# Automatic Abstraction Refinement in Neural Network Verification using Sensitivity Analysis

Tobias Ladner
tobias.ladner@tum.de
Technical University of Munich
Munich, Germany

Matthias Althoff
althoff@tum.de
Technical University of Munich
Munich, Germany

## ABSTRACT

The formal verification of neural networks is essential for their application in safety-critical environments. However, the set-based verification of neural networks using linear approximations often obtains overly conservative results, while nonlinear approximations quickly become computationally infeasible in deep neural networks. We address this issue for the first time by automatically balancing between precision and computation time without splitting the propagated set. Our work introduces a novel automatic abstraction refinement approach using sensitivity analysis to iteratively reduce the abstraction error at the neuron level until either the specifications are met or a maximum number of iterations is reached. Our evaluation shows that we can tightly over-approximate the output sets of deep neural networks and that our approach is up to a thousand times faster than a naive approach. We further demonstrate the applicability of our approach in closed-loop settings.

## CCS CONCEPTS

• **Computer systems organization** → **Neural networks**; • **General and reference** → **Verification**; • **Theory of computation** → **Abstraction**.

## KEYWORDS

Neural networks, formal verification, automatic abstraction refinement, sensitivity analysis, set-based computing, polynomial zonotopes.

**ACM Reference Format:**
Tobias Ladner and Matthias Althoff. 2023. Automatic Abstraction Refinement in Neural Network Verification using Sensitivity Analysis. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3575870.3587129

## 1 INTRODUCTION

Neural networks have achieved significant progress in a variety of applications, including safety-critical tasks [30]. However, they are vulnerable to small perturbations of the input, also known as adversarial examples [13, 41]. This demonstrates the limitations of neural networks in safety-critical environments. Thus, the formal verification of neural networks has gained importance in recent years [28], both in terms of open-loop neural networks [3] and neural-network-controlled systems [20]. A short overview of different approaches is given below.

*Related Work.* Some approaches formulate the verification of neural networks as satisfiability modulo theories (SMT) [21, 22, 31, 46] or symbolic interval propagation [14, 38] problems. These approaches reason about neural networks by introducing relaxed constraints for each neuron and solve these instances using SMT and linear programming solvers. These solvers can return a counterexample if a specification is violated. Most approaches provide a sound verification procedure for neural networks with specific activation functions, such as piecewise-linear neurons. Branch-and-bound strategies [9] can improve the results by splitting the problem at the neuron level [8, 37], e.g., splitting ReLU neurons at the intersection of the linear parts makes each subproblem linear. Further performance improvements are made by implementing the solvers on a GPU [45]. The major limitation of these approaches is that recursive splitting leads to an exponential growth of subproblems [4, 17], which requires more advanced branch-and-bound approaches [43].

More closely related to this work are approaches deploying reachability analysis. Here, the input set is propagated through the network, applying specific operations on the chosen set representation for each layer. While linear layers can be computed exactly, this is not possible for nonlinear layers. Thus, these approaches often compute an over-approximation of the true output set and thus require additional methods to obtain counterexamples [26]. Early approaches focused on convex set representations such as zonotopes [11, 36]. Several publications [27, 44] have shown that splitting sets helps to reduce the over-approximations, however, splitting does not usually scale well with the dimension of the problem. Since neural networks are universal approximators [15], more advanced set representations are necessary to obtain tight over-approximations without splitting. Examples are Taylor models [7, 16, 18], star sets [42], and polynomial zonotopes [25]. In this work, we use polynomial zonotopes [24] as they are closed under many typical set operations with polynomial complexity with respect to the dimension, such as linear map, quadratic map, and order reduction (Sec. 2).

A major limitation of the related works is their shortcoming in improving the computed over-approximation in the event that the specifications cannot be verified. To the best of our knowledge, there is no work on automatically refining the degree of approximation to reduce the over-approximation of the output set without splitting sets. This is especially useful in closed-loop settings [20], where


*HSCC '23, May 09–12, 2023, San Antonio, TX, USA*

ACM ISBN 979-8-4007-0033-0/23/05…$15.00
https://doi.org/10.1145/3575870.3587129

splitting sets is particularly computationally expensive due to the repeated evaluation of the neural network. In this work, we address this research gap.

*Contributions.* We present a novel automatic approach for iteratively reducing the over-approximation of the output $\mathcal{Y}$ of a neural network for a given input set $\mathcal{X}$. Our approach continues to refine $\mathcal{Y}$ until it no longer intersects with given specifications $\mathcal{S}$, defining an unsafe set in the output space, up to a maximum number of iterations. Our automatic refinement is guided by a refinement heuristic, which determines the neurons for which the abstraction is refined. We compare multiple heuristics, specifically a novel heuristic derived from sensitivity analysis. Further, we reuse expensive computations from previous iterations to save computation time. Our approach falls into the category of counter-example-triggered abstraction refinement (CETAR [33]), because no specific counterexample is used as in counter-example-guided abstraction refinement (CEGAR [10]). Further, we present an efficient approach to remove redundancies in the representation of polynomial zonotopes during the evaluation on polynomials, which is used throughout our approach. The applicability of our approach is shown in an extensive evaluation. Our algorithm does not split the propagated sets, making it attractive for open-loop [3] and closed-loop [20] settings.

The remainder of this paper is structured as follows. Sec. 2 states the background on set-based neural network verification. In Sec. 3, we present an efficient approach to evaluate polynomial zonotopes on polynomials. Sec. 4 provides a detailed description of our novel automatic abstraction refinement approach. Finally, we evaluate our approach in Sec. 5.

## 2 PRELIMINARIES

### 2.1 Notation

We denote vectors with lower-case letters, matrices with upper-case letters, and sets with calligraphic letters. The $i$-th element of a vector $b \in \mathbb{R}^n$ is written as $b_{(i)}$. Consequently, $b_{2(i)}$ is the $i$-th element of a vector $b_2$. The element in the $i$-th row and $j$-th column of a matrix $A \in \mathbb{R}^{n \times m}$ is written as $A_{(i,j)}$, the entire $i$-th row and $j$-th column are written as $A_{(i,\cdot)}$ and $A_{(\cdot,j)}$, respectively. We indicate the $i$-th power of $c \in \mathbb{R}$ by $c^i$, whereas $c^{(i)}$ refers to $c$ in iteration $i$. We denote the $p$-norm of a vector $c$ with $\|c\|_p$. The empty matrix is represented by $[]$. The concatenation of two matrices $A$ and $B$ is denoted by $[A\ B]$. The symbols $\mathbf{0}$ and $\mathbf{1}$ refer to matrices with all zeros and ones of proper dimensions, respectively. Given two sets $\mathcal{S}_1, \mathcal{S}_2 \subset \mathbb{R}^n$, then the Minkowski sum is defined as $\mathcal{S}_1 \oplus \mathcal{S}_2 = \{s_1 + s_2 \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\}$. Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and a set $\mathcal{S} \subset \mathbb{R}^n$, then $f(\mathcal{S}) = \{f(x) \mid x \in \mathcal{S}\}$. We denote the $i$-th component of $f$ by $f_{(i)} : \mathbb{R}^n \to \mathbb{R}$. An interval with bounds $a, b \in \mathbb{R}^n$ is denoted by $[a, b]$. The term $a \to b$ denotes that $a$ approaches $b$. We reference an element $e$ of an object $O$ using $O.e$.

### 2.2 Neural Networks

In this work, we intentionally focus on feed-forward neural networks [6] with alternating linear and activation layers; however, our approach can be extended to a variety of neural networks.

*Definition 2.1. (Layers of Neural Networks [6, Sec. 5.1])* Let $v_k$ denote the number of neurons in a layer $k$, $h_{k-1} \in \mathbb{R}^{v_{k-1}}$ the input, and $h_k \in \mathbb{R}^{v_k}$ the output. Further, let $W \in \mathbb{R}^{v_k \times v_{k-1}}$, $b \in \mathbb{R}^{v_k}$, and $\sigma_k(\cdot)$ be the respective continuous activation function (e.g. sigmoid or ReLU), which is applied element-wise. Then, the operation $L_k : \mathbb{R}^{v_{k-1}} \to \mathbb{R}^{v_k}$ on layer $k$ is given by

$$h_k = L_k(h_{k-1}) = \begin{cases} W_k h_{k-1} + b_k, & \text{if layer } k \text{ is linear,} \\ \sigma_k(h_{k-1}), & \text{otherwise.} \end{cases} \tag{1}$$

*Definition 2.2. (Neural Networks [6, Sec. 5.1])* Let there be given $K$ alternating linear and nonlinear layers, $v_0$ input and $v_K$ output neurons, the input $x \in \mathbb{R}^{v_0}$, and the output $y \in \mathbb{R}^{v_K}$ of the neural network. Then, a neural network can be formulated as

$$\begin{aligned} h_0 &= x, \\ h_k &= L_k(h_{k-1}),\ k = 1 \dots K, \\ y &= h_K. \end{aligned} \tag{2}$$

We denote the number of neurons per layer in a network by $[v_0, v_1, v_3 \dots, v_{K-1}]$ because nonlinear layers do not change the number of neurons. We can measure the linear influence of each neuron on the output using sensitivity analysis [40, 47] based on backpropagation [35].

*Definition 2.3. (Neuron Sensitivity [40])* The sensitivity of the neurons in the $k$-th layer on the output $y \in \mathbb{R}^{v_K}$ of a neural network w.r.t. its input $h_{k-1} \in \mathbb{R}^{v_{k-1}}$ can be computed as

$$S_k = \widehat{S}_k \cdot \ldots \cdot \widehat{S}_K,$$
$$\text{with } \widehat{S}_{k(i,j)} = \frac{\partial L_k(j)}{\partial h_{k-1(i)}} = \begin{cases} W_{k(j,i)}, & \text{if layer } k \text{ is linear,} \\ \frac{\partial \sigma_{k(j)}}{\partial h_{k-1(i)}}, & \text{otherwise.} \end{cases} \tag{3}$$

Then, $S_{k(i,j)}$ provides the linear influence of small changes on the input of neuron $i$ in the $k$-th layer on the output neuron $j$.

### 2.3 Set-Based Computation

Given an input set $\mathcal{X}$, the true output sets $\mathcal{H}_k^*$ of each layer $k$ are given by:

$$\begin{aligned} \mathcal{H}_0^* &= \mathcal{X}, \\ \mathcal{H}_k^* &= L_k(\mathcal{H}_{k-1}^*),\ k = 1 \dots K, \\ \mathcal{Y}^* &= \mathcal{H}_K^*. \end{aligned} \tag{4}$$

We use (sparse) polynomial zonotopes [24] as set representation for over-approximating the propagated sets $\mathcal{H}_k \supseteq \mathcal{H}_k^*$, as they can represent non-convex sets and have polynomial time complexity for many operations on them, particularly nonlinear maps.

*Definition 2.4. (Polynomial Zonotope [24])* [1] Given an offset $c \in \mathbb{R}^n$, a generator matrix of dependent generators $G \in \mathbb{R}^{n \times h}$, an exponent matrix $E \in \mathbb{N}_0^{p \times h}$, and a generator matrix of independent generators $G_I \in \mathbb{R}^{n \times q}$, a polynomial zonotope $\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ}$ is defined as

$$\mathcal{PZ} = \left\{ c + \sum_{i=1}^{h} \left( \prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) G_{(\cdot,i)} + \sum_{j=1}^{q} \beta_j G_{I(\cdot,j)} \;\middle|\; \alpha_k, \beta_j \in [-1, 1] \right\}. \tag{5}$$

[1] As in [23, 25], we adapt the definition from [24] and do not integrate the offset $c$ into the generator matrix $G$ and omit the identifier vector for simplicity.

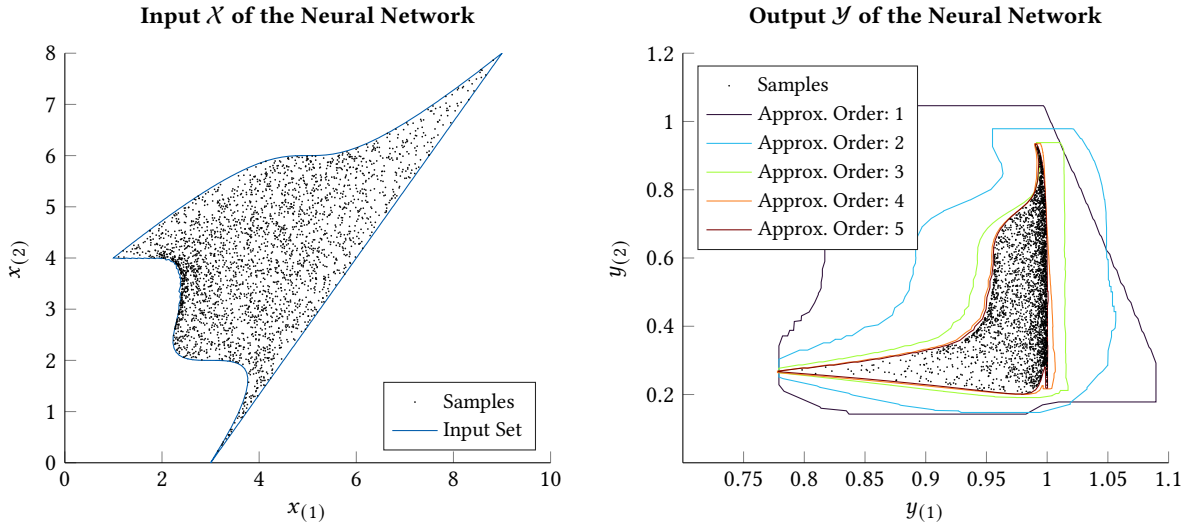


**Figure 1: Over-approximation of the output set to arbitrary precision on a random network with one linear layer and one sigmoid activation layer. Note that the over-approximation with order 5 tightly encloses the gap in the bottom right.**

A polynomial zonotope is *regular* if the exponent matrix $E$ does not contain duplicate or all-zero columns. A non-regular polynomial zonotope can be made regular using the `compact` operation [23, Prop. 3.1.7] with computational complexity

$$O(\texttt{compact}) = O(h(n + p\log(h))). \tag{6}$$

For some operations, we need zonotopes [12, Def. 1], which are a special case of polynomial zonotopes.

*Definition 2.5. (Zonotope [12, Def. 1])* Given a center vector $c \in \mathbb{R}^n$ and a generator matrix $G \in \mathbb{R}^{n\times q}$, a zonotope is defined as

$$\mathcal{Z} = \langle c, G\rangle_Z = \left\{ c + \sum_{j=1}^{q} \beta_j G_{(\cdot,j)} \;\middle|\; \beta_j \in [-1, 1] \right\}. \tag{7}$$

Further, we introduce enclosures to compute bounds for polynomial zonotopes using the following two propositions.

Proposition 2.6. *(Zonotope Enclosure [23, Prop. 3.1.14]) Given a $\mathcal{PZ} = \langle c, G, G_I, E\rangle_{PZ}$, then $\mathcal{PZ} \subseteq \mathcal{Z} = \texttt{zonotope}(\mathcal{PZ})$ with*

$$\mathcal{Z} = \left\langle c + \sum_{i\in\mathcal{H}} 0.5G_{(\cdot,i)}, \begin{bmatrix} 0.5G_{(\cdot,\mathcal{H})} & G_{(\cdot,\mathcal{K})} & G_I \end{bmatrix} \right\rangle_Z, \tag{8}$$

*where $\mathcal{H}$ contains the indices of the generators with all even exponents and $\mathcal{K} = \{1 \dots h\} \setminus \mathcal{H}$.*

Proposition 2.7. *(Interval Enclosure [1, Prop. 2.2]) Given a zonotope $\mathcal{Z} = \langle c, G\rangle_Z$, the interval bounds $[l, u]$ with $l \le z \le u$, $\forall z \in \mathcal{Z}$, where the inequalities are applied element-wise, are given by*

$$\begin{matrix} l = c - \Delta g \\ u = c + \Delta g \end{matrix}, \text{ with } \Delta g = \sum_{j=1}^{q} |G_{(\cdot,j)}|. \tag{9}$$

Next, we briefly state all operations on polynomial zonotopes required for the verification of neural networks. Some operations are simplified as the more complex case is not needed in this work and the notation is harmonized. For linear layers, we require linear maps:

Proposition 2.8. *(Linear Map [23, Prop. 3.1.18/19]) Given a $\mathcal{PZ} \subset \mathbb{R}^n$, a matrix $W \in \mathbb{R}^{m\times n}$, and an offset vector $b \in \mathbb{R}^m$, the result of a linear map is given by*

$$W\,\mathcal{PZ} + b = \langle Wc + b, WG, WG_I, E\rangle_{PZ}. \tag{10}$$

For nonlinear layers, we require polynomial maps, which are based on quadratic maps. As nonlinear layers are evaluated element-wise, we only consider one-dimensional polynomial zonotopes ($n = 1$) here. Additionally, we only consider dependent generators to reduce over-approximations.

Proposition 2.9. *(Quadratic Map [23, Prop. 3.1.30]) Given $\mathcal{PZ}_1 = \langle c_1, G_1, [], E_1\rangle_{PZ}$, $\mathcal{PZ}_2 = \langle c_2, G_2, [], E_2\rangle_{PZ} \subset \mathbb{R}$ with a common identifier vector, the result of the standard quadratic map is*

$$\begin{aligned} &\texttt{sq}(\mathcal{PZ}_1, \mathcal{PZ}_2) = \\ &\left\langle c, \begin{bmatrix} c_2G_1 & c_1G_2 & \overline{G}_1 & \dots & \overline{G}_{h_1} \end{bmatrix}, [], \begin{bmatrix} E_1 & E_2 & \overline{E}_1 & \dots & \overline{E}_{h_1} \end{bmatrix} \right\rangle_{PZ}, \end{aligned} \tag{11}$$

*where $c = c_1c_2$, the generator matrix is given by $\overline{G}_j = G_{1(\cdot,j)}^T G_2$, and the exponent matrix by $\overline{E}_j = E_2 + E_{1(\cdot,j)} \cdot \mathbf{1}$, $j = 1 \dots h_1$.*

Proposition 2.10. *(Polynomial Map [23, Prop. A.1]) Given a polynomial zonotope $\mathcal{PZ} \subset \mathbb{R}$, the standard polynomial map of order $o \in \mathbb{N}_{\ge 2}$ defined by $o$ multiplications of $\mathcal{PZ}$ is given by*

$$\texttt{poly}(\mathcal{PZ}, o) = \mathcal{PZ}^o = \texttt{sq}\left(\mathcal{PZ}, \mathcal{PZ}^{o-1}\right) \tag{12}$$

*with $\mathcal{PZ}^1 = \mathcal{PZ}$.*

To evaluate the nonlinear layers element-wise, we introduce the projection and the Cartesian product to split a polynomial zonotope by dimension and recombine the separated sets afterwards.

Proposition 2.11. *(Projection [23, Prop. 3.1.16]) The projection of a polynomial zonotope $\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ} \subset \mathbb{R}^n$ on dimension $i$ is computed as*

$$\texttt{project}(\mathcal{PZ}, i) = \left\langle c_i, G_{(i,\cdot)}, G_{I(i,\cdot)}, E \right\rangle_{PZ} \subset \mathbb{R}. \tag{13}$$

Proposition 2.12. *(Cartesian Product [25, Prop. 2]) Let $\mathcal{PZ}_1 = \langle c_1, G_1, G_{I1}, E_1 \rangle_{PZ}$ and $\mathcal{PZ}_2 = \left\langle c_2, \begin{bmatrix} G_2 & \widehat{G}_2 \end{bmatrix}, \begin{bmatrix} G_{I2} & \widehat{G}_{I2} \end{bmatrix}, [E_1 \; E_2] \right\rangle_{PZ}$ with a common identifier vector, their Cartesian product is*

$$\mathcal{PZ}_1 \times \mathcal{PZ}_2 = \left\langle \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \begin{bmatrix} G_1 & \boldsymbol{0} \\ G_2 & \widehat{G}_2 \end{bmatrix}, \begin{bmatrix} G_{I1} & \boldsymbol{0} \\ G_{I2} & \widehat{G}_{I2} \end{bmatrix}, [E_1 \; E_2] \right\rangle_{PZ}. \tag{14}$$

The order $\rho$ of a polynomial zonotope is defined as $\rho = \frac{h+q}{n}$ with $h, q, n$ as in Def. 2.4. While propagating the polynomial zonotopes through the neural network, $\rho$ increases and order reduction methods become necessary to remain computationally feasible.

Proposition 2.13. *(Order Reduction [23, Prop. 3.1.39]) Given a $\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ}$ and a desired order $\rho_d > 1 + 1/n$, the operation* reduce *returns a new polynomial zonotope with an order $\rho \leq \rho_d$ that encloses $\mathcal{PZ}$:*

$$\texttt{reduce}(\mathcal{PZ}, \rho_d) = \left\langle c_z, G_{(\cdot,\widehat{\mathcal{K}})}, \begin{bmatrix} G_{I(\cdot,\widehat{\mathcal{H}})} & G_z \end{bmatrix}, E_{(\cdot,\widehat{\mathcal{K}})} \right\rangle_{PZ} \tag{15}$$

*with $\langle c_z, G_z \rangle_Z = \texttt{reduce}(\mathcal{Z}, 1)$ [23, Def. 2.6.1] and*

$$\mathcal{Z} = \texttt{zonotope}\left( \left\langle c, G_{(\cdot,\mathcal{K})}, G_{I(\cdot,\mathcal{H})}, E_{(\cdot,\mathcal{K})} \right\rangle_{PZ} \right) \tag{16}$$

*where $\mathcal{K}, \mathcal{H}$ contain the indices of the smallest generators of $G, G_I$, respectively, and $\widehat{\mathcal{H}} = \{1, \dots, h\} \backslash \mathcal{K}$, $\widehat{\mathcal{H}} = \{1, \dots, q\} \backslash \mathcal{H}$.*

## 2.4 Neural Network Verification

For the propagation of the set through the neural network, we recall the previous approaches deploying reachability analysis, specifically using zonotopes [11, 36] and polynomial zonotopes [25].

Proposition 2.14. *(Image Enclosure [25, Sec. 3]) Let $\mathcal{H}_{k-1} \supseteq \mathcal{H}^*_{k-1}$ be an input set to layer $k$, then*

$$\mathcal{H}_k = \texttt{enclose}(L_k, \mathcal{H}_{k-1}) \supseteq \mathcal{H}^*_k \tag{17}$$

*is the over-approximative output set.*

The propagation through linear layers is computed according to Prop. 2.8 and the propagation through nonlinear layers is summarized in Alg. 1 and Fig. 2: We apply the transformation of each neuron $w$ in layer $k$ sequentially by projecting it to the respective dimension in line 2. Line 3 refers to finding lower and upper bounds $l_{k-1}, u_{k-1}$ of $\mathcal{H}_{k-1}$ (Prop. 2.7). We use polynomial regression [32] to obtain an approximating polynomial $p_{k,w}$ in line 4 for the previously computed lower and upper bound. The approximation error $d_{k(w)}$ is bounded in line 5: Given a user-defined precision $\delta > 0$, we sample points $x \in \widehat{\mathcal{X}} \subset \mathbb{R}$ evenly within $l_{k-1,w}, u_{k-1,w}$ such that

$$d_{k(w)} \leq \overline{d}_{k(w)} = \max_{x \in \widehat{\mathcal{X}}} \left| \sigma_k(x) - p_{k,w}(x) \right| + \delta. \tag{18}$$

Note that sampling is not required for piecewise linear functions, for which the error can be computed exactly [25, Sec. 3.2]. In line 6, we approximate $\sigma_k$ by evaluating $\mathcal{H}_{k-1}$ on the polynomial $p_{k,w}$ (Sec. 3). Finally, line 7 introduces the over-approximation by appending $d_{k(w)}$ as an additional dependent generator to $\widetilde{\mathcal{H}}_{k,w}$. The resulting polynomial zonotopes of each neuron are then combined in line 9 using Prop. 2.12. Alt. 1 can compute tight over-approximations of nonlinear layers (see Fig. 1).

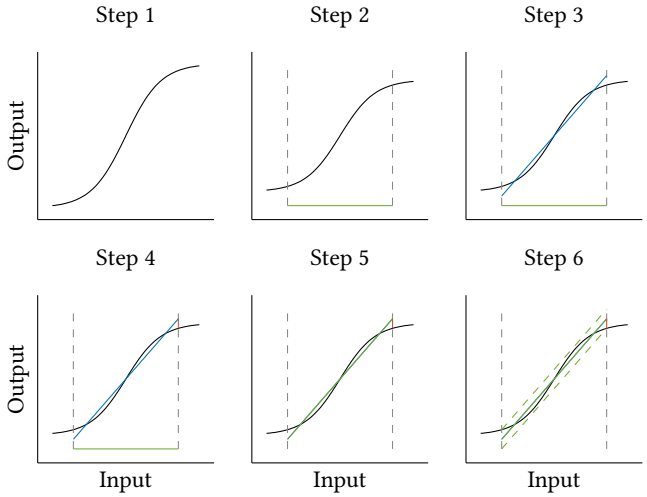


**Figure 2: Six main steps to over-approximate nonlinear layers corresponding to line 2–7 in Alg. 1.**

**Algorithm 1** Image enclosure for nonlinear layer $k$ [25]

**Require:** $\mathcal{H}_{k-1} \subset \mathbb{R}^{v_{k-1}}$, nonlinear function $\sigma_k$
1: **for** $w = 1 \dots v_{k-1}$ **do**
2: $\quad \mathcal{H}_{k-1,w} \leftarrow \texttt{project}(\mathcal{H}_{k-1}, w)$ ▷ Prop. 2.11
3: $\quad$ Find bounds $l_{k-1,w}, u_{k-1,w}$ of $\mathcal{H}_{k-1,w}$ ▷ Prop. 2.7
4: $\quad$ Find polynomial $p_{k,w}(x)$ approximating $\sigma_k(x)$ ▷ [32]
5: $\quad \overline{d}_{k(w)} \leftarrow \max_{x \in \widehat{\mathcal{X}}} \left| \sigma_k(x) - p_{k,w}(x) \right| + \delta$ ▷ (18)
6: $\quad \widetilde{\mathcal{H}}_{k,w} \leftarrow$ Evaluate $\mathcal{H}_{k-1,w}$ on $p_{k,w}(x)$ ▷ Sec. 3
7: $\quad \mathcal{H}_{k,w} \leftarrow$ Append $\overline{d}_{k(w)}$ to generators of $\widetilde{\mathcal{H}}_{k,w}$
8: **end for**
9: $\mathcal{H}_k \leftarrow \mathcal{H}_{k,1} \times \dots \times \mathcal{H}_{k,v_{k-1}}$ ▷ Prop. 2.12
10: **return** $\mathcal{H}_k$

## 2.5 Problem Statement

Given an input set $\mathcal{X}$, a neural network as specified in Def. 2.2 for which the neuron sensitivity (Def. 2.3) is computable, and an unsafe set $\mathcal{S}$, then the problem statement is to formally verify that $y \notin \mathcal{S}$ for an output $y$ of the network for all inputs $x \in \mathcal{X}$.

# 3 EFFICIENT POLYNOMIAL EVALUATION

Higher-order polynomials allow us to over-approximate the image enclosure of nonlinear layers (Alg. 1) and thus the final output set to an arbitrary precision (Fig. 1). In this section, we present a more efficient polynomial evaluation of polynomial zonotopes than a naive approach, which we derive in the appendix (Prop. A.2). We consider polynomials of the form $p(x) = a_0 + \sum_{i=1}^{o} a_i x^i$ and polynomial zonotopes of the form $\mathcal{PZ} = \langle c, G, [\,], E \rangle_{PZ} \subset \mathbb{R}$. However, the extension to higher-dimensional sets is straight-forward. The

computational complexity of the naive approach is

$$\begin{aligned} O(p(\mathcal{PZ})) + O(\texttt{compact}) &= O(ph^o) + O(ph^o \log h^o) \\ &= O(oph^o \log h), \end{aligned} \tag{19}$$

with $p, h$ as defined in Def. 2.4, $n = 1$, and `compact` (6) makes the resulting $\widetilde{\mathcal{PZ}}$ regular. Note that the computational complexity is dominated by the `compact` operation, which is necessary even in simple cases:

*Example 3.1.* Let $\mathcal{PZ} = \langle 1, [1\ 2], [], [1\ 2] \rangle_{PZ}$ and $p(x) = x + x^2$. Then,

$$\begin{aligned} p(\mathcal{PZ}) = \langle 2, [\ \overbrace{[1\ 2]}^{\mathcal{PZ}^1}\ \overbrace{[1\ 2\ 1\ 2\ 1\ 2\ 2\ 4]}^{\mathcal{PZ}^2}\ ], [], \\ [\ [1\ 2]\ [1\ 2\ 1\ 2\ 2\ 3\ 3\ 4]\ ]\ \rangle_{PZ} \end{aligned} \tag{20}$$

is non-regular due to the redundant exponents.

We present a novel approach that avoids the `compact` operation. Note that $p(\mathcal{PZ})$ boils down to a polynomial substitution by a sparse polynomial [19], for which basic arithmetic operations are well studied. Intuitively, our novel approach sorts the columns of the exponent matrix of $\mathcal{PZ}$ once and maintains this sorting throughout the polynomial evaluation:

Proposition 3.2. *(`sort`) Given a regular $\mathcal{PZ} = \langle c, G, [], E \rangle_{PZ}$, then*

$$\widehat{\mathcal{PZ}} = \langle\ c, \widehat{G}, [], \widehat{E}\ \rangle_{PZ} = \texttt{sort}(\mathcal{PZ}) \tag{21}$$

*with $\widehat{G} = G_{(\cdot,\mathcal{K})}, \widehat{E} = E_{(\cdot,\mathcal{K})}$, where $\mathcal{K}$ is a permutation of $\{1, \dots, h\}$ such that $\widehat{E}_{(\cdot,i)} \leq \widehat{E}_{(\cdot,i+1)}$, $i = 1 \dots h-1$, and $\leq$ refers to the lexicographic ordering. The computational complexity is $O(ph \log h)$.*

Proof. The computational complexity follows from the sorting of the columns. □

We introduce the operation `sortPlus` to efficiently sum a list of sorted polynomial zonotopes. For a convenient definition of the `sortPlus` operation, we additionally introduce the `merge` operation:

Proposition 3.3. *(`merge`) Given a sorted non-regular $\mathcal{PZ} = \langle c, G, [], E \rangle_{PZ}$, then $\widehat{\mathcal{PZ}} = \texttt{merge}(\mathcal{PZ})$ is sorted and regular with a computational complexity of $O(ph)$.*

Proof. $\widehat{\mathcal{PZ}}$ is computed by iterating over the $h$ columns of $E$ and merging subsequent equal columns. □

Proposition 3.4. *(`sortPlus`) Given $m$ sorted $\mathcal{PZ}_1, \dots, \mathcal{PZ}_m$ with $\mathcal{PZ}_i = \langle c_i, G_i, [], E_i \rangle_{PZ} \subset \mathbb{R}$, a common identifier vector, and a total number of $\widehat{h}$ generators. Then,*

$$\begin{aligned} \widetilde{\mathcal{PZ}} &= \texttt{sortPlus}\ (\mathcal{PZ}_1, \dots, \mathcal{PZ}_m) \\ &= \texttt{merge}\left(\left\langle \widehat{c}, \widehat{G}_{(\cdot,\mathcal{K})}, [], \widehat{E}_{(\cdot,\mathcal{K})} \right\rangle_{PZ}\right) \end{aligned} \tag{22}$$

*computes the Minkowski sum with the common identifier vector explicitly considered. $\widetilde{\mathcal{PZ}}$ is regular and sorted, where $\widehat{c} = \sum_{i=1}^{m} c_i$, $\widehat{G} = [G_1, \dots, G_m], \widehat{E} = [E_1, \dots, E_m]$, and $\mathcal{K}$ contains the indices such that $\widehat{E}_{(\cdot,\mathcal{K})}$ is sorted. The computational complexity of* `sortPlus` *is $O(\widehat{ph} \log m)$.*

Proof. The operation `sortPlus` computes the same set as the exact addition by construction (Prop. A.1). The sorted indices $\mathcal{K}$ can be computed using a min-heap [19, Sec. 3.2], where we only store the first unprocessed column of each $E_i$ in the heap as all $\mathcal{PZ}_i$ are sorted. Then, we can repeatedly remove the current minimum from the heap and add the next column of the corresponding $E_i$ into the heap in $O(p \log m)$ until all $\widehat{h}$ columns are processed. □

**Algorithm 2** Efficient Polynomial Evaluation

**Require:** $\mathcal{PZ} = \langle c, G, [], E \rangle_{PZ} \subset \mathbb{R},\ p(x) = a_0 + \sum_{i=1}^{o} a_i x^i$

1: $\overline{\mathcal{PZ}}^1 \leftarrow \texttt{sort}(\mathcal{PZ})$ ▷ Prop. 3.2
2: **for** $i = 2 \dots o$ **do** ▷ Compute sorted $\mathcal{PZ}^i$
3: $\quad \mathcal{PZ}^i \leftarrow \texttt{sq}\left(\overline{\mathcal{PZ}}^1, \overline{\mathcal{PZ}}^{i-1}\right)$ ▷ Prop. 2.10
4: $\quad [G_1\ G_2\ [G_3\ \dots\ G_{h+2}]] \leftarrow \mathcal{PZ}^i.G$ ▷ Prop. 2.9
5: $\quad [E_1\ E_2\ [E_3\ \dots\ E_{h+2}]] \leftarrow \mathcal{PZ}^i.E$
6: $\quad$ **for** $j = 1 \dots h+2$ **do**
7: $\quad\quad \mathcal{PZ}^i_j \leftarrow \langle 0, G_j, [], E_j \rangle_{PZ}$ ▷ $\mathcal{PZ}^i_j$ is sorted
8: $\quad$ **end for**
9: $\quad c^i \leftarrow \mathcal{PZ}^i.c$
10: $\quad \overline{\mathcal{PZ}}^i \leftarrow c^i + \texttt{sortPlus}(\mathcal{PZ}^i_1, \dots, \mathcal{PZ}^i_{h+2})$ ▷ Prop. 3.4
11: **end for**
12: **for** $i = 1 \dots o$ **do**
13: $\quad \overline{\mathcal{PZ}}^i \leftarrow a_i \cdot \overline{\mathcal{PZ}}^i$ ▷ Prop. 2.8
14: **end for**
15: $\widetilde{\mathcal{PZ}} \leftarrow a_0 + \texttt{sortPlus}\left(\overline{\mathcal{PZ}}^1, \dots, \overline{\mathcal{PZ}}^o\right)$ ▷ Prop. 3.4

Proposition 3.5. *(Efficient Polynomial Evaluation) Given a polynomial zonotope $\mathcal{PZ} = \langle c, G, [], E \rangle_{PZ} \subset \mathbb{R}$ and a polynomial $p(x) = a_0 + \sum_{i=1}^{o} a_i x^i$, Alg. 2 computes a sound polynomial evaluation $\widetilde{\mathcal{PZ}} = p(\mathcal{PZ})$ where $\overline{\mathcal{PZ}}$ is regular, sorted, and has $O(h^o)$ generators. The computational complexity is $O(ph^o(\log h + \log o))$.*

Proof. All subsequent line numbers are with respect to Alg. 2.

*Soundness.* Alg. 2 computes $a_i\overline{\mathcal{PZ}}^i$ soundly as in the naive approach (Prop. A.2), but each $a_i\overline{\mathcal{PZ}}^i$ is regular and sorted: $\overline{\mathcal{PZ}}^1$ is sorted and regular (line 1). For $i = 2$ each $E_j$ of $PZ^i$ (lines 3–5) is sorted per construction (Prop. 2.9). Thus, `sortPlus` computes a regular and sorted $\overline{\mathcal{PZ}}^i$ because $\overline{\mathcal{PZ}}^1, \overline{\mathcal{PZ}}^{i-1}$ are regular and sorted (line 10), and per induction, it holds $\forall i \in \{1, \dots, o\}$. The multiplication with $a_i$ does not change the exponents (line 13). Finally, `sortPlus` sums all $\overline{\mathcal{PZ}}^i$ and the resulting $\widetilde{\mathcal{PZ}}$ is regular and sorted because each $\overline{\mathcal{PZ}}^i$ is regular and sorted (Prop. 3.4).

*Number of Generators.* Follows from Prop. A.2.

*Computational Complexity.* The initial sorting is in $O(ph \log h)$ (line 1). For $i = 2 \dots o$ holds: $\mathcal{PZ}^i$ can be computed in $O(ph^i)$ with $O(h^i)$ generators (line! 3), and `sortPlus` (line 10) in $O(ph^i \log h)$. Thus, the computation of all $\overline{\mathcal{PZ}}^i$ is in $O(ph^o \log h)$. The construction of all $a_i\overline{\mathcal{PZ}}^i$ is in $O(h^o)$. Finally, as we have a total number of $O(h^o)$ generators and $o$ sorted polynomial zonotopes, the `sortPlus` operation in line 15 is in $O(ph^o \log o)$. Thus, the computational complexity of Alg. 2 is $O(ph^o(\log h + \log o))$. □

Our novel approach reduces the computational complexity from $O(oph^o \log h)$ to $O(ph^o(\log o + \log h))$. As the polynomial order $o$ is typically much smaller than the number of generators $h$, our approach is linearly faster in the polynomial order than the naive approach.

# 4 AUTOMATIC ABSTRACTION REFINEMENT APPROACH

In this section, we present our novel approach for automatically refining the abstraction of neurons to balance precision and computational feasibility, which is also summarized in Alg. 3. The refinement is performed by increasing the order of the approximation polynomial used in Prop. 2.14. The order of the approximation polynomial used in each neuron of a nonlinear layer $k$ is stored in an order pattern tuple

$$\theta = (\theta_2, \theta_4, \dots, \theta_K), \quad \theta_k \in \mathbb{N}_0^{v_k}. \tag{23}$$

To remain computationally feasible using higher-order polynomials, we apply an order reduction in Alg. 3 (line 5) before propagating the set through each layer (line 6), thus limiting the number of generators to $h_{\max}$:

$$\begin{aligned} \rho_k^{(\text{pre})} &= h_{\max}^{1/\max \theta_k} / v_k \\ \implies \rho_k^{(\text{post})} &\approx h_{\max}/v_k. \qquad \text{(Prop. 3.5)} \end{aligned} \tag{24}$$

Sec. 4.1 shows how the expensive range bounding during the image enclosure for nonlinear layers (Alg. 1, line 3) from previous iterations can be reused for later computations. In Sec. 4.2, we indicate how many neuron abstractions are refined per iteration. Sec. 4.3 defines multiple heuristics to determine the neurons, for which we refine the abstraction. Finally, we provide the properties of our novel approach in Sec. 4.4.

**Algorithm 3** Automatic Abstraction Refinement

**Require:** $\mathcal{X}$, $\mathcal{S}$, $K$ layers $L_k$, $k = 1 \dots K$, max. iteration $I$

1: Initialize order pattern $\theta$ with 1
2: **for** $i = 1 \dots I$ **do** ▷ Iteration $i$
3: $\quad \mathcal{H}_0^{(i)} \leftarrow \mathcal{X}$
4: $\quad$ **for** $k = 1 \dots K$ **do** ▷ Network Evaluation
5: $\quad\quad \mathcal{H}_{k-1}^{(i)} \leftarrow \texttt{reduce}\left(\mathcal{H}_{k-1}^{(i)}, \rho_k^{(\text{pre})}\right)$ ▷ Prop. 2.13, (24)
6: $\quad\quad \mathcal{H}_k^{(i)} \leftarrow \overline{\texttt{enclose}}\left(L_k, \mathcal{H}_{k-1}^{(i)}\right)$ ▷ Sec. 4.1
7: $\quad$ **end for**
8: $\quad \mathcal{Y}^{(i)} \leftarrow \mathcal{H}_K^{(i)}$ ▷ $\mathcal{Y}^{(i)} \supseteq \mathcal{Y}^*$
9:
10: $\quad$ **if** $\mathcal{Y}^{(i)} \cap \mathcal{S} = \emptyset$ **then** ▷ Check Specifications
11: $\quad\quad$ **return** VERIFIED
12: $\quad$ **end if**
13: ▷ Refine Abstraction
14: $\quad H \leftarrow$ Compute refinement heuristics ▷ Sec. 4.3
15: $\quad$ Refine neuron abstractions by increasing $\theta$ ▷ Sec. 4.2
16: **end for**
17: **return** UNKNOWN

## 4.1 Reuse of Bounds

During the execution of Alg. 1, we need to compute the bounds of $\mathcal{H}_{k-1,w}$ (line 3) of a neuron $w$ in the $k$-th layer. Tight range bounding is in general computationally expensive for polynomial zonotopes due to recursive splitting [23, Prop. 3.1.44][29]. On the other hand, the exact bounds of zonotopes can be computed efficiently (Prop. 2.7). Unfortunately, the zonotope enclosure (Prop. 2.6) removes all dependencies between generators stored in the exponent matrix, which can lead to a very conservative over-approximation as shown in the following example.

*Example 4.1.* Given a $\mathcal{PZ} = \left\langle 0, [1\ -1], [], \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} \right\rangle_{PZ}$. Using Prop. 2.6 we obtain $\mathcal{Z} = \texttt{zonotope}(\mathcal{PZ}) = \langle 0, [1\ -1] \rangle_Z$. The exact interval enclosure of $\mathcal{PZ}$ is $[-1, 1]$, whereas the exact interval enclosure of $\mathcal{Z}$ is $[-2, 2]$ (Prop. 2.7).

As new dependencies are introduced with each refinement step (Prop. 2.10), we reuse bounds from previous iterations for later computations. Specifically, we use

$$\begin{aligned} \underline{l}_{k-1(w)}^{(i)} &= \max\left(l_{k-1(w)}^{(1)}, \dots, l_{k-1(w)}^{(i)}\right), \\ \overline{u}_{k-1(w)}^{(i)} &= \min\left(u_{k-1(w)}^{(1)}, \dots, u_{k-1(w)}^{(i)}\right), \end{aligned} \tag{25}$$

where $l_{k-1(w)}^{(j)}, u_{k-1(w)}^{(j)}$ are the conservative bounds of the input $\mathcal{H}_{k-1,w}^{(j)}$ using the zonotope enclosure in iteration $j = 1 \dots i$. We indicate the use of $\underline{l}_{k-1(w)}^{(i)}, \overline{u}_{k-1(w)}^{(i)}$ in Alg. 1 by $\overline{\texttt{enclose}}$.

Proposition 4.2. *(Reuse of Bounds) Let $\mathcal{H}_{k-1}^{(i)}$ be the input of layer $k$ in iteration $i$. If $\mathcal{H}_{k-1}^* \subseteq \mathcal{H}_{k-1}^{(j)}, \forall j \in \{1, \dots, i\}$, then it holds*

$$\mathcal{H}_k^* \subseteq \mathcal{H}_k^{(i)} = \overline{\texttt{enclose}}\left(L_k, \mathcal{H}_{k-1}^{(i)}\right). \tag{26}$$

Proof. From Prop. 2.14 it follows that all points within the bounds $\underline{l}_{k-1(w)}^{(i)}, \overline{u}_{k-1(w)}^{(i)}$ are enclosed. Thus, (26) holds as $\underline{l}_{k-1(w)}^{(i)}$, $\overline{u}_{k-1(w)}^{(i)}$ are bounds of $\mathcal{H}_{k-1}^*$ by construction (25). □

In practice, many specifications of the input set are based on intervals [3, 20] with no dependencies between the dimensions. In the first iteration, our Alg. 3 does not introduce additional dependencies between generators as we only use linear abstractions (line 1). Thus, we can efficiently compute the exact bounds of the linear over-approximations $\mathcal{H}_{k-1}^{(1)}, k = 1 \dots K$, using Prop. 2.7 and reuse them in all subsequent iterations (Sec. 4.1). Although $\mathcal{H}_{k-1}^{(1)}$ might not enclose $\mathcal{H}_{k-1}^*$ tightly, the bounds are usually sufficiently viable for polynomials to approximate the activation function well. As lower-order polynomials can approximate sigmoid and tanh well, our approach can compute particularly tight over-approximations for networks using these activations in just a few iterations.

## 4.2 Number of Refinements

Evaluating the over-approximation of the output set is the most expensive step of our approach (Prop. 3.5). Thus, we want to limit the number of iterations by refining multiple neuron abstractions

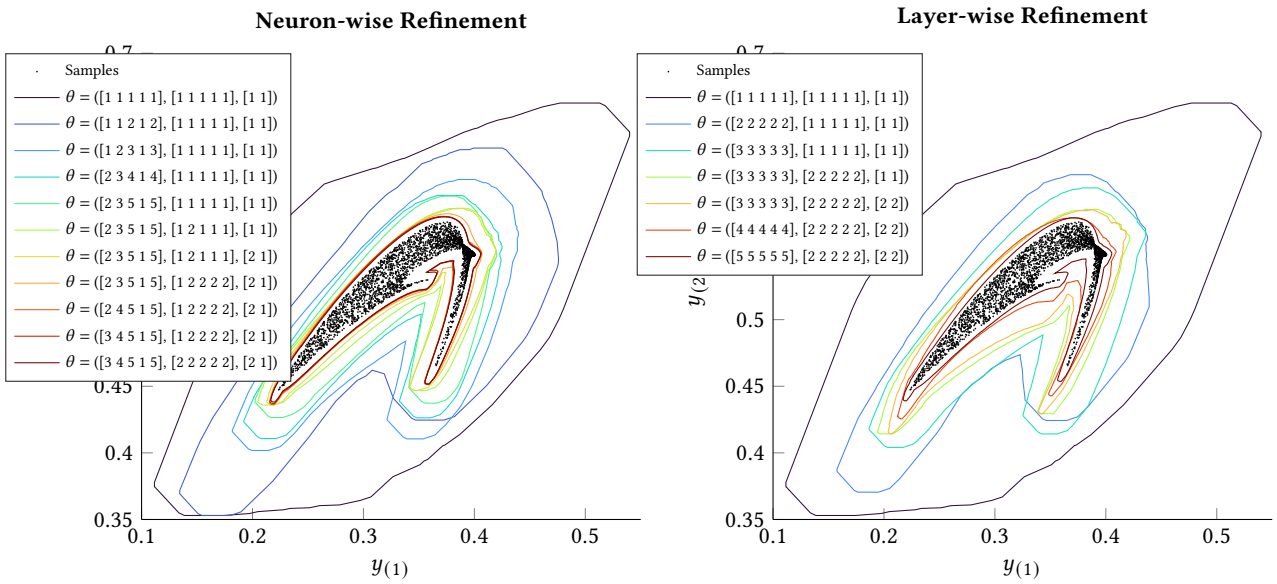


**Figure 3: Automatic abstraction refinement on a random neural network with [2, 5, 5, 2] neurons. The figure shows the output of the network and how each refinement step reduces the over-approximation $\mathcal{Y} \supseteq \mathcal{Y}^*$.**

before recomputing the over-approximation. Our approach automatically determines which neuron abstractions should be refined based on a refinement heuristic $H$ as defined in Sec. 4.3. Let us first discuss the number of generators of $\mathcal{H}_k$ after each layer $k$:

Lemma 4.3. *(Number of Generators) Given a nonlinear layer $k$, an input set $\mathcal{H}_{k-1}$, and an order pattern $\theta_k$ of that layer, then the number of generators of the output $\mathcal{H}_k = \texttt{enclose}(\mathcal{H}_{k-1})$ is determined by the maximum order in $\theta_k$.*

Proof. Follows from Alg. 1, line 9 and Prop. 2.12. □

Thus, we limit the number of order reductions by refining only neuron abstractions within one layer per iteration. We present a neuron-wise and a layer-wise refinement approach. Let us start with the neuron-wise refinement:

*Definition 4.4. (Neuron-wise Refinement)* Given a refinement heuristic $H$, let $k^*, w^* = \arg\max_{k,w} H_{k(w)}$ be the neuron within layer $k^*$ with the maximum heuristic $H^* = H_{k^*(w^*)}$. We refine all abstractions of neurons $w$ of layer $k^*$ for which

$$H_{k^*(w)} \geq \gamma H^*, \ \gamma \in [0, 1]. \tag{27}$$

*Example 4.5.* In the example shown in Fig. 3, $k^* = 2$ and $w^* = 3$ in the first iteration. Thus, with $\gamma = 0.5$ we refine the abstraction of the neurons 3 and 5 of layer 2 by increasing the respective order in $\theta_{k^*}$.

Due to Lemma 4.3, we can also set the order of the approximation polynomial of all neurons of a layer $k$ to the maximum of $\theta_k$ without increasing the overall number of generators:

*Definition 4.6. (Layer-wise Refinement)* Given a refinement heuristic $H$, we refine all neuron abstractions of layer $k^*$, where

$$k^* = \arg\max_k \|H_k\|_2. \tag{28}$$

*Example 4.5. (cont.)* Using the layer-wise refinement approach, $k^* = 2$. Thus, we increase the order of the approximation polynomial in $\theta_2$ for all neurons in layer 2. This is also shown in Fig. 3 (right, second order pattern in legend).

However, refining the neuron abstraction also increases the time to compute the image enclosure (Prop. 3.5). We evaluate this trade-off in Sec. 5 by comparing the neuron-wise refinement with the layer-wise refinement. Note that we might also refine neuron abstractions unnecessarily using the layer-wise refinement as can be seen in Fig. 3: The abstraction of neuron 4 in layer 2 is not refined using the neuron-wise refinement but still obtains a similar over-approximation of the output set as the layer-wise refinement in the last iteration.

## 4.3 Refinement Heuristics

A refinement heuristic $H_{k(w)} \in \mathbb{R}$ should provide information on how beneficial it is to refine the abstraction of a neuron $w$ in a nonlinear layer $k$, that is the largest reduction of the over-approximation, which we again store in a tuple:

$$H = (H_2, H_4, \ldots, H_K), \quad H_k \in \mathbb{R}^{v_k}. \tag{29}$$

Which neuron abstraction refinement has the largest impact on the reduction of the over-approximation is non-trivial, thus, we define multiple refinement heuristics in this section that are later evaluated in Sec. 5, specifically a novel heuristic derived from sensitivity analysis.

*Definition 4.7. (H-All)* The heuristic considers all neurons to be equally beneficial:

$$H^{(A)}_{k(w)} = 1. \tag{30}$$

*Definition 4.8. (H-Random)* The heuristic is randomly sampled between 0 and 1:

$$H^{(R)}_{k(w)} \sim \text{Uniform}(0,1). \tag{31}$$

*Definition 4.9. (H-Approximation Error)* The approximation error $d$ is used as a heuristic:

$$H^{(AE)}_{k(w)} = d_{k(w)}. \tag{32}$$

*Definition 4.10. (H-Layer Bias)* The heuristic has a bias towards earlier layers as later layers depend on good results from earlier layers:

$$H^{(LB)}_{k(w)} = d_{k(w)}/k. \tag{33}$$

*Definition 4.11. (H-Sensitivity)* The heuristic is based on sensitivity analysis:

$$\begin{aligned} H^{(S)}_{k(w)} &= s_{k(w)} \\ \text{with } s_{k(w)} &= \|S_{k(w,\cdot)}\|_2. \end{aligned} \tag{34}$$

As the heuristic $H_{k(w)}$ must be a scalar value, we reduce the sensitivity $S_{k(w,\cdot)}$ (3) of a neuron $w$ in layer $k$ on all output neurons to a scalar value using the Euclidean norm. We draw the evaluation point randomly from the input set.

*Definition 4.12. (H-Weighted Error)* The heuristic weights the approximation error using sensitivity analysis:

$$H^{(E+S)}_{k(w)} = d_{k(w)} \cdot s_{k(w)}. \tag{35}$$

## 4.4 Properties

Finally, we state the properties of Alg. 3. For simplicity, we only consider layer-wise refinements.

Proposition 4.13. *(Layer Convergence) Given a neural network with two layers, that is $f(x) = \sigma(Wx + b)$, and an input set $\mathcal{X}$, then for $I \to \infty, \delta \to 0$, and $h_{\max} \to \infty$, it holds that the output of the network $\mathcal{Y}^{(I)}$ converges to $\mathcal{Y}^*$ using Alg. 3.*

Proof. The linear layer is computed exactly. It remains to show that the nonlinear layer does not induce an over-approximation, where the over-approximation could be induced from the order reduction, the polynomial evaluation, and the approximation error. From $h_{\max} \to \infty$ it follows that no order reduction is applied (24). Polynomial zonotopes can be evaluated exactly on polynomials (Prop. 2.10). For a neuron $w = 1 \dots v_1$, the approximation error $d_{(w)}$ is bounded by $\overline{d}_{(w)}$ (18) using a discrete set $\widehat{\mathcal{X}}$ within the bounds $\overline{l}_{(w)}, \overline{u}_{(w)}$, where $\overline{d}_{(w)}$ has to converge to 0 for no induced over-approximation, and thus $\mathcal{Y}^{(I)}$ to converge to $\mathcal{Y}^*$. From (18) it also follows that

$$d^{(i)}_{(w)} \leq \overline{d}^{(i)}_{(w)} \leq d^{(i)}_{(w)} + \delta. \tag{36}$$

We find the coefficients of the approximation polynomial $p_w(x)$ by regression, which minimizes the sum of squared differences (*SSD*) between $p_w(x)$ and $\sigma$ over $\widehat{\mathcal{X}}$ [6, Sec. 3.1.1]. With the number of iterations $I \to \infty$, it follows that $\theta_{(w)} \to \infty$ (Alg. 3, line 15) and thus the order of $p_w(x)$ (23). Together with the bounds $\overline{l}_{(w)}, \overline{u}_{(w)}$ and $\sigma$ being continuous, from [39, Thm. 15] it follows that $SSD \to 0$. Thus, also $d^{(I)}_{(w)} \to 0$ and $0 \leq \overline{d}^{(I)}_{(w)} \leq \delta$ holds due to (36). With $\delta \to 0, \overline{d}^{(I)}_{(w)} \to 0$ and thus the over-approximation $\mathcal{Y}^{(I)}$ converges to $\mathcal{Y}^*$. □

Theorem 4.14. *(Network Convergence) Alg. 3 is sound and the computed over-approximation $\mathcal{Y}^{(I)}$ converges to $\mathcal{Y}^*$ for $I \to \infty$, $\delta \to 0$, and $h_{\max} \to \infty$ if the approach uses a heuristic $H$ that is proportional to the approximation error $d$. Thus, our approach can verify the problem statement in Sec. 2.5 with an arbitrarily small over-approximation.*

Proof. *Soundness and Termination.* The approach is sound because each operation is over-approximative (Prop. 4.2). Further, if $I \in \mathbb{N}$, the algorithm terminates in finite time as each operation needs finite time and the maximum number of iterations is $I$.

*Convergence.* We give a proof for the heuristic $H^{(AE)}$ (Def. 4.9). An analogous proof can be given for $H^{(LB)}$ (Def. 4.10) and $H^{(S+E)}$ (Def. 4.12). Consider each two subsequent layers as a separate subnetwork as in Prop. 4.13. In each iteration, Alg. 3 refines the nonlinear layer $k$ with the largest approximation error (Def. 4.6). We show that from Prop. 4.13 it follows that $\overline{d}^{(i)}_{k(w)} \to 0 \; \forall k$ by contradiction: If there were a neuron $w_1$ of a nonlinear layer $k_1$ with $\overline{d}^{(I)}_{k_1(w_1)} > 0$, then the layer $k_1$ would not have been refined infinitely often because of Prop. 4.13. Thus, a neuron $w_2$ of another nonlinear layer $k_2$ needs to exist with $\overline{d}^{(I)}_{k_2(w_2)} > \overline{d}^{(I)}_{k_1(w_1)} > 0$ which got refined infinitely often because of $I \to \infty$ and the finite number of layers $K$. However, such a layer cannot exist because of Prop. 4.13. Thus,

$$\mathcal{H}^*_k = \mathcal{H}^{(I)}_k = \overline{\texttt{enclose}}(\mathcal{H}^{(I)}_{k-1}), \quad k = 1 \dots K, \tag{37}$$

and $\mathcal{H}^{(I)}_K = \mathcal{Y}^*$ for $\mathcal{H}^{(I)}_0 = \mathcal{X}$. □

# 5 EVALUATION

Our approach is implemented in the MATLAB toolbox CORA [2] and will be made publicly available with the next release. All computations were performed on an Intel® Core™ Gen. 11 i7-11800H CPU @2.30GHz with 64GB memory.

## 5.1 Comparison of Refinement Heuristics

We compare the heuristics by evaluating the resulting outputs $\mathcal{Y}^{(i)}$ of 10 neural networks with $K$ layers, randomly initialized weights, and sigmoid activation. The networks have 2 input neurons, 2 output neurons, and 50 neurons in each intermediate layer. The input set $\mathcal{X} = [-\mathbf{1}, \mathbf{1}] \subset \mathbb{R}^2$ is used for all networks. We intentionally choose a two-dimensional output of the networks to compute a tight over-approximation of the size of the output $\mathcal{Y}^{(i)}$ by recursive splitting, where the size of $\mathcal{Y}^{(i)}$ is given by the area covered in $\mathbb{R}^2$. The area of $\mathcal{Y}^{(i)}$ is normalized, where we define the area of $\mathcal{Y}^{(1)}$ as 1 per network. Our approach refines the output until $I = 15$ or the set explodes due to order reduction. The maximum number of generators is limited to $h_{\max} = 10^6$ and the largest used polynomial order is 5. For the neuron-wise refinement, we choose $\gamma = 0.1$ to refine many neuron abstractions in each iteration because we find this setting obtains good results. Additionally, we present the results using a naive approach, where we refine all neuron abstractions after each iteration and do not apply Prop. 4.2. Refining all

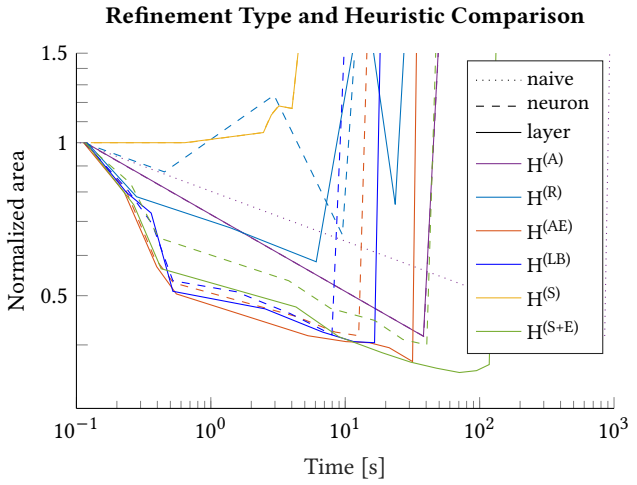


**Figure 4: Average change in normalized area over computation time on 10 random neural networks ($K = 14$) with sigmoid activation (double-log scale).**

**Table 1: Comparison of refinement heuristics: Number of times the respective heuristic leads to the tightest over-approximation, i.e. the smallest area, per number of layers $K$.**

| $K$ | $H^{(A)}$ | $H^{(R)}$ | $H^{(AE)}$ | $H^{(LB)}$ | $H^{(S)}$ | $H^{(S+E)}$ |
|---|---|---|---|---|---|---|
| **4** | 0 | **5** | **5** | **5** | 0 | 1 |
| **6** | 0 | 0 | 4 | 0 | 0 | **6** |
| **8** | 0 | 0 | 0 | 0 | 0 | **10** |
| **10** | 0 | 0 | 2 | 0 | 0 | **8** |
| **12** | 0 | 0 | 1 | 0 | 0 | **9** |
| **14** | 0 | 0 | 1 | 0 | 0 | **9** |
| **16** | 0 | 0 | 0 | 0 | 0 | **10** |

neuron abstractions while applying Prop. 4.2 is essentially carried out using the heuristic $H^{(A)}$.

Fig. 4 shows the averaged results for all refinement heuristic and type combinations on 10 networks with $K = 14$. While all heuristics utilizing the approximation error work well, we find that using $H^{(S+E)}$ results in the smallest area and more refinement steps are possible before the set explodes due to order reduction. While our approach using $H^{(AE)}$ tends to refine neuron abstractions primarily in earlier layers, the refinements are better distributed across all layers using $H^{(S+E)}$. Thus, $H^{(S+E)}$ considers a relatively large approximation error in an earlier layer to be less important than a smaller error in later layers, if such an error contributes more to the output. We observe similar results for different network sizes (Tab. 1), where the benefit of $H^{(S+E)}$ increases with the size of the network. Additionally, our approach is up to a thousand times faster and obtains tighter results than the naive approach for deep neural networks (Fig. 4). Note that the naive approach has to use splitting to obtain viable bound estimates to compute tight $\mathcal{Y}^{(i)}$ after the first iteration (Ex. 4.1). Lastly, we find that refining the abstraction

**Table 2: Examples for increased perturbation radii $r$ per image, where `idx` refers to the corresponding vnnlib specification file. The original radius was $r = 0.012$ for all images.**

| `idx` | 3370 | 5087 | 3867 | 5747 | 7142 | 2683 |
|---|---|---|---|---|---|---|
| $r$ | 0.0167 | 0.0145 | 0.0138 | 0.0132 | 0.0257 | 0.0125 |

of all neurons in a layer usually obtains better results. Thus, in our current implementation there is no benefit in refining at the neuron level, even for small $\gamma$.

## 5.2 Open-Loop Verification Benchmark

We show the applicability of our approach on the eran benchmark from the VNN'21 competition [3]. This benchmark takes correctly classified images from the MNIST handwritten digits dataset. The images have $28 \times 28$ pixels with 10 possible labels for the digits $0 - 9$. The goal of the benchmark is to verify that these images are still classified correctly with a $l_\infty$-norm perturbation of at most $r = 0.012$. Thus, for a flattened image $c \in \mathbb{R}^{784}$ we construct an input set

$$\mathcal{X} = \langle c, r \cdot I_{784}, [\,], I_{784} \rangle_{PZ}\,, \tag{38}$$

where $I_n$ is the identity matrix of dimension $n$. Let $w_0 \in \{1, \ldots, 10\}$ be the dimension of the correct label, then the unsafe set is

$$\mathcal{S} = \left\{ y \in \mathbb{R}^{10} \,\middle|\, \exists w \in \{1, \ldots, 10\} : y_{(w)} > y_{(w_0)} \right\}. \tag{39}$$

The network has $[784, 200, 200, 200, 200, 200, 200, 10]$ neurons ($K = 14$) and sigmoid activation. Further, we apply some smaller verification tricks as described in Appendix B.

To demonstrate our approach, we take 10 images that we can verify using linear abstractions in all neurons. For these images, we increase the perturbation radius $r$ until they can no longer be verified using linear abstractions in all neurons. Our approach is still able to verify this more challenging version of the benchmark using the increased perturbation radius $r$. An example is shown in Fig. 5. Examples for increased perturbation radii $r$ per image are provided in Tab. 2. The average computation time is 48s, where the original eran benchmark has a timeout limit of 5 minutes.

## 5.3 Closed-Loop Verification Benchmark

We additionally show that our approach is also applicable for closed-loop verification: A neural network controls an actor in a continuous dynamic system with uncertain initial state. The goal is to verify a given specification for a given time horizon, where the neural-network controller is evaluated every $\Delta t$. The controller receives the current system state as input and the output is used as a static input to the system dynamics for the next time interval. We demonstrate our automatic refinement approach on the QUAD benchmark [5] of the ARCH'22 competition[2]. The goal of the benchmark is to show that a neural-network-controlled quadrotor stabilizes within 5 seconds within a given goal set. This benchmark is particularly difficult to verify as it requires a tight over-approximative computation of the network's output set. To the best of our knowledge, the only other tool able to verify this benchmark is POLAR [16].

[2] https://cps-vo.org/group/ARCH/FriendlyCompetition

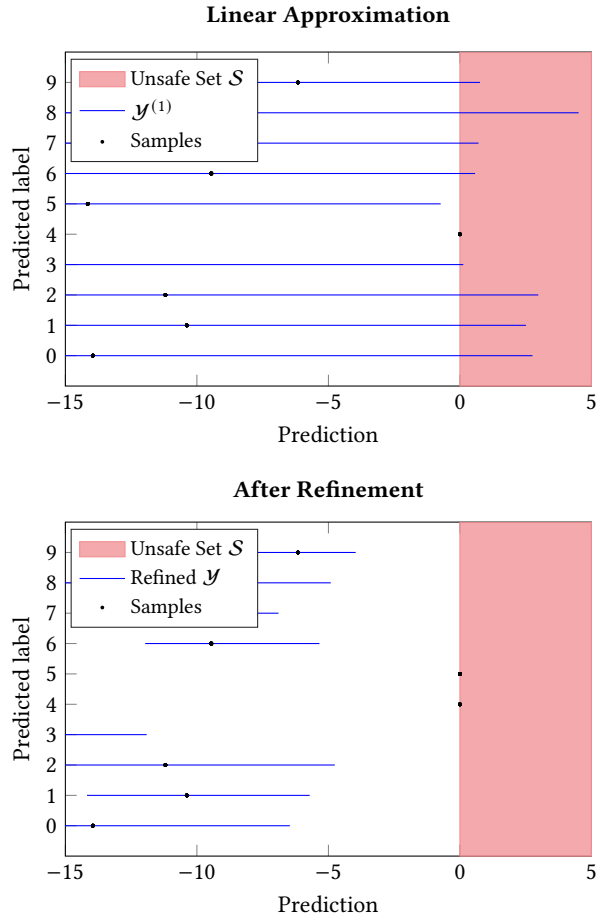


**Figure 5: Example verification of an MNIST image (`idx` = 2683) with correct label 4. The figures show the bounds of the prediction for each label, where we applied the argmax trick (Prop. B.2). Already verified dimensions are not further considered in later iterations and thus appear at 0 (Prop. B.3).**

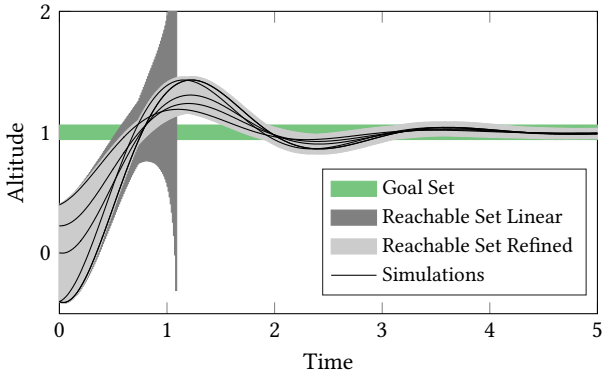


**Figure 6: Our approach is able to verify the QUAD benchmark after one refinement step.**

**Table 3: Comparison of the closed-loop QUAD benchmark.**

| Tool | Approach | Result | Time |
|---|---|---|---|
| POLAR [16] | Taylor Models | `VERIFIED` | 1533s |
| CORA [25] | Polynomial Zonotopes | `UNKNOWN` | 15s |
| Ours | Abstraction Refinement | `VERIFIED` | 296s |

We refine the abstraction of the neural network based on the input set and reuse the returned order pattern throughout the entire time horizon. We use $H^{(S+E)}$ as a heuristic and layer-wise refinement. The result of our approach is shown in Fig. 6 (compare with [16, Fig. 5a]). While we are unable to verify the benchmark using linear approximation in all layers due to a set explosion after a few time steps, one refinement iteration is enough to verify the benchmark, where the first nonlinear layer is refined. Our approach is five times faster than POLAR (Tab. 3) and obtains tighter results. This is a promising result for future research in this direction.

## 6 CONCLUSION

In this paper, we present a novel automatic abstraction refinement approach for the set-based verification of neural networks based on a novel heuristic derived from sensitivity analysis. To the best of our knowledge, our approach is the first approach to automatically refine the abstraction without splitting sets, where the abstraction refinement is done at the neuron level. Our approach works on a variety of common activation functions including ReLU, sigmoid, and tanh. We find that the image enclosure of networks with sigmoid and tanh activation is particularly tight after a few iterations as they can be approximated well using lower-order polynomials. Further, for the first time, we present a method for reusing expensive bound computations in nonlinear layers to save computation time. Additionally, we present a more efficient approach for evaluating polynomial zonotopes on polynomials than a naive approach. In our evaluation, we demonstrate the applicability of our approach to open-loop and closed-loop verification. Our approach can compute a tight over-approximation of the output set in a few iterations, allowing a fast verification of the specifications. Since splitting sets is particularly computationally expensive in closed-loop settings, we expect further progress in closed-loop verification in future work using our approach.

## ACKNOWLEDGMENTS

The authors gratefully acknowledge partial financial support from the Munich Center for Machine Learning (MCML) under the project number MCML22B and the project FAI funded by the Deutsche Forschungsgemeinschaft (DFG) under the project number 286525601.

## REFERENCES

[1] Matthias Althoff. 2010. *Reachability analysis and its application to the safety assessment of autonomous cars.* Ph. D. Dissertation. Technische Universität München.
[2] Matthias Althoff. 2015. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems.* 120–151.
[3] Stanley Bak, Changliu Liu, and Taylor T. Johnson. 2021. The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. *arXiv preprint arXiv:2109.00498* (2021).
[4] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring neural net robustness with constraints. *Advances in Neural Information Processing Systems* 29 (2016).

[5] Randal W. Beard. 2008. Quadrotor dynamics and control. *Brigham Young University* 19, 3 (2008), 46–56.

[6] Christopher M. Bishop and Nasser M. Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[7] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. 2019. JuliaReach: A toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*.

[8] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. 2020. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3291–3299.

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

[10] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2000. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*. Springer, 154–169.

[11] 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.

[12] Antoine Girard. 2005. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 291–305.

[13] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

[14] Patrick Henriksen and Alessio Lomuscio. 2020. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*. IOS Press, 2513–2520.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (1989), 359–366.

[16] Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. 2022. POLAR: A polynomial arithmetic framework for verifying neural-betwork controlled systems. In *Automated Technology for Verification and Analysis*. Springer International Publishing, 414–430.

[17] 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.

[18] Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. 2021. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *International Conference on Computer Aided Verification*. Springer, 249–262.

[19] Stephen C. Johnson. 1974. Sparse polynomial arithmetic. *ACM SIGSAM Bulletin* 8, 3 (1974), 63–71.

[20] Taylor T. Johnson, Diego Manzanas Lopez, Luis Benet, Marcelo Forets, Sebastián Guadalupe, Christian Schilling, Radoslav Ivanov, Taylor J. Carpenter, James Weimer, and Insup Lee. 2021. ARCH-COMP21 category report: Artificial intelligence and neural network nontrol systems (AINNCS) for continuous and hybrid systems plants. *EPiC Series in Computing* 80 (2021), 90–119.

[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.

[22] 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.

[23] Niklas Kochdumper. 2022. *Extensions of polynomial zonotopes and their application to verification of cyber-physical systems*. Ph. D. Dissertation. Technische Universität München.

[24] Niklas Kochdumper and Matthias Althoff. 2020. Sparse polynomial zonotopes: A novel set representation for reachability analysis. *IEEE Trans. Automat. Control* 66, 9 (2020), 4043–4058.

[25] Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. 2022. Open- and closed-loop neural network verification using polynomial zonotopes. *arXiv preprint arXiv:2207.02715* (2022).

[26] Niklas Kochdumper, Bastian Schürmann, and Matthias Althoff. 2020. Utilizing dependencies to obtain subsets of reachable sets. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 1–10.

[27] Anna-Kathrin Kopetzki and Stephan Günnemann. 2021. Reachable sets of classifiers and regression models: (Non-)robustness analysis and robust training. *Machine Learning* 110, 6 (2021), 1175–1197.

[28] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J. Kochenderfer, et al. 2021. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization* 4, 3-4 (2021), 244–404.

[29] Kyoko Makino and Martin Berz. 2005. Verified global optimization with Taylor model-based range bounders. *Transactions on Computers* 11, 4 (2005), 1611–1618.

[30] Debasmita Mukherjee, Kashish Gupta, Li Hsin Chang, and Homayoun Najjaran. 2022. A survey of robot learning strategies for human-robot collaboration in industrial settings. *Robotics and Computer-Integrated Manufacturing* 73 (2022). https://doi.org/10.1016/j.rcim.2021.102231

[31] 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. *Proceedings of the ACM on Programming Languages* 6, POPL (2022), 1–33. https://doi.org/10.1145/3498704

[32] Eva Ostertagová. 2012. Modelling using polynomial regression. *Procedia Engineering* 48 (2012), 500–506.

[33] Luca Pulina and Armando Tacchella. 2010. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*. Springer, 243–257.

[34] Kenneth H Rosen. 2017. *Handbook of discrete and combinatorial mathematics*. CRC press.

[35] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[36] 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).

[37] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2018. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*.

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

[39] Marshall H. Stone. 1948. The generalized Weierstrass approximation theorem. *Mathematics Magazine* 21, 5 (1948), 237–254.

[40] Andrew H. Sung. 1998. Ranking importance of input parameters of neural networks. *Expert systems with Applications* 15, 3-4 (1998), 405–411.

[41] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations*.

[42] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. 2020. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*. Springer, 3–17.

[43] 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. *Advances in Neural Information Processing Systems* 34 (2021).

[44] 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.

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

[46] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems* 31 (2018).

[47] Jacek M. Zurada, Aleksander Malinowski, and Ian Cloete. 1994. Sensitivity analysis for minimization of input data dimension for feedforward neural network. In *Proceedings of IEEE International Symposium on Circuits and Systems*, Vol. 6. IEEE, 447–450.


# A EFFICIENT POLYNOMIAL EVALUATION

Subsequently, we provide the proofs for the naive approach in Sec. 3. The naive approach computes each term of the polynomial using polynomial maps (Sec. 3) and sums up the terms using the exact addition:

Proposition A.1. *(Exact Addition [23, Prop. 3.1.20]) Let* $\mathcal{PZ}_1 = \langle c_1, G_1, G_{I1}, E_1 \rangle_{PZ}$, $\mathcal{PZ}_2 = \langle c_2, G_2, G_{I2}, E_2 \rangle_{PZ} \subset \mathbb{R}^n$ *with a common identifier vector, then their exact addition is*

$$\mathcal{PZ}_1 \boxplus \mathcal{PZ}_2 = \langle c_1 + c_2, [G_1\ G_2], [G_{I1}\ G_{I2}], [E_1\ E_2] \rangle_{PZ}. \quad (40)$$

Proposition A.2. *(Naive Approach) Given a polynomial zonotope* $\mathcal{PZ} = \langle c, G, [], E \rangle_{PZ} \subset \mathbb{R}$ *with* $h > 0$ *generators and a polynomial*

*$p(x) = a_0 + \sum_{i=1}^{o} a_i x^i$, the polynomial evaluation can be obtained by*

$$\widetilde{\mathcal{PZ}} = p(\mathcal{PZ}) = a_0 + a_1\mathcal{PZ}^1 \boxplus \ldots \boxplus a_o\mathcal{PZ}^o. \tag{41}$$

*The number of generators of $\widetilde{\mathcal{PZ}} = p(\mathcal{PZ})$ is given by $\widetilde{H}(o) = \frac{(h+1)^{o+1}-1}{h} - (o+1) \in O(h^o)$. The computational complexity to make $\widetilde{\mathcal{PZ}}$ regular is $O(p(\mathcal{PZ})) + O(\texttt{compact}) = O(oph^o \log(h))$.*

Proof. *Soundness & Termination.* Follows directly from Prop. 2.10, 2.8, A.1, and (6). *Number of Generators.* Note that

$$\widetilde{\mathcal{PZ}} = \left\langle \widetilde{c}, \widetilde{G}, [], \widetilde{E} \right\rangle_{PZ} \quad \text{with } \widetilde{c} = a_0 + \sum_{i=1}^{o} a_i c^i, \tag{42}$$

$$\widetilde{G} = \left[ \overbrace{[a_1 G^{(1)}]}^{a_1\mathcal{PZ}^1.G}, \overbrace{[a_2\widehat{G}_1^{(2)} \; a_2\widehat{G}_2^{(2)} \; [a_2\overline{G}_1^{(2)} \; \ldots \; a_2\overline{G}_h^{(2)}]]}^{a_2\mathcal{PZ}^2.G} \right. \\ \left. \ldots \overbrace{[a_o\widehat{G}_1^{(o)} \; a_o\widehat{G}_2^{(o)} \; [a_o\overline{G}_1^{(o)} \; \ldots \; a_o\overline{G}_h^{(o)}]]}^{a_o\mathcal{PZ}^o.G} \right], \tag{43}$$

$$\widetilde{E} = \left[ \overbrace{[\mathcal{PZ}.E]}^{\mathcal{PZ}^1.E}, \overbrace{[\mathcal{PZ}.E \; \mathcal{PZ}^1.E \; [\overline{E}_1^{(2)} \; \ldots \; \overline{E}_h^{(2)}]]}^{\mathcal{PZ}^2.E} \right. \\ \left. \ldots \overbrace{[\mathcal{PZ}.E \; \mathcal{PZ}^{o-1}.E \; [\overline{E}_1^{(o)} \; \ldots \; \overline{E}_h^{(o)}]]}^{\mathcal{PZ}^o.E} \right], \tag{44}$$

where each matrix $i = 1 \ldots o$ within $\widetilde{G}, \widetilde{E}$ corresponds to the computation of $\mathcal{PZ}^i$. Let $H(i)$ return the number of generators of $\mathcal{PZ}^i$. With $H(1) = h$, $H(i)$, $i \geq 2$, is given by

$$\begin{aligned} H(i) &= \overbrace{H(1)}^{\mathcal{PZ}.G} + \overbrace{H(i-1)}^{\mathcal{PZ}^{i-1}.G} + \overbrace{h \cdot H(i-1)}^{\left[\overline{G}_1^{(i)} \; \ldots \; \overline{G}_h^{(i)}\right]} && \text{(Prop. 2.10)} \\ &= h + (h+1) \cdot H(i-1) \\ &= h + \Sigma_{j=1}^{i-1}(h+1)^j h && \text{(expand recursion)} \\ &= \left(\Sigma_{j=0}^{i-1}(h+1)^j\right) h \\ &= \frac{(h+1)^i - 1}{(h+1) - 1} h && \text{[34, Sec. 3.5]} \\ &= (h+1)^i - 1 \in O(h^i). \end{aligned} \tag{45}$$

Note that $H(1) = h = (h+1)^1 - 1$. Thus,

$$\begin{aligned} \widetilde{H}(o) &= \Sigma_{i=1}^{o} H(i) && \text{(Prop. A.1)} \\ &= \Sigma_{i=1}^{o}(h+1)^i - 1 && (45) \\ &= -o + (1-1) + \Sigma_{i=1}^{o}(h+1)^i \\ &= -(o+1) + \Sigma_{i=0}^{o}(h+1)^i \\ &= \frac{(h+1)^{o+1} - 1}{(h+1) - 1} - (o+1) && \text{[34, Sec. 3.5]} \\ &= \frac{(h+1)^{o+1} - 1}{h} - (o+1). \end{aligned} \tag{46}$$

*Computational Complexity.* The computational complexity of $\mathcal{PZ}^i = \text{poly}(\mathcal{PZ}, \mathcal{PZ}^{i-1})$ with $n = w = 1$, $i = 2 \ldots o$, is given by $O(ph^i)$ [23, Prop. 3.1.30: (3.24)], where we do not apply assumption 3.1.3 in the last line of (3.24) and do not call `compact` immediately. Thus, $\mathcal{PZ}^o$ can be computed in $O(ph^o)$, as $\sum_{i=2}^{o} O(ph^i) \in O(ph^o)$ (Prop. 2.10). Further, $a_i\mathcal{PZ}^i$ can be computed in $O(ph^i)$ using (45) and the exact additions in $O(ph^o)$ using (46). Finally, `compact` is computed in $O(ph^o \log(h^o))$ using (46). □

# B VERIFICATION TRICKS

In this section, we show some tricks to improve the verification in certain scenarios.

*Example B.1.* Consider a classification task with two outputs, thus, the network should output a larger value for the correct class 1 for all elements in an input set $\mathcal{X}$. Thus, the unsafe set is

$$\mathcal{S} = \left\{ y \in \mathbb{R}^2 \;\middle|\; y_{(2)} > y_{(1)} \right\}. \tag{47}$$

Let

$$\mathcal{Y} = \left\langle \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, [], [1] \right\rangle_{PZ} \tag{48}$$

be the computed output set of a neural network for $\mathcal{X}$. The property $\mathcal{Y} \cap \mathcal{S} = \emptyset$ holds for $\mathcal{Y}$. However, in complex polynomial zonotopes we have to check the interval bounds of $\mathcal{Y}$ (Prop. 2.7), obtaining $[0, 2]$ for the first dimension and $[-1, 1]$ for the second dimension. As these intervals overlap, we cannot guarantee safety.

We can apply the following proposition to mitigate such overlapping examples in classification tasks by transforming the output space and making the specifications $\mathcal{S}$ axis-aligned.

Proposition B.2. *(Argmax Trick) Given an output set $\mathcal{Y} \subset \mathbb{R}^{v_K}$ for a classification task with $v_K$ classes and unsafe specifications $\mathcal{S} = \left\{ y \in \mathbb{R}^{v_K} \;\middle|\; \exists w \in \{1, \ldots, v_K\} : y_{(w)} > y_{(w_0)} \right\}$, where the prediction for the correct label is given in dimension $w_0 \in \{1, \ldots, v_K\}$. Then, we can transform the output space by a multiplication with a matrix $M \in \mathbb{R}^{v_K \times v_K}$ such that*

$$M\mathcal{S} = \left\{ y \in \mathbb{R}^{v_K} \;\middle|\; \exists w \in \{1, \ldots, v_K\} : y_{(w)} > 0 \right\} \tag{49}$$

*without additional over-approximation, where*

$$M_{(w_1, w_2)} = \begin{cases} 1, & w_1 = w_2 \wedge w_1 \neq w_0 \\ -1, & w_2 = w_0 \wedge w_1 \neq w_0 \\ 0, & \textit{otherwise} \end{cases} \tag{50}$$

*with $w_1, w_2 = 1 \ldots v_K$.*

Proof. Follows by construction of $M$ and Prop. 2.8. □

*Example B.1. (cont.)* Using Prop. B.2, the interval bounds of $M\mathcal{Y}$ are $[0, 0]$ and $[-1, -1]$. Note that the interval for dimension $w_0$ is always $[0, 0]$ but the interval for all other dimensions can be arbitrary.

Proposition B.3. *(Masking) Given an output set $\mathcal{Y} \subset \mathbb{R}^{v_K}$, specifications $\mathcal{S} \subset \mathbb{R}^{v_K}$, and a dimension $w'$ where all specifications are verified w.r.t. $\mathcal{S}$. Then, we can ignore the dimension $w'$ in future iterations by transforming the output space by a multiplication with a matrix $M \in \mathbb{R}^{v_K \times v_K}$, where*

$$M_{(w_1,w_2)} = \begin{cases} 1, & w_1 = w_2 \wedge w_1 \neq w' \\ 0, & \textit{otherwise} \end{cases} \tag{51}$$

*with $w_1, w_2 = 1 \dots v_K$, $M \in \mathbb{R}^{v_K \times v_K}$, and remove the verified specifications from $\mathcal{S}$. Thus, the time to check the specifications is decreased and sensitivity analysis only considers neurons contributing to violated dimensions in future iterations.*

Proof. Follows directly from Prop. 4.14 and Def. 2.3. □

Note that both, Prop. B.2 and B.3, can be implemented by appending a linear layer at the end of the network using $M$ as weight matrix and $\mathbf{0}$ as bias.