), (e), and (f). Note that this definition of the PDD allows many different $\mathcal{V}$-representations for a given $\mathcal{H}$-representation (see (d), (e), and (f) in Figure 6) some of which are quite imprecise (see (e) and (f)).

Partial double description method. Now, we define the PDDM to compute approximate convex hulls in PDD leveraging two key ideas: (i) instead of intersecting in dual space by adding the constraints of one polytope to the other one-at-a-time (as per DDM), we add them all in a single step. Crucially, this leads to an *overall* number of vertices at most quadratic (instead of exponential) in the number of original vertices (in dual space), and (ii) this single-step approach is asymmetric and we can greatly increase the intersection accuracy, by performing it in both directions and combining the resulting vertices. Overall, our approach yields a polynomial complexity (see Theorem 4.5) algorithm for sound convex hull approximations (see Theorem 4.1), guarantees exactness for low

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

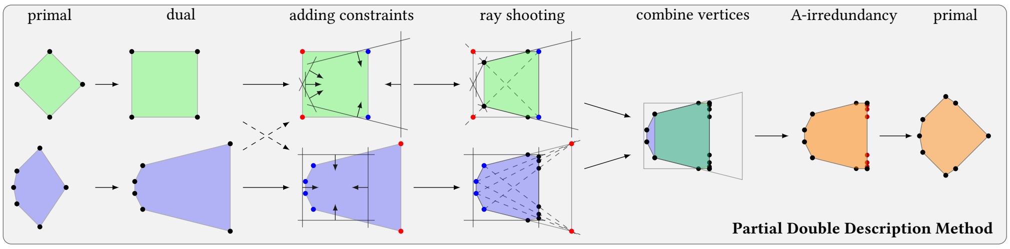


Fig. 7. Partial Double Description Method for a 2-dimensional example. The input polytopes (1st column) are translated to their dual representation (2nd column), then all their constraints are added to the other dual polytope (3rd column). The points are separated based on whether they are included in the intersection of the $\mathcal{H}$-representations. Now ray-shooting is used to discover vertices on the rays between points in the intersection (blue points) to those outside (red points) by intersecting the rays with the constraints added in the previous step (4th column). The vertices of both $\mathcal{V}$-representations are then combined (5th column) before A-irredundancy is enforced (6th column) and the result is translated back to primal space (7th column).

dimensions (see Theorem 4.4), and empirically is two orders of magnitude faster for the challenging cases in our experiments (see Figure 16c), while losing precision only slowly as dimensionality increases (see Figure 16b). We illustrate the Partial Double Description Method in Figure 7 and provide more technical details in Section 4.

3.3 Layerwise Abstraction

So far we have seen how to compute the multi-neuron convex approximation for a single group of k activations. To compute the final abstraction of the whole activation layer, we combine the constraints forming the $\mathcal{H}$-representations of the computed output polyhedra of each group, thereby obtaining the $\mathcal{H}$-representation of the polytope describing the layerwise over-approximation.

4 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 the PDDM step by step as illustrated in Figure 7.

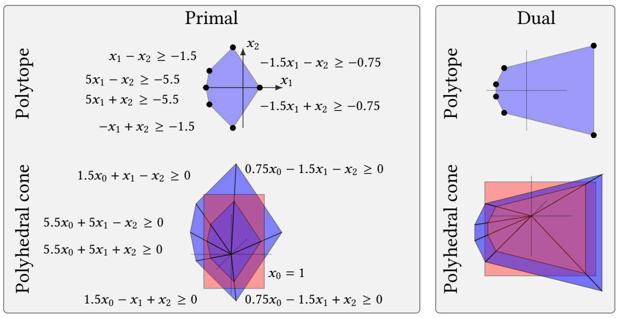

Primal
Dual
Polytope
Polyhedral cone
$x_1 - x_2 \geq -1.5$
$5x_1 - x_2 \geq -5.5$
$5x_1 + x_2 \geq -5.5$
$-x_1 + x_2 \geq -1.5$
x_2
x_1
$-1.5x_1 - x_2 \geq -0.75$
$-1.5x_1 + x_2 \geq -0.75$
$1.5x_0 + x_1 - x_2 \geq 0$
$5.5x_0 + 5x_1 - x_2 \geq 0$
$5.5x_0 + 5x_1 + x_2 \geq 0$
$1.5x_0 - x_1 + x_2 \geq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

Fig. 8. Top: polytope in primal (left) and dual (right) space. Bottom: equivalent polyhedral cones in homogenized coordinates. In red: the plane the cone can be intersected with to recover the polytope.

The 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 2.2) 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. In the following explanations we will use either term, depending on convenience. The original polytope can be recovered from the cone, by intersecting it with the hyperplane $x'_0 = 1$ in primal, or with $x'_0 = -1$ in dual space (explained next) as visualized in Figure 8.

Duality. The dual $\overline{\mathcal{P}}$ of a polytope $\mathcal{P}$ with a minimal set (containing no redundancy) of extremal vertices $\mathcal{R}$ enclosing the origin but not containing it in its boundary (to ensure a bounded dual) is defined as

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

and for polyhedral cones $\mathcal{P}'$ as [Genov 2015]

$$\overline{\mathcal{P}'} = \{\boldsymbol{y}' \in \mathbb{R}^{d+1} \mid \boldsymbol{x}'^\top \boldsymbol{y}' \leq 0 \; \forall \boldsymbol{x}' \in \mathcal{P}'\}. \tag{2}$$

Figure 8 shows an example of the dual of a polytope. Important for the remaining section are four properties of the transform between primal and dual. 1) The dual of a polyhedron is also a polyhedron. 2) It is inclusion reversing: $\mathcal{P} \subset \mathcal{Q}$ if and only if $\overline{\mathcal{Q}} \supset \overline{\mathcal{P}}$, 3) 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 (note that this implies that the vertices of the primal correspond to the supporting hyperplanes of the dual and vice-versa), and 4) the dual of the dual of a polyhedron is the original primal polyhedron $\overline{\overline{\mathcal{P}}} = \mathcal{P}$.

Partial double description. We leverage these duality properties in two ways: We translate the convex hull problem in primal space to an intersection problem in dual space (only involving a transpose given a DD or PDD) where we compute a $\mathcal{V}$-representation *under-approximating* the intersection in dual space to obtain an $\mathcal{H}$-representation *over-approximating* the convex hull in primal space (using inclusion reversion). To compute these intersections efficiently, we introduce the Partial Double Description (PDD) as a relaxation of the Double Description (DD) (Section 2.2) as discussed in the overview.

Formally, the PDD of a $(d+1)$-dimensional polyhedral cone is the pair of constraints and rays $(\boldsymbol{A}', \mathcal{R}')$ 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 constraint $\boldsymbol{a} \in \boldsymbol{A}'$, $\boldsymbol{ar} \geq 0$ 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 $\mathcal{I}_{i,j} = 0$ otherwise. Further, we define the partial ordering on $\mathcal{I}$: $\mathcal{I}_i \subseteq \mathcal{I}_j$ iff $\mathcal{I}_{i,k} \leq \mathcal{I}_{j,k}, \forall\, 1 \leq k \leq m$. Intuitively this corresponds to a row in the incidence matrix being only lesser than another if the set of active constraints of the associated ray is a strict subset of that of the other. Next, we describe PDDM as illustrated in Figure 7.

4.1 Conversion to Dual

Given the two polyhedral cones $\mathcal{P}_1$ and $\mathcal{P}_2$ in PDD representation $(\boldsymbol{A}'_1, \mathcal{R}'_1)$ and $(\boldsymbol{A}'_2, \mathcal{R}'_2)$ (1st column in Figure 7), the first step of the PDDM is to convert them to their dual space representations $(\mathcal{R}'^\top_1, \boldsymbol{A}'^\top_1)$ and $(\mathcal{R}'^\top_2, \boldsymbol{A}'^\top_2)$ [Fukuda 2020] (2nd column).

4.2 Intersection

The next step in the PDDM is the intersection in dual space (columns 3 to 5 in Figure 7). Recall that the standard approach (DDM) 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 intermediate representation. Instead, we add all constraints jointly in one step, leveraging our PDD. In the following description of the intersection, we adopt the polytope (not cone) view and consider a general polytope $(\boldsymbol{A}, \mathcal{R})$.

Batch intersection. To intersect a polytope $(\boldsymbol{A}, \mathcal{R})$ in PDD with a batch of constraints represented by the matrix $\widetilde{\boldsymbol{A}}$ and inducing the polyhedron $\mathcal{P}(\widetilde{\boldsymbol{A}})$, we separate the vertices in $\mathcal{R}$ into three sets depending on whether they satisfy all to-be-added constraints with inequality ($\mathcal{R}_+$), some only

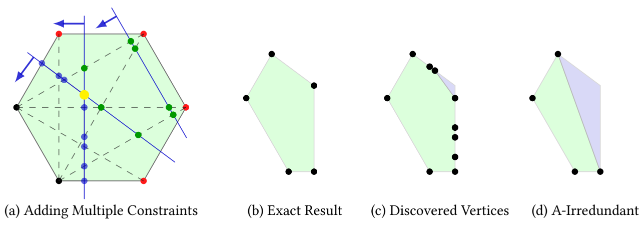


Fig. 9. Adding a batch of three constraints (blue thick lines) to a polytope in PDD. Vertices are separated into $\mathcal{R}'_+$ (black), $\mathcal{R}'_0$ (none), and $\mathcal{R}'_-$ (red). Ray-shooting discovers new vertices $\mathcal{R}'_*$ (blue), avoiding the superfluous green points, but missing an extremal vertex (yellow) (a). Exact intersection (b), result of joint constraint processing (c), and under-approximation after enforcing A-irredundancy (d).

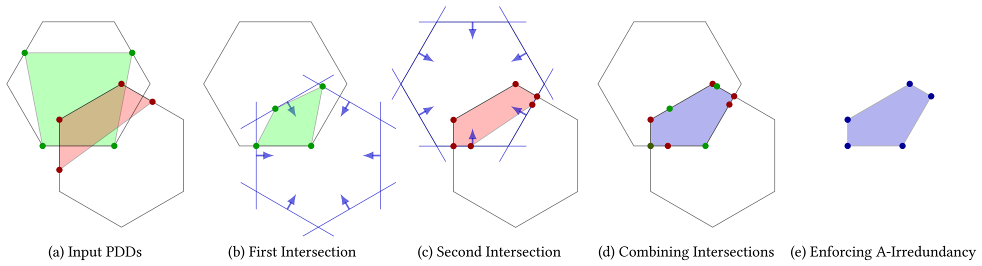


Fig. 10. Boosting intersection precision by combining both directions of batch intersection. Input polytopes in PDD with exact $\mathcal{H}$-representation (black) and approximate $\mathcal{V}$-representation ($\mathcal{P}_1$ green and $\mathcal{P}_2$ red) (a), batch intersection of $\mathcal{P}_1$ with the $\mathcal{H}$-representation of $\mathcal{P}_2$ (b), batch intersection in the opposite direction (c), combining both intersections (d), and applying A-irredundancy (e).

with equality ($\mathcal{R}_0$), or violate at least one ($\mathcal{R}_-$). This corresponds to these points lying inside, on the boundary of, or outside of the polyhedron $\mathcal{P}(\widetilde{\boldsymbol{A}})$. An example is shown in Figure 9(a): the three added constraints are shown in blue and the vertices in $\mathcal{R}'_+$ (black), $\mathcal{R}'_0$ (none), and $\mathcal{R}'_-$ (red).

Now we employ a technique called ray-shooting [Maréchal and Périn 2017] and shoot a ray $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$ from a vertex $\boldsymbol{r}_+ \in \mathcal{R}_+$ inside the intersection $\mathcal{P}(\boldsymbol{A} \cap \widetilde{\boldsymbol{A}})$ to a vertex $\boldsymbol{r}_- \in \mathcal{R}_-$ outside the intersection. We record the first hyperplane $\mathcal{H} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \widetilde{\boldsymbol{a}}_i \boldsymbol{x} = 0\}$ corresponding to one of the new constraints $\widetilde{\boldsymbol{a}}_i \in \widetilde{\boldsymbol{A}}$ that intersects with the ray $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$. We add the point $\boldsymbol{r}_*$ at which $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$ intersects $\mathcal{H}$ to the set of discovered points $\mathcal{R}_*$. Doing so for all combinations of $(\boldsymbol{r}_+, \boldsymbol{r}_-) \in \mathcal{R}_+ \times \mathcal{R}_-$ yields the set of points

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

The $\mathcal{V}$-representation of the resulting intersection is now the union $\mathcal{R}_+ \cup \mathcal{R}_0 \cup \mathcal{R}_*$. In Figure 9 (a) the rays $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$ are dashed lines from all black to all red vertices and discover new vertices $\mathcal{R}_*$ (blue). Only using the first intersections, immediately discards the green points, however, we also do not discover the yellow point, which is an extremal vertex of the exact intersection (b), obtaining instead the under-approximation (c).

Boosting precision. Batch intersection is asymmetric: The PDD of one polytope is intersected with the $\mathcal{H}$-representation of another, to obtain an exact $\mathcal{H}$-representation and under-approximating $\mathcal{V}$-representation of the intersection (compare Figure 10 (b) and (c)). By performing it in both directions, i.e., intersecting $(\mathcal{R}_1'^{\top}, \boldsymbol{A}_1'^{\top})$ with $(\mathcal{R}_2'^{\top}, \boldsymbol{A}_2'^{\top})$ and vice-versa in our example, we obtain two different under-approximations of the intersection (see Figure 10 (b) and (c)). Their convex hull (obtained by the union of vertices) is still a sound under-approximation of the exact intersection and more precise than the individual under-approximations. This is illustrated in Figure 10, where the exact intersection (blue in (d)) of the two $\mathcal{H}$-representations (grey in (a)) is recovered despite the union of the input $\mathcal{V}$-representations (green and red in (a)) not covering it. This is due to the synergy between PDD and PDDM: the under-approximate $\mathcal{V}$-representation of the first polytope is intersected with the exact $\mathcal{H}$-representation of the second one and vice versa. We see the same behaviour in Figure 7, where both uni-directional intersections (4th column) are under-approximations, but their union is exact (5th column).

Empirically we find that this is crucial to minimize the precision loss due to using approximations. Further, the intersection results are exact for small dimensions $d \leq 4$ of cones (see Theorem 4.4).

4.3 Enforcing A-Irredundancy

Despite using batch intersection, the number of vertices can grow quickly when computing multiple convex hulls sequentially in the Split-Bound-Lift Method. 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 does not lose precision and can be computed by retaining only rays with rank $d-1$ (which can be cheaply computed using the incidence matrix $\mathcal{I}$). 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 irredundancy definition could remove all rays. Enforcing the second definition is expensive to compute in the absence of a full set of extremal rays, as the full convex hull problem has to be solved to assess the removal af a ray.

Therefore, we propose *A-irredundancy* 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. Formally and using the partial ordering defined above, we require for an A-irredundant PDD:

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

Any ray fulfilling a subset (including the same) constraints with equality as another ray, is removed until the above definition is satisfied 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. Intuitively, this enforces that no two rays lie in the interior of the same face of the polyhedron.

We illustrate the effect of enforcing A-irredundancy once in Figure 10 where we use it to obtain the polytope 10 (e) from 10 (d) and see that all extremal rays are retained and no precision is lost. In Figure 9 we apply it to polytope 9 (c) where the PDD misses one extremal vertex to obtain 9 (d) and see that here the resulting reduction in generator set size can come at the cost of a precision loss. Enforcing A-irredundancy in the 6th column of Figure 7 (removing the red vertices), recovers the minimal set of extremal rays. Note that for rays of equal incidence there are multiple possibilities which to retain, as is illustrated in Figure 6 (e) and (f).

4.4 Conversion to Primal

Translating the A-irredundant PDD obtained as described above, back to primal space concludes the PDDM and yields the (generally) approximate convex hull of $\mathcal{P}_1$ and $\mathcal{P}_2$ illustrated in the 7th column of Figure 7.

4.5 Formal Guarantees

In this subsection, we first show that the PDDM is sound and exact in low dimensions, before analysing its worst-case complexity.

Soundness guarantee. Computing a sound over-approximation of the convex hull of two polytopes in primal space, by inclusion-inversion, is equivalent to computing a sound under-approximation of the intersection of their dual space representations. Since the primal-dual conversion employed in the PDDM is exact, a sound under-approximation of the intersection of two polytopes in PDD in dual space implies overall soundness. Enforcing A-irredundancy on a polytope $\mathcal{P}$ to yield Q can only remove generators, yielding $Q \subseteq \mathcal{P}$. It follows directly that Q is a sound under-approximation, if $\mathcal{P}$ is. If both polytopes $\mathcal{P}'_q$ and $\mathcal{P}'_p$ 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 from the soundness of the batch intersection step:

Algorithm 1: Batch Intersection

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

Theorem 4.1. *The batch intersection $\mathcal{P}'_p = (\boldsymbol{A}', \mathcal{R}'_p)$ of a polytope $\mathcal{P}$ in PDD $(\boldsymbol{A}_p, \mathcal{R}_p)$ with the exact constraints $\boldsymbol{A}_q$ of a polytope Q computed as described above and detailed in Algorithm 1, is a sound under-approximation of the intersection of the two exact $\mathcal{H}$-representations $\boldsymbol{A}_p$ and $\boldsymbol{A}_q$:*

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

$$\left\{ \sum_{\boldsymbol{r}_i \in \mathcal{R}'_p} \lambda_i \boldsymbol{r}_i | \sum_i \lambda_i \leq 1, \lambda_i \in \mathbb{R}_0^+ \right\} \subseteq \{\boldsymbol{x} \in \mathbb{R}^d | \boldsymbol{A}_p\boldsymbol{x} \geq 0 \wedge \boldsymbol{A}_q\boldsymbol{x} \geq 0\}.$$

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 $\boldsymbol{r} \in \mathcal{R}'_p$ satisfies all constraints in $\boldsymbol{A}'$. Recall that $\mathcal{R}'_p$ is the union of three groups of vertices (see Section 4.2 or Algorithm 1):

- $\mathcal{R}_+$ 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 all constraint in $\boldsymbol{A}_q$) with the hyperplanes defined by $\boldsymbol{A}_q$. Since $\boldsymbol{r}_-$ lies outside Q while $\boldsymbol{r}_+$ lies inside, an intersection $\boldsymbol{r}_*$ is guaranteed to exist and lie between the two. By convexity of $\mathcal{P}$, $\boldsymbol{r}_*$ satisfies all constraints of $\boldsymbol{A}_p$. Further, since $\boldsymbol{r}_*$ is the first intersection of the ray with a constraint in $\boldsymbol{A}_q$ as seen from $\boldsymbol{r}_+$, which satisfies all constraints in $\boldsymbol{A}_q$, $\boldsymbol{r}_*$ also satisfies all constraints in $\boldsymbol{A}_q$.

Consequently, all vertices in the generating set $\mathcal{R}'_p$ satisfy all constraints of both $\mathcal{P}$ and Q. It follows that $\mathcal{R}'_p \subseteq Q \cap \mathcal{P}$ and hence that the generated polytope is a sound under-approximation. □

Exactness guarantee. Further, we can show that for relatively low dimensional polyhedra in Double Description, as they are often encountered during the first step of lifting in the SBLM, the PDDM as described above is not only sound but actually exact. To this end, let us first show the following guarantee for the intersection of a cone in DD with a matrix of constraints:

Algorithm 2: PDDM Intersection

```
Result: Intersected polytope $(\boldsymbol{A}_p \cup \boldsymbol{A}_q, \mathcal{R}')$
Input: polytope $(\boldsymbol{A}_p, \mathcal{R}_p)$ and $(\boldsymbol{A}_q, \mathcal{R}_q)$
Compute $(\boldsymbol{A}', \mathcal{R}'_p) = (\boldsymbol{A}_p \cup \boldsymbol{A}_q, \mathcal{R}_p)$ with Alg.1
Compute $(\boldsymbol{A}', \mathcal{R}'_q) = (\boldsymbol{A}_p \cup \boldsymbol{A}_q, \mathcal{R}_q)$ with Alg.1
Construct new PDD $(\boldsymbol{A}', \mathcal{R}'_p \cup \mathcal{R}'_q)$
Make PDD A-irredundant
return PDD
```

Theorem 4.2. *Given a Double Description $(\boldsymbol{A}_p, \mathcal{R}_p)$ of a polyhedral cone and the constraint matrix $\boldsymbol{A}_q$, adding all constraints jointly as per Algorithm 1 is guaranteed to yield a double description $(\boldsymbol{A}_p \cup \boldsymbol{A}_q, \mathcal{R}'_p)$ enumerating* all *extremal rays r' of the $\boldsymbol{A}_p \cup \boldsymbol{A}_q$-induced cone with one of the following properties:*

(1) r' is extremal (rank $d-1$) in the $\boldsymbol{A}_p$-induced cone.
(2) r' is of rank $d-2$ in the $\boldsymbol{A}_p$-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 $\mathcal{R}_p$ not violating any constraint $a \in \boldsymbol{A}_q$
- $\mathcal{R}_*$: Rays discovered by ray-shooting

Since $(\boldsymbol{A}_p, \mathcal{R}_p)$ is a DD of the $\boldsymbol{A}_p$-induced cone it enumerates all extremal rays. If r' is extremal in both the $\boldsymbol{A}$-induced and the $\boldsymbol{A}_p \cup \boldsymbol{A}_q$-induced cones, it is included in $\mathcal{R}_p$ and does not violate any constraints. Therefore, it is included in the first group above and will be part of $\mathcal{R}'_p$, 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 $\boldsymbol{A}_p \cup \boldsymbol{A}_q$-induced cone and therefore have rank $d-1$, it necessarily intersects at least one constraint $\boldsymbol{a} \in \boldsymbol{A}_q$ and is extremal to the $\boldsymbol{A}_p \cup \boldsymbol{a}$-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 $\boldsymbol{a}$. 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. □

Using this 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 4.2 and Algorithm 2:

Theorem 4.3. *Given the double descriptions $(\boldsymbol{A}_p, \mathcal{R}_p)$ and $(\boldsymbol{A}_q, \mathcal{R}_q)$ of two polyhedral cones, their intersection computed as per Algorithm 2 is guaranteed to be a partial double description $(\boldsymbol{A}_p \cup \boldsymbol{A}_q, \mathcal{R}')$ enumerating* all *extremal rays r' of the $(\boldsymbol{A}_p \cup \boldsymbol{A}_q)$-induced cone with one of the following properties:*

(1) r' is extremal in the $\boldsymbol{A}_p$-induced cone.
(2) r' is extremal in the $\boldsymbol{A}_q$-induced cone.
(3) r' is of rank $d-2$ in the $\boldsymbol{A}_p$-induced cone.
(4) r' is of rank $d-2$ in the $\boldsymbol{A}_q$-induced cone.

Proof. The proof follows directly from applying Lemma 4.2 to both applications of Algorithm 1, the insight that every extremal ray discovered by either will be included in the final generating set $\mathcal{R}'$ and the observation that the intersection of the exact $\mathcal{H}$-representations, trivially is the union of their respective constraints, leading to a valid partial double description. □

Using these results, we can in turn proof that the intersection of two polyhedral cones of up to dimension 4 in DD using the approach described above is exact:

Theorem 4.4. *Given the Double Descriptions $(\boldsymbol{A}_p, \mathcal{R}_p)$ and $(\boldsymbol{A}_q, \mathcal{R}_q)$ of two polyhedral cones $\mathcal{P}$ and $\mathcal{Q}$ of dimension $d \leq 4$, the PDD of their intersection $(\boldsymbol{A}_p \cup \boldsymbol{A}_q, \mathcal{R}')$ computed as described above and detailed in Algorithm 2 is an exact 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 $(\boldsymbol{A}_p \cup \boldsymbol{A}_q)$-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 $\boldsymbol{A}_p \cup \boldsymbol{A}_q$ with equality. This leads to the following four exhaustive options:

(1) all 3 constraints are part of $\boldsymbol{A}_p$, $\boldsymbol{r}^*$ is extremal in $\mathcal{P}_p$,
(2) all 3 constraints are part of $\boldsymbol{A}_q$, $\boldsymbol{r}^*$ is extremal in $\mathcal{P}_q$,
(3) 2 constraints are part of $\boldsymbol{A}_p$ and 1 of $\boldsymbol{A}_q$, $\boldsymbol{r}^*$ is of rank $d - 2 = 2$ in $\mathcal{P}_p$,
(4) 2 constraints are part of $\boldsymbol{A}_q$ and 1 of $\boldsymbol{A}_p$, $\boldsymbol{r}^*$ is of rank $d - 2 = 2$ in $\mathcal{P}_q$.

All of those are enumerated by Algorithm 4.3. Hence, $\mathcal{R}'$ will include all extremal rays of the $(\boldsymbol{A}_p \cup \boldsymbol{A}_q)$-induced cone. In this case A-irredundancy is equivalent to irredundancy. □

Complexity analysis. Finally, we can show that computing an over-approximation of the convex hull of two d-dimensional, bounded polytopes in PDD using the PDDM has polynomial complexity:

Theorem 4.5. *Given the PDD of two d-dimensional, bounded polytopes with a $\mathcal{V}$-representation of at most n_v vertices and an $\mathcal{H}$-representation of at most n_a constraints, computing a sound over-approximation of their convex hull using the PDDM as described above and detailed in Algorithm 2 has a worst-case time complexity of $O(n_v \cdot n_a^4 + n_a^2 \log(n_a^2))$.*

Proof. The PDDM can be broken down into its six components illustrated in Figure 7:

(1) Conversion from primal to dual representation (Section 4.1)
(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 4.2 or first half of Algorithm 1)
(3) Discovery of new vertices via ray-shooting (Section 4.2 or second half of Algorithm 1)
(4) Combining the vertices of the two intersection directions (Section 4.2 or Algorithm 2)
(5) Enforcing of A-irredundancy (Section 4.3 or Algorithm 2)
(6) Conversion from dual to primal representation (Section 4.1)

Primal-dual conversions and combining of vertices can be computed in constant time, as this only involves computing the transpose and concatenation which can be done implicitly 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, of two d-dimensional, bounded polytopes which in dual-space are defined by $\mathcal{P} = (\boldsymbol{A}_p, \mathcal{R}_p)$ and $\mathcal{Q} = (\boldsymbol{A}_q, \mathcal{R}_q)$. For convenience's sake, we assume the number of vertices to be $n_v = \max(|\mathcal{R}_p|, |\mathcal{R}_q|)$ and number of constraints $n_a = \max(|\boldsymbol{A}_p|, |\boldsymbol{A}_q|)$. Note that their roles are reversed compared to a primal space representation.

Adding constraints and separating vertices. Recall that in dual space we compute the intersection of the two polytopes $\mathcal{P}$ and $\mathcal{Q}$. The first step of intersecting $\mathcal{P}$ with $\mathcal{Q}$ is to split all points in $\mathcal{R}_p$ into the three groups $\mathcal{R}_+$, $\mathcal{R}_0$, and $\mathcal{R}_-$ defined in Section 4.2 depending on whether the lie inside, on the border of or outside the polytope defined by $\boldsymbol{A}_q$ as per the first half of Algorithm 1. This requires (at worst) evaluating $\boldsymbol{a}_i \boldsymbol{r}_j - b_i \{>, =, <\} 0$ for all $\boldsymbol{r}_j \in \mathcal{R}_p$ and $\boldsymbol{a}_i, b_i \in \boldsymbol{A}_q$. Where the addition and comparison 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 incidence matrix columns corresponding to the new constraints are added and populated without any extra computation with 0s for the vertices in $\mathcal{R}_+$ and 1s 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 $\mathcal{P}$ lying inside Q, $\boldsymbol{r}_+ \in \mathcal{R}_+$, to all vertices lying outside Q, $\boldsymbol{r}_- \in \mathcal{R}_-$, and all constraints in $\boldsymbol{A}_q$ are computed. At worst there are no vertices in group $\mathcal{R}_0$ and all vertices are spread equally between $\mathcal{R}_+$ and $\mathcal{R}_-$, leading to $n_v^2/4$ 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 $O(d \cdot n_a \cdot n_v^2)$. Note that this adds new incidence matrix rows corresponding to the new vertices $\mathcal{R}_*$, which can then be populated with the row obtained by the elementwise *and* of the two vertices generating the ray and a 1 in the column associated with the constraint of the first intersection which is linear $O(n_v)$ and dominated by the previous term.

Enforcing A-irredundancy. The intermediate state prior to enforcing A-irredundancy contains at most $n = 2(n_v + n_v^2/4)$ vertices, consisting of the at most n_v vertices in $\mathcal{R}_+$ and the at most $n_v^2/4$ 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 $(n^2 - n)/2$ 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 $O(n_v \cdot n_a^4 + n_a^2 \log(n_a^2))$. □

5 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 $\text{conv}(\{(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x})) \mid \boldsymbol{x} \in \mathcal{P} \subseteq [l_x, u_x]^k\})$ for a group of k neurons and their activation functions $\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1), ..., f_k(x_k)]^\top$, assuming that their inputs are constrained by the polytope $\mathcal{P}$.

Algorithm 3: Split-Bound-Lift Method (SBLM)

```
Input: Variable ordering $\mathcal{I}$, input polytope $\mathcal{P}$, 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 $\mathcal{D}^i, \mathcal{B}^i$ in $\mathcal{D}, \mathcal{B}$ do
        Split region: $\mathcal{P}_i = \mathcal{P} \cap \mathcal{D}^i$
        Apply SBLM: $\mathcal{K}_i \leftarrow \texttt{SBLM}(\mathcal{I}_{1:end}, \mathcal{P}_i, \mathcal{D}, \mathcal{B})$
        Extend into space including $y$: $\mathcal{K}_i \leftarrow \mathcal{K}_i \times \mathbb{R}$
        Apply bounds $\mathcal{B}^i$: $\mathcal{K}_i \leftarrow \mathcal{K}_i \cap \mathcal{B}^i$
    Compute convex hull: $\mathcal{K} = \texttt{PDDM}(\{\mathcal{K}_i\}_i)$
    return $\mathcal{K}$
else
    return $\mathcal{P}$
```

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 corresponding to the (relaxed) activations. 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*. This minimizes the dimensionality in which we have to compute the convex hulls. We formalize this in Algorithm 3 and explain both splitting and lifting below after stating the prerequisites for the SBLM.

5.1 Prerequisites

For simplicities' sake, we assume just one type of activation function $f\colon \mathbb{D} \longrightarrow \mathbb{R}$, with domain $\mathbb{D}$, is to be bounded. Now the SBLM requires a set of intervals $\mathcal{D}^i$ (e.g., $x_j \leq 0, x_j \geq 0$ for ReLU), covering the domain $\mathbb{D}$ (e.g., $\mathbb{R}$ for ReLU), and a pair of tight linear constraints $\mathcal{B}^i$ upper and lower bounding the function output (e.g., $y_j \leq 0$ and $y_j \geq 0$, and $y_j \leq x_j$ and $y_j \geq x_j$, respectively, for ReLU) on each of the intervals obtained by intersecting the interval $[l_x, u_x]_i$ defined by the neuron-wise bounds with the intervals $\mathcal{D}^i$. More formally, we require the intervals

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

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

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

to be provided to instantiate SBLM and by extension PRIMA. We note that the bounds c_i and d_i of the bounding regions can depend on the concrete input bounds l_x and u_x and the slope a and intercept b of $a_i^{\{\leq,\geq\}}$ can in turn depend on the corresponding concrete interval bounds $[\max(l_x, c_i), \min(u_x, d_i)]$.

Generalization. While we focus on the univariate case using only two bounding regions $\mathcal{D}^1$ and $\mathcal{D}^2$ in the following, SBLM and by extension PRIMA can be generalized to allow for neuron groups combining different multivariate activation functions $f\colon \mathbb{D} \subseteq \mathbb{R}^d \longrightarrow \mathbb{R}$. Further, more than one upper- and lower-bound $\mathcal{B}^i$ per bounding region can be provided and $\mathcal{D}^i$ can be specified as polyhedral regions instead of as intervals, as long as their union covers the domain $\mathbb{D} \subseteq \bigcup_i \mathcal{D}^i$ of the individual functions f.

5.2 Splitting the Input Polytope

To apply the bounds $\mathcal{B}^i$, the input polytope $\mathcal{P}$ has to be split 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 $\mathcal{D}^i$, that is all combinations of neuron-wise bounding regions for the group of k neurons. 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 on an abstract level, we will explain one case assuming the parent polytope $\mathcal{P}_1$, the output variable $y_j = f(x_j)$, and the corresponding bounding regions $\mathcal{D}_j^1 = \{\boldsymbol{x} \in \mathbb{R}^k \,|\, x_j \geq c_1\}$ and $\mathcal{D}_j^2 = \{\boldsymbol{x} \in \mathbb{R}^k \,|\, x_j \leq d_2\}$. We compute the children nodes by intersecting $\mathcal{P}_1$ with $\mathcal{D}_j^1$ and $\mathcal{D}_j^2$ to obtain $\mathcal{P}_{1,1} = \mathcal{P}_1 \cap \mathcal{D}_j^1$ and $\mathcal{P}_{1,2} = \mathcal{P}_1 \cap \mathcal{D}_j^2$. Starting with $\mathcal{P}$ at the root and recursively applying this splitting rule for every $y_j \in \mathcal{I}$, generates a polytope tree, which we call the decomposition tree, with 2^k leaf polytopes $\mathcal{P}_{\{1,2\}^k}$, which we call *quadrants*. This is illustrated in the blue portion of the central panel in Figure 4, where $\mathcal{D}^1$ and $\mathcal{D}^2$ are $\mathbb{R}_0^+$ and $\mathbb{R}_0^-$, respectively.

5.3 Lifting

We now extend these quadrants $\mathcal{P}_{\{1,2\}^k}$ to the output space and bound them using the corresponding constraints on the activation function $\mathcal{B}_j^i$, before taking their convex hull. This yields a polytope $\mathcal{K}$, jointly constraining the inputs and outputs of a neuron group. The constraints of its $\mathcal{H}$-representation form the desired k-neuron abstraction. We call this process *lifting* and propose a recursive approach: We lift sibling polytopes on the decomposition tree until only the desired polytope $\mathcal{K}$ remains.

Again, we explain a single step of lifting, as they are equivalent. We assume the sibling polytopes $\mathcal{K}_{1,1}$ and $\mathcal{K}_{1,2}$, corresponding to $\mathcal{P}_{1,1}$ and $\mathcal{P}_{1,2}$ in the decomposition tree, with the associated input- and output-variables x_j and y_j, respectively, and the pairs of bounds $\mathcal{B}_j^1$ and $\mathcal{B}_j^2$ instantiated for y_j. A single step consist of three parts:

- extending $\mathcal{K}_{1,1}$ and $\mathcal{K}_{1,2}$ by the output variable y_j,
- bounding y_j on the extended polytopes, by intersecting them with the constraints $\mathcal{B}_j^1$ and $\mathcal{B}_j^2$ to obtain $\mathcal{K}'_{1,1}$ and $\mathcal{K}'_{1,2}$,
- computing their (approximate) convex hull using the PDDM: $\mathcal{K}_1 = \text{conv}(\mathcal{K}'_{1,1}, \mathcal{K}'_{1,2})$.

Applying this lifting rule recursively to the decomposition tree starting with $\mathcal{K}_{\{1,2\}^k} = \mathcal{P}_{\{1,2\}^k}$, 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 is illustrated in the right portion of the central panel in Figure 4. The decompositional approach has two benefits: Precision – computing approximate convex hulls via the PDDM is exact for polytopes of dimension up to 3 and starts to lose precision only slowly as dimensionality increases. Directly computing $2k$-dimensional convex hulls with PDDM will therefore lose more precision than using our decomposed method. Speed – a lower-dimensional polytope with fewer constraints and generally also fewer vertices significantly reduces the runtime for the individual convex hull operations. In fact, computing the convex hulls for the approximation of non-piecewise-linear functions directly in the input-output space is intractable even for groups of only size $k = 3$, as the number of vertices increases exponentially with k during the extension and bounding process in that case.

5.4 Instantiation for Various Functions

We instantiate SBLM for common network functions next.

ReLU. We can capture all univariate, piecewise-linear functions, such as ReLU, exactly on the intervals $\mathcal{D}^i$ where they are linear. Further, if the neuron-wise bounds $[l_x, u_x]$ only contain one such linear region, the neuron behaves linearly, can be encoded exactly and is excluded from the k-neuron abstraction. Therefore, we consider $y = max(x, 0)$ with $x \in [l_x, u_x]$ for $l_x < 0 < u_x$. We choose $\mathcal{D}^1 = [-\infty, 0]$ and $\mathcal{D}^2 = [0, \infty]$, with $\mathcal{B}^1 = (y \geq 0,\ y \leq 0)$ and $\mathcal{B}^2 = (y \geq x,\ y \leq x)$, obtaining exact bounds on both intervals.

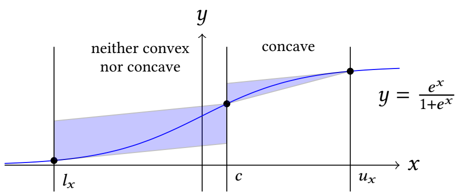


Fig. 11. Interval-wise bounds for the Sigmoid function on the intervals $[l_x, c]$ and $[c, u_x]$.

Tanh and Sigmoid. Let f be an S-curve function with domain $[l_x, u_x]$, that is $f''(x) \geq 0$ for $x \leq 0$, $f''(x) \leq 0$ for $x \geq 0$ and $f'(x) > 0$ for $x \in [l_x, u_x]$. Both Sigmoid $\sigma(x) = \frac{e^x}{e^x+1}$ and Tanh $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ have these properties. We split the domain at $c \in [l_x, u_x]$ into $\mathcal{D}^1 = [-\infty, c]$ and $\mathcal{D}^2 = [c, \infty]$, choosing c to minimize the area between upper and lower bound in the input-output plane, using the bounds from Singh et al. [2019b]:

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

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

where we denote the lower bound of the intersection $\mathcal{D}^i \cap [l_x, u_x]$ as l_d and the upper one as u_d. We show these bounds in Figure 11 for the Sigmoid function and, for illustration purposes, a non-optimal c. In practice, we choose c to minimize the area of the abstraction of a single neuron in the input-output plane.

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 polyhedral bounding regions $\mathcal{D}^i = \{\boldsymbol{x} \in \mathbb{R}^d | x_i \geq x_j,\ 1 \leq j \leq d,\ i \neq j\}_i$, separating the domain into the d regions where one variable dominates all others (illustrated for $d = 2$ in Figure 12). 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.

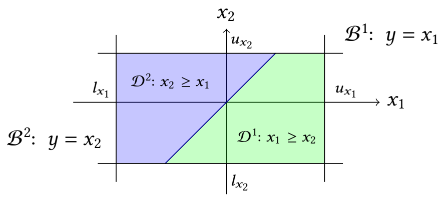

x_2
u_{x_2}
$\mathcal{B}^1$: $y = x_1$
l_{x_1}
$\mathcal{D}^2$: $x_2 \geq x_1$
u_{x_1}
x_1
$\mathcal{B}^2$: $y = x_2$
$\mathcal{D}^1$: $x_1 \geq x_2$
l_{x_2}

Fig. 12. Polyhedral bounding regions $\mathcal{D}^i$ and corresponding bounds $\mathcal{B}^i$ for the 2d MaxPool function on the input region $[l_{x_1}, u_{x_1}] \times [l_{x_2}, u_{x_2}]$.

6 PRIMA VERIFICATION FRAMEWORK

Prima is based on three high-level steps: (i) accumulate a set of constraints encoding a (convex) abstraction of the network for a given pre-condition (as discussed so far), (ii) define a linear optimization objective representing the post-condition, and (iii) use an LP or MILP solver to derive a bound on this optimization objective. If this bound exceeds a threshold depending on the post-condition, certification succeeds, otherwise, if the optimal solution violates this bound, it could be a true counterexample or a false positive due to approximation. Hence, we evaluate any such possible counterexample with the concrete network to determine whether it is a true counterexample.

While all affine layers are encoded exactly, two considerations have to be balanced when encoding non-linear activation layers with Prima: more precise encodings (e.g., considering more or larger neuron groups) improve the optimal bound of the optimization problem, but the increased number of constraints can make this problem impractical to solve. We navigate this trade-off by leveraging abstraction refinement – using increasingly more precise but also more costly methods until we are able to either decide a property (verify or falsify) or reach a timeout.

6.1 Abstraction Refinement Approaches

Fundamentally, we can refine our abstraction in three ways: (i) compute tighter abstractions of the group-wise inputs, (ii) compute tighter layer-wise multi-neuron constraints for the given input abstraction from (i), and (iii) encode part of the network using an exact MILP encoding.

Input bound refinement. Since SBLM and PDDM abstract a group of neurons for a given polyhedral input region, the tightness of the resulting constraints depends directly on the tightness of the input abstraction. These are computed using a fast, incomplete verifier (e.g., [Müller et al. 2021; Singh et al. 2019b; Xu et al. 2020a]) based on single-neuron abstractions and can be tightened significantly by computing more precise neuron-wise bounds [Singh et al. 2019c] using an LP or MILP encoding.

Tighten multi-neuron constraints. The layer-wise tightness of our multi-neuron constraints depends on (i) the tightness of the group-wise constraints, mostly determined by the quality of the input region, and (ii) on capturing the important neuron-interdependencies with the chosen groups. Using larger neuron groups (increasing k) and considering more groupings by allowing more overlap (increasing s) and partitioning the neurons into fewer sets before grouping (increasing n_s), allows capturing more and more complex interactions. While the constraints themselves can be computed quickly, the resulting LP problems become harder to solve.

Network encoding. PRIMA encodes non-linear activations in four different ways: (i) exact encoding via equality constraints for stable (those exhibiting linear behavior) piecewise-linear activations, (ii) single-neuron constraints, (iii) multi-neuron constraints computed via SBLM and PDDM, and (iv) exact (for piecewise-linear functions) MILP encodings. While stable activations are always encoded exactly and all unstable activations are encoded using both the single- and multi-neuron constraints, we only selectively use a MILP encoding on the (typically relatively narrow) last layers of convolutional networks due to their large computational cost.

6.2 Abstraction Refinement Cascade

PRIMA leverages our multi-neuron constraints as part of an abstraction refinement cascade using increasingly more precise and expensive approaches: We first attempt verification using single-neuron constraints via DEEPPOLY [Singh et al. 2019b] or GPUPOLY [Müller et al. 2021]. If this fails, we encode all activation layers using our multi-neuron constraints and solve the resulting LP. If this also fails, we attempt to decide the property by tightening the multi-neuron constraints Section 6.1, encoding the final network layer(s) using MILP, and refining individual neuron bounds.

7 EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness of PRIMA and show that it significantly improves over 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. Further, we show that PRIMA can scale to real-world problems, obtaining tight bounds in an autonomous driving steering-angle-prediction task. Finally, we demonstrate the effectiveness and benefits of computing relaxations with SBLM and PDDM compared to directly using the exact convex hull.

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	48 064	6	ReLU
CIFAR10	`ConvSmall`	Conv	4 852	3	ReLU
	`CNN-A-Mix`	Conv	6 244	3	ReLU
	`CNN-B-Adv`	Conv	16 634	3	ReLU
	`ConvBig`	Conv	62 464	6	ReLU
	`ResNet`	Residual	107 496	10	ReLU
Self-Driving	`DAVE`	Conv	107 032	8	ReLU + Tanh

7.1 Experimental Setup

The neural network certification benchmarks for fully connected networks were run on a 20 core 2.20GHz Intel Xeon Silver 4114 CPU with 100 GB of main memory and those for convolutional networks on a 16 Core 3.6GHz Intel i9-9900K with 64GB of main memory and an NVIDIA RTX 2080Ti. We use Gurobi 9.0 for solving MILP and LP problems [Gurobi Optimization, LLC 2018].

7.2 Benchmarks

We evaluate PRIMA on a wide range of networks based on ReLU, Tanh, and Sigmoid activations:

- The set of fully-connected and convolutional ReLU networks[5] from [Singh et al. 2019a] trained using DiffAI [Mirman et al. 2018], PGD [Madry et al. 2017], Wong [Wong et al. 2018], and natural training (see results on MNIST and CIFAR10 in Table 2).

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

Table 2. Number of verified adversarial regions of the first 1 000 samples and runtime for Prima, OptC2V [Tjandraatmadja et al. 2020], and κPoly [Singh et al. 2019a]. Natural (NOR), adversarial (PGD [Madry et al. 2017]), or provable (DiffAI [Mirman et al. 2018], Wong [Wong et al. 2018]) training was used.

Dataset	Model	Training	Accuracy	ϵ	n_s	κPoly		OptC2V †		Prima (ours)		# Upper Bound
						# Ver	Time	# Ver	Time	# Ver	Time	
MNIST	5×100	NOR	960	0.026	100	441	307	429	137	**510**	159	842
	8×100	NOR	947	0.026	100	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	**612**	395	911
	`ConvSmall`	NOR	980	0.120	100	347	477	436	55	**640**	51	733
	`ConvBig`	DiffAI	929	0.300	100	736	40	771	102	**775**	5.5	790
CIFAR10	`ConvSmall`	PGD	630	2/255	100	399	86	398	105	**458**	16	481
	`ConvBig`	PGD	631	2/255	100	459	346	n/a†	n/a†	**482**	128	550
	`ResNet`	Wong	290	8/255	50	245	91	n/a†	n/a†	**248**	1.9	248

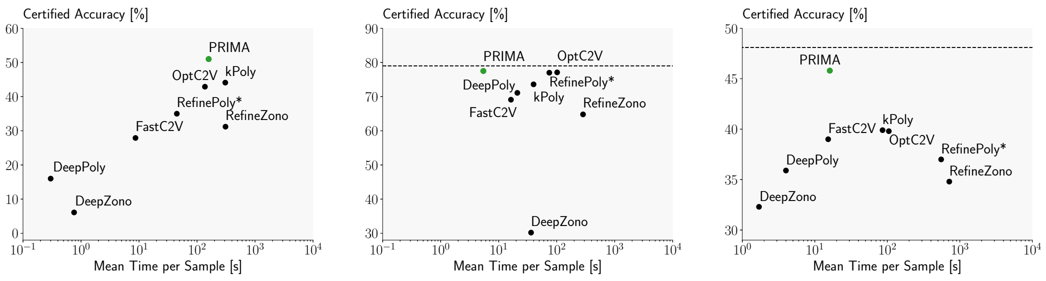
†The OptC2V [Tjandraatmadja et al. 2020] code has not been released; we report their runtimes and results where available.

- The published set of CIFAR10 convolutional networks from [Dathathri et al. 2020], trained using either just PGD or a mix of standard and PGD training (see results on CIFAR10 in Table 3).
- The set of fully-connected and convolutional Tanh and Sigmoid networks from [Singh et al. 2019a] trained using natural training (see results on MNIST in Table 5).
- The NVIDIA self-driving car network architecture `DAVE` [Bojarski et al. 2016] trained on a steering angle prediction task using the Udacity self-driving car dataset [Udacity 2016] with 31 834 train and 1 974 test samples[6], an input resolution of $3 \times 66 \times 200$, and PGD [Madry et al. 2017] training (see results in Table 6).

While we evaluate performance for the widely considered and challenging ℓ_∞ perturbations[7], Prima can also be applied to other specifications including individual fairness [Ruoss et al. 2020b], global safety properties [Katz et al. 2017], acoustic [Ryou et al. 2020], geometric [Balunović et al. 2019], and spatial [Ruoss et al. 2020a] based perturbations.

For classification tasks and ReLU networks, we compare Prima with a range of state-of-the-art incomplete verifiers notably also the ReLU-specialized κPoly [Singh et al. 2019a], OptC2V [Tjandraatmadja et al. 2020], and additionally the highly optimized and fully GPU-based β-Crown [Wang et al. 2021] (in incomplete mode). For classification using Tanh and Sigmoid activations, fewer verifiers are available and thus we compare with the state-of-the-art incomplete verifier DeepPoly [Singh et al. 2019b]. Few verification methods consider the regression setting and to the best of our knowledge, we are the first to analyze the full-size `DAVE` network. Neurify [Wang et al. 2018] analyses a heavily scaled-down version in a binary classification setting, but in complete mode it does not scale to the much larger networks analysed here. In incomplete mode, it uses the same bounds as DeepZono [Singh et al. 2018] and is less precise than GPUPoly [Müller et al. 2020] to which we compare. β-Crown does not support regression tasks and while an extension might be possible, it is non-trivial. It is also unclear if the approach scales to networks of this size.

[6]The labels of the original test set are not available (anymore), so we used videos 1, 2, 5, and 6 as train and video 4 (instead of 3) as test dataset.
[7]That is, $y := c(\boldsymbol{x})_i = c(\boldsymbol{x}'), \forall \boldsymbol{x}' \in \mathbb{B}_\epsilon^\infty := \{\boldsymbol{x} \in \mathcal{X} \mid ||\boldsymbol{x} - \boldsymbol{x}'||_\infty \leq \epsilon\} \Leftrightarrow \min_{\boldsymbol{x}' \in \mathbb{B}_\epsilon^\infty} \boldsymbol{h}(\boldsymbol{x}')_y - \boldsymbol{h}(\boldsymbol{x}')_i > 0, \forall i \neq y$



(a) MNIST 5 × 100, $\epsilon = 0.026$ (b) MNIST `ConvBig`, $\epsilon = 0.3$ (c) CIFAR10 `ConvSmall`, $\epsilon = 2/255$

Fig. 13. Comparison of the runtime/accuracy trade-off of Prima (ours), OptC2V [Tjandraatmadja et al. 2020], FastC2V [Tjandraatmadja et al. 2020], kPoly [Singh et al. 2019a], RefinePoly [Singh et al. 2019b], DeepPoly [Singh et al. 2019c] (equivalent bounds to Crown [Zhang et al. 2018] and CNN-Cert [Boopathy et al. 2019]), RefineZono [Singh et al. 2019c] and DeepZono [Singh et al. 2018] (equivalent bounds to Fast-Lin [Weng et al. 2018] and Neurify [Wang et al. 2018] in incomplete mode), evaluated on the first 1000 samples (100 for RefinePoly) of the corresponding test sets. The tightest known upper bound to the certifiable accuracy is shown as dashed line. Higher and further left is better.

For our experiments, we use the setup outlined in Section 6 which is similar to kPoly in [Singh et al. 2019a]. We use DeepPoly or GPUPoly (for convolutional networks) to determine the octahedral input bounds required to compute the multi-neuron constraints with Prima. For fully-connected networks, we refine the neuron-wise bounds of unstable neurons using the MILP encoding from Tjeng et al. [2017] for the second activation layer (the first layer bounds are already exact) and an LP encoding for the remaining layers. We note that encoding more layers with MILP does not scale on these networks. For convolutional networks, we encode some of the neurons in the last one or two layers using the MILP encoding from [Tjeng et al. 2017]. We note that the concurrent bound optimization in β-Crown corresponds to simultaneous bound-refinement on all neurons of all layers, which is orthogonal to our approach and a promising direction to be explored in future work (though intractable without a GPU-based LP solver). We report as *Accuracy* the number of correctly classified samples out of the considered test set, as *# Upper Bound* the number of properties that could not be falsified and hence form an upper bound to the number of certifiable properties, as *# Ver* the number of verified regions, and as *Time* the average runtime per correctly classified sample in seconds.

7.3 Image Classification with ReLU Activation

We compare Prima against the state-of the art methods kPoly and OptC2V in Table 2 and β-Crown in Table 3. Computing multi-neuron constraints for groups of $k = 4$ ReLU neurons becomes feasible with SBLM and PDDM reducing the time per group from several minutes, when directly computing exact convex hulls as in kPoly, to less than 50 milliseconds. Nevertheless, we find empirically that the best strategy to leverage this speed-up is to evaluate a large variety of small groups. Unless reported differently, we consider overlapping groups of size $k = 3$ with $n_s = 100$.

Comparison with the state-of-the-art. Figure 13 shows scatter plots comparing the runtime and precision of Prima with those of other state-of-the-art verifiers on the robustness certification of a normally trained 5 × 100 MLP, a provably trained `ConvBig` (MNIST) and an adversarially trained `ConvSmall` (CIFAR10). We note that adversarially and provably trained networks sacrifice accuracy for ease of certification, making normally trained networks more relevant and challenging. Here, fast, purely propagation-based, incomplete verifiers like DeepPoly verify only about 16% of the images. In contrast, Prima verifies 51% in < 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 310 and 140 seconds, respectively. Based on these observations, we compare Prima with kPoly and OptC2V on the remaining benchmarks from [Singh et al. 2019a].

Comparison with kPoly and OptC2V. For all normally trained networks, Prima is significantly more accurate than both kPoly [Singh et al. 2019a] and OptC2V [Tjandraatmadja et al. 2020], verifying between 44 and 201 more regions than the better of the two while sometimes also being significantly faster. These results are summarized in Table 2. For the, comparatively easy to verify (as can be seen in Figure 13(b)), DiffAI trained ConvBig MNIST network, we gain less precision verifying only 4 more regions than OptC2V. However, the easier proofs come at the cost of reduced accuracy, making them less relevant for real-world applications. For both PGD-trained CIFAR10 networks, Prima verifies between 23 and 59 more regions than kPoly and OptC2V while being around four times faster. On the provably trained ResNet, Prima is 50x faster than kPoly and able to decide all properties. However, this network is so heavily regularized that even complete verification via a MILP encoding is tractable. In summary, Prima is usually faster than kPoly and OptC2V, especially on larger networks, and is always more precise, sometimes substantially so.

Table 3. Number of verified adversarial regions of the 100 random samples from the CIFAR10 test set evaluated by [Wang et al. 2021]. CNN-A-Mix is trained using a combination of adversarial and natural training and CNN-B-Adv only adversarially. Both are taken from [Dathathri et al. 2020].

Model	ϵ	Acc	β-Crown		Prima (ours)		# Bound
			# Ver	Time	# Ver	Time	
CNN-A-Mix	2/255	100	43	209	**57**	53	68
CNN-B-Adv	2/255	100	**46**	234	43	260	81

Comparison with β-Crown. β-Crown [Wang et al. 2021] is a highly optimized, fully GPU-based complete BaB [Morrison et al. 2016] solver, supporting only ReLU activations[8] and the classification setting. When comparing complete and incomplete verifiers on accuracy, it is crucial to ensure that similar runtimes were achieved, as complete verifiers can, given sufficient time, decide any property. The GPU-based LP solver underlying β-Crown is an orthogonal development to the Prima multi-neuron constraints. Prima currently uses a much slower CPU-based solver which is the main bottleneck for large networks as the runtime for computing multi-neuron constraints becomes small via our improved algorithms (see Section 7.8). We consider combining the GPU-based solver from β-Crown with our multi-neuron approximations as an interesting item for future work. Despite the discrepancy in LP-solver performance distorting the comparison, Prima is still significantly faster on CNN-A-Mix while also achieving notably higher precision. On the larger network CNN-B-Adv, where LP-solver performance is more dominant, β-Crown achieves slightly

Table 4. Evaluation of a range of parameters for grouping set size n_s, group size k, and overlap s, partial MILP refinement, and neuron-wise bound refinement for the first 100 samples of the MNIST test set and the normally trained 5×100. Of the first 100 samples, 99 are classified correctly and for 9 of those a counterexample is known.

n_s	k	s	Partial MILP		Refinement		# Ver	Time [s]
			# layers	# neurons	LP	MILP		
1	1	-	-	-	-	-	21	2.56
10	3	1	-	-	-	-	26	5.75
20	3	1	-	-	-	-	28	6.52
20	3	2	-	-	-	-	28	67.79
20	4	1	-	-	-	-	28	54.05
100	3	1	-	-	-	-	28	16.59
1	1	-	1	30	-	-	23	4.58
100	3	1	1	30	-	-	30	42.00
100	3	1	1	100	-	-	30	44.03
100	3	1	2	100	-	-	35	117.37
1	1	-	-	-	y	-	27	24.15
100	3	1	-	-	y	-	45	99.40
100	3	1	-	-	y	y	54	115.24
100	3	1	2	100	y	y	60	189.21

[8]Extensions to piecewise-linear activations with more than $m = 2$ linear regions would significantly increase runtime ($O(m^d)$ with split depth d), while precision would be significantly lower for non-piecewise linear activations.

higher precision and smaller runtime. Unfortunately, we could not run the public version of β-Crown without soundness issues on the networks from [Singh et al. 2019a] and consequently only compare on networks they provide. The recent SDP-based (semidefinite programming) SDP-FO [Dathathri et al. 2020] takes many hours per sample and is outperformed by β-Crown. Thus we do not compare to it directly.

7.4 Parameter Study

In Table 4, we compare the effect of different parameter combinations on runtime and accuracy for the 5×100 MLP, which allows also more expensive settings to be evaluated while still representing a challenging verification problem with $\epsilon = 0.026$. Using the single-neuron triangle relaxation ($k = 1$) only 21 regions can be verified. Adding our multi-neuron constraints with partition sizes of $n_s = 10$ and $n_s = 20$ increases this to 26 and 28 regions, respectively. Neither considering a larger overlap ($s = 2$), nor larger groups ($k = 4$), nor larger partition sizes ($n_s = 100$) can increase the number of verified regions, despite significantly increased the runtime. While using triangle relaxations with a partial MILP encoding is relatively fast it also only increases the accuracy to 23 regions. In contrast, combining a partial MILP encoding with multi-neuron constraints yields, depending on the exact setting, an almost 75% increase to 35 verified regions, although at the price of increased runtime. Refining the neuron-wise bounds using a triangle relaxation and LP encoding only improves the number of verified regions to 27, while additionally using multi-neuron constraints yields a significant jump to 45. This further improves to 54 when using MILP to refine the second layer bounds and 60 when additionally encoding the last two layers with MILP. The significant increase in precision when combining tight multi-neuron constraints computed via SBLM and PDDM with other methods demonstrates their utility and highlights the potential of our abstraction-refinement-based approach.

7.5 Effect of Grouping Strategy

We evaluate the sensitivity of Prima to the chosen neuron groupings, by comparing the performance[9] of random groups with those generated by our sparse grouping heuristic in Figure 14 for the first 100 test images of CIFAR10 and the `ConvSmall` network. Concretely, we first generate a deterministic sparse grouping with our heuristic for a group size of $k = 3$, a partition size of $n_s = 100$, and a maximum overlap of $s = 1$. Then we (randomly) reduce this grouping to a fraction r (x-axis in Figure 14) of the original number of groups. The random groupings are generated to have the same size (number of groups) by repeatedly drawing k indices uniformly at random and rejecting duplicates.

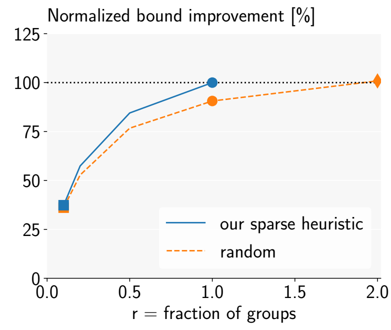


Fig. 14. Normalized bound improvement over the fraction of groups used to compute multi-neuron constraints, r. Our method is the blue circle, whose gain is normalized to 100%.

We observe that considering fewer groups from our heuristic (blue in Figure 14) reduces the bound improvement notably, e.g., to 37% at $r = 0.1$ (blue square). Choosing random groups (orange in Figure 14) is consistently worse (vertical gap in Figure 14); by around 10% at $r = 1.0$ (circles) closing to 3.4% at $r = 0.1$ (squares). While our heuristic generates groups with small overlap to evenly cover all neurons, random sampling can lead to some groups with large overlap, while potentially not covering some neurons at all, leading to worse performance.

[9]Concretely, we compare the obtained improvement of $\underline{h_{y,i}}$, the lower bound to the optimization objective $\min_{\boldsymbol{x}' \in \mathbb{B}_\epsilon^\infty} \boldsymbol{h}(\boldsymbol{x}')_y - \boldsymbol{h}(\boldsymbol{x}')_i$, over the triangle relaxation (Δ) normalized using our standard sparse heuristic (Prima): $(\underline{h_{y,i}} - \underline{h^{\Delta}_{y,i}})/(\underline{h^{\text{Prima}}_{y,i}} - \underline{h^{\Delta}_{y,i}})$

Considering fewer groups makes overlaps between groups less likely, making the groupings resulting from the two sampling strategies more similar and explaining the shrinking performance gap. To obtain the same precision with random groups as with our heuristic, about twice as many ($r = 2.0$, diamond) groups are needed (horizontal gap in Figure 14). We repeated these experiments several times with different random seeds and obtained consistent results.

Overall, we conclude that while our heuristic consistently outperforms random groups, PRIMA is relatively insensitive to the exact groupings, as long as sufficiently many are used.

7.6 Image Classification with 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 using direct exact convex hull computation, whereas SBLM using PDDM takes only 10 milliseconds. This dramatic speed-up is a result of SBLM's decompositional approach of solving the problem in lower dimensions (see Section 5), significantly reducing its complexity. Note that both methods compute only approximations of the optimal group-wise convex relaxation for these cases, as the underlying interval-wise bounds are not exact.

Table 5. Number of verified adversarial regions and runtime in seconds 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**	186.0
	6×200	98	0.002	39	0.6	**68**	170.0
	`ConvSmall`	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
	`ConvSmall`	99	0.014	30	0.5	**51**	47.0

We evaluate our method on normally trained, fully-connected and convolutional networks for the MNIST dataset. We choose an ϵ for the B_ϵ^∞ region such that the state-of-the-art verifier for Tanh and Sigmoid activations, DEEPPOLY, verifies less than 50% of the 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 [Boopathy et al. 2019] and CROWN [Zhang et al. 2018].

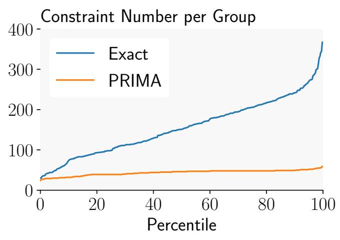
We use overlapping groups with $n_s = 10$ and again refine neuron-wise lower- and upper-bounds for fully-connected networks. We verify between 14% and 34% more regions than the current state-of-the-art, in some cases doubling the number of verified samples, while maintaining a reasonable runtime comparable to that for ReLU networks (see Table 5).

7.7 Autonomous Driving

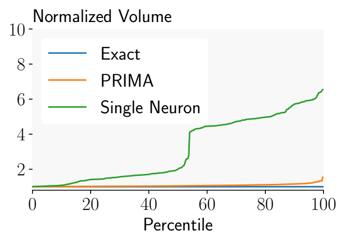
We evaluate PRIMA in the setting of autonomous driving, deriving upper and lower bounds to the predicted steering angle under an ℓ_∞ threat-model in a regression setting. We thereby demonstrate scalability to large networks (> 100k neurons and over 27 million connections) and inputs ($3 \times 66 \times 200$) of real-world relevance. We report the certified maximum absolute steering angle error and the width of reachable steering angles. We

Table 6. Standard (std.), empirically maximal (emp.) and certifiably maximal (cert.) mean absolute steering angle error (MAE) (smaller is better) for PRIMA vs. GPUPOLY evaluated on every 20^{th} sample and mean evaluation time.

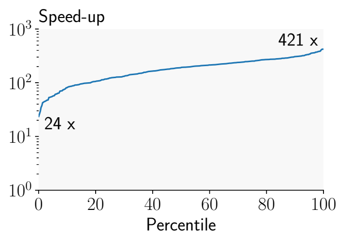
ϵ	Method	std. MAE	emp. MAE	cert. MAE	cert. Width	Time [s]
1/255	GPUPOLY	7.37°	9.41°	10.35°	5.75°	1.55
	PRIMA	7.37°	9.41°	**10.17°**	**5.30°**	154.2
2/255	GPUPOLY	7.37°	11.46°	18.35°	19.63°	2.41
	PRIMA	7.37°	11.46°	**17.05°**	**17.03°**	239.5



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

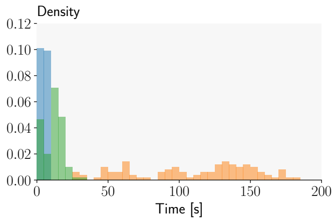


(b) Volume of PRIMA and single-neuron constraint polytopes compared to the exact convex hull.

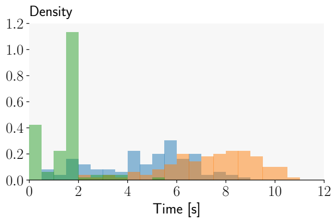


(c) Speedup of constraint computation using SBLM and PDD compared to an exact convex hull.

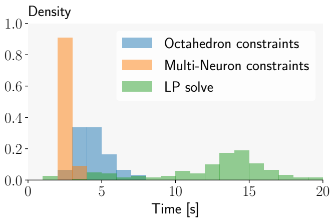
Fig. 16. Case study: Analysis of the distribution of the number of discovered constraints, abstraction volume, and runtime over all ($\approx$ 360) individual 3-neuron groups processed during the verification of a single MNIST image on the 5×100 ReLU network.



(a) Exact – MNIST 5×100



(b) SBLM + PDDM – MNIST 5×100



(c) SBLM + PDDM – CIFAR10 `ConvSmall`

Fig. 17. Comparison of the runtime contribution of the octahedral input constraint computation, multi-neuron constraint computation and LP solve.

use PGD [Madry et al. 2017] to compute empirical bounds (emp). We use the CNN architecture proposed by Bojarski et al. [2016] and adversarial training [Madry et al. 2017] on the Udacity autonomous driving dataset [Udacity 2016] to obtain the network evaluated here.

When the permissible perturbation size is small and the standard error of the model is larger than the perturbation effect, cheaper methods such as GPUPoly already yield good results. However, for larger perturbations, PRIMA reduces the gap between empirical and certified error around 20% (see Table 6). In Figure 15, we show two representative samples, where the certified steering angle range for $\epsilon = 2/255$ is shaded blue, the empirical bounds on the steering angle are shown in red, the target in green and the prediction on the unperturbed sample in blue. Qualitatively, we find that while the network often still performs well on unperturbed samples with poor lighting or contrast (see lower example in Figure 15) the sensitivity to perturbations and consequently the width of the reachable steering angle range is much larger than for samples in better conditions (see upper example in Figure 15).

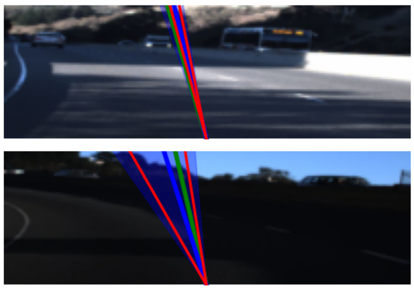


Fig. 15. Samples from the self-driving car dataset. The target steering angle is illustrated in green, the predicted one in blue. The empirical bounds for $\epsilon = 2/255$ are shown in red and the certified range is shaded blue.

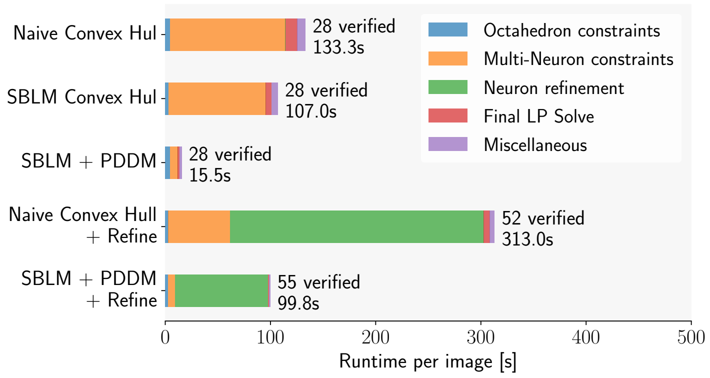


Fig. 18. Runtime comparison of using SBLM vs. exact convex hull for computing relaxations in PRIMA. Evaluated on 100 images and the MNIST 5×100 ReLU network.

7.8 Effectiveness of SBLM and PDDM for Convex Hull Computations

Computing approximations with SBLM using PDDM has two main advantages compared to the direct convex hull approach: It is significantly faster and produces fewer constraints, making the resulting LP easier to solve, while barely losing any precision.

For example, verifying the 5×100 network with PRIMA and comparing abstractions for groups of $k = 3$ computed with SBLM and PDDM or naively and neuron-wise triangle relaxation (Figure 16), we observe the following: Using SBLM and PDDM we reduce the mean number of constraints computed per neuron-group by over 70% from 156 to 44 significantly reducing the number of constraints in the resulting LP, as many hundred such neuron groups are considered. The mean volume of the constraint polytopes defined by these constraints in the 6-dimensional input-output space of the individual neuron groups, meanwhile, is only around 5% larger. Single neuron constraints, in contrast, yield 4-times larger volumes. Additionally, computing the approximate constraints is about 200 times faster than the exact convex hull.

Not only are PRIMA constraints faster to generate and allow the verification of the same properties, but a runtime analysis for the first 100 samples (illustrated in Figure 18) shows that they also speed up the final LP solve 8-fold compared to the naive approach, as significantly fewer constraints have to be considered. This effect is also observed in the time-intensive neuron-wise bound-refinement where PRIMA constraints reduce the runtime by 70% while allowing 3 additional regions to be verified. This can be explained by the fewer but more diverse PRIMA constraints also speeding up the final LP solve in the refinement step reducing the number of timeouts and allowing tighter neuron-wise bounds to be computed. Using neuron-refinement with PRIMA is in fact still quicker than the naive approach without any refinement, while almost verifying twice as many samples. SBLM combined with exact convex hulls computations already yields a small speed-up of around 20%, but the synergy with PDDM is key to unlock its full potential.

An analysis of the runtime contributions of the octahedral input constraint computation, the multi-neuron constraint computation and the final LP solve (illustrated in Figure 17), shows the following: Using the naive approach, the multi-neuron constraint computation clearly dominates the runtime, while only contributing around 50% when using SBLM and PDDM. For larger networks, the input constraint computation and LP-solve become more expensive, reducing the multi-neuron constraints computation runtime contribution further and further, e.g., 7% for the CIFAR10 `ConvSmall`, and shifting the performance bottleneck to the LP-solver, especially when neuron-wise bound-refinement or partial MILP encodings are used.

8 RELATED WORK

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

Complete methods are mostly based on satisfiability modulo theory (SMT) [Ehlers 2017; Huang et al. 2017; Katz et al. 2017, 2019] or the branch-and-bound approach [Anderson et al. 2020; Botoeva et al. 2020; Bunel et al. 2020b; Lu and Kumar 2020; Palma et al. 2021; Tjeng et al. 2017; Wang et al. 2021; Xu et al. 2020b], often implemented using mixed integer linear programming (MILP). These methods offer exactness guarantees but are based on solving NP-hard optimization problems, which can make them intractable even for small networks. Incomplete methods can be divided into bound propagation approaches [Gowal et al. 2019; Mirman et al. 2018; Müller et al. 2020; Singh et al. 2018, 2019b; Weng et al. 2018; Zhang et al. 2018] and those that generate polynomially-solvable optimization problems [Bunel et al. 2020a; Dathathri et al. 2020; Lyu et al. 2019; Raghunathan et al. 2018; Singh et al. 2019a; Tjandraatmadja et al. 2020; Xiang et al. 2018] such as linear programming (LP) or semidefinite programming (SDP) optimization problems. Compared to deterministic certification methods, randomized smoothing [Cohen et al. 2019; Lecuyer et al. 2018; Salman et al. 2019a] is a defence method providing only probabilistic guarantees and incurring significant runtime costs at inference time, with the generalization to arbitrary safety properties still being an open problem.

A new avenue towards more precision are methods [Palma et al. 2021; Singh et al. 2019a; Tjandraatmadja et al. 2020] breaking the so-called convex barrier [Salman et al. 2019b] by considering activation functions jointly. 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 [Avis and Fukuda 1991, 1992; Barber et al. 1993; Dantzig 1998; Edelsbrunner 2012; Fukuda and Prodon 1995; Joswig 2003; Motzkin et al. 1953], in contrast to few approximate methods which either sacrifice soundness [Bentley et al. 1982; Khosravani et al. 2013; Sartipizadeh and Vincent 2016; Zhong et al. 2014] or still exhibit exponential complexity [Xu et al. 1998], prohibiting their use in neural network verification.

Our work follows the line of convex barrier-breaking methods, generalizing the concept to arbitrary bounded, multivariate activations. In contrast to prior work, we decompose the underlying convex hull problem into lower-dimensional spaces and solve it approximately using a novel relaxed Double Description, irredundancy formulation, and a new ray-shooting-based algorithm to add multiple constraints jointly. The resulting speed-ups make Prima tractable for non-piecewise-linear activations, a first for convex barrier-breaking methods.

9 CONCLUSION

We presented Prima, a general framework that substantially advances the state-of-the-art in neural network verification by providing efficient multi-neuron abstractions for arbitrary, bounded, multivariate non-linear activation functions. Our key idea is to compute tighter overall abstractions by considering many overlapping neuron groups thereby capturing more inter-neuron dependencies. To enable this, we decompose the bottleneck convex hull computation into lower-dimensional spaces and solve it approximately. Our extensive experimental evaluation shows that our algorithmic advances shift the bottleneck to the LP-solver while significantly improving both precision and scalability over prior work.

REFERENCES

Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. 2019. Optimization and Abstraction: A Synergistic Approach for Analyzing Neural Network Robustness. In *Proc. Programming Language Design and Implementation (PLDI)*. 731–744.

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.

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

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.

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).

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.

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

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

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)*.

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.

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

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

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

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

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

Patrick Cousot. 1996. Abstract Interpretation. *ACM Comput. Surv.* 28, 2 (1996), 324–328. https://doi.org/10.1145/234528.234740

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

Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy R Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy S Liang, et al. 2020. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems* 33 (2020).

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

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.

Komei Fukuda. 2020. Polyhedral Computation. https://doi.org/10.3929/ethz-b-000426218

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.

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.

Blagoy Genov. 2015. *The convex hull problem in practice: improving the running time of the double description method.* Ph.D. Dissertation.

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.

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

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.

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

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.

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.

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.

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).

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

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).

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).

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

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

David R Morrison, Sheldon H Jacobson, Jason J Sauppe, and Edward C Sewell. 2016. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization* 19 (2016), 79–102.

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.

Christoph Müller, Francois Serre, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2021. Scaling Polyhedral Neural Network Verification on GPUs. *Proc. Machine Learning and Systems (MLSys)* (2021).

Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2020. Neural Network Robustness Verification on GPUs. arXiv:cs.LG/2007.10868

Alessandro De Palma, Harkirat S. Behl, Rudy R. Bunel, Philip H. S. Torr, and M. Pawan Kumar. 2021. Scaling the Convex Barrier with Active Sets. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=uQfOy7LrlTR

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.

Anian Ruoss, Maximilian Baader, Mislav Balunović, and Martin Vechev. 2020a. Efficient Certification of Spatial Robustness. *arXiv preprint arXiv:2009.09318* (2020).

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

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).

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

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

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

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

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

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.

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

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

Gagandeep Singh, Markus Püschel, and Martin Vechev. 2017. Fast Polyhedra Abstract Domain. In *Proc. Principles of Programming Languages (POPL)*. 46–59.

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)*.

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).

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

Udacity. 2016. Using Deep Learning to Predict Steering Angles. https://github.com/udacity/self-driving-car.

Caterina Urban and Antoine Miné. 2021. A Review of Formal Methods applied to Machine Learning. *CoRR* abs/2104.02466 (2021). arXiv:2104.02466 https://arxiv.org/abs/2104.02466

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.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Verification. *arXiv preprint arXiv:2103.06624* (2021).

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).

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

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.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020a. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. (2020).

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2020b. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824* (2020).

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.

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*.

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.