Provable Preimage Under-Approximation for Neural Networks

Xiyue Zhang(✉), Benjie Wang, and Marta Kwiatkowska

Department of Computer Science, University of Oxford, Oxford, UK
{xiyue.zhang, benjie.wang, marta.kwiatkowska}@cs.ox.ac.uk

Abstract. Neural network verification mainly focuses on local robustness properties, which can be checked by bounding the image (set of outputs) of a given input set. However, often it is important to know whether a given property holds globally for the input domain, and if not then for what proportion of the input the property is true. To analyze such properties requires computing *preimage* abstractions of neural networks. In this work, we propose an efficient anytime algorithm for generating symbolic under-approximations of the preimage of any polyhedron output set for neural networks. Our algorithm combines a novel technique for cheaply computing polytope preimage under-approximations using linear relaxation, with a carefully-designed refinement procedure that iteratively partitions the input region into subregions using input and ReLU splitting in order to improve the approximation. Empirically, we validate the efficacy of our method across a range of domains, including a high-dimensional MNIST classification task beyond the reach of existing preimage computation methods. Finally, as use cases, we showcase the application to quantitative verification and robustness analysis. We present a sound and complete algorithm for the former, which exploits our disjoint union of polytopes representation to provide formal guarantees. For the latter, we find that our method can provide useful quantitative information even when standard verifiers cannot verify a robustness property.

1 Introduction

Despite the remarkable empirical success of neural networks, guaranteeing their correctness, especially when using them as decision-making components in safety-critical autonomous systems [7, 13, 43], is an important and challenging task. Towards this aim, various approaches have been developed for the verification of neural networks, with extensive effort devoted to local robustness verification [20, 22, 44, 11, 35, 32, 40, 41, 36]. While local robustness verification focuses on deciding the absence of adversarial examples within an ϵ-perturbation neighbourhood, an alternative approach for neural network analysis is to construct the preimage of its predictions [27, 15]. Given a set of outputs, the preimage is defined as the set of all inputs mapped by the neural network to that output set. By characterizing the preimage symbolically in an abstract representation, e.g., polyhedra, one can perform more complex analysis for a wider class of properties beyond local robustness, such as computing the *proportion* of inputs satisfying a property (quantitative verification) even if standard robustness verification fails.

Exact preimage generation [27] is intractable, taking time exponential in the number of neurons in a network; thus approximations are necessary. Unfortunately, existing methods are limited in their applicability. The inverse abstraction method in [15] bypasses the intractability of exact preimage generation by leveraging symbolic interpolants [14, 2] for abstraction of neural network layers. However, due to the complexity of interpolation, the time to compute the abstraction also scales exponentially with the number of neurons in hidden layers. A concurrent work [23] proposed an input bounding algorithm targeting backward reachability analysis for control policies and out-of-distribution (OOD) detection in low-dimensional domains. Their method produces a preimage *over-approximation*, which cannot be used for quantitative verification. Therefore, more efficient and flexible computation methods for (symbolic abstraction of) preimages of neural networks are needed.

The main contribution of this paper is a scalable method for preimage approximation, which can be used for a variety of robustness analysis tasks. More specifically, we propose an efficient *anytime* algorithm for generating symbolic under-approximations of the preimage of piecewise linear neural networks as a union of disjoint polytopes. The algorithm computes a sound preimage under-approximation leveraging linear relaxation based perturbation analysis (LiRPA) [40, 41, 32], applied backwards from a polyhedron output set. It iteratively refines the preimage approximation by adding input and/or intermediate (ReLU) splitting (hyper)planes to partition the input region into disjoint subregions, which can be approximated independently in parallel in a divide-and-conquer approach. The refinement scheme uses a novel differential objective to optimize the quality (volume) of the polytope subregions. We also show that our method can be generalized to generate preimage over-approximations. We illustrate the application of our method to quantitative verification, input bounding for control tasks, and robustness analysis against adversarial and patch attacks. Finally, we conduct an empirical analysis on a range of control and computer vision tasks, showing significant gains in efficiency compared to exact preimage generation methods and scalability to high-input-dimensional tasks compared to existing preimage approximation methods.

For space reasons, proofs and additional technical details have been moved to Appendix of the full version of the paper [45].

2 Preliminaries

We use $f : \mathbb{R}^d \to \mathbb{R}^m$ to denote a feedforward neural network. For layer i, we use $\mathbf{W}^{(i)}$ to denote the weight matrix, $\mathbf{b}^{(i)}$ the bias, $h^{(i)}$ the pre-activation neurons, and $a^{(i)}$ the post-activation neurons, such that we have $h^{(i)} = \mathbf{W}^{(i)} a^{(i-1)} + \mathbf{b}^{(i)}$. In this paper, we focus on ReLU neural networks with $a^{(i)}(x) = ReLU(h^{(i)}(x))$,

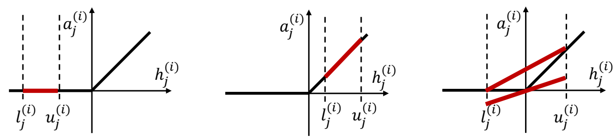


Fig. 1: Linear bounding functions for inactive, active, unstable ReLU neurons.

where $\mathrm{ReLU}(h) := \max(h, 0)$ is applied element-wise. However, our method can be generalized to other activation functions bounded by linear relaxation [44].

Linear Relaxation of Neural Networks. Nonlinear activation functions lead to the NP-completeness of the neural network verification problem [22]. To address such intractability, linear relaxation is often used to transform the nonconvex constraints into linear programs. As shown in Figure 1, given *concrete* lower and upper bounds $\mathbf{l}^{(i)} \le h^{(i)}(x) \le \mathbf{u}^{(i)}$ on the pre-activation values of layer i, there are three cases to consider. In the *inactive* ($u_j^{(i)} \le 0$) and *active* ($l_j^{(i)} \ge 0$) cases, the post-activation neurons $a_j^{(i)}(x)$ are linear functions $a_j^{(i)}(x) = 0$ and $a_j^{(i)}(x) = h_j^{(i)}(x)$ respectively. In the *unstable* case, $a_j^{(i)}(x)$ can be bounded by $\alpha_j^{(i)} h_j^{(i)}(x) \le a_j^{(i)}(x) \le -\frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} + \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} h_j^{(i)}(x)$, where $\alpha_j^{(i)}$ is a configurable parameter that produces a valid lower bound for any value in $[0, 1]$. Linear bounds can also be obtained for other non-piecewise linear activation functions [44].

Linear relaxation can be used to compute linear upper and lower bounds of the form $\underline{\mathbf{A}}x + \underline{\mathbf{b}} \le f(x) \le \overline{\mathbf{A}}x + \overline{\mathbf{b}}$ on the output of a neural network, for a given bounded input region $\mathcal{C}$. These methods are known as linear relaxation based perturbation analysis (LiRPA) algorithms [40, 41, 32]. In particular, *backward-mode* LiRPA computes linear bounds on f by propagating linear bounding functions backward from the output, layer-by-layer, to the input layer.

Polytope Representations. Given an Euclidean space $\mathbb{R}^d$, a polyhedron T is defined to be the intersection of a set of half spaces. More formally, suppose we have a set of linear constraints defined by $\psi_i(x) := c_i^T x + d_i \ge 0$ for $i = 1, ...K$, where $c_i \in \mathbb{R}^d, d_i \in \mathbb{R}$ are constants, and $x = x_1, ..., x_d$ is a set of variables. Then a polyhedron is defined as $T = \{x \in \mathbb{R}^d | \bigwedge_{i=1}^K \psi_i(x)\}$, where T consists of all values of x satisfying the first-order logic (FOL) formula $\alpha(x) := \bigwedge_{i=1}^K \psi_i(x)$. We use the term polytope to refer to a bounded polyhedron, that is, a polyhedron T such that $\exists R \in \mathbb{R}^{>0} : \forall x_1, x_2 \in T, \|x_1 - x_2\|_2 \le R$ holds. The abstract domain of polyhedra [32, 6, 8] has been widely used for the verification of neural networks and computer programs. An important type of polytope is the hyperrectangle (box), which is a polytope defined by a closed and bounded interval $[\underline{x_i}, \overline{x_i}]$ for each dimension, where $\underline{x_i}, \overline{x_i} \in \mathbb{Q}$. More formally, using the linear constraints $\phi_i := (x_i \ge \underline{x_i}) \land (x_i \le \overline{x_i})$ for each dimension, the hyperrectangle takes the form $\mathcal{C} = \{x \in \mathbb{R}^d | x \models \bigwedge_{i=1}^d \phi_i\}$.

3 Problem Formulation

3.1 Preimage Approximation

In this work, we are interested in the problem of computing preimages for neural networks. Given a subset $O \subset \mathbb{R}^m$ of the codomain, the preimage of a function $f : \mathbb{R}^d \to \mathbb{R}^m$ is defined to be the set of all inputs $x \in \mathbb{R}^d$ that are mapped to an element of O by f. For neural networks in particular, the input is typically restricted to some bounded input region $\mathcal{C} \subset \mathbb{R}^d$. In this work, we restrict the output set O to be a polyhedron, and the input set $\mathcal{C}$ to be an axis-aligned hyperrectangle region $\mathcal{C} \subset \mathbb{R}^d$, as these are commonly used in neural network verification. We now define the notion of a restricted preimage:

Definition 1 (Restricted Preimage). *Given a neural network $f : \mathbb{R}^d \to \mathbb{R}^m$, and an input set $\mathcal{C} \subset \mathbb{R}^d$, the restricted preimage of an output set $O \subset \mathbb{R}^m$ is defined to be the set $f_{\mathcal{C}}^{-1}(O) := \{x \in \mathbb{R}^d | f(x) \in O \wedge x \in \mathcal{C}\}$.*

Example 1. To illustrate our problem formulation and approach, we introduce a vehicle parking task [3] as a running example. In this task, there are four parking lots, located in each quadrant of a 2×2 grid $[0, 2]^2$, and a neural network with two hidden layers of 10 ReLU neurons $f : \mathbb{R}^2 \to \mathbb{R}^4$ is trained to classify which parking lot an input point belongs to. To analyze the behaviour of the neural network in the input region $[0, 1] \times [0, 1]$ corresponding to parking lot 1, we set $\mathcal{C} = \{x \in \mathbb{R}^2 | (0 \leq x_1 \leq 1) \wedge (0 \leq x_2 \leq 1)\}$. Then the restricted preimage $f_{\mathcal{C}}^{-1}(O)$ of the set $O = \{\boldsymbol{y} \in \mathbb{R}^4 | \bigwedge_{i \in \{2,3,4\}} y_1 - y_i \geq 0\}$ is the subspace of the region $[0, 1] \times [0, 1]$ that is *labelled* as parking lot 1 by the network.

We focus on *provable* approximations of the preimage. Given a first-order formula A, α is an *under-approximation* (resp. *over-approximation*) of A if it holds that $\forall x.\alpha(x) \implies A(x)$ (resp. $\forall x.A(x) \implies \alpha(x)$). In our context, the restricted preimage is defined by the formula $A(x) = (f(x) \in O) \wedge (x \in \mathcal{C})$, and we restrict to approximations α that take the form of a disjoint union of polytopes (DUP). The goal of our method is to generate a DUP approximation $\mathcal{T}$ that is as tight as possible; that is, to maximize the volume $\text{vol}(\mathcal{T})$ of an under-approximation, or minimize the volume $\text{vol}(\mathcal{T})$ of an over-approximation.

Definition 2 (Disjoint Union of Polytopes). *A disjoint union of polytopes (DUP) is a FOL formula α of the form $\alpha(x) := \bigvee_{i=1}^{D} \alpha_i(x)$, where each α_i is a polytope formula (conjunction of a finite set of linear half-space constraints), with the property that $\alpha_i \wedge \alpha_j$ is unsatisfiable for any $i \neq j$.*

3.2 Quantitative Properties

One of the most important verification problems for neural networks is that of proving guarantees on the output of a network for a given input set [18, 19, 30]. This is often expressed as a property of the form (I, O) such that $\forall x \in I \implies f(x) \in O$. We can generalize this to *quantitative* properties:

Definition 3 (Quantitative Property). *Given a neural network $f : \mathbb{R}^d \to \mathbb{R}^m$, a measurable input set with non-zero measure (volume) $I \subseteq \mathbb{R}^d$, a measurable output set $O \subseteq \mathbb{R}^m$, and a rational proportion $p \in [0,1]$ we say that the neural network satisfies the property (I, O, p) if $\frac{\mathrm{vol}(f_I^{-1}(O))}{\mathrm{vol}(I)} \geq p$.* [1]

Neural network verification algorithms [25] can be divided into two categories: sound, which always return correct results, and complete, guaranteed to reach a conclusion on any verification query. We now define soundness and completness of verification algorithms for quantitative properties.

Definition 4 (Soundness). *A verification algorithm QV is sound if, whenever QV outputs* True*, the property (I, O, p) holds.*

Definition 5 (Completeness). *A verification algorithm QV is complete if (i) QV never returns* Unknown*, and (ii) whenever QV outputs* False*, the property (I, O, p) does not hold.*

If the property (I, O) holds, then the quantitative property $(I, O, 1)$ holds, while quantitative properties for $0 \leq p < 1$ provide more information when (I, O) does not hold. Most neural network verification methods produce approximations of the *image* of I in the output space, which cannot be used to verify quantitative properties. Preimage *over-approximations* include false regions, thereby not applicable for quantitative verification. In contrast, preimage *under-approximations* provide a lower bound on the volume of the preimage, allowing us to soundly verify quantitative properties.

4 Methodology

Overview. In this section we present the main components of our methodology. Firstly, in Section 4.1, we show how to cheaply and soundly under-approximate the (restricted) preimage with a single polytope, using linear relaxation methods (Algorithm 2). Secondly, in Section 4.2, we propose a novel differentiable objective to optimize the quality (volume) of the polytope under-approximation. Thirdly, in Section 4.3, we propose a refinement scheme that improves the approximation by partitioning a (sub)region into subregions with splitting planes, with each subregion then being under-approximated more accurately. The main contribution of this paper (Algorithm 1) integrates these three components and is described in Section 4.4. Finally, in Section 4.5, we apply our method to quantitative verification (Algorithm 3) and prove its soundness and completeness.

4.1 Polytope Under-Approximation via Linear Relaxation

We first show how to adapt linear relaxation techniques to efficiently generate valid under-approximations to the restricted preimage for a given input region $\mathcal{C}$.

[1] In particular, the restricted preimage of a polyhedron under a neural network is Lebesgue measurable since polyhedra (intersection of a finite set of half-spaces) are Borel measurable and NNs are continuous functions.

Algorithm 1: Preimage Approximation

Input: Neural network f, Input region $\mathcal{C}$, Output region O, Volume threshold v, Maximum iterations R, Boolean *SplitOnInput*

Output: Disjoint union of polytopes $\mathcal{T}$

$T \leftarrow \text{GenUnderApprox}(\mathcal{C}, O)$; // Initial preimage polytope

$\widehat{\text{vol}}_T, \widehat{\text{vol}}_{f_{\mathcal{C}}^{-1}(O)} \leftarrow \text{EstimateVol}(T), \text{EstimateVol}(f_{\mathcal{C}}^{-1}(O))$;

$\text{Dom} \leftarrow \{(\mathcal{C}, T, \widehat{\text{vol}}_{f_{\mathcal{C}}^{-1}(O)} - \widehat{\text{vol}}_T)\}$; // Priority queue

// $\mathcal{T}_{\text{Dom}}$ is the union of polytopes in Dom

while $\text{EstimateVol}(\mathcal{T}_{\text{Dom}}) < v$ **and** $\text{Iterations} \leq R$ **do**

  $\mathcal{C}_{\text{sub}}, T, \text{Priority} \leftarrow \text{Pop(Dom)}$; // Subregion with highest priority

  if *SplitOnInput* **then**

    $id \leftarrow \text{SelectInputFeature}(\text{Feature}_I)$; // Feature_I is the set of input features/dimensions

  else

    $id \leftarrow \text{SelectReLUNode}(\text{Node}_Z)$; // Node_Z is the set of unstable ReLU nodes

  $[\mathcal{C}_{sub}^l, \mathcal{C}_{sub}^u] \leftarrow \text{SplitOnNode}(\mathcal{C}_{sub}, id)$; // Split on the selected node

  $[T^l, T^u] \leftarrow \text{GenUnderApprox}([\mathcal{C}_{sub}^l, \mathcal{C}_{sub}^u], O)$; // Generate preimage

  $[\widehat{\text{vol}}_{T^l}, \widehat{\text{vol}}_{T^u}] \leftarrow \text{EstimateVol}([T^l, T^u])$;

  $\widehat{\text{vol}}_{f_{\mathcal{C}_{sub}^l}^{-1}(O)}, \widehat{\text{vol}}_{f_{\mathcal{C}_{sub}^u}^{-1}(O)} \leftarrow \text{EstimateVol}(f_{\mathcal{C}_{sub}^l}^{-1}(O)), \text{EstimateVol}(f_{\mathcal{C}_{sub}^u}^{-1}(O))$;

  $\text{Dom} \leftarrow \text{Dom} \cup \{(\mathcal{C}_{sub}^l, T^l, \widehat{\text{vol}}_{f_{\mathcal{C}_{sub}^l}^{-1}(O)} - \widehat{\text{vol}}_{T^l})\} \cup \{(\mathcal{C}_{sub}^u, T^u, \widehat{\text{vol}}_{f_{\mathcal{C}_{sub}^u}^{-1}(O)} - \widehat{\text{vol}}_{T^u})\}$; // Disjoint polytope

return $\mathcal{T}_{\text{Dom}}$

Recall that LiRPA methods enable us to obtain linear lower and upper bounds on the output of a neural network f, that is, $\underline{\mathbf{A}}x + \underline{\mathbf{b}} \leq f(x) \leq \overline{\mathbf{A}}x + \overline{\mathbf{b}}$, where the linear coefficients depend on the input region $\mathcal{C}$.

Now, suppose that we are interested in computing an under-approximation to the restricted preimage, given the input hyperrectangle $\mathcal{C} = \{x \in \mathbb{R}^d | x \models \bigwedge_{i=1}^d \phi_i\}$, and the output polytope specified using the half-space constraints $\psi_i(y) = (c_i^T y + d_i \geq 0)$ for $i = 1, ..., K$ over the output space. Given a constraint ψ_i, we append an additional linear layer at the end of the network f, which maps $y \mapsto c_i^T y + d_i$, such that the function $g_i : \mathbb{R}^d \to \mathbb{R}$ represented by the new network is $g_i(x) = c_i^T f(x) + d_i$. Then, applying LiRPA bounding to each g_i, we obtain lower bounds $\underline{g_i}(x) = \underline{a}_i^T x + \underline{b}_i$ for each i, such that $\underline{g_i}(x) \geq 0 \implies g_i(x) \geq 0$ for $x \in \mathcal{C}$. Notice that, for each $i = 1, ..., K$, $\underline{a}_i^T x + \underline{b}_i \geq 0$ is a half-space constraint in the input space. We conjoin these constraints, along with the restriction to the input region $\mathcal{C}$, to obtain a polytope $T_{\mathcal{C}}(O) := \{x | \bigwedge_{i=1}^K (\underline{g_i}(x) \geq 0) \wedge \bigwedge_{i=1}^d \phi_i(x)\}$.

Proposition 1. *$T_{\mathcal{C}}(O)$ is an under-approximation to the restricted preimage $f_{\mathcal{C}}^{-1}(O)$.*

Algorithm 2: GenUnderApprox

Input: List of subregions $\mathcal{C}$, Output set O, number of samples N
Output: List of polytopes $\mathbf{T}$

1 $\mathbf{T} = []$;
2 **for** subregion $\mathcal{C}_{sub} \in \mathcal{C}$ `// Parallel over subregions`
3 **do**
4 $\quad [\underline{g_1}(x, \boldsymbol{\alpha}_1), ..., \underline{g_K}(x, \boldsymbol{\alpha}_K)] \leftarrow \text{LinearLowerBound}(\mathcal{C}_{sub}, O)$;
5 $\quad x_1, ..., x_N \leftarrow \text{Sample}(\mathcal{C}_{sub}, N)$;
6 $\quad \text{Loss}(\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K) \leftarrow -\sum_{j=1,...,N} \sigma(-\text{LSE}(-\underline{g_1}(x_j, \boldsymbol{\alpha}_1), ..., -\underline{g_K}(x_j, \boldsymbol{\alpha}_K))$;
7 $\quad \boldsymbol{\alpha}_1^*, ..., \boldsymbol{\alpha}_K^* \leftarrow \text{Optimize}(\text{Loss}(\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K))$;
8 $\quad \mathbf{T} = \text{Append}(\mathbf{T}, [\underline{g_1}(x, \boldsymbol{\alpha}_1^*) \geq 0, ..., \underline{g_K}(x, \boldsymbol{\alpha}_K^*) \geq 0, x \in \mathcal{C}_{sub}])$
9 **return** $\boldsymbol{T}$

Example 2. Returning to Example 1, the output constraints (for $i = 2, 3, 4$) are given by $\psi_i = (y_1 - y_i \geq 0) = (c_i^T y + d_i \geq 0)$, where $c_i := e_1 - e_i$ (where e_i is the i^{th} standard basis vector) and $d_i := 0$. Applying LiRPA bounding, we obtain the linear lower bounds $\underline{g_2}(x) = -3.79x_1 + x_2 + 2.65 \geq 0; \underline{g_3}(x) = 0.34x_1 - x_2 - 0.60 \geq 0; \underline{g_4}(x) = -1.11x_1 - x_2 + 1.99 \geq 0$ for each constraint. The intersection of these constraints, shown in Figure 2a, represents the region where any input is guaranteed to satisfy the output constraints.

We generate the linear bounds in parallel over the output polyhedron constraints $i = 1, ..., K$ using the *backward mode* LiRPA [44], and store the resulting input polytope $T_{\mathcal{C}}(O)$ as a list of constraints. This highly efficient procedure is used as a sub-routine `LinearLowerBound` when generating a preimage under-approximation as a polytope union using Algorithm 2 (Line 4).

4.2 Local Optimization

One of the key components behind the effectiveness of LiRPA-based bounds is the ability to efficiently improve the tightness of the bounding function by optimizing the relaxation parameters $\boldsymbol{\alpha}$, via projected gradient descent. In the context of local robustness verification, the goal is to optimize the concrete lower or upper bounds over the (sub)region $\mathcal{C}$ [40], i.e., $\min_{x \in \mathcal{C}} \underline{\mathbf{A}}(\boldsymbol{\alpha})x + \underline{\mathbf{b}}(\boldsymbol{\alpha})$, where we explicitly note the dependence of the linear coefficients on $\boldsymbol{\alpha}$. In our case, we are instead interested in optimizing $\boldsymbol{\alpha}$ to refine the polytope under-approximation, that is, increase its volume. Unfortunately, computing the volume of a polytope exactly is a computationally expensive task, and requires specialized tools [12] that do not permit easy optimization with respect to the $\boldsymbol{\alpha}$ parameters.

To address this challenge, we propose to use statistical estimation. In particular, we sample N points $x_1, ..., x_N$ uniformly from the input domain $\mathcal{C}$ then employ Monte Carlo estimation for the volume of the polytope approximation:

$$\widehat{\text{vol}}(T_{\mathcal{C},\boldsymbol{\alpha}}(O)) = \frac{\sum_{i=1}^{N} \mathbb{1}_{x_i \in T_{\mathcal{C},\boldsymbol{\alpha}}(O)}}{N} \times \text{vol}(\mathcal{C}) \tag{1}$$

where we highlight the dependence of $T_{\mathcal{C}}(O) = \{x | \bigwedge_{i=1}^{K} \underline{g_i}(x, \boldsymbol{\alpha}_i) \geq 0 \wedge \bigwedge_{i=1}^{d} \phi_i(x)\}$ on $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K)$, and $\boldsymbol{\alpha}_i$ are the α-parameters for the linear relaxation of the neural network g_i corresponding to the i^{th} half-space constraint in O. However, this is still non-differentiable w.r.t. $\boldsymbol{\alpha}$ due to the identity function. We now show how to derive a differentiable relaxation which is amenable to gradient-based optimization:

$$\begin{aligned}
\widehat{\text{vol}}(T_{\mathcal{C},\boldsymbol{\alpha}}(O)) &= \frac{\text{vol}(\mathcal{C})}{N} \sum_{j=1}^{N} \mathbb{1}_{x_j \in T_{\mathcal{C},\boldsymbol{\alpha}}(O)} = \frac{\text{vol}(\mathcal{C})}{N} \sum_{j=1}^{N} \mathbb{1}_{\min_{i=1,...K} \underline{g_i}(x_j, \boldsymbol{\alpha}_i) \geq 0} \\
&\approx \frac{\text{vol}(\mathcal{C})}{N} \sum_{j=1}^{N} \sigma \left(\min_{i=1,...K} \underline{g_i}(x_j, \boldsymbol{\alpha}_i) \right) \\
&\approx \frac{\text{vol}(\mathcal{C})}{N} \sum_{j=1}^{N} \sigma \left(-\text{LSE}(-\underline{g_1}(x_j, \boldsymbol{\alpha}_1), ..., -\underline{g_K}(x_j, \boldsymbol{\alpha}_K)) \right)
\end{aligned}$$

The second equality follows from the definition of the polytope $T_{\mathcal{C},\boldsymbol{\alpha}}(O)$; namely that a point is in the polytope if it satisfies $\underline{g_i}(x_j, \boldsymbol{\alpha}_i) \geq 0$ for all $i = 1, ..., K$, or equivalently, $\min_{i=1,...K} \underline{g_i}(x_j, \boldsymbol{\alpha}_i) \geq 0$. After this, we approximate the identity function using a sigmoid relaxation, where $\sigma(y) := \frac{1}{1+e^{-y}}$, as is commonly done in machine learning to define classification losses. Finally, we approximate the minimum over specifications using the log-sum-exp (LSE) function. The log-sum-exp function is defined by $LSE(y_1, ..., y_K) := \log(\sum_{i=1,...,K} e^{y_i})$, and is a differentiable approximation to the maximum function; we employ it to approximate the minimization by adding the appropriate sign changes. The final expression is now a differentiable function of $\boldsymbol{\alpha}$. We employ this as the loss function in Algorithm 2 (Line 6) for generating a polytope approximation, and optimize volume using projected gradient descent.

Example 3. We revisit the vehicle parking problem in Example 1. Figure 2a and 2b show the computed under-approximations before and after local optimization. We can see that the bounding planes for all three specifications are optimized, which effectively improves the approximation quality.

4.3 Global Branching and Refinement

As LiRPA performs crude linear relaxation, the resulting bounds can be quite loose even with $\boldsymbol{\alpha}$-optimization, meaning that the polytope approximation $T_{\mathcal{C}}(O)$ is unlikely to constitute a tight under-approximation to the preimage. To address this challenge, we employ a divide-and-conquer approach that iteratively refines our under-approximation of the preimage. Starting from the initial region $\mathcal{C}$ represented at the root, our method generates a tree by iteratively partitioning a subregion $\mathcal{C}_{sub}$ represented at a leaf node into two smaller subregions $\mathcal{C}_{sub}^{l}, \mathcal{C}_{sub}^{u}$, which are then attached as children to that leaf node. In this way, the subregions represented by all leaves of the tree are disjoint, such that their union is the initial region $\mathcal{C}$.

For each leaf subregion $\mathcal{C}_{sub}$ we compute, using LiRPA bounds (Line 4, Algorithm 2), an associated polytope that under-approximates the preimage in $\mathcal{C}_{sub}$. Thus, irrespective of the number of refinements performed, the union of the polytopes corresponding to all leaves forms an *anytime* DUP under-approximation $\mathcal{T}$ to the preimage in the original region $\mathcal{C}$. The process of refining the subregions continues until an appropriate termination criterion is met.

Unfortunately, even with a moderate number of input dimensions or unstable ReLU nodes, naïvely splitting along all input- or ReLU-planes quickly becomes computationally infeasible. For example, splitting a d-dimensional hyperrectangle using bisections along each dimension results in 2^d subdomains to approximate. It thus becomes crucial to identify the subregion splits that have the most impact on the quality of the under-approximation. Another important aspect is how to prioritize which leaf subregion to split. We describe these in turn.

Subregion Selection. Searching through all leaf subregions at each iteration is computationally too expensive. Thus, we propose a subregion selection strategy that prioritizes splitting subregions according to (an estimate of) the difference in volume between the exact preimage $f^{-1}_{\mathcal{C}_{sub}}(O)$ and the (already computed) polytope approximation $T_{\mathcal{C}_{sub}}(O)$ on that subdomain, that is:

$$\text{Priority}(\mathcal{C}_{sub}) = \text{vol}(f^{-1}_{\mathcal{C}_{sub}}(O)) - \text{vol}(T_{\mathcal{C}_{sub}}(O)) \tag{2}$$

which measures the gap between the polytope under-approximation and the optimal approximation, namely, the preimage itself.

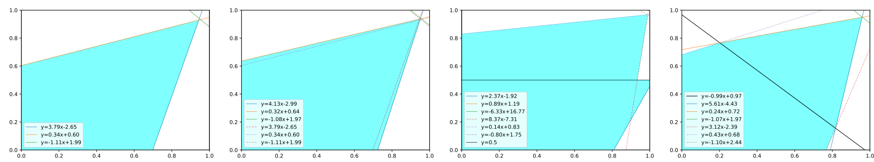
Suppose that a particular leaf subdomain attains the maximum of this metric among all leaves, and we partition it into two subregions $\mathcal{C}^l_{sub}, \mathcal{C}^u_{sub}$, which we approximate with polytopes $T_{\mathcal{C}^l_{sub}}(O), T_{\mathcal{C}^u_{sub}}(O)$. As tighter intermediate concrete bounds, and thus linear bounding functions, can be computed on the partitioned subregions, the polytope approximation on each subregion will be refined compared with the single polytope restricted to that subregion.

Proposition 2. *Given any subregion $\mathcal{C}_{sub}$ with polytope approximation $T_{\mathcal{C}_{sub}}(O)$, and its children $\mathcal{C}^l_{sub}, \mathcal{C}^u_{sub}$ with polytope approximations $T_{\mathcal{C}^l_{sub}}(O), T_{\mathcal{C}^u_{sub}}(O)$ respectively, it holds that:*

$$T_{\mathcal{C}^l_{sub}}(O) \cup T_{\mathcal{C}^u_{sub}}(O) \supseteq T_{\mathcal{C}_{sub}}(O) \tag{3}$$

Corollary 1. *In each refinement iteration, the volume of the polytope approximation $\mathcal{T}_{Dom}$ does not decrease.*

Since computing the volumes in Equation 2 is intractable, we sample N points $x_1, ..., x_N$ uniformly from the subdomain $\mathcal{C}_{sub}$ and employ Monte Carlo estimation to estimate the volume for both the preimage and the polytope approximation using the same set of samples, i.e., $\widehat{\text{vol}}(f^{-1}_{\mathcal{C}_{sub}}(O)) = \text{vol}(\mathcal{C}_{sub}) \times \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{x_i \in f^{-1}_{\mathcal{C}_{sub}}(O)}$, and $\widehat{\text{vol}}(T_{\mathcal{C}_{sub}}(O)) = \text{vol}(\mathcal{C}_{sub}) \times \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{x_i \in T_{\mathcal{C}_{sub}}(O)}$. We



(a) No optimization (b) After optimization (c) Input split (d) ReLU split

Fig. 2: Refinement and optimization for preimage approximation.

stress that volume estimation is only used to prioritize subregion selection, and does not affect the soundness of our method.

Input Splitting. Given a subregion (hyperrectangle) defined by lower and upper bounds $x_i \in [\underline{x}_i, \overline{x}_i]$ for all dimensions $i = 1, ..., d$, input splitting partitions it into two subregions by cutting along some feature i. This splitting procedure will produce two subregions which are similar to the original subregion, but have updated bounds $[\underline{x}_i, \frac{\underline{x}_i + \overline{x}_i}{2}], [\frac{\underline{x}_i + \overline{x}_i}{2}, \overline{x}_i]$ for feature i instead. In order to determine which feature/dimension to split on, we propose a greedy strategy. Specifically, for each feature, we generate a pair of polytopes for the two subregions resulting from the split, and choose the feature that results in the greatest total volume of the polytope pair. In practice, another commonly-adopted splitting heuristic is to select the dimension with the longest edge [10], that is, to select feature i with the largest range: $\arg\max_i(\overline{x}_i - \underline{x}_i)$. However, this method falls short in per-iteration approximation volume improvement compared to our greedy strategy.

Example 4. We revisit the vehicle parking problem in Example 1. Figure 2b shows the polytope under-approximation computed on the input region $\mathcal{C}$ before refinement, where each solid line represents the bounding plane for each output specification ($y_1 - y_i \geq 0$). Figure 2c depicts the refined approximation by splitting the input region along the vertical axis, where the solid and dashed lines represent the bounding planes for the two resulting subregions. It can be seen that the total volume of the under-approximation has improved significantly.

Intermediate ReLU Splitting. Refinement through splitting on input features is adequate for low-dimensional input problems such as reinforcement learning agents. However, it may be infeasible to generate sufficiently fine subregions for high-dimensional domains. We thus propose an algorithm for ReLU neural networks that uses intermediate ReLU splitting for preimage refinement. After determining a subregion for refinement, we partition the subregion based upon the pre-activation value of an intermediate unstable neuron $z_j^{(i)} = 0$. As a result, the original subregion $\mathcal{C}_{sub}$ is split into two new subregions $\mathcal{C}^+_{z_j^{(i)}} = \{x \in \mathcal{C}_{sub} \mid z_j^{(i)} = h_j^{(i)}(x) \geq 0\}$ and $\mathcal{C}^-_{z_j^{(i)}} = \{x \in \mathcal{C}_{sub} \mid z_j^{(i)} = h_j^{(i)}(x) < 0\}$.[2]

[2] To obtain a polytope under-approximation, we can utilize linear lower/upper bounds on $h_j^{(i)}(x)$ as an approximation to the subregion boundary.

In this procedure, the order of splitting unstable ReLU neurons can greatly influence the refinement quality and efficiency. Existing heuristic methods of ReLU prioritization select ReLU nodes that lead to greater improvement in the final bound (maximum or minimum value) of the neuron network on the input domain [10], i.e. $\min_{x \in \mathcal{C}} \underline{f}(x)$. However, these ReLU prioritization methods are not effective for preimage analysis, because our objective is instead to refine the overall preimage approximation. We thus propose a heuristic method to prioritize unstable ReLU nodes for preimage refinement. Specifically, we compute (an estimate of) the volume difference between the split subregions $|\mathrm{vol}(\mathcal{C}^{+}_{z_j^{(i)}}) - \mathrm{vol}(\mathcal{C}^{-}_{z_j^{(i)}})|$, using a single forward pass for a set of sampled datapoints from the input domain; note that this is bounded above by the total subregion volume $\mathrm{vol}(\mathcal{C}_{sub})$. We then propose to select the ReLU node that minimizes this difference. Intuitively, this choice results in balanced subdomains after splitting.

Another advantage of ReLU splitting is that we can replace the unstable neuron bound $\underline{c}h_j^{(i)}(x)+\underline{d} \leq a_j^{(i)}(x) \leq \overline{c}h_j^{(i)}(x)+\overline{d}$ with the exact linear function $a_j^{(i)}(x) = h_j^{(i)}(x)$ and $a_j^{(i)}(x) = 0$, respectively, as shown in Figure 1 (unstable to stable). This can then tighten the linear bounds for the other neurons, thus tightening the under-approximation on each subdomain.

Example 5. We now apply our algorithm with ReLU splitting to the vehicle parking problem in Example 1. Figure 2d shows the refined preimage polytope by adding the splitting plane (black solid line) along the direction of a selected unstable ReLU node. Compared with Figure 2b, we can see that the volume of the approximation is improved.

Remark 1 (Preimage Over-approximation). While Algorithms 1 and 2 focus on preimage under-approximations, they can be easily configured to generate over-approximations with two key modifications. Firstly, we generate polytope over-approximations by using LiRPA to propagate a linear *upper* bound $\overline{g_i}(x) = \overline{a}_i^T x + \overline{b}_i$ for each output constraint, such that $g_i(x) \geq 0 \implies \overline{g_i}(x) \geq 0$ for $x \in \mathcal{C}$. Secondly, the refinement and optimization objective is to *minimize* the volume of the over-approximation instead of maximizing the volume as in the case of under-approximation.

4.4 Overall Algorithm

Our overall preimage approximation method is summarized in Algorithm 1. It takes as input a neural network f, input region $\mathcal{C}$, output region O, target polytope volume threshold v (a proxy for approximation precision), termination iteration number R, and a Boolean indicating whether to use input or ReLU splitting, and returns a disjoint polytope union $\mathcal{T}$ representing an underapproximation to the preimage.

The algorithm initiates and maintains a priority queue of (sub)regions according to Equation 2. The *initialization* step (Lines 1-3) generates an initial polytope approximation on the whole region using Algorithm 2 (Sections 4.1,

Algorithm 3: Quantitative Verification

Input: Neural network f, Property (I, O, p), Maximum iterations R
Output: Verification Result $\in$ {True, False, Unknown}

$\mathrm{vol}(I) \leftarrow \mathrm{ExactVolume}(I)$;
$\mathcal{C} \leftarrow \mathrm{OuterBox}(I)$; `// For general polytopes` I
$\mathcal{T} \leftarrow \mathrm{InitialRun}(f, \mathcal{C}, O)$;
while Iterations $\leq R$ **do**
  $\mathcal{T} \leftarrow \mathrm{Refine}(f, \mathcal{T}, \mathcal{C}, O)$;
  if $\mathrm{EstimateVolume}(\mathcal{T}) \geq p \times \mathrm{vol}(I)$ **then**
    if $\mathrm{ExactVolume}(\mathcal{T}) \geq p \times \mathrm{vol}(I)$ **then**
      return True
  if AllReLUSplit **then**
    return False
return Unknown

4.2). Then, the *preimage refinement* loop (Lines 4-14) partitions a subregion in each iteration, with the preimage restricted to the child subregions then being re-approximated (Line 10-11). In each iteration, we choose the region to split (Line 5) and the splitting plane to cut on (Line 7 for input split and Line 9 for ReLU split), as detailed in Section 4.3. The preimage under-approximation is then updated by computing the priorities for each subregion (by approximating volumes) (Lines 12-14). The loop terminates and the approximation returned when the target volume threshold v or maximum iteration limit R is reached.

4.5 Quantitative Verification

We now show how to use our efficient preimage under-approximation method (Algorithm 1) to verify a given quantitative property (I, O, p), where O is a polyhedron, I a polytope and p the desired proportion value, summarized in Algorithm 3. To simplify assume that I is a hyperrectangle, so that we can take $\mathcal{C} = I$ (in view of space constraints the case of general polytopes is discussed in Appendix of [45]). We utilize Algorithm 1 by setting the volume threshold to $p \times \mathrm{vol}(I)$, such that we have $\frac{\widehat{\mathrm{vol}}(\mathcal{T})}{\mathrm{vol}(I)} \geq p$ if the algorithm terminates before reaching the maximum number of iterations. However, the Monte Carlo estimates of volume cannot provide a sound guarantee that $\frac{\mathrm{vol}(\mathcal{T})}{\mathrm{vol}(I)} \geq p$. To resolve this problem, we propose to run exact volume computation [5] only when the Monte Carlo estimate reaches the threshold. If the exact volume $\mathrm{vol}(\mathcal{T}) \geq p \times \mathrm{vol}(I)$, then the property is verified. Otherwise, we continue running the preimage refinement.

In Algorithm 3, `InitialRun` generates an initial approximation to the preimage as in Lines 1-3 of Algorithm 1, and `Refine` performs one iteration of approximation refinement (Lines 5-14). Termination occurs when we have verified or falsified the quantitative property, or when the maximum number of iterations has been exceeded.

Proposition 3. *Algorithm 3 is sound for quantitative verification with input splitting.*

Proposition 4. *Algorithm 3 is sound and complete for quantitative verification on piecewise linear neural networks with ReLU splitting.*

5 Experiments

We have implemented our approach as a prototype tool [3] for preimage approximation for polyhedron-type output sets/specifications. In this section, we perform experimental evaluation of the proposed approach on a set of benchmark tasks and demonstrate its effectiveness in approximation generation and its application to quantitative analysis of neural networks.

5.1 Benchmark and Evaluation Metric

We evaluate our preimage analysis approach on a benchmark of reinforcement learning and image classification tasks. Besides the vehicle parking task [3] shown in the running example, we use the following (trained) benchmarks: (1) aircraft collision avoidance system (VCAS) [21] with 9 feed-forward neural networks (FNNs); (2) neural network controllers from VNN-COMP 2022 [1] for three reinforcement learning tasks (Cartpole, Lunarlander, and Dubinsrejoin) [9]; and (3) the neural network from VNN-COMP 2022 for MNIST classification. Details of the models and additional experiments can be found in Appendix of [45].

Evaluation Metric To evaluate the quality of the preimage approximation, we define the *coverage ratio* to be the ratio of volume covered to the volume of the exact preimage, i.e., $\mathrm{cov}(\mathcal{T}, f_{\mathcal{C}}^{-1}(O)) := \frac{\mathrm{vol}(\mathcal{T})}{\mathrm{vol}(f_{\mathcal{C}}^{-1}(O))}$. Note that this is a normalized measure for assessing the quality of the approximation, as shown in Algorithm 3 when comparing with target coverage proportion p for termination of the refinement loop, and not as a measure for formal verification guarantees. In practice, we estimate $\mathrm{vol}(f_{\mathcal{C}}^{-1}(O))$ as $\widehat{\mathrm{vol}}(f_{\mathcal{C}}^{-1}(O)) = \mathrm{vol}(\mathcal{C}) \times \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{f(x_i)\in O}$, where $x_1, ...x_N$ are samples from $\mathcal{C}$. In Algorithm 1, the target volume (stopping criterion) is set as $v = r \times \widehat{\mathrm{vol}}(f_{\mathcal{C}}^{-1}(O))$, where r is the *target coverage ratio*.

5.2 Evaluation

Effectiveness in Preimage Approximation with Input Split We apply Algorithm 1 with input splitting to the input bounding problem for low-dimensional reinforcement learning tasks to evaluate its effectiveness. For comparison, we also run the exact preimage (Exact) [27] and preimage over-approximation (Invprop) [23, 24] methods.

Vehicle Parking & VCAS. Table 1 presents experimental results on the vehicle parking and VCAS tasks. In the table, we show the number of polytopes (#Poly)

[3] The source code is at `https://github.com/Zhang-Xiyue/PreimageApproxForNNs`.

Table 1: Performance comparison in preimage generation.

Models	**Exact**		**Invprop**		**Our**		
	#Poly	**Time**	**Time**	**Cov(%)**	**#Poly**	**Time**	**Cov(%)**
Vehicle (FNN 1×20)	10	3110.979	2.642	92.1	4	1.175	95.7
VCAS (FNN 1×21)	131	6363.272	-	-	12	11.281	91.0

Table 2: Performance of preimage approximation for reinforcement learning tasks.

Task	**Property**	**Config**	**#Poly**	**Cov(%)**	**Time**
Cartpole (FNN 2×64)	$\{y \in \mathbb{R}^2 \mid y_1 \geq y_2\}$	$\dot{\theta} \in [-2, -1]$	8	82.0	8.933
		$\dot{\theta} \in [-2, -0.5]$	17	75.5	14.527
		$\dot{\theta} \in [-2, 0]$	32	76.5	27.344
Lunarlander (FNN 2×64)	$\{y \in \mathbb{R}^4 \mid \wedge_{i \in \{1,3,4\}} y_2 \geq y_i\}$	$\dot{v} \in [-0.5, 0]$	38	75.5	34.311
		$\dot{v} \in [-1, 0]$	71	75.1	63.333
		$\dot{v} \in [-2, 0]$	159	75.0	134.929
Dubinsrejoin (FNN 2×256)	$\{y \in \mathbb{R}^8 \mid \wedge_{i \in [2,4]} \; y_1 \geq y_i \; \bigwedge \wedge_{i \in [6,8]} \; y_5 \geq y_i\}$	$x_v \in [-0.1, 0.1]$	26	75.8	34.558
		$x_v \in [-0.2, 0.2]$	61	75.4	78.437
		$x_v \in [-0.3, 0.3]$	1002	57.6	1267.272

in the preimage, computation time (Time(s)), and the approximate coverage ratio (Cov(%)) when the preimage approximation algorithm terminates with target coverage 90%. Compared with the exact method, our approach yields *orders-of-magnitude* improvement in efficiency. It can also characterize the preimage with much fewer (and also disjoint) polytopes (average reduction of 91.1% for VCAS).

The Invprop method [23] cannot be directly applied as it computes preimage over-approximations. We adapt it to produce an under-approximation by computing over-approximations for the complement of each output constraint; the resulting approximation is then the complement of a union of polytopes, rather than a DUP. On the 2D vehicle parking task, we find that the results (see Table 1) are comparable with ours in time and approximation coverage. Their implementation currently only supports two-dimensional input tasks [24]. While their algorithm, which employs input splitting, can in theory be extended to higher-dimensional tasks, a significant unaddressed technical challenge is in how to choose the input splits effectively in high dimensions. This is confounded by the fact that, to generate an under-approximation, we need separate runs of their algorithm for each output constraint. In contrast, our method naturally incorporates a principled splitting and refinement strategy, and can also effectively employ ReLU splitting for further scalability, as we will show below. Our method can also be configured to generate over-approximations (Section 4.3, Remark 1).

Neural Network Controllers. In this experiment, we consider preimage under-approximation for neural network controllers in reinforcement learning tasks. Note that [27] (Exact) is unable to deal with neural networks of these sizes and

Table 3: Refinement with ReLU split for MNIST (FNN 6×100)

L_∞ attack	#Poly	Cov(%)	Time	Patch attack	#Poly	Cov(%)	Time
0.05	2	100.0	3.107	3×3(center)	1	100.0	2.611
0.07	247	75.2	121.661	4×4(center)	678	38.2	455.988
0.08	522	75.1	305.867	6×6(corner)	2	100.0	9.065
0.09	733	16.5	507.116	7×7(corner)	7	84.2	10.128

[23, 24] (Invprop) does not support these higher-dimensional input domains. Table 2 summarizes the experimental results. We evaluate Algorithm 1 with input split on a range of tasks/properties and configurations of the input region (e.g., angular velocity $\dot{\theta}$ for Cartpole). Empirically, for the same coverage ratio, our method requires a number of polytopes and time roughly linear in the input region size, with the exception of Dubinsrejoin, where the larger number of output constraints and larger network size contribute to greater relaxation error.

MNIST Preimage Approximation with ReLU Split Next, we evaluate the scalability of Algorithm 1 with ReLU splitting by applying it to MNIST image classifiers. To our knowledge, this is the first time preimage computation has been attempted for this challenging, high-dimensional task.

Table 3 summarizes the evaluation results for two types of image attacks: l_∞ and patch attack. For L_∞ attacks, bounded perturbation noise is applied to all image pixels. The patch attack applies only to a smaller patch area but allows arbitrary perturbations covering the whole valid range $[0, 1]$. The task is then to produce a DUP under-approximation of the perturbation region that is guaranteed to be classified correctly. For L_∞ attack, our approach generates a preimage approximation that achieves the targeted coverage of 75% for noise up to 0.08. Notice that, from e.g. 0.05 to 0.07, the volume of the input region increases by tens of orders of magnitude due to the high dimensionality. The fact that the number of polytopes and computation time remains manageable is due to the effectiveness of ReLU splitting. Interestingly, for the patch attack, we observe that the number of polytopes required increases sharply when increasing the patch size at the center of the image, while this is not the case for patches in the corners of the image. We hypothesize this is due to the greater influence of central pixels on the neural network output, and correspondingly a greater number of unstable neurons over the input perturbation space.

Comparison with Robustness Verifiers We now illustrate empirically the utility of preimage computation in robustness analysis compared to robustness verifiers. Table 4 shows comparison results with α, β-CROWN, winner of the VNN competition [1]. We set the tasks according to the problem instances from VNN-COMP 2022 for local robustness verification (localized perturbation regions). For Cartpole, α, β-CROWN can provide a verification guarantee (yes/no or safe/unsafe) for both of the problem instances. However, in the case where the robustness property does not hold, our method explicitly generates a preimage approximation in the form of a disjoint polytope union (where correct classi-

Table 4: Comparison with a robustness verifier.

Task	α,β-**CROWN**		**Our**		
	Result	Time	Cov(%)	#Poly	Time
Cartpole ($\dot{\theta} \in [-1.642, -1.546]$)	yes	3.349	100.0	1	1.137
Cartpole ($\dot{\theta} \in [-1.642, 0]$)	no	6.927	94.9	2	3.632
MNIST (L_∞ 0.026)	yes	3.415	100.0	1	2.649
MNIST (L_∞ 0.04)	unknown	267.139	100.0	2	3.019

fication is guaranteed), and covers 94.9% of the exact preimage. For MNIST, while the smaller perturbation region is successfully verified, α, β-CROWN with tightened intermediate bounds by MIP solvers returns unknown with a timeout of 300s for the larger region. In comparison, our algorithm provides a concrete union of polytopes where the input is guaranteed to be correctly classified, which we find covers 100% of the input region (up to sampling error). Note also (Table 3) that our algorithm can produce non-trivial under-approximations for input regions far larger than α, β-CROWN can verify.

Quantitative Verification We now demonstrate the application of our preimage generation framework to quantitative verification of the property (I, O, p); that is, to check whether $f(x) \in O$ for at least proportion p of input values $x \in I$. This leverages the disjointness of our approximation, such that we can exactly compute the volume covered by exactly computing the volume of each polytope.

Vehicle Parking. We consider the quantitative property with input set $I = \{x \in \mathbb{R}^2 \mid x \in [0,1]^2\}$, output set $O = \{y \in \mathbb{R}^4 | \bigwedge_{i=2}^4 y_1 - \ y_i \geq 0\}$, and quantitative proportion $p = 0.95$. We use Algorithm 3 to verify this property, with iteration limit 1000. The computed under-approximation is a union of two polytopes, which takes 0.942s to reach the target coverage. We then compute the exact volume ratio of the under-approximation against the input region. The final quantitative proportion reached by our under-approximation is 95.2%, verifying the quantitative property.

Aircraft Collision Avoidance. In this example, we consider the VCAS system and a scenario where the two aircraft have negative relative altitude from intruder to ownship ($h \in [-8000, 0]$), the ownship aircraft has a positive climbing rate $\dot{h}_A \in [0, 100]$ and the intruder has a stable negative climbing rate $\dot{h}_B = -30$, and time to the loss of horizontal separation is $t \in [0, 40]$, which defines the input region I. For this scenario, the correct advisory is "Clear Of Conflict" (COC). We apply Algorithm 3 to verify the quantitative property where $O = \{y \in \mathbb{R}^9 | \bigwedge_{i=2}^9 y_1 - y_i \geq 0\}$ and the proportion $p = 0.9$, with an iteration limit of 1000. The under-approximation computed is a union of 6 polytopes, which takes 5.620s to reach the target coverage. The exact quantitative proportion reached by the generated under-approximation is 90.8%, which verifies the quantitative property.

6 Related Work

Our paper is related to a series of works on robustness verification of neural networks. To address the scalability issues with *complete* verifiers [20, 22, 35] based on constraint solving, convex relaxation [31] has been used for developing highly efficient *incomplete* verification methods [44, 39, 32, 40]. Later works employed the branch-and-bound (BaB) framework [11, 10] to achieve completeness, using incomplete methods for the bounding procedure [41, 36, 17]. In this work, we adapt convex relaxation for efficient preimage approximation. Further, our divide-and-conquer procedure is analogous to BaB, but focuses on maximizing covered volume rather than maximizing a function value. There are also works that have sought to define a weaker notion of local robustness known as *statistical robustness* [37, 26], which requires that a proportion of points under some perturbation distribution around an input point are classified in the same way. Verification of statistical robustness is typically achieved by sampling and statistical guarantees [37, 4, 34, 42]. In this paper, we apply our symbolic approximation approach to quantitative analysis of neural networks, while providing *exact quantitative* rather than *statistical* guarantees [38].

Another line of related works considers deriving exact or approximate abstractions of neural networks, which are applied for explanation [33], verification [16, 29], reachability analysis [28], and preimage approximation [15, 23]. [15] leverages symbolic interpolants [2] for preimage approximations, facing exponential complexity in the number of hidden neurons. Concurrently, [23] employs Lagrangian dual optimization for preimage over-approximations. Our anytime algorithm, which combines convex relaxation with principled splitting strategies for refinement, is applicable for both under- and over-approximations. Their work may benefit from our splitting strategies to scale to higher dimensions.

7 Conclusion

We present an efficient and flexible algorithm for preimage under-approximation of neural networks. Our *anytime* method derives from the observation that linear relaxation can be used to efficiently produce under-approximations, in conjunction with custom-designed strategies for iteratively decomposing the problem to rapidly improve the approximation quality. Unlike previous approaches, it is designed for, and scales to, both low and high-dimensional problems. Experimental evaluation on a range of benchmark tasks shows significant advantage in runtime efficiency and scalability, and the utility of our method for important applications in quantitative verification and robustness analysis.

Acknowledgments This project received funding from the ERC under the European Union's Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement No. 834115) and ELSA: European Lighthouse on Secure and Safe AI project (grant agreement No. 101070617 under UK guarantee). This work was done in part while Benjie Wang was visiting the Simons Institute for the Theory of Computing.

References

1. VnnComp 2022. https://github.com/ChristopherBrix/vnncomp2022_benchmarks, accessed: 2022-09-30
2. Albarghouthi, A., McMillan, K.L.: Beautiful interpolants. In: Computer Aided Verification - 25th International Conference, CAV 2013, Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 313–329. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_22
3. Ayala, D., Wolfson, O., Xu, B., DasGupta, B., Lin, J.: Parking slot assignment games. In: 19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS, Proceedings. pp. 299–308. ACM (2011)
4. Baluta, T., Chua, Z.L., Meel, K.S., Saxena, P.: Scalable quantitative verification for deep neural networks. In: Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings. p. 248–249. ICSE '21, IEEE Press (2021)
5. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. pp. 469–483 (1996). https://doi.org/10.1145/235815.235821
6. Benoy, P.M.: Polyhedral domains for abstract interpretation in logic programming. Ph.D. thesis, University of Kent, UK (2002)
7. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
8. Boutonnet, R., Halbwachs, N.: Disjunctive relational abstract interpretation for interprocedural program analysis. In: Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Proceedings. Lecture Notes in Computer Science, vol. 11388, pp. 136–159. Springer (2019). https://doi.org/10.1007/978-3-030-11245-5_7
9. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. CoRR (2016), http://arxiv.org/abs/1606.01540
10. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H., Kohli, P., Kumar, M.P.: Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research pp. 1–39 (2020)
11. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS. pp. 4795–4804 (2018)
12. Chevallier, A., Cazals, F., Fearnhead, P.: Efficient computation of the the volume of a polytope in high-dimensions using piecewise deterministic markov processes. In: International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event. Proceedings of Machine Learning Research, vol. 151, pp. 10146–10160. PMLR (2022)
13. Codevilla, F., Müller, M., López, A.M., Koltun, V., Dosovitskiy, A.: End-to-end driving via conditional imitation learning. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation. pp. 1–9. IEEE, Brisbane, Australia (2018). https://doi.org/10.1109/ICRA.2018.8460487
14. Craig, W.: Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. The Journal of Symbolic Logic pp. 269–285 (1957)
15. Dathathri, S., Gao, S., Murray, R.M.: Inverse abstraction of neural networks using symbolic interpolation. In: The Thirty-Third AAAI Conference

on Artificial Intelligence, AAAI 2019. pp. 3437–3444. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33013437

16. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Computer Aided Verification: 32nd International Conference, CAV 2020, Proceedings, Part I 32. pp. 43–65. Springer (2020)
17. Ferrari, C., Müller, M.N., Jovanovic, N., Vechev, M.T.: Complete verification via multi-neuron relaxation guided branch-and-bound. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net (2022)
18. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE symposium on security and privacy (SP). pp. 3–18. IEEE (2018)
19. Gopinath, D., Converse, H., Păsăreanu, C.S., Taly, A.: Property inference for deep neural networks. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. p. 797–809. ASE '19, IEEE Press (2020). https://doi.org/10.1109/ASE.2019.00079
20. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Computer Aided Verification - 29th International Conference, CAV 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10426, pp. 3–29. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_1
21. Julian, K.D., Kochenderfer, M.J.: A reachability method for verifying dynamical systems with deep neural network controllers. CoRR (2019), http://arxiv.org/abs/1903.00520
22. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: Computer Aided Verification: 29th International Conference, CAV 2017, Proceedings, Part I 30. pp. 97–117. Springer (2017)
23. Kotha, S., Brix, C., Kolter, Z., Dvijotham, K., Zhang, H.: Provably bounding neural network preimages. Accepted to NeurIPS 2023, CoRR (2023). https://doi.org/10.48550/arXiv.2302.01404
24. Kotha, S., Brix, C., Kolter, Z., Dvijotham, K., Zhang, H.: INVPROP for provably bounding neural network preimages. https://github.com/kothasuhas/verify-input (accessed October, 2023)
25. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J., et al.: Algorithms for verifying deep neural networks. Foundations and Trends in Optimization pp. 244–404 (2021)
26. Mangal, R., Nori, A.V., Orso, A.: Robustness of neural networks: a probabilistic and practical approach. In: Sarma, A., Murta, L. (eds.) Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019. pp. 93–96. IEEE / ACM (2019)
27. Matoba, K., Fleuret, F.: Exact preimages of neural network aircraft collision avoidance systems. In: Proceedings of the Machine Learning for Engineering Modeling, Simulation, and Design Workshop at Neural Information Processing Systems 2020 (2020)
28. Prabhakar, P., Afzal, Z.R.: Abstraction based output range analysis for neural networks. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019. pp. 15762–15772 (2019)

29. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Computer Aided Verification: 22nd International Conference, CAV 2010, Proceedings 22. pp. 243–257. Springer (2010)
30. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018. pp. 2651–2659. ijcai.org (2018)
31. Salman, H., Yang, G., Zhang, H., Hsieh, C., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019. pp. 9832–9842 (2019)
32. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages pp. 1–30 (2019)
33. Sotoudeh, M., Thakur, A.V.: Syrenn: A tool for analyzing deep neural networks. In: Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Proceedings, Part II 27. pp. 281–302. Springer (2021)
34. Tit, K., Furon, T., Rousset, M.: Efficient statistical assessment of neural network corruption robustness. In: Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. pp. 9253–9263 (2021)
35. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: 7th International Conference on Learning Representations, ICLR 2019. OpenReview.net (2019)
36. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. pp. 29909–29921 (2021)
37. Webb, S., Rainforth, T., Teh, Y.W., Kumar, M.P.: A statistical approach to assessing neural network robustness. In: 7th International Conference on Learning Representations, ICLR 2019. OpenReview.net (2019)
38. Wicker, M., Laurenti, L., Patane, A., Kwiatkowska, M.: Probabilistic safety for bayesian neural networks. In: In Proc. 36th Conference on Uncertainty in Artificial Intelligence (UAI-2020). PMLR (2020)
39. Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: International conference on machine learning. pp. 5286–5295. PMLR (2018)
40. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.: Automatic perturbation analysis for scalable certified robustness and beyond. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020)
41. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event. OpenReview.net (2021)

42. Yang, P., Li, R., Li, J., Huang, C., Wang, J., Sun, J., Xue, B., Zhang, L.: Improving neural network verification through spurious region guided refinement. In: Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12651, pp. 389–408. Springer (2021)
43. Yun, S., Choi, J., Yoo, Y., Yun, K., Choi, J.Y.: Action-decision networks for visual tracking with deep reinforcement learning. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. pp. 1349–1358. IEEE Computer Society (2017)
44. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018. pp. 4944–4953 (2018)
45. Zhang, X., Wang, B., Kwiatkowska, M.: Provable preimage under-approximation for neural networks. arXiv preprint arXiv:2305.03686 (2023)